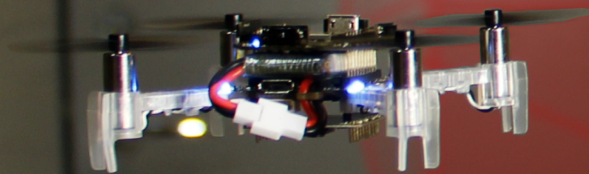
BACHELOR'S THESIS 2022

Prototype Circuit Board with Multiple Simultaneous Multizone Solid-State Lidars for Small Robots

Victor Krook



ISSN 1651-2197
LU-CS/HBG-EX: 2022-03

DEPARTMENT OF COMPUTER SCIENCE
LTH SCHOOL OF ENGINEERING AT CAMPUS HELSINGBORG

Prototype Circuit Board with Multiple Simultaneous Multizone Solid-State Lidars for Small Robots

Prototyp Kretskort med Flera Simultan-Multizone Solid-State Lidar för Små Robotar

Victor Krook

LUND UNIVERSITY
Campus Helsingborg

LTH School of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor's thesis work carried out at Bitcraze.

Supervisor: Bertil Lindvall, bertil.lindvall@eit.lth.se

Deputy Supervisor: Kristoffer Richardsson, Bitcraze AB, kristoffer@bitcraze.io

Examiner: Christin Lindholm, christin.lindholm@cs.lth.se

© Copyright Victor Krook

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2022

Abstract

Lidars have been getting more attention in recent years and are popular sensors among autonomous vehicles. Recent technology advancements have enabled solid-state (non-moving) lidars to simultaneously measure distances to multiple zones to output a 2D matrix of distances. This can help vehicles create a 3D map of its surrounding environment, which can be especially useful for small robots and drones since the sensors can be lightweight and small. Having lightweight sensors is a necessity for tiny drones such as the Crazyflie, which is a pocket-sized drone developed by the Swedish company Bitcraze, and is used among universities and enthusiasts world-wide.

During this bachelor's thesis project, a prototype circuit board is built, which has five simultaneous multizone solid-state lidars, a microcontroller, and can be mounted on the Crazyflie. Having multiple simultaneous multizone solid-state lidars oriented in different directions on the drone, can allow it to get one step closer to becoming fully autonomous. The project is done in collaboration with Bitcraze, whom have aided the project financially, in the form of materials, as well as with their expertise.

The making of the prototype is divided into four phases: 1) The requirements of the prototype are set, and a list of the main necessary components is made. 2) An electronic schematic is created and a printed circuit board is designed. The circuit board and the components are then ordered. 3) Once the circuit board and the components have arrived, the prototype is assembled. This is followed by writing firmware for the microcontroller and writing software for the ground control station. The software on the ground control station allows wireless communication to the microcontroller, and offers three different methods to visualize the distance data. 4) An evaluation of the prototype is given, where measured results are presented. Then, the thesis questions are answered, which is followed by a discussion of the results.

After these four phases, a final conclusion of the thesis is presented, and at last, a discussion of future work and improvements for the prototype is given.

The prototype built during this thesis was able to collect distance data from the sensors and send it to a ground control station, which could then be visualized in two or three dimensions. Thus, this bachelor's thesis can be considered successful.

Keywords: BSc, Multizone, ToF, SSL, lidar, prototype, Crazyflie, ESP32

Sammanfattning

Lidar har fått mer uppmärksamhet de senaste åren och är populära sensorer bland autonoma fordon. De senaste tekniska utvecklingarna har gjort det möjligt för solid-state (icke-rörliga) lidar att simultant mäta avstånd till flera zoner, för att mata ut en 2D-matris av avstånd. Detta kan hjälpa fordon att skapa en 3D-karta över den omgivande miljön och kan vara särskilt användbart för små robotar och drönare, eftersom sensorerna kan vara lätta och små. Att ha lätta sensorer är en nödvändighet för små drönare som Crazyflie, vilket är en drönare i fickstorlek, utvecklad av det svenska företaget Bitcraze, och används bland universitet och entusiaster världen över.

Under detta examensarbete byggs ett prototypkretskort, som har fem simultan multizon solid-state lidar, en mikrokontroller, och kan monteras på en Crazyflie. Att ha flera simultan multizon solid-state lidar orienterade i olika riktningar på drönaren, kan tillåta den att komma ett steg närmare att bli helt autonom. Projektet görs i samarbete med Bitcraze, som har bistått projektet ekonomiskt, i form av material, såväl som med sin expertis.

Tillverkningen av prototypen är uppdelad i fyra faser: 1) Kraven på prototypen fastställs och en lista över de primära nödvändiga komponenterna görs. 2) Ett elektroniskt schema skapas och ett kretskort utformas. Därefter beställs kretskortet och komponenterna. 3) När kretskortet och komponenterna har mottagits, byggs prototypen. Detta följs av att skriva firmware för mikrokontrollern och skriva programvara för markkontrollstationen. Programvaran på markkontrollstationen möjliggör trådlös kommunikation till mikrokontrollern och erbjuder tre olika metoder för att visualisera avståndsdatan. 4) En utvärdering av prototypen ges, där mätresultat presenteras. Därefter besvaras examensarbetets frågeställningar som följs av en diskussion om resultaten.

Efter dessa fyra faser presenteras en slutsats av examensarbetet och till sist diskuteras framtida arbete och förbättringar av prototypen.

Prototypen som byggdes under detta projekt kunde samla in data från sensorerna och skicka den till en markkontrollstation, som sedan kunde visualiseras i två eller tre dimensioner. Således kan detta examensarbete för högskoleingenjörer anses vara lyckat.

Nyckelord: Kandidatarbete, Multizone, ToF, SSL, lidar, prototyp, Crazyflie, ESP32

Acknowledgements

I would like to give a huge thanks to Bitcraze, whom have allowed me to work on such an exciting project, which is completely aligned with my personal interests. Being able to work on a project full-time, that has included going from an idea to a final, working prototype, has been truly inspiring, challenging and educational. I also want to thank Bitcraze for always taking their time to discuss and answer any of my questions, inspiring me to becoming a better engineer, as well as for financing the project.

I would like to thank my thesis Supervisor Bertil Lindvall, for taking on a final thesis project, before his retirement. Bertil held our introductory course in digital systems, which sparked my interests for embedded systems.

Thank you Christin Lindholm, my Program Director, for being my thesis Examiner, even though I know you already had too much to do.

I would also like thank my girlfriend Josefine, for answering my academic questions, listening to my frustration, and giving wise advice when needed.

At last, I would like to take this opportunity to express my gratitude to my family for their support, and specifically to my young niece Ella, whom I hope can benefit from the amazing technology advancements that the future holds.

Preface

This bachelor's thesis presents the work that has been carried out during a time period of 10 weeks, in the spring of 2022, at LTH School of Engineering at Campus Helsingborg. The project has been done in collaboration with Bitcraze, whom have supported the project financially, in the form of materials, as well as with their expertise. The reader is expected to have a background in Computer Science & Computer Engineering.

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Goal . . . 3
1.3 Motivation . . . 3
1.4 Constraints . . . 4
1.5 Related Work . . . 4

2 Methodology **5**
2.1 Benefits & Downsides of Prototyping . . . 5
2.2 Prototype Model . . . 5
2.2.1 Phase 1 - Prototype Requirements . . . 6
2.2.2 Phase 2 - Prototype Design . . . 7
2.2.3 Phase 3 - Building the Prototype . . . 7
2.2.4 Phase 4 - Prototype Evaluation . . . 8
2.3 Evaluation of Thesis Sources . . . 8

3 Theoretical Background **11**
3.1 Basic Electronics . . . 11
3.2 Common Communication Protocols . . . 15
3.3 FreeRTOS . . . 17
3.4 ESP32 . . . 17
3.5 KiCad . . . 18

4 Crazyflie **19**
4.1 The Crazyflie . . . 20
4.2 Coordinate System of the Crazyflie . . . 21
4.3 Radio . . . 21
4.4 CRTP . . . 23
4.5 Client Software . . . 24
4.6 Expansion Decks . . . 24
4.6.1 Multi-Ranger Deck . . . 25
4.6.2 Flow Deck . . . 25
4.6.3 AI Deck . . . 26
4.6.4 Expansion Deck Connection Pins . . . 26
4.7 CPX - Crazyflie Packet eXchange . . . 27
4.7.1 Transport - UART . . . 29

4.7.2 Transport - SPI 30
4.7.3 Transport - WiFi 30
4.7.4 Packet Splitting 31

5 Prototype Requirements **33**
5.1 Formalized Requirements 33
5.1.1 Deck Size and Connections 34
5.1.2 VL53L5CX (SSL Sensor) 35
5.1.3 ESP32-S3 (MCU) 39
5.1.4 TPS62A01 (DC-DC Converter) 40
5.1.5 Other Components 41
5.2 Main Components List 41
5.3 Prototype Name - Slamdeck 42

6 Prototype Design **43**
6.1 Schematic 43
6.1.1 USB & UART 45
6.1.2 Power-Chain 45
6.1.3 VL53L5CX 47
6.1.4 LEDs 48
6.1.5 Button 49
6.1.6 1-Wire Memory 49
6.1.7 Expansion Connection Pins 49
6.1.8 ESP32-S3 GPIO Connections 50
6.1.9 Satellite Board 52
6.2 PCB 52
6.3 Order Components 55

7 Building the Prototype **57**
7.1 Assembly 57
7.2 Firmware 62
7.2.1 ESP32-S3 62
7.2.2 Crazyflie 67
7.2.3 Working Example 67
7.3 Software 69
7.3.1 Data Visualization 70

8 Prototype Evaluation **77**
8.1 Functionality 77
8.2 Results 78
8.2.1 Initialization Time 78
8.2.2 Operating Temperature 78
8.2.3 Distance Data 79
8.2.4 Power 79
8.2.5 Data Rates 79
8.2.6 GUI & Data Visualization 80
8.2.7 Price 84

8.3 Answers to Thesis Questions 84
8.4 Discussion 85

9 Conclusion **89**
9.1 Ethical Reflection 90

10 Future Work **91**

Appendix A VL53L5CX Settings **99**
A.1 Run-Time Configurable Settings 99
A.2 Compile-Time Configurable Settings 101

Appendix B Firmware **103**
B.1 Slamdeck 103
B.2 Slamdeck API 106

List of Figures

2.1 The four phases used in order to build and evaluate the prototype. 6

3.1 Simplified structure of a normal diode. 12
3.2 Simplified working principles of a laser diode. 13
3.3 Simplified differences between EELs and VCSELs. 13
3.4 Simplified working principles of a FET. Current is flowing from drain to source. 14
3.5 Pull-up and pull-down resistors. High (digital 1) is at 5V and low (digital 0) is at ground. 14
3.6 Building and flashing code to an ESP32, directly from Visual Studio Code. . 18

4.1 Crazyflie eco-system overview. Image taken from [1], with Bitcrazes permission. 19
4.2 Crazyflie drone, weighs only 27g. Image taken from [2], with Bitcrazes permission. 20
4.3 The coordinate system of the Crazyflie. The front of the drone is in the direction of the x-axis. Image taken from [3], with Bitcrazes permission. . . 21
4.4 Crazyradio PA USB dongle. Image taken from [4], with Bitcrazes permission. 22
4.5 ESB ACK containing payload. 22
4.6 CRTP packet structure. 23
4.7 Crazyflie software stack. Image taken from [5], with Bitcrazes permission. . 24
4.8 Multi-ranger deck with 5 VL53L1X sensors mounted in directions: front, right, back, left and up. Image taken from [6], with Bitcrazes permission. . . 25
4.9 Flow deck v2. Image taken from [7], with Bitcrazes permission. 25
4.10 AI deck with GAP8 RISC-V MCU, ESP32 and an on-board camera. Image taken from [8], with Bitcrazes permission. 26
4.11 Crazyflie expansion connection pins. Image taken from [9], with Bitcrazes permission. 26
4.12 CPX communication stack. Image taken from [10], with Bitcrazes permission. 28
4.13 CPX packet structure. Destination and source are both `target`s, and can thus be set to any of the values in table 4.3. The `function` (bits 8-15) can be given any value from table 4.4. 28
4.14 CPX UART packet structure. Image taken from [10], with Bitcrazes permission. 29
4.15 CPX SPI packet structure. Image taken from [10], with Bitcrazes permission. 30
4.16 CPX WiFi packet structure. Image taken from [10], with Bitcrazes permission. 30

4.17 Simple example of how CPX splits a packets that is bigger than the MTU. A benefit of CPX is that the sender (Application in the figure), does not have to care about the size of the packet when sending data. This is all handled by CPX. 31

5.1 Expansion deck template, provided by Bitcraze [11]. As one can see, there is only about 16.8mm between the expansion connection pins. 34
5.2 Simplified working principles of the VL53L5CX sensor. The yellow dashed arrow represents a travelling laser beam that is reflected by the target to the DOE, which then hits the SPAD array. 35
5.3 VL53L5CX effective orientation. The output 2D matrix is flipped vertically and horizontally. The resolution in the figure is 4x4. The numbers in each zone is the index of the measurement result data. 36
5.4 VL53L5CX dimensions. 36
5.5 VL53L5CX pins. 36
5.6 ESP32-S3 modules. Module one and two has an on-board antenna, and only module number two has solder pads exposed. 40

6.1 How the satellite board was expected to look. A separate PCB, with the VL53L5CX sensor and a few passive components. 43
6.2 The satellite board would be mounted on the main board through connection holes, and then soldered in place. 44
6.3 Prototype schematic overview. Each satellite block to the right should result in a separate satellite board, except the sensor facing up, which should be on the main board. 45
6.4 Voltage selector using a PMOS transistor, schottky diode and pull-down resistor. `VCOM_CF` is the voltage of the battery on the Crazyflie, and `VBUS` is the voltage from the USB. 46
6.5 The red arrows indicate current flow. (a) When USB is connected, the transistor is opened, and no current flows from the battery. Instead, current flows from USB, and due to the schottky diode, there is a ~0.4 V voltage drop. (b) When USB is *not* connected, the gate of the transistor is pulled down to ground and thus allows current to flow from the battery through the transistor. 46
6.6 Complete power-chain of the prototype. The output of the voltage selector (`VSYS`) is fed into the DC-DC converter. On the output of the DC-DC converter there are feedback resistors (`R5` and `R19`) that sets the correct output voltage. At the end of the output voltage, there is a big 22 µF capacitor (`C5`) to keep the 3V stable. `D4` is a TVS diode, to protect the USB from voltage spikes. 47
6.7 VL53L5CX schematic with the necessary passive components. VCC is 3V. Capacitors `C7` and `C8` are decoupling capacitors, which are used to ensure stable voltage levels at the sensors input power pins. Lines `SENSOR_1`, `SCL` and `SDA` are connected to the ESP32-S3. 48
6.8 Overview of the prototype from above. The main board contains the ESP32-S3, expansion connections pins, and all satellite boards. 52

6.9 Working principles of a via. These are used to connect different layers on a PCB. The insulation is colored green in the figure. 53
6.10 Main PCB board front view, in the 3D viewer of PcNew. Most small components have their reference id next to them. 53
6.11 Main PCB board front view with description of certain components. OW is the 1-Wire memory. 54
6.12 Main PCB board back view, in the 3D viewer of PcNew. Note the UART solder pads in the top left corner, which were placed there due to lack of space on the top side, while trying to make them easily accessible. 54
6.13 Satellite PCB board front view, in the 3D viewer of PcNew. 55
6.14 Satellite PCB board back view, in the 3D viewer of PcNew. 55

7.1 Prototype main PCB. (a) Front side. (b) Bottom side. 57
7.2 Prototype satellite PCB. (a) Front side. (b) Bottom side. 58
7.3 Workbench at Bitcraze, where the prototype was assembled, with the InteractiveHtmlBom plugin running on the laptop. 58
7.4 Most of the component soldered on the circuit board. The ESP32-S3 is in the middle, and the VL53L5CX sensor is on the top right of the image, still not soldered. 59
7.5 ESP32-S3 first successful USB connection. The ESP32-S3 registers itself as a new USB device, and is available through a serial port such as `/dev/ttyACM0` (on Linux). 59
7.6 Testing the LEDs on the deck. 60
7.7 Complete prototype deck after assembly, top view. 61
7.8 Complete prototype deck after assembly, side view. 61
7.9 ESP32-S3 firmware architecture overview. 62
7.10 A typical I^2C transaction with the VL53L5CX. The numbers below the squares indicate bit index. Note that the transaction is slightly simplified, and does not include ACKs/NACKs, which are necessary for I^2C communication. . . 63
7.11 Simplified overview how CPX should work on the prototype. WiFi and API are two separate software modules on the ESP32-S3. Note that the router is not a physical device, but a software module. 64
7.12 Flow chart of getting sensor data from the Slamdeck API to the GCS, via WiFi. The italic text under the arrows indicates what the underlying data link is, and GCS, WiFi, Router and API can be seen as separate software modules. 65
7.13 Flow chart of getting sensor data from the Slamdeck API via the nRF on the Crazyflie. The process is slightly simplified, as the data is actually sent between the nRF51822 and STM32 as well. Note that the right part (highlighted) remains the same as in figure 7.12. The italic text under the arrows indicates what the underlying data link is. GCS, WiFi, Router and API can be seen as separate software modules. 65
7.14 WiFi Driver overview. *Tasks*, refers to a FreeRTOS task. 66
7.15 UART driver overview. 66

7.16 Working example of the firmware when a ground control station wants to read distance data from the sensors. The solid black lines indicate that the data goes *from* the GCS, and dashed black lines indicate that the data goes *towards* the GCS. 68
7.17 GUI Overview. Each box represents a separate software module. Referring to the MVC architecture; the *Models* are the gray boxes to the left, the *Controller* is the blue box in the middle, and the *View* is the green box at the top. 69
7.18 Simplified example of 2D matrix visualization. A hand is covering half of the sensors FoV. The darker red in the 2D matrix means that an object is closer to the sensor. 70
7.19 Simplified example of 2D matrix visualization with all five sensors. The hand is completely covering the FoV of the sensor facing up, and the other sensors has not detected any objects at all. 71
7.20 2D Point cloud, using only the top row of the distance data (left), or using only the bottom row of the distance data (right). 72
7.21 2D Point cloud, using row 1 and 2. The distance data is averaged between the rows, for each column. 73
7.22 2D distance data matrix projected to 3D around the sensors local coordinate system. The third dimension is given value 1 for each cell in the matrix. Note that the origin (0,0) is in the center of the matrix, thus the y and z components of the zones are not integers. 74
7.23 3D point cloud visualization in VisPy. All points are projected in the Crazyflies local coordinate system. 75

8.1 Screenshot of GUI, visualizing the distance data that is read from the prototype deck, which is mounted on the Crazyflie. 80
8.2 2D matrix visualization with the sensor facing front. The values in the grids are the distances in millimeters. (a) A hand is placed close to the front sensor. (b) The hand is moved away from the sensors FoV and the values takes some time to update to match the distances to the wall. 81
8.3 2D Matrix visualization result for all five sensors. Red grids indicates shorter distances. (a) A hand is put on top of the deck. (b) A hand is put to the back of the deck. 81
8.4 2D point cloud visualization result. With the menu to the left it is possible to select which of the rows of distance data that should be displayed, as explained in section 7.3.1. 82
8.5 3D point cloud visualization result. (a) The Crazyflie is flying away from the camera. (b) The Crazyflie has flown away from the camera, and entered another room. 83
8.6 3D point cloud visualization when the Crazyflie is facing a corner. Notice that the bulge on the wall to the right of the Crazyflie can be seen in the 3D visualization. However, the corner to the left of the drone is *not* seen in the visualization, due to the sensors having a FoV of 45°, and thus missing this area. 83

A.1 VCSEL emission timeline. Resolution is 4x4, and ranging mode is Autonomous. 100

A.2 VCSEL emission timeline. Resolution is 8x8, and ranging mode is Autonomous. 100
A.3 Example result of different sharpener settings with a given scene. The resolution is 4x4. One target is centered in the FoV at distance 100 mm, and another target further is 500 mm behind it. 101
A.4 Example of Xtalk margin setting. The brown blocks represents objects in front of the sensor. The margin is set to 400 mm, which means that the sensor will ignore distances that are below 400 mm. 102

B.1 Slamdeck API example. The GCS requests data through the *GET_DATA* command, and Slamdeck returns the data. 106

List of Tables

4.1 Crazyflie main components. 20
4.2 Some of the ports used in CRTP. 23
4.3 Available CPX targets. 28
4.4 Available CPX functions. 29

5.1 Available fields in a Ranging result of the VL53L5CX. *N* stands for number of Targets per zone. 38
5.2 Estimated max current consumptions for the main components on the deck. 41
5.3 Main components of the prototype deck. 42

6.1 Crazyflie expansion connection pins, left side. 49
6.2 Crazyflie expansion connection pins, right side. 50
6.3 Connections of the ESP32-S3 on the prototype deck. "-" means that the pin is not connected to anything and left floating. *Expansion pins* refers to the expansion connection pins on the Crazyflie. *Sensor enable* refers to the LPn pin of each sensor. 51
6.4 VL53L5CX connections pins on the satellite board. 52
6.5 All components required for the main board. 56
6.6 All components required for the satellite board. 56

8.1 Approximate price of all components on the prototype deck. Note that only the ESP32-S3 and the VL53L5CX are listed separately, which is because the other components are typically cheap, and thus grouped together. The price of shipping varies depending on the shipping time. 84

A.1 Maximum and minimum ranging frequencies, at different resolutions. . . . 99

B.1 Data structure that represents the settings of the sensor. 104
B.2 Possible data lengths (in bytes) for a single data reading from the VL53L5CX sensor, with different values for the resolution, and Targets per zone. 105
B.3 Example data buffer, with resolution = 8x8, and Targets per zone = 1 105

Glossary

ACK An acknowledgment signal that can be used between microcontrollers or computers. 22, 63, 104

AGI Artificial General Intelligence. 1, 90

AI Artificial Intelligence. 1, 26, 90, 92

API Application Programming Interface. 17, 24, 62, 64, 65, 68, 78, 106

BJT Bipolar Junction Transistor. 13, 15

BLE Bluetooth Low Energy. 21, 24, 39

Cfclient GUI that is used to configure and control the Crazyflie. 24, 69, 87

Cflib Python library to control the Crazyflie from a PC. 24, 70, 75

CPU Central Processing Unit. 15, 39

CPX Crazyflie Packet eXchange. 26–31, 62, 64, 66–68, 77–79, 84

CRTP Crazy RealTime Protocol. 23, 27, 29, 67, 79, 85

CS Chip Select. 16, 30

CTS Clear To Send. 16, 29

DOE Diffractive Optical Element. 35

EEL Edge-Emitting Lasers. 13

EEPROM Electrically Erasable Programmable Read-Only Memory. 42, 49, 50

ESB Enhanced ShockBurst. 22

FET Field-Effect Transistor. 13–15

Footprint Arrangements of pads or through-holes of a component. 56, 91

FoV Field of View. 2–4, 35, 70–72, 81, 83, 86, 88, 89, 101

FreeRTOS Popular open-source Real-Time Operating System. 17, 66

GCS Ground Control Station. 3, 7, 21, 23, 39, 62–65, 67, 68, 77–79, 85–87, 92, 106

GNU GPLv3 The GNU General Public License version 3 - free, copyleft license for software and other kinds of works. 78

GPIO General Purpose Input/Output. 27, 41, 42, 44, 48, 50, 64, 103

GUI Graphical User Interface. 24, 57, 69, 70, 75, 78, 80, 87

I^2C Inter-Integrated Circuit. 15, 16, 25, 27, 37, 38, 47, 48, 50, 52, 60, 63, 77, 84, 85, 87, 91, 103, 104

I/O Input/Output. 15, 59

IC integrated circuit. 15

IEEE Institute of Electrical and Electronics Engineers. 9

IoT Internet of Things. 17

ISM Industrial, Scientific, Medical. 15, 22

JTAG Industry standard that is used to verify designs and test PCBs after manufacture. It can also be used to flash and debug embedded devices. 39

KiCad Free open-source Schematic Capture & PCB Design Software. 7, 18, 58

LED Light Emitting Diode. 4, 12, 15, 33, 41, 44, 48, 51, 60, 62, 64

lidar A type of ToF method that uses laser to measure distances. 1–4, 33, 88, 89

MCU Microcontroller. 3, 4, 7, 17, 20, 21, 26, 27, 33, 35, 37, 39, 42, 56, 58, 64, 67, 84

MISO Master In Slave Out. 16, 50, 51

MOSFET Metal-Oxide-Silicon Field-Effect Transistor. 14, 20

MOSI Master Out Slave In. 16, 50, 51

MTU Maximum Transmission Unit. 27, 29–31

MVC Model View Controller. 69

NMOS N-channel Metal-Oxide Semiconductor. 14

nRF nRF Radio Link on the Crazyflie. 62, 64, 65, 67, 69, 78, 79, 85

OS Operating System. 17

OW 1-Wire. 27, 50, 54

PCB Printed Circuit Board. 7, 18, 39, 40, 43, 44, 52–55, 57, 58

PcNew PCB design software, as a part of KiCad. 18, 53–55

PMOS P-channel Metal-Oxide Semiconductor. 14, 45, 46, 56

PWM Pulse Width Modulation. 15, 20

PyQt Python QT bindings. 24, 69, 73, 87

radar A type of ToF method that uses radio waves to measure distances. 1

Ranging result The result data of a single reading from the VL53L5CX sensor. 37, 38, 63, 78, 84–87, 91, 105

ROM Read Only Memory. 39, 59

RTOS Real-Time Operating System. 17, 85

RTS Ready To Send. 16, 29

SCL Serial Clock. 16, 47, 49, 51, 52

SCLK Serial Clock. 16, 30

SDA Serial Data. 16, 47, 49, 51, 52

SDLC Software Development Life Cycle. 5

SLAM Simultaneous Localization and Mapping. 2, 3, 42, 90

SoC System on Chip. 15, 17, 21, 39, 41

SPAD Single-Photon Avalanche Diode. 12, 35, 38

SPI Serial Peripheral Interface. 15–17, 21, 22, 27, 29, 30, 39, 50, 67, 84, 85

SS Slave Select. 16, 30

SSL Solid-State Lidar. 2–4, 7, 25, 33, 35, 42, 56, 84, 85, 89

ST STMicroelectronics. 2, 4, 35, 63, 99

Target status Measurement of the validity of the distance data. 37, 63, 86, 91

Targets per zone How many targets are measured for each zone on the VL53L5CX sensor. 37, 38, 79, 87, 101, 105

ToF Time-of-Flight. 1, 2, 4

TVS Transient Voltage Suppression. 12, 47, 56

UART Universal Asynchronous Receiver Transmitter. 15, 16, 27, 29, 39, 44, 45, 49, 50, 59, 62, 64, 66, 67, 77, 79, 84, 85

UAV Unmanned Aerial Vehicle. 2, 3, 19, 89, 90, 92

UI User Interface. 70, 78

VCSEL Vertical Cavity Surface Emitting Laser. 13, 35, 38, 43, 99, 100

1
Introduction

This chapter gives an introduction to this thesis work and is structured as follows: First, a brief overview of relevant background is given, which is followed by the thesis goal being presented. After this, the thesis motivation is given, and this is followed by a discussion of the constraints of the thesis work. At last, the chapter ends with a discussion of related work.

1.1 Background

The interest for autonomous vehicles has never been bigger, and this interest is further increased as advancements are made in the fields of Machine Learning and Artificial Intelligence (AI). Having fully autonomous vehicles can lead to benefits that includes saving lives and money [12], but can also be seen as an important step towards Artificial General Intelligence (AGI). However, while the algorithms and computations behind the AI is often what gets the most attention, it is important to not only focus on this, but also on the hardware that is responsible for collecting the data that is fed to the AI. There is a saying in Computer Science that goes: "Garbage In, Garbage Out", which refers to the fact that even if one were to have the most perfect machine to perform the most perfect computation, if the input data to the machine is bad, the output can only get so good [13]. In the world of autonomous vehicles, this means that if a vehicles sensors collect bad data, the vehicle will have a harder time to be fully autonomous, or even function at all.

For autonomous vehicles the input signals that are fed to the AI are usually gathered from image cameras, lidar and radar [14]. A disadvantage with image cameras is that the gathered data is a two-dimensional representation of the three-dimensional world. In order to *estimate* the third dimension, smart image processing algorithms must be used, and even though image processing techniques are getting better, lidar and radar can still be advantageous due to the fact that they are *measuring* distance.

Time-of-Flight (ToF) is a technique where the distance to an object can be measured by sending out a wave and measuring the time it takes for the wave to reflect back. The wave could in theory be of any medium; for instance a sound-wave or a light-wave. Lidar stands for *light detection and ranging*, and is a type of ToF that works by emitting laser-waves (light) to

measure distances, and with the knowledge of the speed of light, the distance that the wave has traveled can then be determined. There are mainly two types of lidars available:

Rotating Lidar (mechanical) - The lidar is constantly physically rotated with the help of mechanics, in order to cover a Field of View (FoV) up to 360°. The mechanical rotation causes vibrations that the components must be able to handle, and this generally results in bigger, more expensive, and higher quality components [15]. The mechanical rotations also tends to wear the components faster than non-mechanical lidars.

Solid-State Lidar (SSL) (non-mechanical) - A non-rotating lidar that measures distances in a fixed direction, typically covering a FoV of 120° or less [15].

Due to the complexity and size constraints of mechanical lidars, they can be unpractical, or even impossible to use on smaller vehicles, such as small drones and other Unmanned Aerial Vehicles (UAVs). To cover a wider FoV without the use of rotating lidars, one option would be to combine multiple SSLs, that together covers a wider FoV. This way, it could still be possible to cover a FoV up to 360°, without mechanically rotating lidars.

Normal SSLs emit a beam of light that measures the distance to a single point in the environment, and the size of this point depends on the size of the beam. While this allows for measurements of distance to a certain point in space, it is necessary to have multiple SSLs in the same direction if it is desired to detect motion, or to create a more detailed 3D map of the environment in front of the SSL. However, recent technology advancements has resulted in SSLs that can measure distances to multiple points, herein referred to as *zones*, simultaneously, within a given FoV. This enables the SSLs to output a 2D matrix of distances. While SSLs could offer multizone output data prior to this breakthrough, the sensors were not able to collect data from more than one zone at the same time, thus they were not simultaneous. In 2021, STMicroelectronics (ST) released the VL53L5 sensor, which is a simultaneous multizone SSL, which can measure distances up to 4 meters, within a 45° FoV, and output a 2D matrix of distances, with an adjustable resolution of 4x4 or 8x8 [16]. Furthermore, the company ams OSRAM, announced in 2021 that they would release a simultaneous multizone ToF sensor TMF8828 [17], which just like the VL53L5, can measure distances in a matrix of 8x8 within a 45 ° FoV.

A benefit of having simultaneous multizone distance data, instead of distances to a single point, is that it can be easier for an autonomous vehicles to create a 3D map of its surrounding environment. Creating a map of an unknown environment is one part of Simultaneous Localization and Mapping (SLAM), which can be described as the problem for a mobile robot, to be "placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map" [18]. Solving this problem could be seen as one of the biggest breakthroughs for autonomous vehicles, since it would allow a robot to know where it is in an environment (location), and how the environment around it looks (map).

Thus, new technologies such as simultaneous multizone solid-state lidars, can become a vital part for autonomous vehicles and robots in the future. For small UAVs, like drones, being fully autonomous can enable the drones to be useful tools that can help humanity. For instance, Everdrone is a company that creates drones that aims to deliver medical equipment in emergency situations [19]. This could save lives in situations where ambulances have a hard time to get to the target destination. Bitcraze is another company that creates drones, but

primarily for research and educational purposes. Bitcrazes drone; Crazyflie, is a small pocket-sized drone that is used by universities world-wide, and comes with a complete open-source flying development platform, which enables researchers to quickly make progress in fields of UAV autonomy. This thesis project is done in collaboration with Bitcraze.

1.2 Goal

The goal of the thesis is to make a prototype circuit board (also referred to as *deck*) with multiple simultaneous multizone solid-state lidars, which can be mounted on the Crazyflie. There should also be a microcontroller (MCU) on-board the circuit board, which collects the data from the sensors and sends it to a Ground Control Station (GCS), as well as some other components.

In this thesis, the following questions will be answered:

1. What components are necessary on the deck?
2. What communication protocol should be used between the Crazyflie and the deck?
3. Can the data from the SSLs be gathered in real-time and sent to a GCS?
4. Can the data from the SSLs be sent through the existing radio link between the Crazyflie and the GCS, or is it necessary to use another data link, such as WiFi?

If this can be accomplished within the time of the thesis, methods for visualizing the data should also be developed.

1.3 Motivation

While there are several boards and modules that includes multiple SSLs that together covers a wide FoV on the market today, there are very few, if any, that has multiple SSLs which offer simultaneous multizone output data. Furthermore, these boards and modules are typically rather big, expensive or too heavy for smaller UAVs and robots. Because of this, this thesis aims to build a prototype that combines multiple simultaneous multizone SSLs, that together cover a wide FoV, with components that are affordable, even for makers and technology enthusiasts.

Since the project is done in collaboration with Bitcraze, if the project is successful, it could lead to a real product, which could help developers and researchers to make more progress in fields such as autonomous driving or flying. Having a 2D matrix of distance data from multiple directions of a vehicle, could be useful for algorithms that regards SLAM, object detection and/or object avoidance. Thus, a circuit board with multiple simultaneous multizone SSLs, which can be mounted on the Crazyflie, could help solve some of the most foundational problems for autonomous vehicles.

Even if the prototype does not result in a real product, the experience from this project could still be valuable to Bitcraze and others. All the schematics and source code will be open source and available on Github, which should make it easy for future researchers and developers to take leverage of the project.

1.4 Constraints

During this thesis work a prototype is built. Thus, the final result is not meant to be a complete, finished product, but rather serve as a proof of concept. If the project is successful, it could later be used as the basis for an actual product that Bitcraze can manufacture and sell. The prototype is thus constrained to work primarily with the Crazyflie, which means that the size, and connections will be suited to fit the Crazyflie, and the source code is written to match the Crazyflies eco-system. However, it should not be hard to make modifications to the hardware and the source code to make it suitable for other vehicles and robots, as well as other embedded eco-systems.

1.5 Related Work

Bitcraze has created a deck called Multi-ranger, which has five SSLs mounted in directions: front, right, back, left and up. The sensors on the deck are the VL53L1X, which is a multizone solid-state lidar, developed by ST [20]. It can measure distances up to 4 meters, within a 27° FoV. However, since this sensor does not offer *simultaneous* multizone output data, it is only possible to get multiple zones by sampling each zone one by one. Thus, the output data from the sensor is the distance to a single point, and not a 2D matrix. Furthermore, the Multi-ranger deck does not have an on-board MCU, and thus the sensors are controlled and sampled from the main microcontroller on the Crazyflie instead.

Terabee is a company that creates and develops sensor modules, embedded software, and ready-to-deploy solutions for systems integrators [21]. They have a series of ToF sensors called TeraRanger Evo, which has different distance coverage specifications, from 3m up to 60m. The sensors comes in a small 30mm × 30mm package, but uses Light Emitting Diode (LED) technology instead of lasers to measure distances, which according to the datasheet of the TeraRanger Evo enables the sensor to measure distance to an area instead of a single point [22]. One of their newer products is the TeraRanger Hub Evo which is a module that allows the connection of up to eight of these ToF sensors, attached in a circle-like configuration [23]. This way, it is possible for the sensors to together cover a wide FoV up to 360°. With the help of the modules simple plug-and-play mechanics, and its compatibility with popular development frameworks, it can be a viable solution for many applications. However, none of the sensors offer simultaneous multizone output data, and even though the LED technology is suppose to measure distances to an area instead of a point, it might still be difficult to get a detailed three-dimensional description of the environment.

Recently, a research group at ETH Zurich has been using the Crazyflie during their research on autonomous navigation and obstacle avoidance. They have created a custom expansion deck with a VL53L5CX sensor facing front, which could mounted on the Crazyflie [24]. This is indeed similar to what this project aims to build, but since the expansion deck they created was limited to having a single sensor facing front, it would probably need to turn or spin around in order detect or map the environment on the sides of the drone. Furthermore, the deck had no on-board MCU, but required an external microcontroller to communicate and collect data from the sensors.

2 Methodology

This chapter aims to describe the methodology used in this thesis and is structured as follows: First, several benefits and downsides of building a prototype are discussed. Second, a detailed description of the methodology used in this thesis is given. Third, an evaluation of the sources referred to in this thesis is presented.

2.1 Benefits & Downsides of Prototyping

This thesis aims to build a prototype, and even though building a prototype before manufacturing costs time, money, and delays potential sales, the benefits can still overcome the disadvantages. First of all, building a prototype is a quick way to create a proof of concept. Second of all, one can learn from potential mistakes or caveats that presents themselves during the development of the prototype. For instance, during prototype evaluation, it can be noticed that the components do not act as expected. Perhaps they are too close on the circuit board, or they get too hot. It is then significantly easier to adjust this before manufacturing has started, which could save both time and money. Third of all, new technologies or new components can be tested before creating a real product.

2.2 Prototype Model

To build the prototype in this thesis, the methodology used is based on the Prototype Model, which is defined in Software Development Life Cycle (SDLC) [25]. SDLC is a process that is typically followed by software projects and it describes how to develop, maintain, replace, alter or enhance software. In general, it defines a methodology for improving the quality of the software and the overall development process. While the Prototype Model typically consists of six phases, only four of them are followed in this thesis. The reason for this is that the last two phases in the original Prototype Model regards refining the prototype and the implementation of a final product, which is out of scope for this project. Thus, this project can be divided into the following four phases:

1. Prototype Requirements
2. Prototype Design
3. Building the Prototype
4. Prototype Evaluation

The transition between phases is done according to the Waterfall Model, where a phase cannot be started until all previous phases has been completed. Furthermore, within each phase, there are several tasks that must be completed, where the results of the tasks are needed by the posterior phase. Figure 2.1 gives a visual representation of the four phases, their corresponding tasks, and the transitions between them.

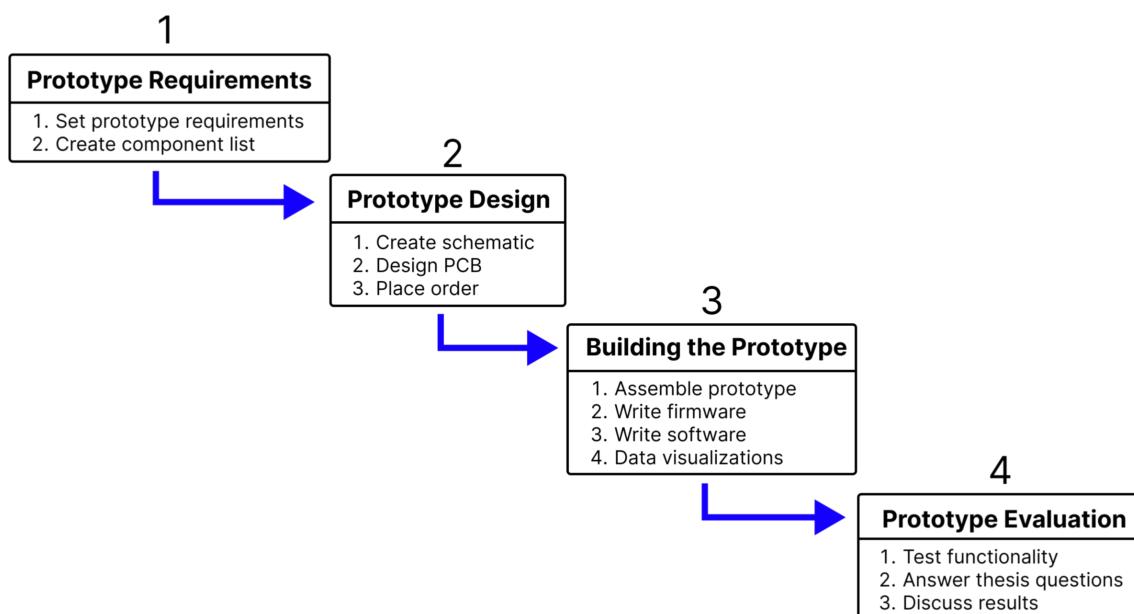


Figure 2.1: The four phases used in order to build and evaluate the prototype.

Each of these four phases are described in more detail below.

2.2.1 Phase 1 - Prototype Requirements

During this phase the requirements for the prototype are set, which means that this phase has a crucial impact on the remainder of the project. In order to specify the requirements, discussions must be held together with the management team, in this case Bitcraze, and a brief analysis of what components are available on the market must be done. Making sure that the necessary components are available on the market is especially important, since there was an on-going chip shortage during the project initialization, making certain components impossible to buy. Once the requirements have been specified, a list of the main necessary components must be created. This list should not contain *every* component required on the circuit board, as some of the components can only be decided once the schematic is being made. Thus, this phase consists of the following two tasks:

1. Gather and formalize all requirements for the prototype.
2. Make a list of the main components that will be needed to fulfill these requirements.

This phase should result in the specified requirements of the prototype and a list of the main necessary components, which are both required in order to start the design of the prototype.

2.2.2 Phase 2 - Prototype Design

Once the requirements of the prototype has been set and a list of main necessary components has been made, the design of the prototype can start. The design phase can be divided into the following three tasks:

1. Create a schematic, containing all necessary components and connections.
2. Design a PCB from the schematics.
3. Order the PCB and the necessary components.

Both the schematic and the PCB are designed in KiCad (see subsection 3.5). Furthermore, this phase contains many critical aspects that deserves special attention. If there is something wrong in the schematics, perhaps something is connected to the wrong pin of the microcontroller, it might be necessary to re-do the schematic and re-order the PCB or certain components. This could cause severe delays of the project, since shipping can take up to several weeks. Because this thesis project only spans a period of 10 weeks, every week is of much importance, and thus, this phase requires careful attention. This phase should result in the finished PCB and the necessary components for the prototype being ordered.

2.2.3 Phase 3 - Building the Prototype

Once the PCB and all components has arrived the prototype can finally be built. The building process of the prototype requires soldering the components to the PCB, as well as writing firmware for the microcontroller and software for the GCS. The building phase can thus be divided into the following four tasks:

1. Mount all components on the circuit board.
2. Write firmware for the MCU, so that it can communicate with the sensors, read data from them, and send the data to a GCS.
3. Write software for a GCS, that can communicate with the MCU on the PCB.
4. If there is time left, extend the software to include methods that can visualize the data.

This phase should result in a finished prototype, which should be able to collect distance data from the SSL sensors, and send the data to a GCS. If time allows, it should then be possible to visualize the distance data on the GCS.

2.2.4 Phase 4 - Prototype Evaluation

Once the prototype has been built, and firmware and software has been written for it, it is time to evaluate the prototype. This is done by testing its functionality and measuring and observing the results. After this, the thesis questions are answered, and at last the results are discussed. Thus, this phase consists of the following three tasks:

1. Test the functionality of the prototype.
2. Answer the thesis questions.
3. Discuss the results of the prototype.

This phase should provide enough results and information about the functionality of the prototype, in order to make a final conclusion of the thesis.

2.3 Evaluation of Thesis Sources

Much of the information that is presented in this thesis is based on external sources, which includes:

- Datasheets
- Research papers
- Textbooks
- Websites
- Source code (from Github)

Each of these sources are described in more detail below.

Datasheets

Since this thesis contains a fair amount of technical details, it is sometimes necessary to dig deep into datasheets to find certain pieces of information. These datasheets usually contains first hand-information, are developed and maintained by the manufacturer or organization of the product or service, and can thus be seen as the "ground truth" of that particular product or service. This is not to say that datasheets cannot be incorrect, but they can typically be trusted, as long as the datasheet is indeed coming from the given manufacturer or organization. The datasheets used in this thesis are the following: [26], [27], [28], [29], [30], [31], [32], [33], [34], [22], [23], [17], [35], [36], [37], [16], [20] and [38].

Research Papers

In order to find detailed descriptions or summaries of specific topics, research papers have been used, which consists of Journal Articles and Conference Papers. Both of these sources typically contains second-hand, or even third-hand information. Thus, these sources may be further away from the "ground truth" than datasheets, and might deserve to be used with more care. However, all papers that are used in this thesis are published from the well-known publishers IEEE and ScienceDirect, so while care must be taken, they can for the most be seen as reliable. The following sources are published by IEEE: [12], [39], [14], [18], [15], [40], [41], [13], [42], and sources [43], [44] [45] are published by ScienceDirect.

Textbooks

Three textbooks have been used in this thesis, which includes [46], [47] and [25]. While the books are from 2012 and 2013, meaning they were approximately 9-10 years old when this thesis work was carried out, they provide information that has most likely not changed since they were written. [46] and [47] presents well-established theories and technicalities in electronics and [25] discusses software development methodologies, which even if they have changed according to today's standard, it does not matter, since it was mainly used as the basis for the methodology used in this thesis.

Websites

Some information is not available in datasheets, research papers nor textbooks, and websites can provide excellent information, as long as the information is handled with care. A difficulty with websites is that there is not always a clear author of the information, nor a date when it was created. However, in this thesis, websites have mainly been used for two use-cases:

1. Finding information *about* a specific company or organization. This information has for the most part been gathered from the websites "About" page, and thus it seems reasonable to believe that the information is valid. The following websites has been used to retrieve short pieces of information about companies/organizations: [48], [49], [50], [19] and [21].

2. Retrieving information from Bitcrazes documentation about the Crazyflie and its eco-system, which is hosted on the web. Since this project is done in collaboration with Bitcraze, it seems reasonable to assume that their website and its information can seen as reliable and the information as valid. The following sources has been used to retrieve information about the Crazyflie and its eco-system: [8], [5], [10], [2], [51], [9], [4], [6], [7], [1], [3], [52] and [11].

Source Code (Github)

Two Github repositories has been used in this thesis, and while both of these are open-sourced, meaning anyone can contribute to them, they have only been used as tools during the project. Thus their reliability is not of much importance, as long as they provide the tools that are necessary. These repositories includes [53] and [54].

3

Theoretical Background

This chapter aims to describe the major theoretical concepts and technologies that are used in this thesis. While multiple of the topics require complete books by themselves, only a brief overview is given since a deeper description would be out of scope for this thesis. If the reader is already familiar with the topics, this chapter can be skipped and used as a reference when needed.

3.1 Basic Electronics

This section describes the fundamental *electrical* concepts and technologies that are used in this thesis.

Diodes

A diode is a semiconductor device that has two terminals, and typically allows current to flow in only one direction. This is accomplished by doping the silicon, with P-type doping on one side, leaving an excess positive "holes", and N-type doping on the other side, leaving an excess of negative electrons. This creates a PN-junction which creates the one-way current flow behavior that is desired [46, p. 407]. However, the current cannot flow through the diode at all times, but a voltage has to be applied to the diode in order to "turn it on". When this is done, and the current flows in the expected direction, the diode is said to be *forward* biased. If a voltage would be applied in the opposite of this direction, the diode is said to be *reverse* biased. The voltage level that is required to turn on a normal diode is typically around 0.6 V [46, p. 408]. This means that if a diode is inserted into a circuit, there will be a voltage drop across the diode, because it "requires" 0.6 V in order to let current flow through it. Figure 3.1 illustrates a normal diode.

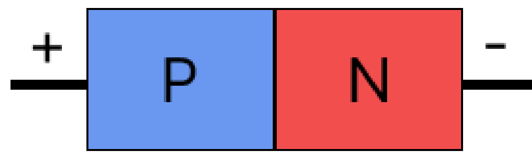


Figure 3.1: Simplified structure of a normal diode.

There are many different types of diodes, each having specific properties that might be desired for a given application. The types of diodes that are used in this thesis include:

Schottky Diode - Schottky diodes typically have a low voltage drop of ~0.4 V (exact value depends on the current through the diode), and are often faster to turn on that normal diodes. This makes them suitable for detecting low-voltage, high-frequency signals that ordinary diodes would not see [46, p. 410].

LED - A LED is a diode that emits visible light when current flows through it [46, p. 428].

TVS Diode - A Transient Voltage Suppression (TVS) diode is an avalanche diode, which uses its properties to protect electronics from high voltage spikes. These voltage spikes (transients) can for instance be caused by a collapsing magnetic field when the current through an inductor is suddenly switched off [46, p. 486].

SPAD - A Single-Photon Avalanche Diode (SPAD) is a type of diode that acts as a mix of a photo diode, avalanche photo diode and a normal diode. This means that a SPAD can convert light into an electrical current, through the photoelectric effect. Some of the benefits of the SPAD diode, compared to normal a photo diode, are that it increases photon detection efficiency, has lower power consumption and is sensitive enough to sense a single photon [45].

Laser Diode - Laser is an acronym for Light Amplification by Stimulated Emission of Radiation, and is a type of PIN diode. A PIN diode is a PN-junction diode, that has an undoped intrinsic region between the P and N junctions, thus the name PIN [47, p. 10]. A laser diode creates spontaneous emissions of photons when positive "holes" and electrons recombine within this intrinsic region.

In a laser diode, the surfaces of the intrinsic region are polished to a mirror-like finish, which causes the photons from the spontaneous emission, to bounce back and forth, which eventually forms a beam that is parallel to the junction [47, p. 11]. This is illustrated in figure 3.2, and in order to focus the beam, a lens is typically added on top of the laser diode. The safety of a laser diodes is classified into a set of classes and is determined by the wavelength as well as the power of the laser.

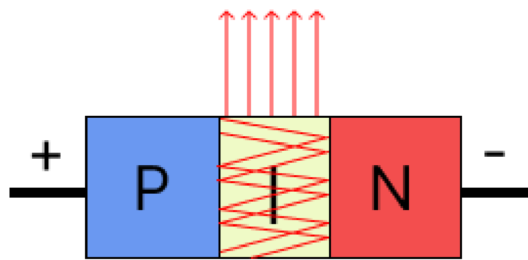


Figure 3.2: Simplified working principles of a laser diode.

VCSEL

Laser diodes can typically be grouped into two different classes: Edge-Emitting Lasers (EEL) and Vertical Cavity Surface Emitting Laser (VCSEL). EELs have been used for a long time and is the original semiconductor laser, and the light that is emitted propagates in an elliptical way on the *edge* of the semiconductor, parallel to the surface, thus its name *Edge*-Emitting Laser. However, for VCSELs, the light is emitted vertically, orthogonal to the semiconductor surface and the light is propagated as a circular beam, thus its name *Vertical* Cavity Surface Emitting Laser [14]. Figure 3.3 illustrates the (simplified) differences between an EEL and a VCSEL.

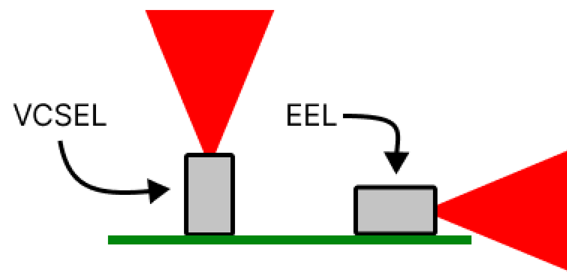


Figure 3.3: Simplified differences between EELs and VCSELs.

Thus, one of the benefits of VCSELs is that they are easier to mount in arrays, because the emitted beam is orthogonal to the surface of the semiconductor. While VCSELs are not as established in the industry as EELs, there has been a market shift in recent years where VCSELs are getting used more in areas such as sensing and optical communication applications [40].

Transistors

A transistor is a semiconductor devices that can amplify or switch electrical signals. Just like diodes, there are different types of transistors; Bipolar Junction Transistor (BJT), and Field-Effect Transistor (FET). Both types has three terminals, but they operate by different physical principles. The terminals of a BJT are called collector, base and emitter, and on the FET they are called source, gate and drain [46, p. 445, 459]. When a FET is turned on, the current typically flows from the drain to the source, which is controlled by applying voltages at the gate terminal of the transistor. A highly simplified FET is illustrated in figure 3.4. The

drain and source can be doped positively, or negatively, which changes the way the FET is turned on.

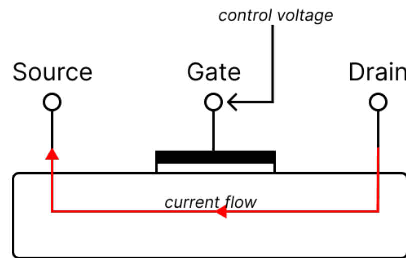


Figure 3.4: Simplified working principles of a FET. Current is flowing from drain to source.

MOSFETs

A Metal-Oxide-Silicon Field-Effect Transistor (MOSFET) is a type of FET, that is commonly used in electronics, for instance; to control the flow of power in an electrical application. There are two types of MOSFETs:

N-channel (NMOS) - Source and drain are N-doped. Current flows when a *positive* voltage is applied to the gate.

P-channel (PMOS) - Source and drain are P-doped. Current flows when a *negative* voltage is applied to the gate.

Pull-Up & Pull-Down Resistors

Pull-up and pull-down resistors are used to prevent electrical electronic signals to be left floating. Floating means that the signal is not connected to anything, and can thus have unpredicted voltage values. Thus, with the help of pull-up and pull-down resistors, signals can be correctly biased, to either a high value (digital 1), or a low value (digital 0), depending on the application. Figure 3.5 illustrates the working principals of the pull-up and pull-down resistors.

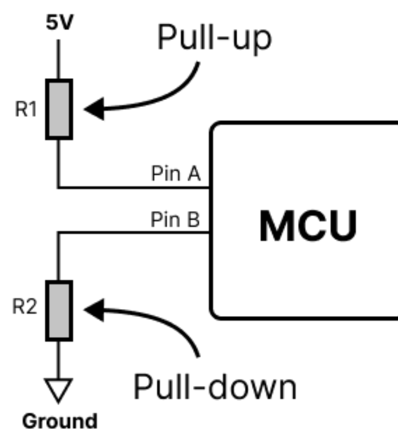


Figure 3.5: Pull-up and pull-down resistors. High (digital 1) is at 5V and low (digital 0) is at ground.

One might think that it would be sufficient to connect the signal lines directly to a high voltage or ground without the resistors, but if a signal is connected straight to ground, or high voltage, this could form a short-circuit that could damage the electronics. Thus, a resistor is placed in between, to limit the current.

Open Collector & Open Drain

An open collector, or open drain, is a type of output that can often be found on many integrated circuits (ICs). Whether it is open collector or open drain, depends on what transistor is used on the IC (BJT or FET). The input collector, or drain, is left floating, and has to be connected externally. Thus, some benefits of this is that the logic is moved out of the system, and the voltage levels is not decided by the IC [46, p. 763].

PWM

Pulse Width Modulation (PWM) is a method of reducing, or increasing, the average power delivered by an electrical signal by turning the signal on and off at a certain frequency, with a specific duty cycle. The length of the duty cycle determines how long the signal should be high for each period. This can range from 0% (output is 0), to 100% (output is highest). PWM is often used by microcontrollers to control motors or LEDs.

SoC

A System on Chip (SoC) is an integrated circuit that has all, or most components of a functioning computer. These components includes a Central Processing Unit (CPU), memory interfaces, Input/Output (I/O) interfaces and more.

Transceiver

In radio communication, a transceiver is an electronic device that can both transmit, and receive radio waves, using an antenna. Thus, it can be seen as a combination of a radio transmitter and radio receiver.

ISM Radio Band

The ISM radio bands are portions of the available radio spectrum, that are reserved specifically for Industrial, Scientific, Medical (ISM) purposes.

3.2 Common Communication Protocols

Microcontrollers typically use different digital communication protocols to communicate. Some of the most common ones are I^2C, UART, SPI, which are described below.

I^2C

Inter-Integrated Circuit (I^2C) is a synchronous, serial communication protocol, which requires only 2 lines to operate: Serial Clock (SCL) and Serial Data (SDA). These lines are half duplex (data can go in both directions, but not on the same time), open-collector or open-drain, and thus, must be pulled up high with resistors. This allows the circuit-designer to decide the voltage levels of the lines, which are typically 3.3V or 5V [42].

The communication is structured in a Master-Slave configuration, where one device acts as the master, and is responsible for initiating the communication, and the other device acts as the slave, waiting for the master to start the communication. There can be multiple masters and multiple slaves, but only one master can talk to a slave at a given time. Each slave is given a unique address of 7-10 bits (typically 7) that the master uses in order to choose which slave device it wants to communicate with.

The I^2C supports different operating modes, which specify the data transfer speed. The modes allows data rates from 100 kbit/s (Standard mode) up to 5 Mbit/s (Ultra-Fast mode). However, higher data rates requires some caveats in order to work, and for most applications, data rates between 100 kbit/s and 1 Mbit/s are sufficient.

UART

Universal Asynchronous Receiver Transmitter (UART) is an asynchronous serial communication protocol, which similar to I^2C, requires two lines: RX and TX. But unlike I^2C, it is a *full*-duplex communication protocol (data can go in both directions, at the same time). In order for two devices to communicate through UART, they must both support and be configured to use the same data rate [44].

Note, that since two devices that communicates through UART only has a transmission line (TX), and a receiving line (RX), there is no way for either device to know when the other device is ready to receive new data. Thus, there is no synchronization; if the receiver is not ready to receive new data, but the transmitter keeps sending data, the data can be lost. To mitigate this problem, *flow control* can be used, which ensures that the transmitter does not overflow the receiver. For UART, two pins can be used for flow control: Ready To Send (RTS) and Clear To Send (CTS).

UART is typically used with transfer rates at 9.6 kbit/s or 115.2 kbit/s, but can work at even higher data rates like 921.6 kbit/s, and even up to 4 Mbit/s [43].

SPI

Serial Peripheral Interface (SPI) is a synchronous serial communication specification that, similar to UART, is full-duplex. However, the interface requires 4 lines to operate: Master Out Slave In (MOSI), Master In Slave Out (MISO), Slave Select (SS) *or* Chip Select (CS), and Serial Clock (SCLK). The protocol works in a Master-Slave configuration, and the bus can operate with a one master and multiple slave devices. In order for the master to select which slave it wants to communicate with, the corresponding SS pin must be driven low, which notifies the slave that the master wants to start communication. Thus, synchronization is built into the protocol.

A downside of SPI is that it requires 4 lines (5 if ground included) between two devices in order to communicate. However, having more lines enables higher transfer speeds, and SPI can work at speeds up to 1 GHz [41].

3.3 FreeRTOS

FreeRTOS is an open-source, Real-Time Operating System (RTOS) that is widely used among microcontrollers around the world. It is distributed freely with MIT open source license, which means that it is free to use in real products, without having to pay any fees [34].

FreeRTOS includes a kernel, as well as an increasing set of Internet of Things (IoT) libraries that makes integration easy for developers. FreeRTOS is developed in C, and the core consists of only 3 source files, and one configuration file, which is specific to the microcontroller in use. It uses a priority based preemptive scheduler, which means that tasks (similar to threads for a normal Operating System (OS)) are paused, stopped or resumed in an order that is decided by the tasks priority. This enables developers to run multi-threaded applications on embedded devices, similar to how it is typically done on a normal OS. Another features that FreeRTOS offers are queues, which are data structures that can be used to synchronize different tasks, and send data between running tasks [34].

One of the benefits of using an RTOS instead of a full OS on embedded devices is that the overhead that is followed by context switching is significantly smaller. Thus, one can achieve *real-time* responses, which can be critical for embedded devices such as UAVs and robots. Also, the memory footprint of the FreeRTOS kernel (6-12 kB) can be more suitable for embedded devices that does not have enough memory for a full OS.

3.4 ESP32

ESP32 is a series of low-power, low-cost SoCs with integrated WiFi and Bluetooth. They are developed by Espressif Systems, whom were established in 2008, and focuses on developing cutting-edge wireless communication, low-power Artificial Intelligence of Things solutions [48]. The ESP32 is the predecessor of the ESP8266, which made a big breakthrough for the company in the IoT market. The ESP8266 was, and still is, used among makers and hobbyists, and compared to other popular microcontrollers in these communities, such as the AVR Atmega328 (which is the main MCU on the Arduino), the ESP8266 and ESP32 offers a significantly more powerful MCU, as well as WiFi and Bluetooth functionalities. Another benefit of the ESP8266 and the ESP32s is their low price, which can get as low as just a few dollars.

ESP-IDF

In order to program the ESP32s, one can use ESP-IDF, which is an open-source software development framework, that is also developed and maintained by Espressif Systems [32]. It is built with FreeRTOS and provides an extensive Application Programming Interface (API) that builds on top of a hardware abstraction layer, which makes it easy for developers to develop and upload code to the microcontrollers. Espressif have also made an extension to

Visual Studio Code [53], that makes it easy for developers to setup and use the development framework, within the IDE. Figure 3.6 illustrates the process of building and flashing code to an ESP32, directly from Visual Studio Code.

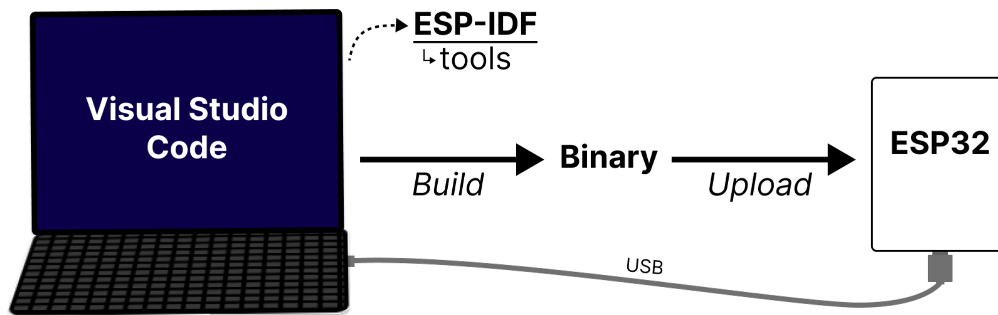


Figure 3.6: Building and flashing code to an ESP32, directly from Visual Studio Code.

3.5 KiCad

KiCad is a free open-source Schematic Capture & PCB Design Software that runs on Windows, Linux and macOS [50]. It has features that makes it possible to create electronic schematics, and then design PCBs from these schematics, whereas the PCB design part of KiCad is called PcNew. The goal of the KiCad project is to "provide the best possible cross platform electronics design application for professional electronics designers." [50]. It is also possible to create plugins to KiCad, which can further extend the functionality of the software.

4

Crazyflie

This chapter gives an overview of the Crazyflie platform and its eco-system. While it is not essential to learn all details by heart, it contains many topics and concepts that are used in the remainder of the thesis.

Crazyflie is a open-source flying development platform that is developed by Bitcraze. At its core, *the* Crazyflie is a small pocket-sized drone that weighs only 27g. The drone and the its eco-system is used primarily for research and educational purposes, but also finds its way to hobbyists and enthusiasts in fields of UAVs and robotics [2]. Figure 4.1 gives an overview of the Crazyflie eco-system.

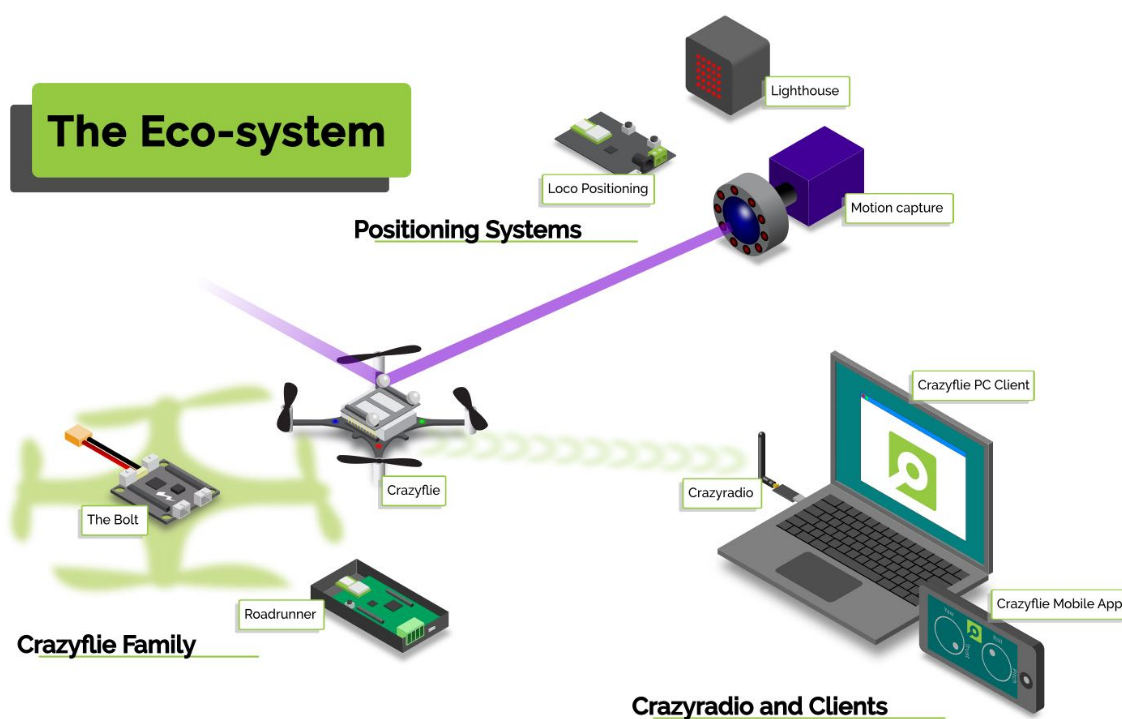


Figure 4.1: Crazyflie eco-system overview. Image taken from [1], with Bitcrazes permission.

All firmware, software and tools within the eco-system is open-sourced, which allows users to inspect, edit and/or improve it by contributing to the project. Thus, together with the Crazyflie community, the platform and its eco-system is constantly updated and improved. The firmware is written purely in C and with the help of Bitcrazes build tools, developers can easily write their own extensions or modifications to the firmware of the drone, flash it, and inspect the results. The drone can be programmed to be fully autonomous, or controlled by a ground control station.

4.1 The Crazyflie

The Crazyflie drone uses brushed DC motors which are controlled by small MOSFETs using PWM. It is powered from a 3.7V lithium polymer battery, which is regulated to 3V. It has a flight time of around 7 minutes, battery charging time of approximately 40 minutes, and can carry up to 15 grams of additional payload [27]. Figure 4.2 shows the Crazyflie.

Figure 4.2: Crazyflie drone, weighs only 27g. Image taken from [2], with Bitcrazes permission.

The main components of the Crazyflie can be seen in table 4.1.

Table 4.1: Crazyflie main components.

Component	Description	Specs
STM32F405	Main MCU	Cortex-M4, 168MHz, 192kB SRAM, 1MB flash
nRF51822	Radio and power management	Cortex-M0, 32MHz, 16kB SRAM, 128kB flash
BMI088	Inertial measurement unit	3-axis accelerometer + 3-axis gyroscope
BMP388	Pressure sensor	High precision pressure sensor

4.2 Coordinate System of the Crazyflie

When interpreting the attitude (orientation) of the Crazyflie, it is important to differentiate between the global coordinate system and Crazyflies local coordinate system. The local coordinate system is the coordinate system seen from the Crazyflie, and the global coordinate system is the coordinate system "seen from outside". This is illustrated in figure 4.3.

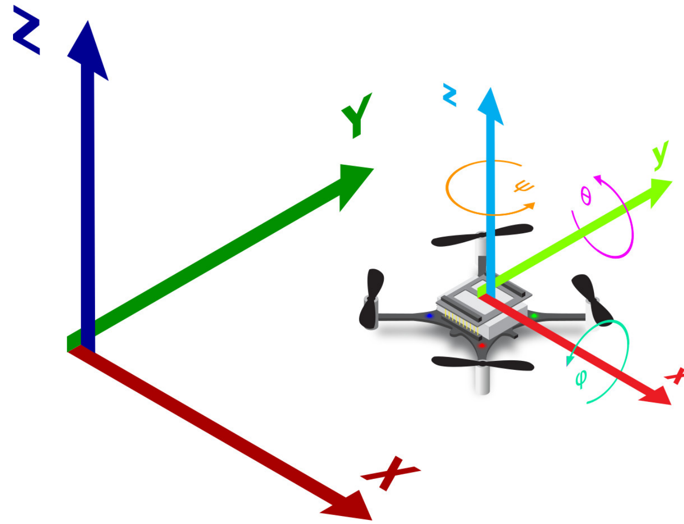


Figure 4.3: The coordinate system of the Crazyflie. The front of the drone is in the direction of the x-axis. Image taken from [3], with Bitcrazes permission.

For the Crazyflie, the coordinate systems (x, y, z) are in East North Up convention. In figure 4.3 above, φ, θ, ψ are the attitude angles, which are called roll, pitch and yaw respectively.

4.3 Radio

The Crazyflie has an on-board nRF51822, which is a general purpose ultra-low power SoC which features a 2.4 GHz radio that supports Bluetooth Low Energy (BLE) as well as other proprietary protocols [37]. It is connected with SPI to the STM32 (main MCU on the Crazyflie).

Since the Crazyflie uses the nRF51822 to communicate with the GCS, the ground station must have a compatible radio transceiver. To handle this, Bitcraze has developed the Crazyradio PA dongle (figure 4.4), which can be plugged into the USB-port of a PC [28]. This dongle is equipped with an nRF24LU1+ radio transceiver, which is a simpler transceiver than the nRF51822, and does not offer BLE functionalities.

Figure 4.4: Crazyradio PA USB dongle. Image taken from [4], with Bitcrazes permission.

Both of these radio chips are developed by Nordic Semiconductor, operates in the 2.4 GHz ISM band, and are configurable to data rates of 250 kbit/s, 1 Mbit/s or 2 Mbit/s. The data is transmitted in the air with the help of Enhanced ShockBurst (ESB) protocol, which is also developed by Nordic Semiconductor. ESB is a master-slave protocol that supports error detection as well as re-transmissions of packets, by the use of acknowledgments (ACKs) and cyclic redundancy checks. The transmission is half-duplex, and thus, the radio can be in either *receiver* mode, or in *sender* mode. In order to send data back and forth between devices one would then have to switch between sender and receiver mode every time. While this switching can be fast, it does add some delays, because it has to be communicated through SPI. To mitigate this, Nordic Semiconductor has built a feature where it is possible for a slave to put payload data in the ACKs, which makes it possible for a slave to send data back to the master, without either of them having to switch modes (in reality, the two devices *does* switch modes, but this happens very fast, so this time can almost be ignored).

ESB supports a payload size of maximum 32 bytes, in both directions [36] and figure 4.5 shows a communication example where the PC is the master (sender), and the Crazyflie is the slave (receiver).

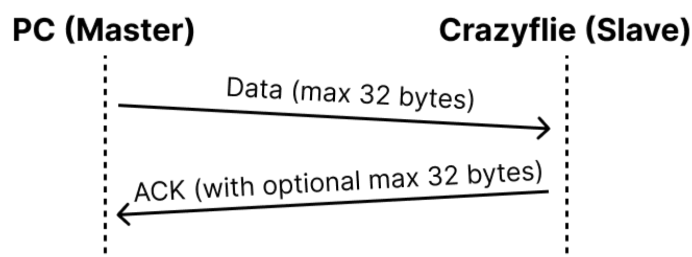


Figure 4.5: ESB ACK containing payload.

The communication of the nRF51822 and nRF24LU1+ is simply referred to as *nRF* during the remainder of the thesis.

4.4 CRTP

Crazy RealTime Protocol (CRTP) is a communication protocol developed by Bitcraze, which is used with to communicate and control the Crazyflie. The protocol is designed to suit the radio chips on the Crazyflie and the Crazyradio (nRF51822, and nRF24LU1+), however, it can work over other data transports as well, such as USB. The following description of the protocol is valid for CRTP version 2022.01 [51], and might change in the future.

In CRTP, the Crazyflie acts as the slave, and the GCS (PC, or other device) acts as the master, and the communication is always initialized by the master. One of the 32 bytes of possible payload is used as header, which means that the payload size of a packet is limited to 31 bytes. However, due to an internal bug in the firmware, only a maximum of 30 bytes of payload data is possible. The current implementation of CRTP allows approximately 1000 packets to be sent per second, which equals ~30 kB/s. The structure of a CRTP packet is visualized in figure 4.6.

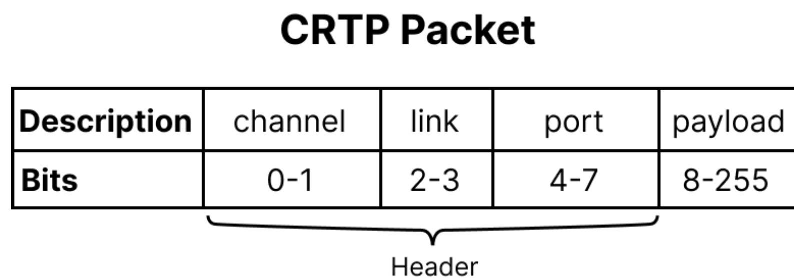


Figure 4.6: CRTP packet structure.

Packet fields explanation:

Channel - Used to identify the action/command.

Link - Reserved for future use.

Port - Used to identify the functionality or task that is associated with this message.

Payload - The data to be sent.

Not all ports are used, so it is possible to add more in the future. Table 4.2 gives some examples of used ports, and their functionality.

Table 4.2: Some of the ports used in CRTP.

Port	Name	Description
3	Commander	Sending control set-points for the roll/pitch/yaw/thrust
6	Localization	Packets related to localization
7	Generic Setpoint	Allows to send setpoint and control modes

4.5 Client Software

In order to communicate with the Crazyflie conveniently from a PC, Bitcraze has developed a Python library: Cflib, which communicates with the drone through the Crazyradio PA dongle. On top of this Python library, there is a higher level API, which makes it easy for users to control the Crazyflie through Python. This way, a user can for instance send setpoints directly to the Crazyflie, from a simple Python script.

In order to make it easier to control and configure the drone, Bitcraze developed a Graphical User Interface (GUI), called Cfclient. It is written in Python, with PyQt (Python bindings for the QT Application framework) and makes it easy to configure settings of the drone, read logging data, fly, or do other things. There is also a mobile Android and IOS app that connects to the drone through BLE and makes it possible to fly it from a mobile phone. Figure 4.7 gives an overview of the Crazyflie software stack.

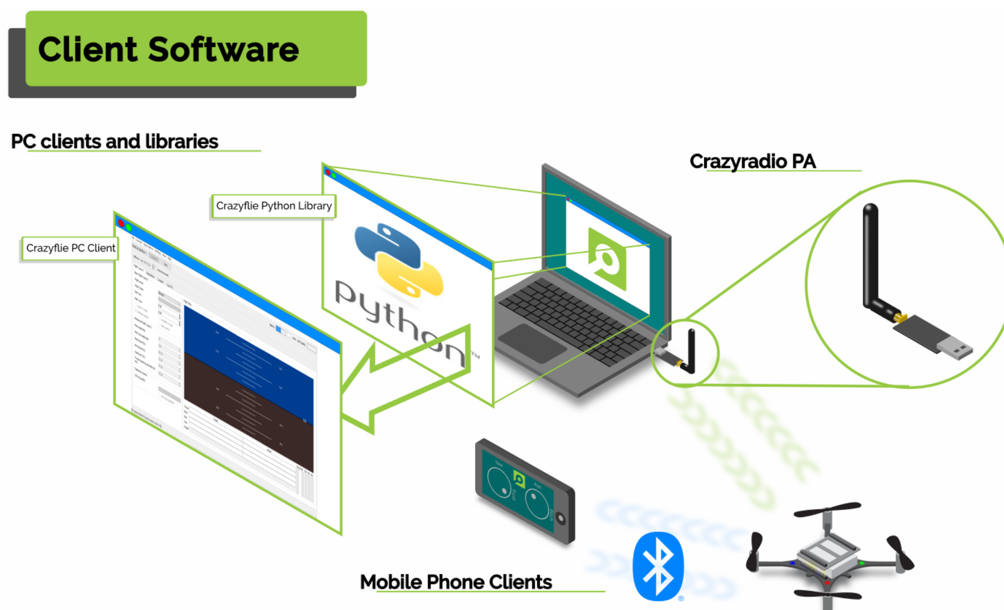


Figure 4.7: Crazyflie software stack. Image taken from [5], with Bitcrazes permission.

4.6 Expansion Decks

To extend the functionality of the Crazyflie, the drone has an expansion connector, which allows additional electronics to be mounted on the top or bottom of the drone. Bitcraze has developed multiple decks (circuit boards), that fits on these expansion connectors, and can thus be mounted on the Crazyflie. Most of the decks features some type of sensor(s), that the STM32 communicates with, and typically requires a deck driver. A deck driver is a piece of code that is specific to the deck and handles deck initialization and deck communication, and is compiled into the firmware of the STM32. Three decks are described below: Multi-ranger deck, Flow Deck and the AI deck.

4.6.1 Multi-Ranger Deck

The Multi-ranger deck uses the SSL sensor VL53L1X, to measure distances in directions: front, right, back, left and up. The sensors are programmable to measure multiple zones, with a FoV of 27º, but does not offer *simultaneous* multizone distance data, like the VL53L5CX or TMF8828 [31]. This means that in order to collect data from multiple zones with the VL53L1X, one must sample one zone at the time. The VL53L1X communicates through I^2C and the Crazyflie reads data from the sensors, one at the time. Figure 4.9 shows the Multi-ranger deck.

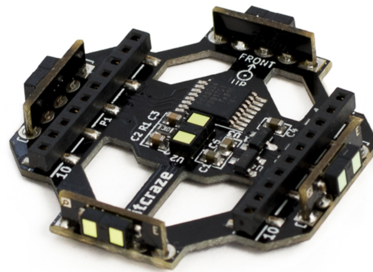


Figure 4.8: Multi-ranger deck with 5 VL53L1X sensors mounted in directions: front, right, back, left and up. Image taken from [6], with Bitcrazes permission.

4.6.2 Flow Deck

The Flow deck also uses the VL53L1X SSL sensor, but measures the distance to the ground [30]. The deck can be mounted underneath the Crazyflie and also has an optical flow sensor, which measures movements in relation to the ground. This allows the drone to estimate its position without any external positioning systems.

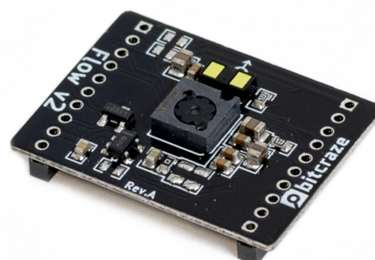


Figure 4.9: Flow deck v2. Image taken from [7], with Bitcrazes permission.

4.6.3 AI Deck

The AI deck is built around the GAP8 RISC-V multi-core MCU and has an on-board camera, as well as an ESP32 MCU. This is a new product that aims to allow developers to create low power AI solutions that can run on-board the drone [26]. While being new and powerful, the deck also brought more complexity with it. With the AI deck mounted, there is a total of *four* different microcontrollers on the drone, which means that debugging and developing code for each of the MCUs can be harder. Also, communication between the MCUs can be difficult, and to mitigate this problem, the Crazyflie Packet eXchange (CPX) protocol was developed, which is described below, in section 4.7. Figure 4.10 shows the AI deck.

Figure 4.10: AI deck with GAP8 RISC-V MCU, ESP32 and an on-board camera. Image taken from [8], with Bitcrazes permission.

4.6.4 Expansion Deck Connection Pins

For the expansion decks to connect to the Crazyflie, they must match the same connection pins as on the Crazyflie. These pins are described in figure 4.11.

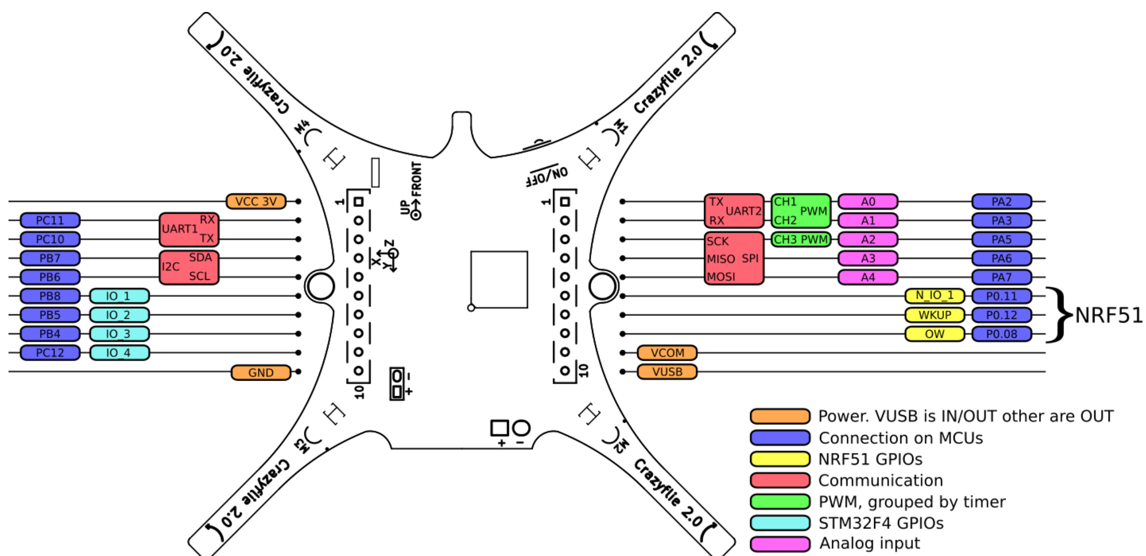


Figure 4.11: Crazyflie expansion connection pins. Image taken from [9], with Bitcrazes permission.

As shown in figure 4.11, the expansion connection pins allows expansion decks to communicate with the Crazyflie through UART, SPI and I^2C. There are also a few General Purpose Input/Output (GPIO) pins available as well as power and ground.

During system initialization of the Crazyflie, it checks for connected decks through the 1-Wire (OW) pin. This pin is connected to a 1-wire memory on the deck, which has a unique ID that is used to identify the deck. This way, the Crazyflie can know which decks are connected to it, and start the correct deck drivers.

4.7 CPX - Crazyflie Packet eXchange

At the time of the project initialization, Bitcraze has just released a new communication protocol, called CPX (Crazyflie Packet eXchange). CPX enables communication between different microcontrollers and the Crazyflie, and ensures that the communication is performed in a structured way. Before CPX, if one wanted an external MCU (like an ESP32) to talk to the Crazyflie, one had to implement the communication logic from scratch. Thanks to CPX, adding external MCUs on decks and communicating between them should now be easier and not as time-consuming. CPX is in some ways similar to the Internet Protocol, and can, as of version 0, be summarized as follows:

- An endpoint (Crazyflie, ESP32, PC etc) is referred to as a `target`.
- A `router`, handles the routing of packages, and ensures that a package is given to the correct target, *or* that the packet is passed on, if the target is more than 1 step away. A *step* refers to a single step along the way; for instance, microcontroller `A` can send a packet to `C`, which is first passed to `B`, and then forwarded to `C`. In this scenario, the packet travelling from `A` to `B` consists of one step, and from `B` to `C` another step. Note that the router is not a physical device, but rather a software module.
- Each packet has a source and destination target ID, as well as a dedicated `function`.
- Each link between targets can have its own Maximum Transmission Unit (MTU). This way, targets can optimize memory usage. This is handled by (potentially) splitting packages along the way, in order to make their size lower or equal to the MTU, which means that the sender does not have to care about the MTU of the transport link, and can send packets of any size.
- Packets are delivered in order.
- Instead of dropping packets if a target is overloaded, the communication blocks (waits until target is ready). This way, it is not possible for a transmitter to send more data to a target, than what the target can handle.

CPX is suppose to work as a base, where an application specific protocol is implemented on top of it. For instance, it is possible to run CRTP *over* CPX. Figure 4.12 visualizes the CPX stack.

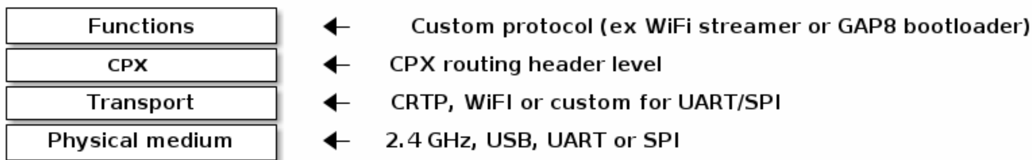


Figure 4.12: CPX communication stack. Image taken from [10], with Bitcrazes permission.

The structure of a CPX packet can be seen in figure 4.13.

CPX Packet

Description	destination	source	lastPacket	reserved	function	dataLength	data
Bits	0-2	3-5	6	7	8-15	16-31	31-*

Routing (destination–function) · Payload (dataLength–data)

Figure 4.13: CPX packet structure. Destination and source are both `target`s, and can thus be set to any of the values in table 4.3. The `function` (bits 8-15) can be given any value from table 4.4.

Table 4.3 shows the available targets, and table 4.4 shows the available functions. However, it is worth mentioning that extending the protocol to support more targets and more functions is simple.

Table 4.3: Available CPX targets.

Target	**Value**	**Description**
STM32	1	STM32 on Crazyflie
ESP32	2	ESP32 on AI-deck
HOST_WIFI	3	Host computer
GAP8	4	GAP8 on AI-deck

Table 4.4: Available CPX functions.

Function	Value	Description
SYSTEM	1	Used for target system functionality
CONSOLE	2	Used for console printouts
CRTP	3	Used to tunnel CRTP
WIFI CTRL	4	Used for controlling the WiFi setup and signaling
APP	5	Intended to be used with user applications
TEST	14	Used for test functionality (ECHO, SOURCE and SINK)
BOOTLOADER	15	Used to communicate with bootloaders.

CPX is unaware of the transport layer, meaning any type of transport can be used. In the current implementation, the following transports are supported:

- UART
- SPI
- WiFi/TCP

It is up to the transport layer to deliver the bits of the packet, and to ensure that they are correctly delivered. Each transport will thus add some extra overhead bytes to each packet, which are specific to each transport. The subsections below highlights specific details for each transport.

4.7.1 Transport - UART

The MTU of UART is 100 bytes, and the packet structure is described in figure 4.14

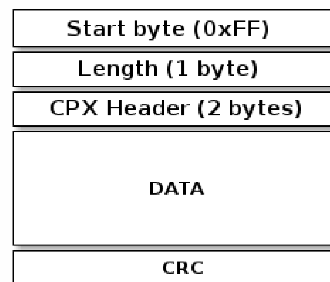


Figure 4.14: CPX UART packet structure. Image taken from [10], with Bitcrazes permission.

Like mentioned in section 3.2, two pins are typically used for flow control in UART: RTS and CTS. However, since this would require two additional pins, in CPX the flow control

is instead accomplished by sending a special message code (`0xFF 0x00`) to indicate when either part is ready to receive new data.

4.7.2 Transport - SPI

The MTU of SPI is 1022 bytes, and the packet structure is described in figure 4.15. SPI includes a shared clock signal SCLK, which is controlled by the master, and requires an additional pin (CS/SS) to select which slave device the master wants to communicate to (see section 3.2). This setup works fine for a master-slave topology where the master always initiates the communication. However, in CPX, the communication can be initiated by either device, and thus two lines are used in order to achieve synchronization. These lines are called Ready To Transfer, and are pulled high by either device when it wants to send data. The device must then wait until the other device pulls the other line high, and only when both lines are high, data can be transferred.

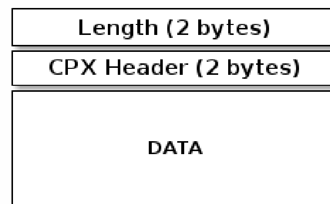


Figure 4.15: CPX SPI packet structure. Image taken from [10], with Bitcrazes permission.

4.7.3 Transport - WiFi

The MTU of WiFi is 1022 bytes, but the size of the actual transferred data per packet will depend on the underlying TCP data. The packet structure is described in figure 4.16.

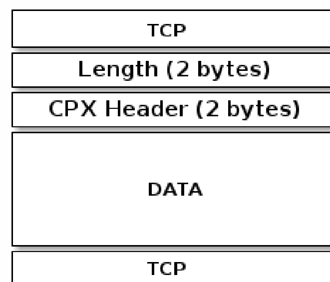


Figure 4.16: CPX WiFi packet structure. Image taken from [10], with Bitcrazes permission.

4.7.4 Packet Splitting

CPX supports packets splitting, which means that developers does not have to care about the sizes of their packages when sending data. However, when receiving data through CPX, one must assemble the (potentially fragmented) packets, which is not handled automatically. This can be accomplished by inspecting the `lastPacket` bit of the CPX packet, which is set to 1 when the last packet is sent. The reason for not having this automatically done in CPX is that it would require a lot of statically allocated memory that might not be necessary. Figure 4.17 shows a simple example where CPX splits a packet of 200 bytes into two packets of 100 bytes each. Note that it is the transport layer that sets the MTU.

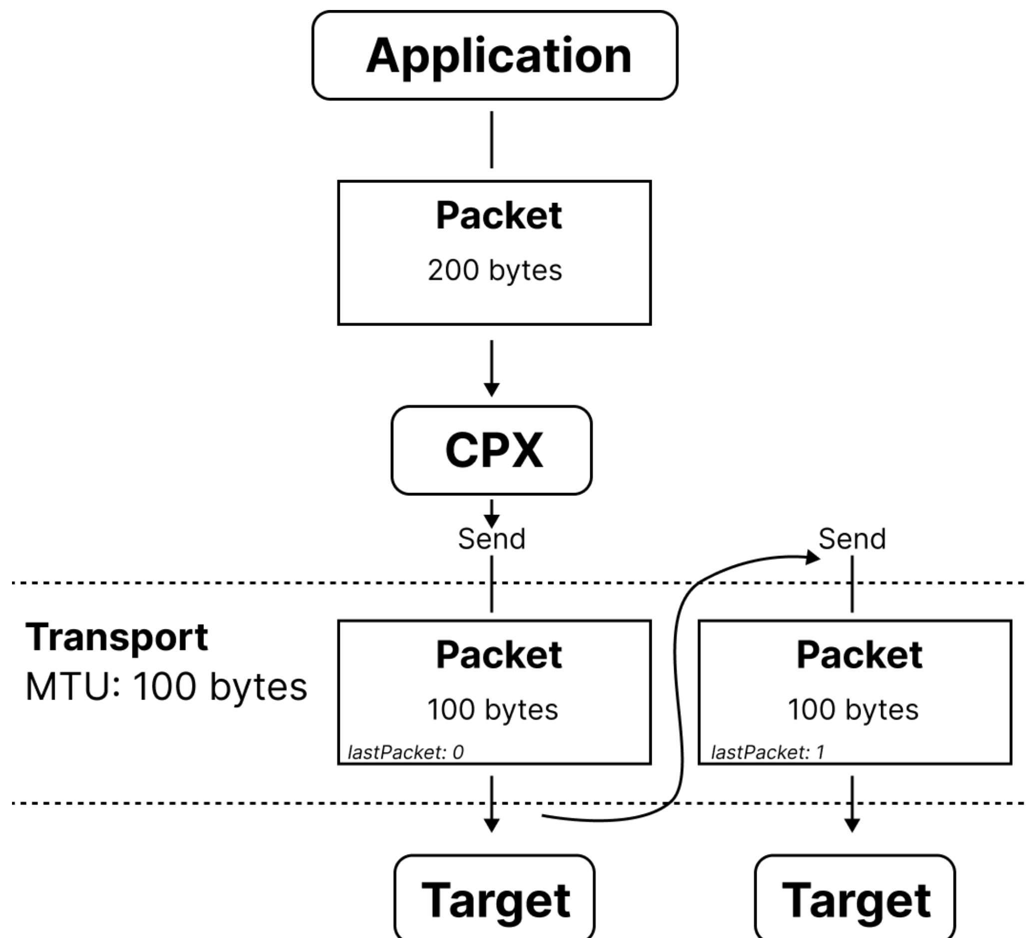


Figure 4.17: Simple example of how CPX splits a packets that is bigger than the MTU. A benefit of CPX is that the sender (Application in the figure), does not have to care about the size of the packet when sending data. This is all handled by CPX.

5 Prototype Requirements

Now that an overview of the relevant theoretical background has been given, and the Crazyflie and its eco-system has been discussed, it is to present the requirements for the prototype. This chapter corresponds to phase one of the methodology, and is structured as follows: First, the requirements of the prototype are formalized and listed, whereas each requirement is discussed in greater detail in sections 5.1.1-5.1.5. Then, the main necessary components are listed in section 5.2, and at last, the chosen name for the prototype is presented.

5.1 Formalized Requirements

Bitcraze was interested to use the new VL53L5CX sensor, which is a simultaneous multizone solid-state lidar, and is of the same family as the VL53L1X sensor, which Bitcraze already had familiarity with. Furthermore, at the time, the VL53L5CX was the only simultaneous multizone SSL on the market with a small package and a reasonable price (around $10) that could be found. How the sensors were suppose to be mounted, or what microcontroller would control them, was not of much importance to Bitcraze, and was left free of choice. After some discussions and considerations, the following requirements were chosen for the prototype:
The prototype will be based on the Multi-ranger deck and must:

1. Be possible to mount on top of the Crazyflie, as an expansion deck. Thus, the size of the deck is constrained, as well as the expansion connection pins.
2. Have five VL53L5CX sensors on the deck, which are mounted in directions: front, right, back, left and up.
3. Have an on-board MCU that can gather the data from the sensors. This MCU should be an ESP32-S3.
4. Have an on-board DC-DC converter to ensure stable voltage levels.
5. Have necessary passive components like resistors, capacitors etc, as well as components that makes developing and debugging easier, such as USB, LEDs and a button.

Each of these requirements are discussed in more detail below.

5.1.1 Deck Size and Connections

To be able to mount the prototype deck on the Crazyflie, it must match the Crazyflies expansion connections pins (see section 4.6). This means that the deck must have pin holes at the same locations, with the same distances between them, as the Crazyflie. Figure 5.1 shows an expansion deck template that Bitcraze has provided.

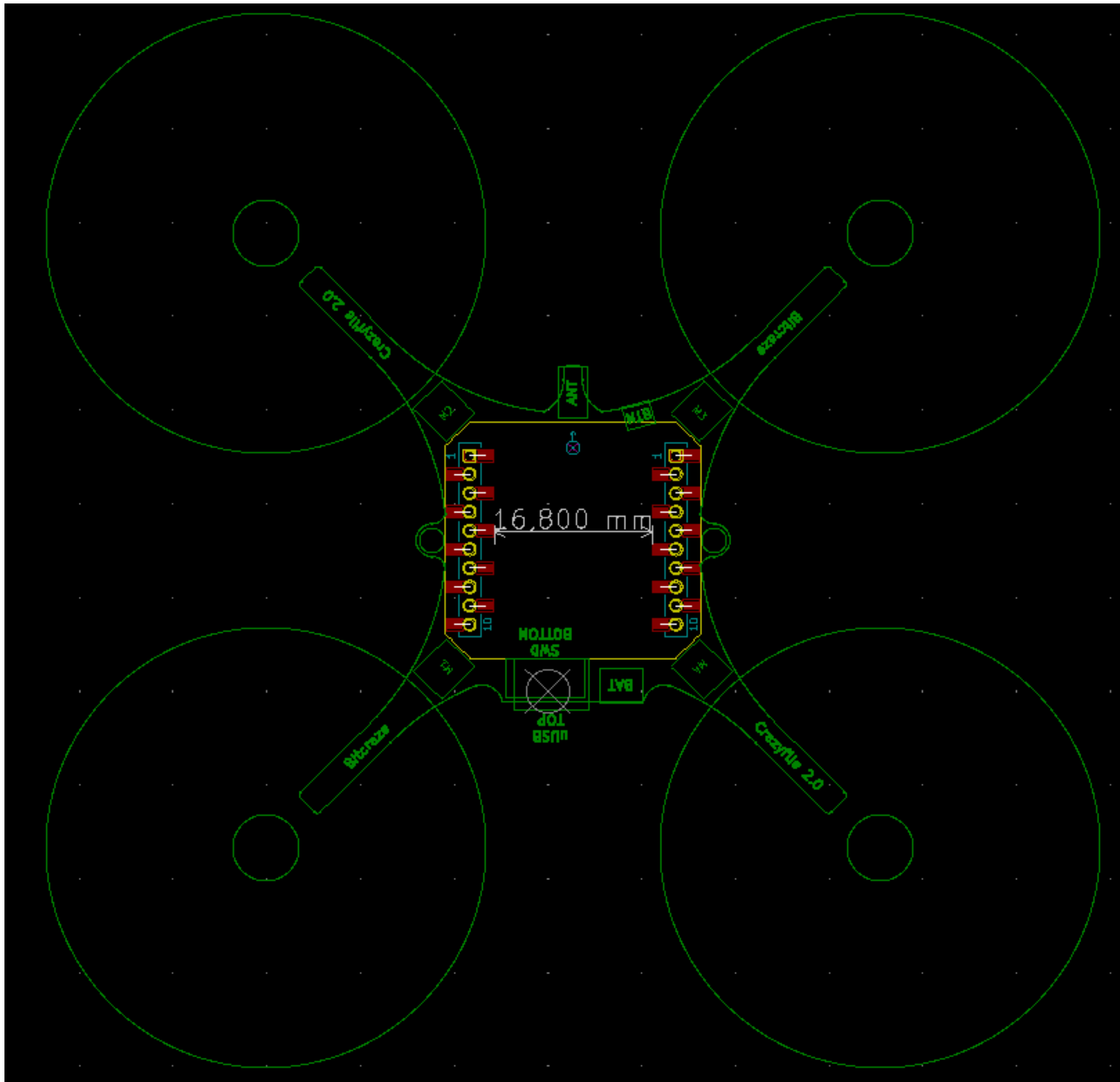


Figure 5.1: Expansion deck template, provided by Bitcraze [11]. As one can see, there is only about 16.8mm between the expansion connection pins.

There is only about 16.8mm between the expansion deck pins, which all components must fit within. While there is more room to the front and back of the Crazyflie (up and down in figure 5.1), the deck cannot be too big, because it might interfere with the propellers. This size constraint had an impact on many of the decisions that involved picking and placing components.

5.1.2 VL53L5CX (SSL Sensor)

VL53L5CX is a state of the art simultaneous multizone SSL sensor, developed by ST [16]. It measures distances by sending out a laser beam and then measures the time it takes for the laser to hit a target and reflect back. It sends the light with a VCSEL, and has a Diffractive Optical Element (DOE) in front of the receiving sensor, which splits the incoming laser beam into multiple beams, which are then projected onto a SPAD array. This way, a 2D matrix consisting of distances to objects in front of the sensor can be measured.

The VL53L5CX features an on-board 32-bit 250 MHz MCU with custom hardware acceleration [39]. The sensor is able to measure absolute distances up to 400 cm, is configurable to output a 4x4 or 8x8 matrix of distances within a 45° FoV (vertical and horizontal), and can work at ranging frequencies up to 60 hz. The VL53L5CX is able to detect different objects within the FoV, thanks to STs histogram algorithms, which also provides immunity to cover glass crosstalk (unwanted light that is reflected by a cover glass, and *not* by the intended target) beyond 60 cm. Figure 5.2 visualizes the (simplified) working principles of the sensor.

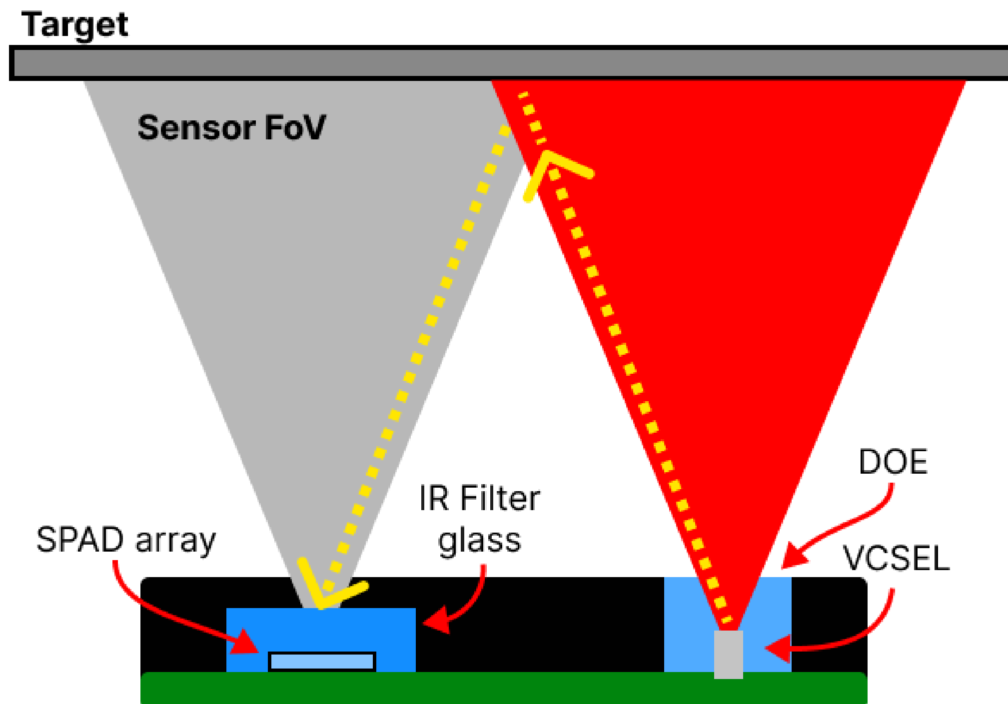


Figure 5.2: Simplified working principles of the VL53L5CX sensor. The yellow dashed arrow represents a travelling laser beam that is reflected by the target to the DOE, which then hits the SPAD array.

The 2D matrix that is returned by the sensor is flipped horizontally and vertically, as illustrated in figure 5.3.

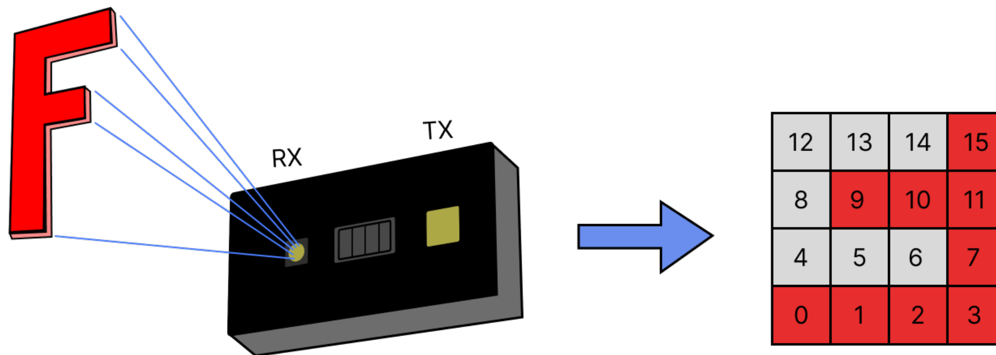


Figure 5.3: VL53L5CX effective orientation. The output 2D matrix is flipped vertically and horizontally. The resolution in the figure is 4x4. The numbers in each zone is the index of the measurement result data.

The device comes in a reflowable package that measures 6.4 x 3.0 x 1.5 mm, and has 16 exposed pins, and a big ground pad in the middle [16], as seen in figure 5.4 and 5.5.

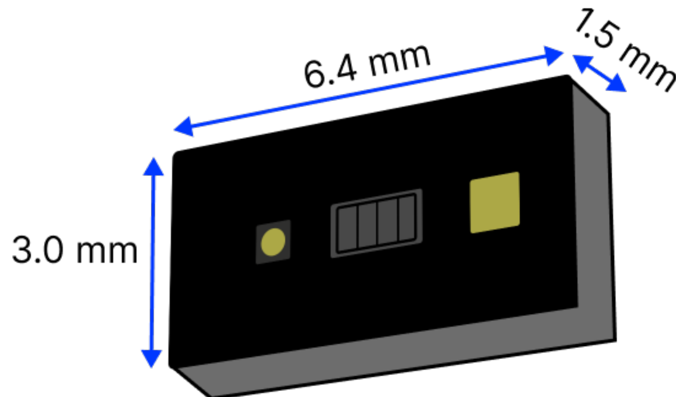


Figure 5.4: VL53L5CX dimensions.

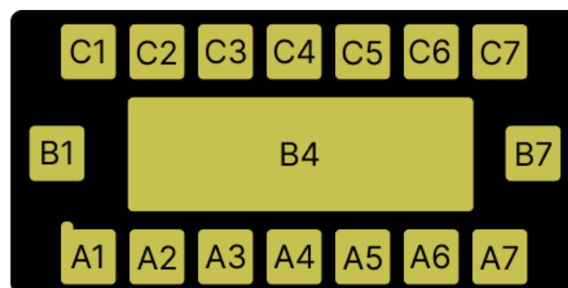


Figure 5.5: VL53L5CX pins.

The device communicates with an external MCU over I^2C and ST has developed a low-level C driver to control it, which is free to use. The driver only requires the user to create a few functions that handles the low-level platform-specific I^2C operations. In order to enable the I^2C communication of the device, a pin called LPn (A5) must be driven high, and to disable the I^2C communication, this pin must be driven low.

The sensor is highly configurable with the help of the low-level C driver, which makes it possible to change its settings. Some of the settings can be set during run-time, through I^2C, while others must be set on compile-time. For instance, with the Targets per zone setting, it is possible to configure the sensor to measure distances of up to four targets *per* zone. A complete list of the available settings can found in appendix A.

During sensor initialization, about 84 kB of firmware must be copied to the device. While it is not mentioned in the datasheet, the reason for this could be that there is no on-board memory flash that can store the firmware. Thus, after the firmware is copied to the sensor, it is probably stored in its ram until powered off. Once the sensor has been initialized, it has to be started in order to measure distances. Once started, the sensor will continuously measure distances, which is referred to as *ranging*. Thus, the data that is gathered from the sensor is called Ranging result.

Ranging Result

When the VL53L5CX is ranging, there are two ways to know if new data is available:

1. **Polling** - Ask the sensor over I^2C if new data is available.
2. **Interrupt** - An interrupt is raised by the sensor, on pin A3 (INT).

When new data is available, it can be gathered from the sensor, and the distance data per zone is returned in millimeters and consists of two bytes. The configured sensor settings will have an impact on the structure of the available data, since the results can consist of more than just distance data. For instance, it can contain a field called Target status, which consists of a single byte, and gives a measurement of the validity of the distance data. Thus, this field could be used to filter the distance data of the sensor, and improve its reliability. Another field, which is always included as a part of the Ranging result is the internal sensor silicon temperature, which makes it possible to monitor the temperature of the sensor. Table 5.1 summarizes all possible results that can be gathered from the sensor.

Table 5.1: Available fields in a Ranging result of the VL53L5CX. *N* stands for number of Targets per zone.

Field	Bytes (RAM)	Unit	Description
Ambient per SPAD	256	Kcps/SPAD	Ambient rate measurement performed on the SPAD array.
Number of targets detected	64	None	Number detected targets in the current zone.
Number of SPADs enabled	256	None	Number of SPADs enabled for the current measurement. A target far away or with low reflectance will activate more SPADs.
Signal per SPAD	256 x N	Kcps/SPAD	Quantity of photons measured during the VCSEL pulse.
Range sigma	128 x N	Millimeter	Standard deviation estimator for the noise in the reported target distance.
Distance	128 x N	Millimeter	Target distance.
Target status	64 x N	None	Measurement validity, can be used to filter the data.
Reflectance	64 x N	None	Estimated target reflectance (in percent).
Motion indicator	140	None	Structure containing the motion indicator results.

All fields are enabled by default, and can only be changed on compile-time, which is done by adding or removing definitions in a header file. Note that the size of the Ranging result has an impact on the ram usage of the driver, because it has to allocate buffers to store the data. It also has an impact on the traffic on the I^2C bus, since bigger results, means that more data has to be sent between the microcontroller and the sensors.

5.1.3 ESP32-S3 (MCU)

The ESP32-S3 is a state of the art SoC of the ESP32 family that has two Xtensa® 32-bit LX7 CPU cores, and supports WiFi, BLE, USB Serial/JTAG controller, and much more [33]. It can be programmed with the help of Espressifs ESP-IDF (see section 3.4) directly through USB, or by pulling a specific pin: GPIO0/BOOT, to low during power-on, and then send the new firmware through UART. ESP-IDF includes all the tools that are necessary for this. A part from being a new SoC at the time of the project initialization, the reasons for choosing ESP32-S3 as the microcontroller on the prototype can be summarized as:

- Bitcraze already has an ESP32 on their AI-deck, which means that they have some familiarity with the development framework, as well some existing firmware for it.
- At the time of the project initialization, there was an on-going global chip shortage, making it difficult to get other MCUs like the one on the Crazyflie (STM32). However, it was still possible to get ESP32 chips.
- The ESP32 series is not only powerful, but also offers both WiFi and BLE, which opens up new communication possibilities for the Crazyflie. WiFi is especially interesting because it allows bandwidths that are significantly higher than the nRF radio on the Crazyflie (see section 4.3). This would also mean, that if the bandwidth of the nRF radio was not sufficient to send the distance data to a GCS, WiFi could be used instead.

Download Mode & Strapping Pins

The ESP32-S3 comes programmed with a Read Only Memory (ROM) that is unbrickable. This ROM contains the bootloader of the chip and during power-on, the MCU can select whether it should boot from external flash memory (with SPI), or if it should enter *Download* mode. In this mode, the ESP32-S3 can be programmed through UART. The MCU has four *strapping* pins, that are sampled during system reset, as 1 or 0. These values are then stored internally in the chip, until it is powered down. By setting the values of these strapping pins before powering it on, one can select different functionalities for the ESP32-S3. Entering Download mode is one of the functionalities that can be configured with the help of these pins.

ESP32-S3 Modules

The ESP32-S3 requires a few external components to work correctly, like a flash memory and an antenna. One can choose to get all these components separately, and assemble them on a circuit board, and while this might save some space, it comes at the cost of complexity. For instance, because the WiFi and Bluetooth works at high frequencies, (2.4 GHz), one has to consider electromagnetic interference when designing the PCB, which makes the circuit design significantly more complex. Luckily, Espressif also produces complete modules, which includes the ESP32-S3 SoC, as well as all the necessary components such as flash and antenna. These modules are slightly bigger, because they have a protection case, but are almost the same price as the individual chips.

It was decided to use a complete module to make prototyping easier, and there were three different modules available for the ESP32-S3:

- ESP32-S3-MINI-1
- ESP32-S3-WROOM-1
- ESP32-S3-MINI-1U

The different modules can be seen in figure 5.6.

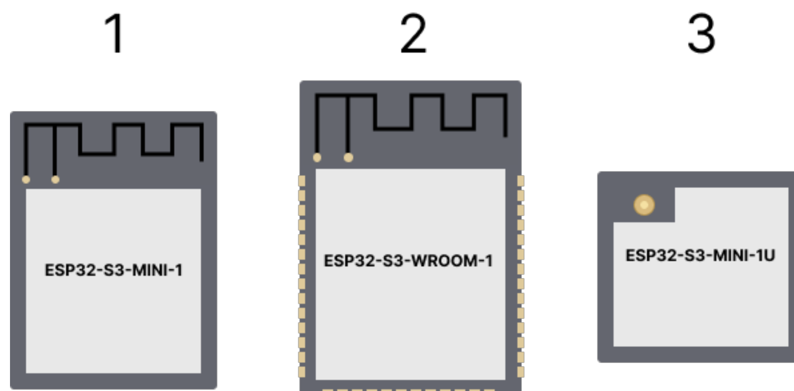


Figure 5.6: ESP32-S3 modules. Module one and two has an on-board antenna, and only module number two has solder pads exposed.

Package number one was chosen, because it includes an on-board antenna, and is smaller than number two. While number three is significantly smaller than the other ones, it requires an external antenna, which was not desired. The caveat with module number one is that the solder pads are underneath the module, which means that soldering it onto the PCB would probably be more difficult, and it would be impossible to probe the pads to check their voltage levels. However, module number two was simply too big to fit between the expansion connection pins, so this trade-off had to be taken.

5.1.4 TPS62A01 (DC-DC Converter)

It was decided that the deck should also have its own DC-DC converter, to ensure as stable voltage levels as possible. This decision was taken due to the fact the ESP32-S3 was expected to consume rather high power when using WiFi [33], and according to the datasheet of VL53L5CX [16], the sensors were suppose to consume more power than the VL53L1X. The ESP32-S3 operates between 3.0V and 3.6V, and the VL53L5CX between 2.5V and 3.3V, and since the battery on the Crazyflie can drop as low as 3.0 V, and it was thus decided to have the main voltage line at 3.0V.

Table 5.2 shows the estimated max current consumption for the main components on the deck. These values are taken from the datasheets, and they are all measured at 3.3V, so the currents should be slightly higher at 3V. The purpose of the table is mainly to give a *hint* of what the maximum current consumption will be.

Table 5.2: Estimated max current consumptions for the main components on the deck.

Component	**Max current consumption at 3.3V (mA)**
ESP32	355
VL53L5CX Up	95
VL53L5CX Front	95
VL53L5CX Right	95
VL53L5CX Back	95
VL53L5CX Left	95
LEDs and others	~20
Total	~850

Note that the average current consumption can be hard to estimate, because it depends on many factors. However, if necessary, this can easily be measured once the prototype is built.

After analyzing the expected (maximum) current consumption, it was decided to use the DC-DC converter TPS62A01, which is an efficient buck-converter that can handle currents up to 1 A. It comes in a very small packet, has an adjustable output voltage (which is set using two feedback resistors that forms a voltage divider), and does not require many external components [35]. Since the estimated maximum current (850 mA) is lower than 1A, the TPS62A01 seemed like a sufficient DC-DC converter.

5.1.5 Other Components

In order for the main components to operate properly, additional passive components were required, such as resistors, capacitors and inductors. The datasheets of the main components contained guidelines of what additional components were necessary, and most of these guidelines were followed.

While not strictly necessary, having on-board LEDs can be very handy when debugging, thus, it was decided to have three LEDs on the deck. Also, to enter Download mode on the ESP32-S3, a button was added. While this might no be necessary since the SoC offers direct USB-programming, having a button could be useful later, because after power-up, the GPIO0/BOOT pin of the ESP32-S3 can be used as a normal GPIO pin. A USB connection was also added, to make flashing and debugging easier, as well as a 1-wire memory for the Crazyflie to identify the deck.

5.2 Main Components List

Table 5.3 lists the main components that the deck was suppose to have, as well as a description for them. Note that this list does not include *all* components, since the schematic had not

yet been created, and certain components will be decided only once the schematic is being made.

Table 5.3: Main components of the prototype deck.

Component	Amount	Description
ESP32-S3	1	Main microcontroller
VL53L5CX	5	Simultaneous multizone SSL sensor
TPS62A01	1	DC-DC converter
LED	3	LEDs can always be useful. For instance, they can indicate the state of the MCU
Button	1	Used to enter Download mode or as normal GPIO
Micro-USB connection	1	Used to program the ESP32-S3
DS28E05R	1	1-Wire EEPROM, used by the Crazyflie to identify the deck (see section 4.6)
Resistors, capacitors & inductors	x	Necessary for the rest of the components to work correctly

5.3 Prototype Name - Slamdeck

While it was not of much importance, it felt useful to come up with a suitable name for the prototype. The name proposed was *Slamdeck*, which seemed like an appropriate name, since the deck could be very useful for SLAM algorithms. It is worth noting that this was *not* an official name for a potential product, but a name purely for this project.

6

Prototype Design

Once the requirements of the prototype had been set and a list of the main necessary components had been made, it was time to start designing the prototype. This chapter corresponds to phase two of the methodology, and is structured as follows: First, the prototype schematic is created and described. Then, the PCB is designed with the help of the schematic. At last, the PCB and the necessary components are ordered.

6.1 Schematic

With the schematics of the Multi-ranger deck as the base, a new schematic was created for the prototype. Just like the Multi-ranger deck, the prototype deck would also consist of two different schematics: One schematic that describes the main board, which has the ESP32-S3, DC-DC converter and other necessary components. The other schematic describes the "satellite" boards, which will contain only the VL53L5CX sensor and its necessary passive components. Figure 6.2 shows the expected look of the satellite boards.

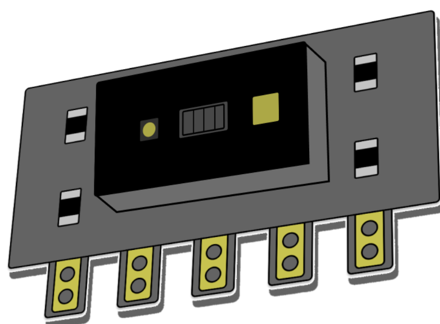


Figure 6.1: How the satellite board was expected to look. A separate PCB, with the VL53L5CX sensor and a few passive components.

This separation will result in two different PCBs and the reason for this, is that the sensors VCSEL emits the laser pulses vertically. Thus, if one wants the sensor to measure distances

horizontally (e.g. in front of the deck), one must either use a flexible PCB, or have a separate PCB for the sensor. This separate PCB can then mounted on the main board, as illustrated in figure 6.2.

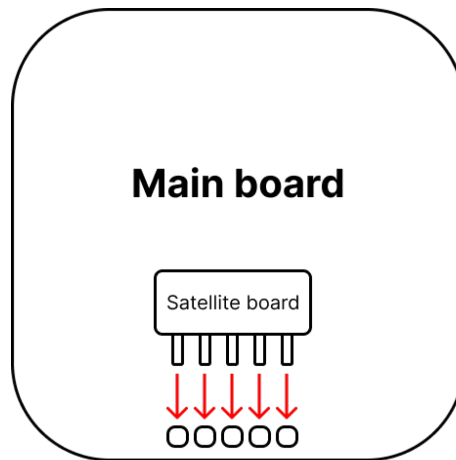


Figure 6.2: The satellite board would be mounted on the main board through connection holes, and then soldered in place.

To keep the size of the deck within the given constraints, the components were chosen to be surface-mounted and as small as possible. This meant that all resistors and most capacitors were of size 0402 (0.4 mm × 0.2 mm). However, some capacitors that had higher capacitance (above 1 µF) were 0603, or even 0805.

The schematic of the prototype can be separated into the following parts:

1. USB & UART pins
2. Power-chain
3. VL53L5CX
4. LEDs
5. Button
6. 1-wire memory
7. Expansion connection pins
8. ESP32-S3 GPIO connections
9. Satellite board

These parts, and how they are related are visualized in figure 6.3.

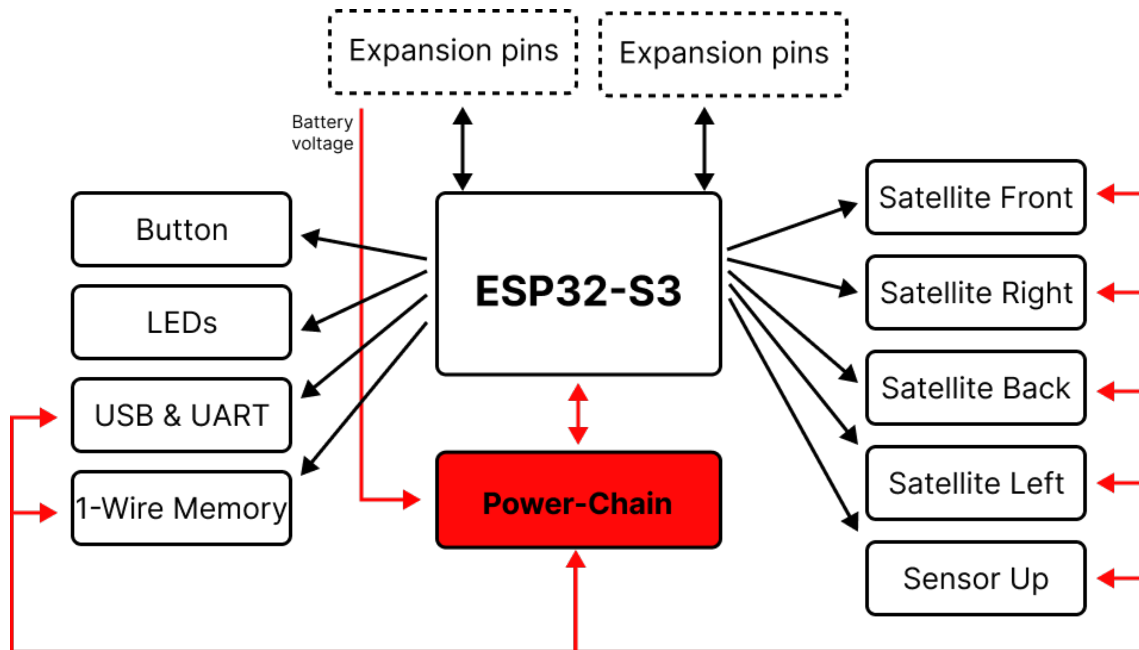


Figure 6.3: Prototype schematic overview. Each satellite block to the right should result in a separate satellite board, except the sensor facing up, which should be on the main board.

6.1.1 USB & UART

The micro-USB connection comes with five lines; VBUS (5V), D+ and D- (data lines), GND (ground) and ID (not used, left floating). The data lines were connected straight to pins GPIO19 and GPIO20 on the ESP32-S3. In case problems were to arise with USB programming, pins connected to UART were also added, together with VBUS and GND.

6.1.2 Power-Chain

For the deck to mounted on the Crazyflie, and still have the USB plugged in, there had to be a voltage source selector that selects only one of the voltage sources (voltage from Crazyflies battery, or the voltage from USB). If this is ignored then unpredictable things could happen when having the USB plugged in at the same time as the Crazyflie is turned on. To handle this, a PMOS-transistor was used, where the gate was controlled by the USB voltage (see figure 6.4). The output voltage of this voltage selector is fed into the DC-DC converter which outputs a stable 3V.

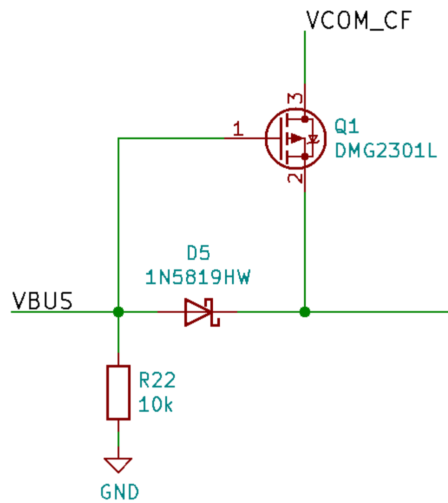


Figure 6.4: Voltage selector using a PMOS transistor, schottky diode and pull-down resistor. `VCOM_CF` is the voltage of the battery on the Crazyflie, and `VBUS` is the voltage from the USB.

Thus, when the USB is plugged in, the transistor is opened, and no current can flow from the battery of the Crazyflie. If there is no USB plugged in, the gate of the transistor is pulled to ground via the 10k Ω resistor, which closes the transistor and allows current from the battery to flow through it. A schottky diode was also placed between the gate and the source of the transistor, which is necessary to separate the two voltage sources. This is illustrated in figure 6.5.

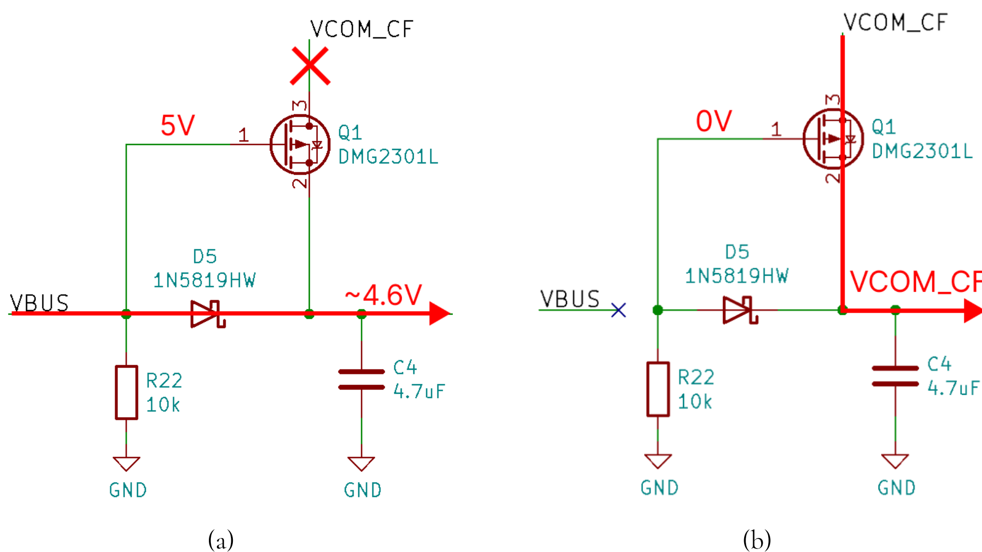


Figure 6.5: The red arrows indicate current flow. (a) When USB is connected, the transistor is opened, and no current flows from the battery. Instead, current flows from USB, and due to the schottky diode, there is a ~0.4 V voltage drop. (b) When USB is *not* connected, the gate of the transistor is pulled down to ground and thus allows current to flow from the battery through the transistor.

A part from the voltage-selector a few other passive components were added to the power-chain:

TVS diode - Used to protect the USB from voltage spikes. While this was most likely not necessary, it was recommended in the ESP32-S3 datasheet, and thus added on the VBUS 5V line.

Capacitors - Used to stabilize the voltages at the input and output of the DC-DC converter.

Inductor and resistors - Used to create a feedback system, which is necessary to set the output voltage of the DC-DC converter. The output voltage can be set according to the following formula:

$$V_{out} = 0.6 \cdot (1 + \frac{\text{R5}}{\text{R19}}) \tag{6.1}$$

A typical application has R5 = 200k Ω, and thus, if $V_{out} = 3V$, R19 must be 200k Ω, because:

$$0.6 \cdot (1 + \frac{200\text{k}\Omega}{50\text{k}\Omega}) = 3V \tag{6.2}$$

Equation 6.1 and 6.2 are deduced from equation 2, section 9.2.2.1 in the datasheet of TPS62A01 [35].

The complete power-chain can be seen in figure 6.6.

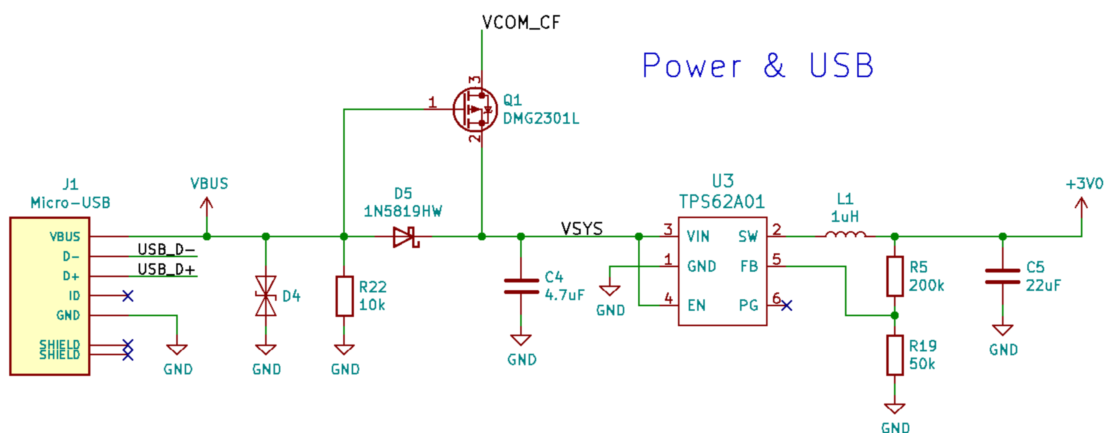


Figure 6.6: Complete power-chain of the prototype. The output of the voltage selector (`VSYS`) is fed into the DC-DC converter. On the output of the DC-DC converter there are feedback resistors (`R5` and `R19`) that sets the correct output voltage. At the end of the output voltage, there is a big 22 µF capacitor (`C5`) to keep the 3V stable. `D4` is a TVS diode, to protect the USB from voltage spikes.

6.1.3 VL53L5CX

The VL53L5CX requires some additional components such as resistor and capacitors to function correctly. In order to communicate with the sensor, three lines were used: SCL, SDA and LPn (see section 5.1.2). All five sensors would initially have the same default I²C address,

so in order to communicate with all of them, it would be necessary to first disable the I^{2}C communication for all five sensors, by pulling their LPn low. Then, one by one, pull the corresponding LPn pin high, to enable the I^{2}C communication again, initialize the sensor, and change its I^{2}C address. Once this is done, it should be possible to communicate with each sensors at their unique I^{2}C address.

Figure 6.7 shows the schematic of the VL53L5CX, which is based on the recommendations from the datasheet. However, some of the recommendations were *not* followed, such as adding a pull-down resistor to A1, which was expected to not have any significant impact on the circuit. The reason for not adding the pull-down resistor was to keep the size of the circuit board as small as possible, and thus, keep the components as few as possible.

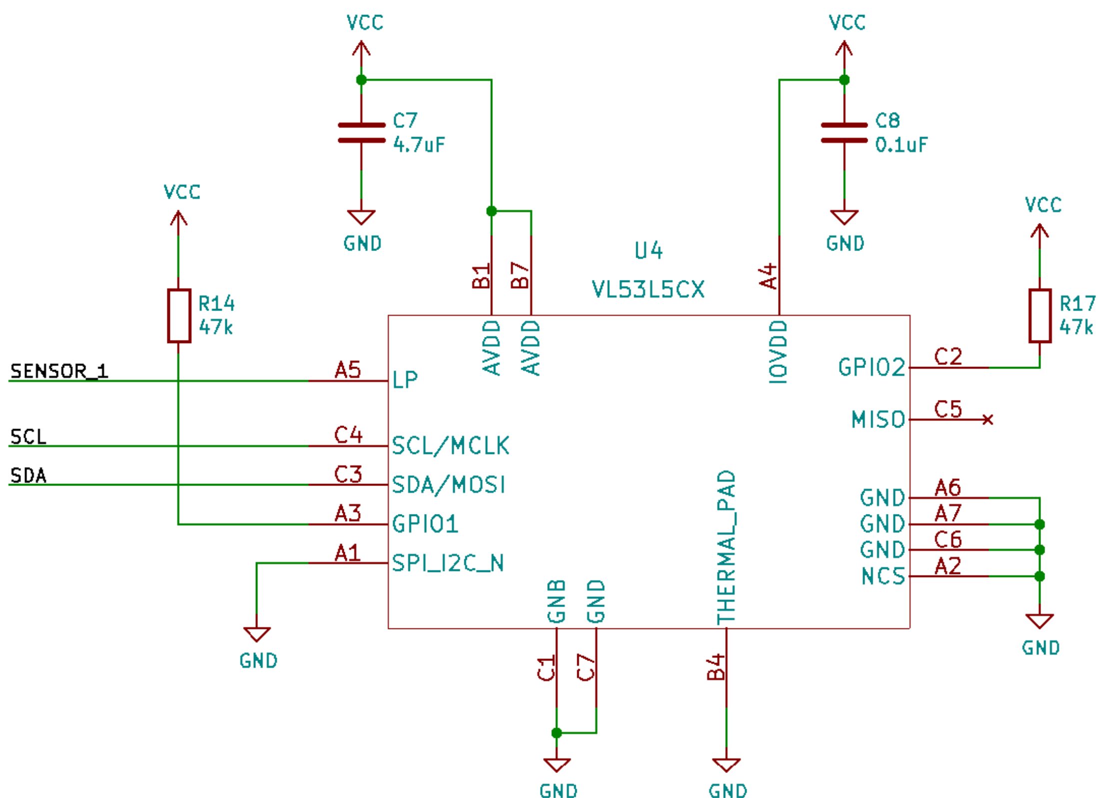


Figure 6.7: VL53L5CX schematic with the necessary passive components. VCC is 3V. Capacitors `C7` and `C8` are decoupling capacitors, which are used to ensure stable voltage levels at the sensors input power pins. Lines `SENSOR_1`, `SCL` and `SDA` are connected to the ESP32-S3.

The schematic in figure 6.7 was used for the sensor on the main board (facing up), as well for the satellite boards.

6.1.4 LEDs

Three LEDs were added, which were connected in series with a 420 Ω resistor, to an individual GPIO on the ESP32-S3.

6.1.5 Button

A button was added to make it possible to connect the GPIO0/BOOT pin of the ESP32-S3 to ground. Like mentioned in 5.1.3, if this pin is low during power-up, the device enters Download mode, allowing the ESP32-S3 to be programmed through UART. After power-up, the pin can be used as a normal I/O, and then the button could be used for anything.

6.1.6 1-Wire Memory

A 3-pin 1-wire EEPROM memory was also added, which was connected to the OW pin of the expansion connection pins. This EEPROM would be used by the Crazyflie to identify the deck.

6.1.7 Expansion Connection Pins

In order for the deck to communicate with the Crazyflie, the connection pins had to match the expansion connection pins on the Crazyflie. These pins and their usage are described in table 6.1 and 6.2.

Table 6.1: Crazyflie expansion connection pins, left side.

Pin	Description
VCC	System voltage, regulated to 2.8V
RX1	RX, UART 1
TX1	TX, UART 1
SDA	Serial Data (I2C)
SCL	Serial Clock (I2C)
IO_1	GPIO 1, can be used for anything
IO_2	GPIO 2, can be used for anything
IO_3	GPIO 3, can be used for anything
IO_4	GPIO 4, can be used for anything
GND	Ground

Table 6.2: Crazyflie expansion connection pins, right side.

Pin	Description
TX2	TX, UART 2
RX2	RX, UART 2
SCK	Clock (SPI)
MISO	Master In, Slave Out (SPI)
MOSI	Master Out, Slave In (SPI)
N_IO_1	IO pin connected to the nRF, can be used for anything.
N_IO_2	IO pin connected to the nRF, can be used for anything.
OW	1-Wire, used to read deck id EEPROM
VCOM	Voltage battery
VUSB	USB Voltage

At this point, it had not been decided yet, *how* the communication between the ESP32-S3 and the Crazyflie would be done, and thus all the available communication buses were connected in the schematic (one I^2C, one SPI and two UARTS). The reason for this was simple; it seemed better to connect too many communication buses, than too few. However, 0 Ω resistors were used in series with all the bus connections, which would make it convenient to disconnect any of them if problems were to arise.

6.1.8 ESP32-S3 GPIO Connections

Table 6.3 contains all the GPIO connections for the ESP32-S3 on the prototype deck.

Table 6.3: Connections of the ESP32-S3 on the prototype deck. "-" means that the pin is not connected to anything and left floating. *Expansion pins* refers to the expansion connection pins on the Crazyflie. *Sensor enable* refers to the LPn pin of each sensor.

Pin	Connected to	Default value
GPIO0/BOOT	Button and IO_1	Pulled up to 3V through 10k Ω resistor
EN	-	Pulled up to 3V through 10k Ω resistor
GPIO1	LED blue	-
GPIO2	LED red	-
GPIO3	-	-
GPIO4	LED green	-
GPIO5	Sensor enable, Back	-
GPIO6	-	-
GPIO7	SCL (Expansion pins)	Pulled up to 3V on the Crazyflie
GPIO8	SDA (Expansion pins)	Pulled up to 3V on the Crazyflie
GPIO9	-	-
GPIO10	-	-
GPIO11	Sensor enable, Right	-
GPIO12	SCL (for sensors)	Pulled up to 3V through 2.2k Ω resistor
GPIO13	SDA (for sensors)	Pulled up to 3V through 2.2k Ω resistor
GPIO14	-	-
GPIO15	Sensor enable, Up	-
GPIO16	-	-
GPIO17	RX2 (Expansion pins)	-
GPIO18	TX2 (Expansion pins)	-
GPIO19	USB -	-
GPIO20	USB +	-
GPIO21	-	-
GPIO26	-	-
GPIO33	Sensor enable, Front	-
GPIO34	-	-
GPIO35	MOSI (Expansion pins)	-
GPIO36	SCK (Expansion pins)	-
GPIO37	MISO (Expansion pins)	-
GPIO38	-	-
GPIO39	-	-
GPIO40	Sensor enable, Left	-
GPIO41	IO_2 (Expansion pins)	-
GPIO42	-	-
GPIO43	RX1 (Expansion pins)	-
GPIO44	TX1 (Expansion pins)	-
GPIO45	-	Pulled low internally (Strapping pin)
GPIO46	IO_3 (Expansion pins)	Pulled low internally (Strapping pin)
GPIO47	-	-
GPIO48	-	-

6.1.9 Satellite Board

The schematic of the satellite board contains the VL53L5CX and some passive components as described as in figure in 6.7, as well as a 5-pin connection, which is described in table 6.4

Table 6.4: VL53L5CX connections pins on the satellite board.

Pin	Description
VCC	Operating voltage, regulated to 3V
GND	Ground
SCL	Serial Clock (I^2C)
SDA	Serial Data (I^2C)
SENSOR_EN	Enables, or disables the I^2C communication

6.2 PCB

Once the schematic was complete, it was time to design the PCB. It was going to be a 2-layer PCB, with a thickness of 0.8 mm. The PCB of the Multi-ranger deck was used as a template, because it included the expansion connection pins at the correct positions. It was then a matter of trying to fit all the components within the same size as the Multi-ranger deck. However, due to the size of the ESP32-S3 module, this was not possible, and thus the size of the prototype had to be slightly bigger than the Multi-ranger deck. Figure 6.8 gives an overview of how the prototype was suppose to look from above.

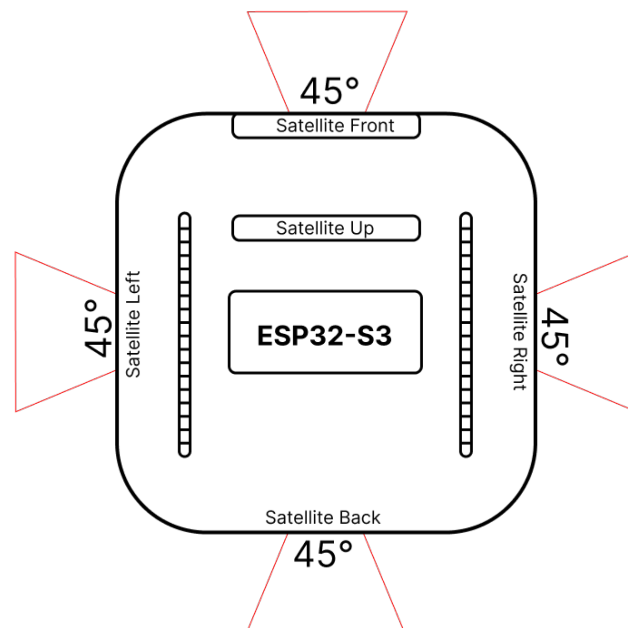


Figure 6.8: Overview of the prototype from above. The main board contains the ESP32-S3, expansion connections pins, and all satellite boards.

Once the size of the board had been increased a few millimeters, all components fit on the board. At this time, the components that belonged together, were placed physically close to each other, which could be important to ensure stable voltage levels. Once all components were in reasonable positions, routing began, in order to connect everything. Ground was connected first, followed by 3V, and then the rest. The ground and 3V lines were thicker than the rest, in order to handle higher currents. Once everything was routed, a ground plane was added to the top and bottom layer of the PCB, to make ground as stable as possible.

Another constraint that had to be considered, was the fact that the ESP32-S3 module has an on-board antenna. This antenna had a "keep-out" zone, which typically means that traces should not be placed over or under this region, to ensure as low electromagnetic interference as possible. Ideally, the antenna would have been placed at the edge of the PCB, so that the antenna would be free in the air, but since a satellite board would be placed on each side of the board, this was not possible. Still, no traces were routed under the keep-out zone.

The top and bottom layers on the PCB were separated though an insulation. In order to connect two layers, vias were used, which are drilled holes, plated with copper. This form an electrical connection between the layers, through the insulation. Figure 6.9 visualizes the working principles of a via.

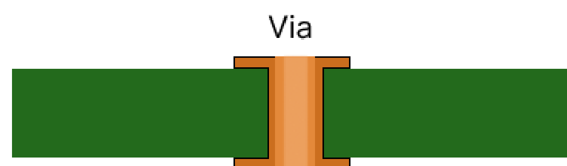


Figure 6.9: Working principles of a via. These are used to connect different layers on a PCB. The insulation is colored green in the figure.

Figures 6.10 to 6.14 shows the final design results of the PCBs, in the 3D viewer of PcNew.

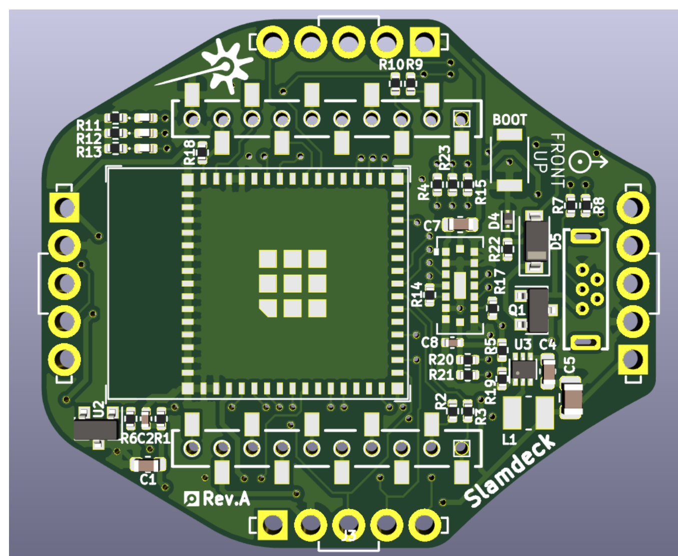


Figure 6.10: Main PCB board front view, in the 3D viewer of PcNew. Most small components have their reference id next to them.

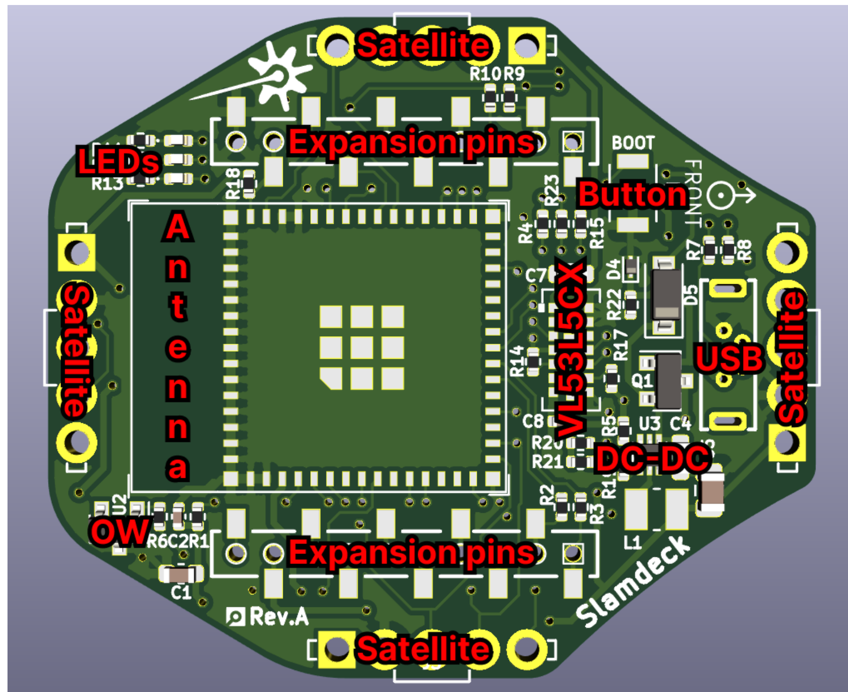


Figure 6.11: Main PCB board front view with description of certain components. OW is the 1-Wire memory.

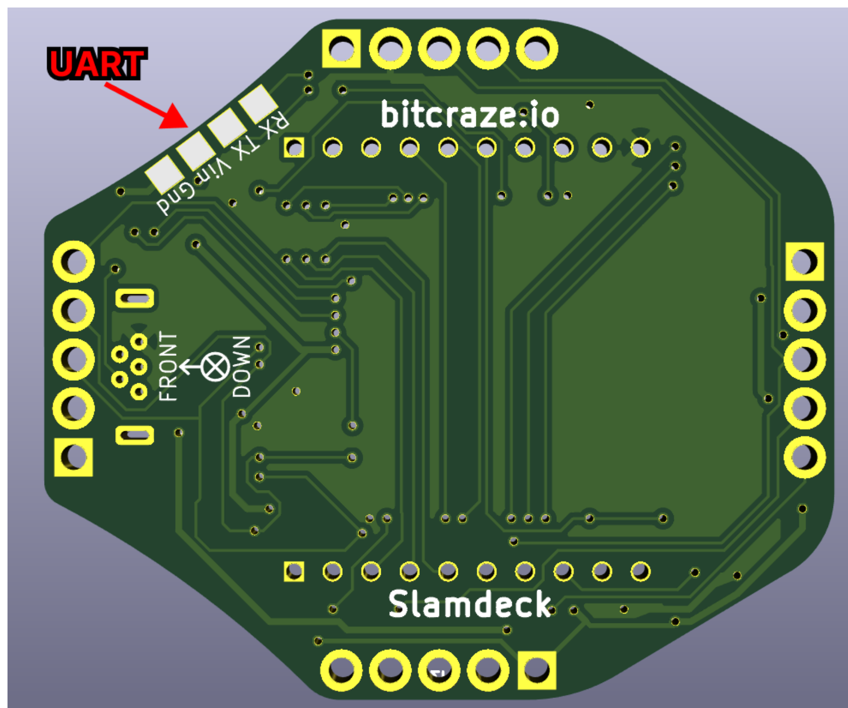


Figure 6.12: Main PCB board back view, in the 3D viewer of PcNew. Note the UART solder pads in the top left corner, which were placed there due to lack of space on the top side, while trying to make them easily accessible.

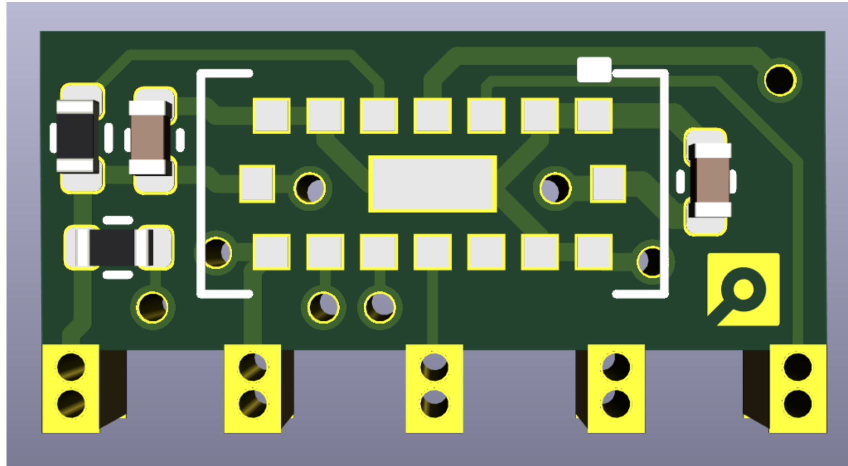


Figure 6.13: Satellite PCB board front view, in the 3D viewer of Pc-New.

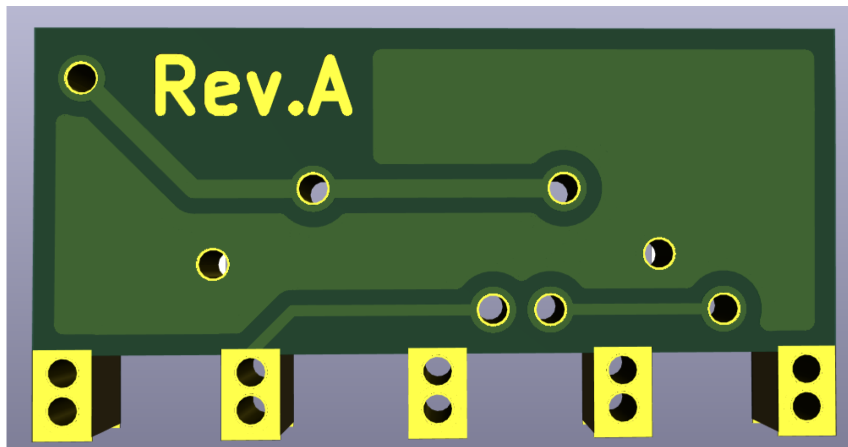


Figure 6.14: Satellite PCB board back view, in the 3D viewer of Pc-New.

6.3 Order Components

Once the PCBs had been designed, it was time to order them, as well as all the necessary components. Table 6.5 includes all components that were necessary for the main board, and table 6.6 includes all components necessary for the satellite boards.

Table 6.5: All components required for the main board.

Component	Quantity	Footprint
Capacitor 10 µF	1	0603
Capacitor 0.11 µF	2	0402
Capacitor 7 µF	2	0603
Capacitor 22 µF	1	0805
Inductor 1 µH	1	Custom
Resistor 0 Ω	6	0402
Resistor 330 Ω	4	0402
Resistor 420 Ω	3	0402
Resistor 2.2k Ω	2	0402
Resistor 10k Ω	3	0402
Resistor 47k Ω	2	0402
Resistor 50k Ω	1	0402
Resistor 200k Ω	1	0402
LED	3	0402
Button	1	Custom
ESD9B3 (TVS diode)	1	SOD-923
1N5819HW (schottky diode)	1	SOD-123
Micro-USB 3130	1	Custom
DMG2301L (PMOS transistor)	1	SOT-23
ESP32-S3 (main MCU)	1	ESP32-S3-MINI-1 (made by Espressif)
DS28E05R (1-wire memory)	1	SOT-23
TPS62A01 (DC-DC converter)	1	SOT-563
VL53L5CX (multizone SSL)	1	Custom

Table 6.6: All components required for the satellite board.

Component	Quantity	Footprint
Capacitor 4.7 µF	1	0402
Capacitor 0.1 µF	1	0402
Resistor 47k Ω	2	0402
VL53L5CX (multizone SSL)	1	Custom

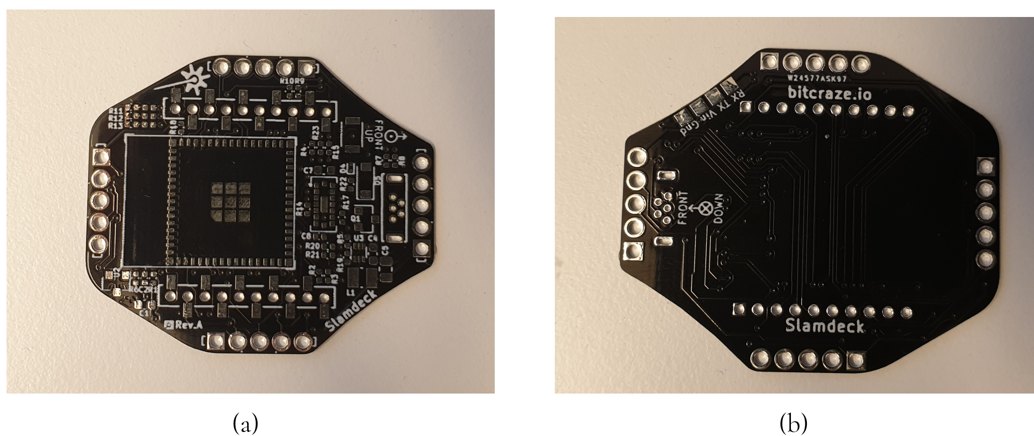
7

Building the Prototype

Once the design of the prototype was finished and the PCBs and necessary components had been ordered, it was time to build the prototype. This chapter corresponds to phase three of the methodology, and is structured as follows: First, the prototype is assembled and built in section 7.1. Second, an overview of the firmware architecture on the ESP32-S3 is given in section 7.2, where each part of the architecture is described in detail. This section continues by presenting the deck driver on the Crazyflie, and ends with a complete example of how the firmware should function. Third, in section 7.3, a quick overview of the GUI architecture is presented and three different data visualizations methods are described.

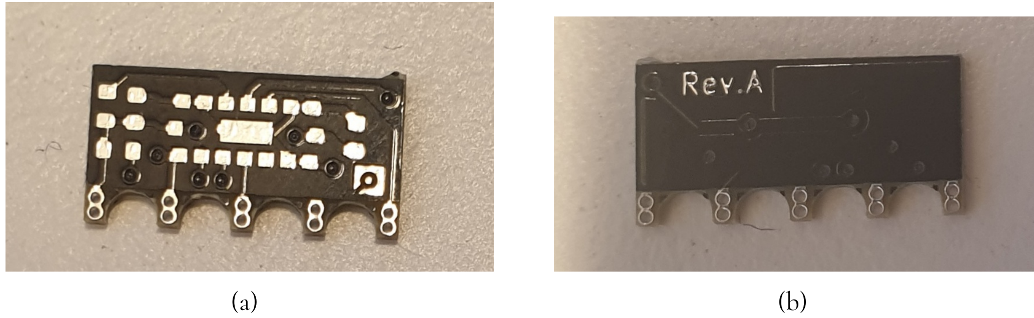
7.1 Assembly

After about two weeks of waiting, the PCBs and all components had arrived. Figure 7.1 shows the main PCB board, and figure and 7.2 shows the satellite PCB board.



(a) (b)

Figure 7.1: Prototype main PCB. (a) Front side. (b) Bottom side.



(a) (b)

Figure 7.2: Prototype satellite PCB. (a) Front side. (b) Bottom side.

Everything was soldered by hand with a soldering iron and a hot air gun, which was especially useful for the components that only had solder pads on the bottom side, such as the ESP32-S3 and the VL53L5CX. Flux was put on some of the components and solder pads to make soldering easier. In order to not get confused of which components had been placed and which were left, a plugin to KiCad, called InteractiveHtmlBom [54] was used, which was helpful during the assembly. Figure 7.3 shows the workbench at Bitcraze, where the prototype was assembled.

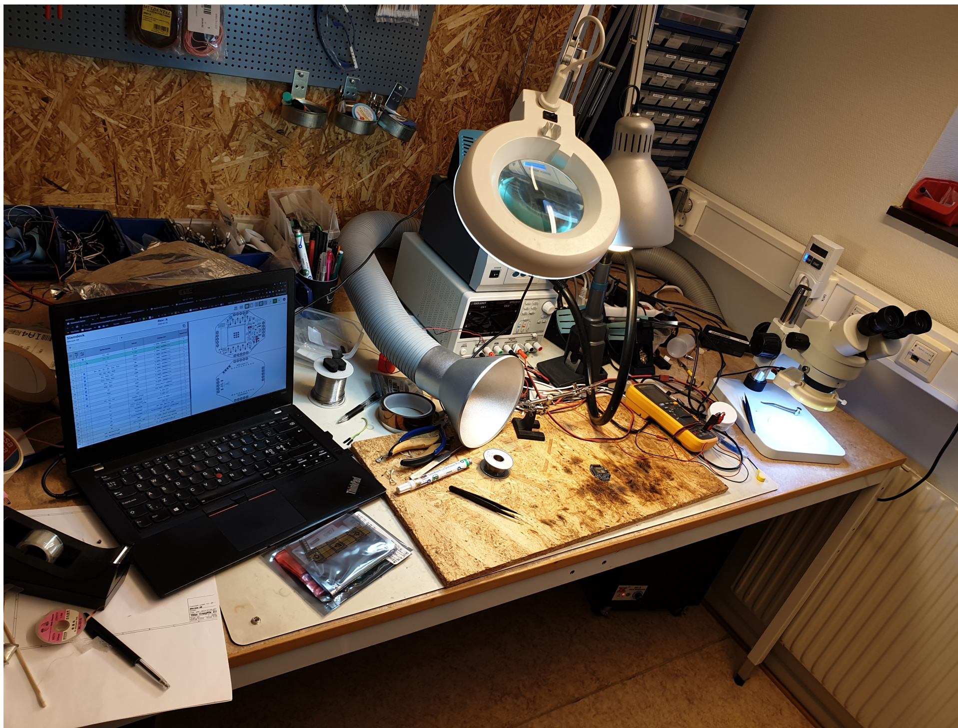


Figure 7.3: Workbench at Bitcraze, where the prototype was assembled, with the InteractiveHtmlBom plugin running on the laptop.

Once most of the components were soldered on the main board, it was time to inspect the connections and try to program the MCU. However, since the pins on the ESP32-S3 module was underneath the chip, there was no way to check that the pins were actually connected, which made inspection significantly harder. Figure 7.4 shows the main board after most of the components had been soldered.

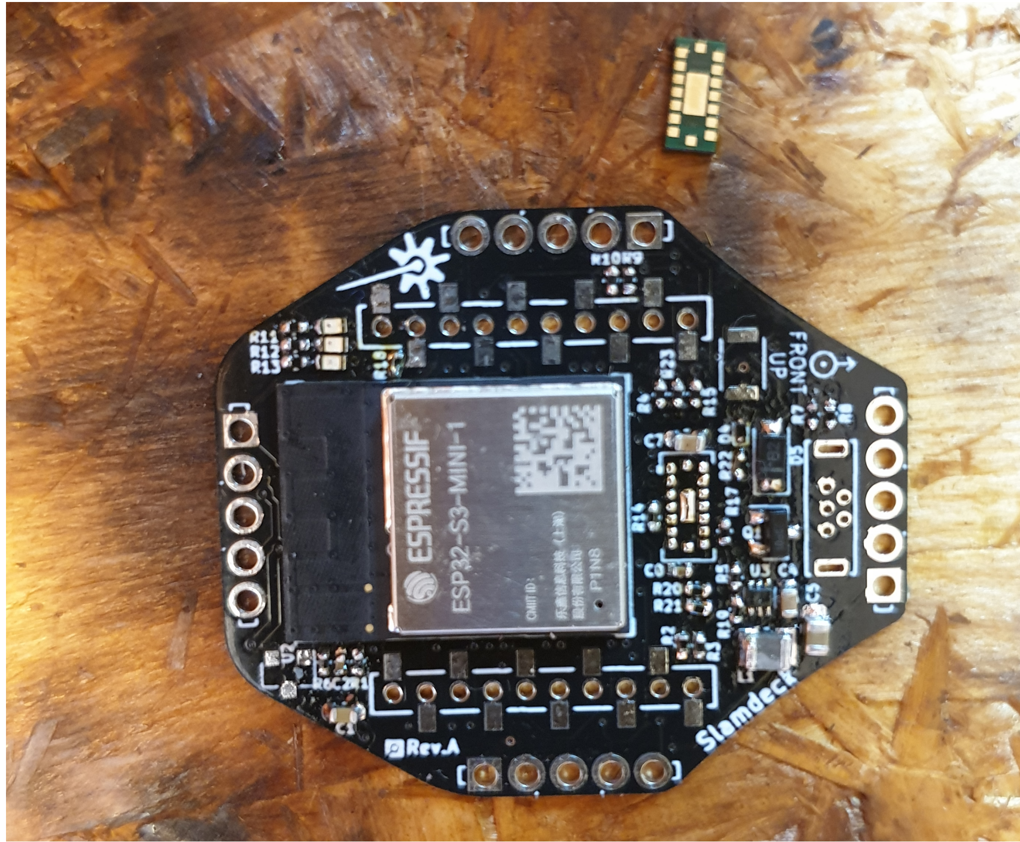


Figure 7.4: Most of the component soldered on the circuit board. The ESP32-S3 is in the middle, and the VL53L5CX sensor is on the top right of the image, still not soldered.

In order to check the functionality of the ESP32-S3, it was plugged in to a PC through USB, and the unix command `dmesg -w` was used in order to monitor the events on the USB bus. Initially, the USB connection did not function properly, which was suspected to be because of bad solder connections. Thus, the ESP32-S3 module had to be re-soldered a few times, and after several iterations, the USB connection was successful, as shown in figure 7.5

```
[110766.022223] usb 1-1: new full-speed USB device number 70 using xhci_hcd
[110766.171792] usb 1-1: New USB device found, idVendor=303a, idProduct=1001, bcdDevice= 1.01
[110766.171806] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[110766.171812] usb 1-1: Product: USB JTAG/serial debug unit
[110766.171816] usb 1-1: Manufacturer: Espressif
[110766.171819] usb 1-1: SerialNumber: 60:55:F9:F5:C5:FC
[110766.175087] cdc_acm 1-1:1.0: ttyACM0: USB ACM device
```

Figure 7.5: ESP32-S3 first successful USB connection. The ESP32-S3 registers itself as a new USB device, and is available through a serial port such as `/dev/ttyACM0` (on Linux).

Once the USB connection worked, simple programs were written, that aimed to check the I/O functionality of the deck. Figure 7.6 shows an example of this, where the LEDs are blinking. Notice that UART pins were also soldered, which were used to check that the ESP32-S3 was booting correctly (during boot, the ESP32-S3 prints out some information from its ROM, like manufacturer, version etc, on the UART).

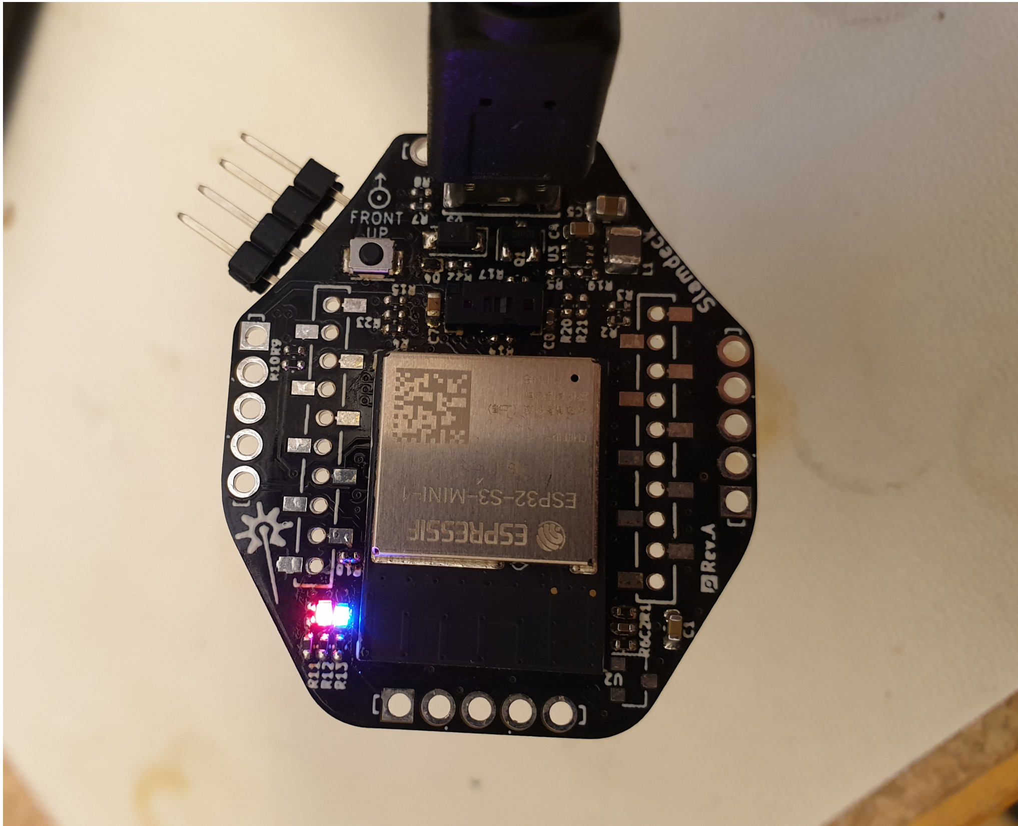


Figure 7.6: Testing the LEDs on the deck.

Once the main board was complete, the satellite boards were soldered as well. Then the communication between the sensors and the ESP32-S3 was tested through the I^2C bus. This also took several attempts to get right and eventually all the satellite boards were working. At this point, all satellite boards were soldered on the main board, and everything but the 1-wire memory had been soldered in place. The reason for this was that the 1-wire memory would not be necessary for the functionality of the deck, and could be added at a later point. Thus, the assembly of the prototype was complete, and is shown in figure 7.8 and 7.7.

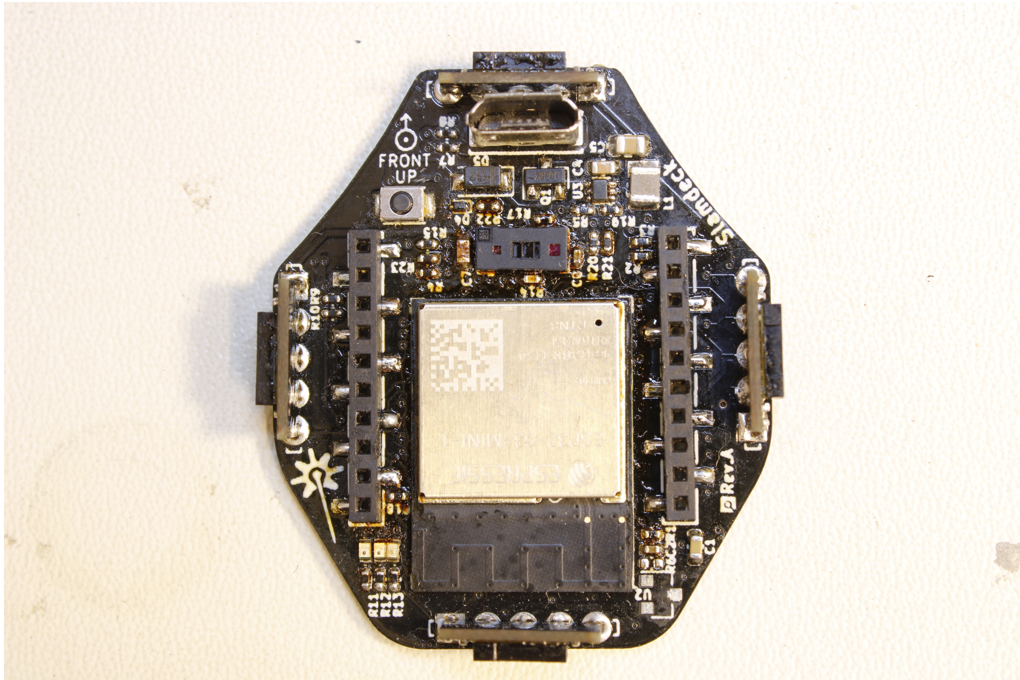


Figure 7.7: Complete prototype deck after assembly, top view.

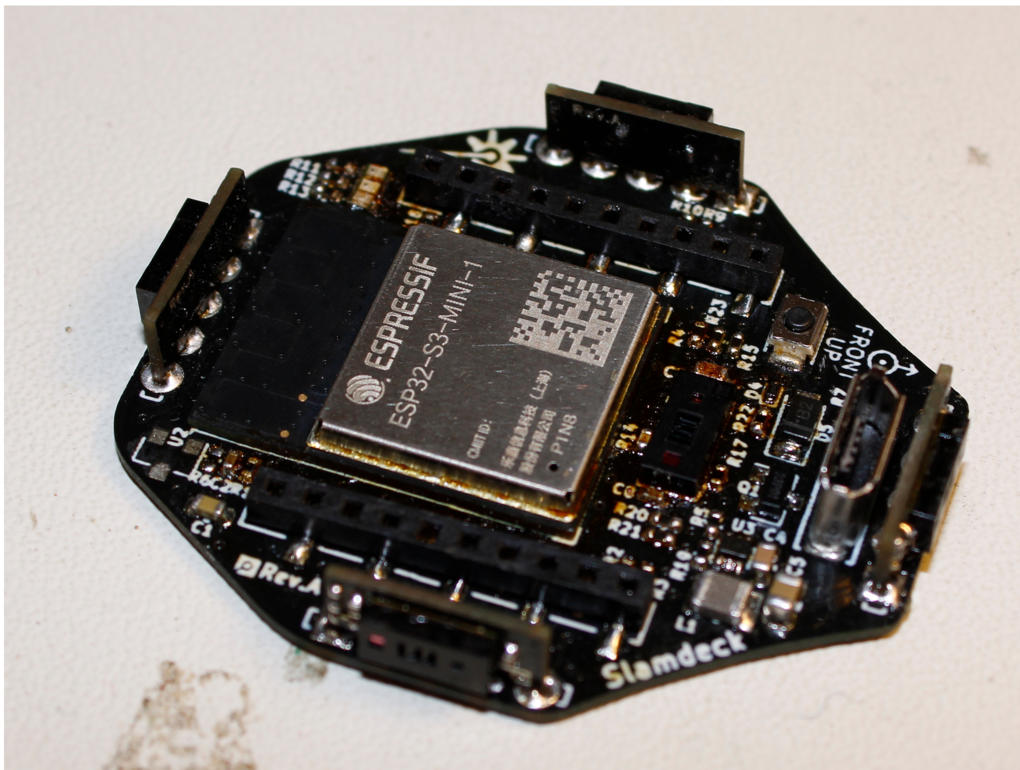


Figure 7.8: Complete prototype deck after assembly, side view.

7.2 Firmware

The ESP32-S3 was programmed with the tools of ESP-IDF, in Visual Studio Code, with the help of the extension Espressif IDF, as explained in section 3.4. While it was possible to setup breakpoint-debugging, it was not free from issues and required a bit of custom tweaks and configurations to get right. While investing time into this might be worth it for some scenarios, for this project it seemed sufficient to use simple printouts as debugging.

7.2.1 ESP32-S3

The main responsibility of the ESP32-S3 was to collect the distance data from the VL53L5CX sensors and send it to a GCS. *How* the data should be sent was not specified, and thus an architecture that would support sending the data through WiFi, *or* any other data link (like the existing nRF radio link on the Crazyflie) was developed. This would make it easy to swap between the different data links if desired. The proposed firmware architecture for the ESP32-S3 is illustrated in figure 7.9 and can be separated into the following modules:

1. VL53L5CX driver
2. Slamdeck
3. Slamdeck API
4. LED driver
5. CPX
6. WiFi driver
7. UART driver

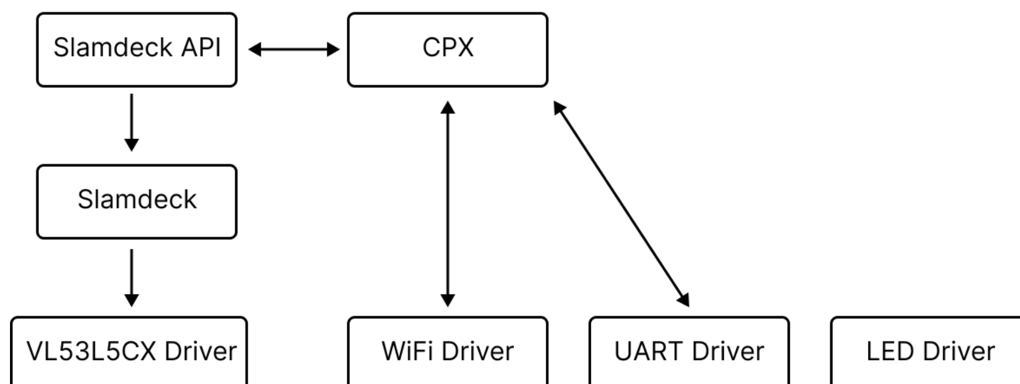


Figure 7.9: ESP32-S3 firmware architecture overview.

VL53L5CX Driver

Like mentioned in section 5.1.2, ST has developed a low-level C driver for the VL53L5CX. This driver required the user to write a few functions that would handle the low-level I^2C communication. The necessary functions were:

- Write 1 byte of data to register X, to sensor with I^2C address Y.
- Write N bytes of data to register X, to sensor with I^2C address Y.
- Read 1 byte of data to register X, to sensor with I^2C address Y.
- Read N bytes of data to register X, to sensor with I^2C address Y.

In order to read or write, to a given register of the VL53L5CX, the ESP32-S3 must first send the sensors I^2C address, followed by the read/write bit. It must then send the register that it wants to read from, or write to, followed by the data. A simplified typical read/write transaction is described in figure 7.10.

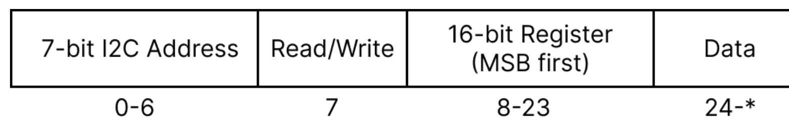


Figure 7.10: A typical I^2C transaction with the VL53L5CX. The numbers below the squares indicate bit index. Note that the transaction is slightly simplified, and does not include ACKs/NACKs, which are necessary for I^2C communication.

Once these functions were implemented, the low-level driver worked, and distance data could be gathered from the sensors.

As presented in table 5.1; one of the available Ranging result of the VL53L5CX is Target status. This is a value that can be read from the sensor as a part of a Ranging result, and it consists of a single byte, that is taken for each zone, and gives a measurement of the validity of the distance data. Thus, it can be used to filter the distance data of the sensor, and improve its reliability. A very primitive filter was implemented for the VL53L5CX driver, where the Target status was checked, to see if a target had been detected at all. If no target was detected, the distance data for that zone was set to the maximum distance (4000 mm).

Slamdeck

It was decided to create a module, that acted as a wrapper around the VL53L5CX driver. This module would initialize and start all 5 sensors, and then continuously collect the distance data from them. Since the interrupt pin of the VL53L5CX was not used, it would be necessary to poll the sensors for new data.

As explained in section 5.1.2, it was possible to configure the settings of the VL53L5CX. While it was not necessary for the project, it seemed useful to include this functionality in the firmware, making it possible for a GCS to read and/or edit sensor settings, without having to re-flash the ESP32-S3. Thus, this module would handle sensor initialization, get/set sensor settings, and collecting distance data. A more comprehensive description of this module can be found in appendix B.1.

Slamdeck API

In order to interact with the Slamdeck module, a simple API was created that would handle the communication to external MCUs/GCS, while keeping the Slamdeck module focusing on reading data from the sensors, and getting/setting their settings.

To make the Slamdeck functionalities available through a data link, like UART or WiFi, a simple protocol was created that made it possible to get/set sensor settings, or read distance data from the sensors. This protocol also enabled streaming mode, where the ESP32-S3 would continuously send distance data through the data link. This way, it would be possible to achieve high data rates which otherwise could be limited, if a GCS had to request the distance data each time it wanted to read new data. A more comprehensive description of this module can be found in appendix B.2.

LED driver

To control the LEDs, a simple LED driver was developed. This driver controlled the GPIO for the LEDs and each led was given a state. The possible states for a LED were:

- ON - LED is constantly on.
- OFF - LED is constantly off.
- Blink at x Hz - LED is blinking at a frequency of x Hz.

x could be any reasonable frequency such as 0.5, 1, 2 Hz. This driver exposed functions that allowed other modules to change the state of the LEDs.

CPX

To make the Slamdeck API independent of the underlying data link, it was decided to use Bitcrazes new CPX protocol (see section 4.7). This way data could be sent to, and from, the Slamdeck API through WiFi, nRF, or any other link in the future, without the Slamdeck API being aware of the underlying data link. Figure 7.11 visualizes how CPX was expected to work on the prototype.

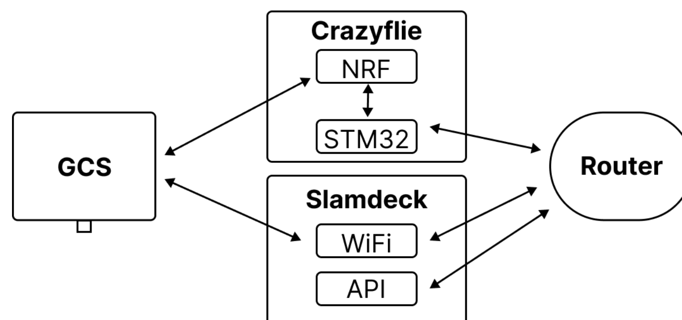


Figure 7.11: Simplified overview how CPX should work on the prototype. WiFi and API are two separate software modules on the ESP32-S3. Note that the router is not a physical device, but a software module.

Note that as seen in figure 7.11, there is no connection between the WiFi module and the API module, as the data always goes through the router. Figure 7.12 shows a flow-chart that describes the process of getting data from the Slamdeck API to the GCS.

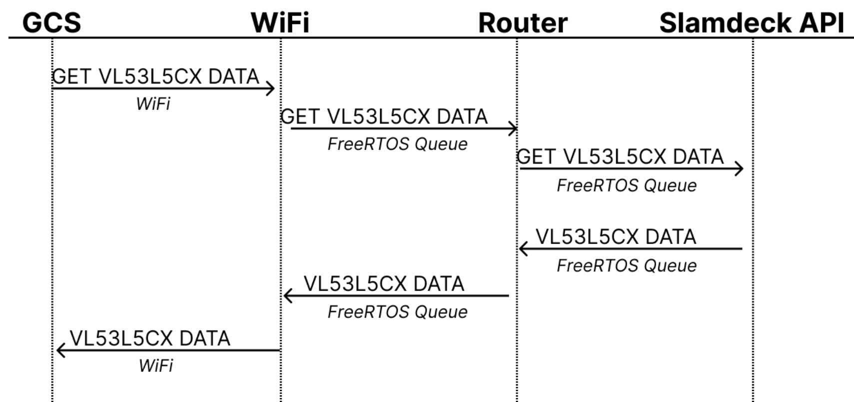


Figure 7.12: Flow chart of getting sensor data from the Slamdeck API to the GCS, via WiFi. The italic text under the arrows indicates what the underlying data link is, and GCS, WiFi, Router and API can be seen as separate software modules.

Figure 7.13 shows the same process of getting data, but this time the WiFi link is changed for the nRF radio link on the Crazyflie.

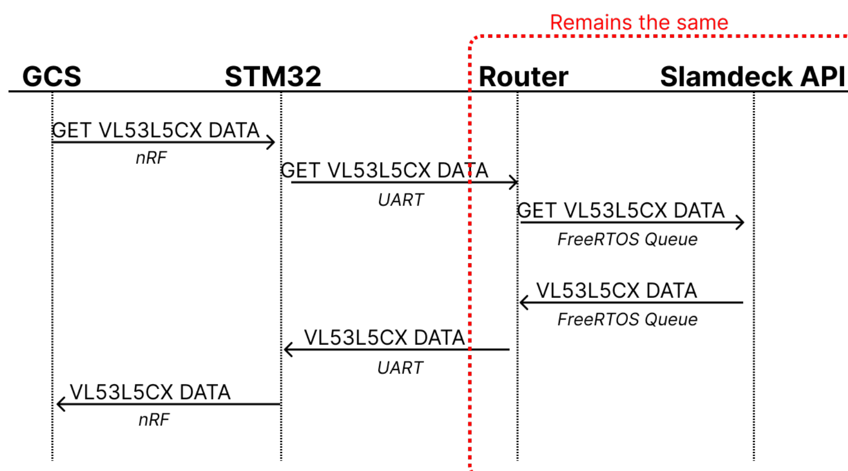


Figure 7.13: Flow chart of getting sensor data from the Slamdeck API via the nRF on the Crazyflie. The process is slightly simplified, as the data is actually sent between the nRF51822 and STM32 as well. Note that the right part (highlighted) remains the same as in figure 7.12. The italic text under the arrows indicates what the underlying data link is. GCS, WiFi, Router and API can be seen as separate software modules.

As one can observe from figure 7.13, the part to the right (highlighted in red) remains the same, regardless of the underlying data link. Thus, an architecture that allows the Slamdeck API to be independent of the underlying data link had been achieved.

WiFi Driver

A simple WiFi driver was developed, based on existing code for the AI deck (see section 4.6.3). This driver had to be able to connect to a WiFi network, listen for incoming connections on a given TCP port, and once a connection has been made, it would send, or receive packets to and/or from CPX. An overview of the driver is visualized in figure 7.14.

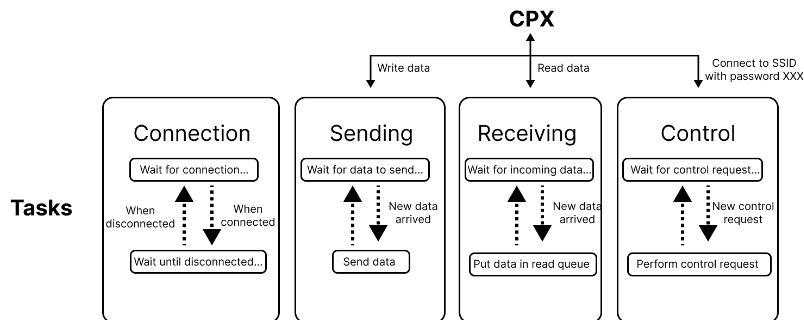


Figure 7.14: WiFi Driver overview. *Tasks*, refers to a FreeRTOS task.

Through the `Control` task, the WiFi Driver can be configured to connect to a network, with a given SSID and password. Once this is done, a TCP socket is opened and the `Connection` task is waiting for someone to connect. Once a client has been connected, the `Receiving` task listens for incoming data, and once new data has been received, it is put in a queue. Other modules can then collect the data from the queue through CPX. The `Sending` task waits for packets to arrive from CPX, and then sends them through the socket. Note that only one client could be connected at the same time, which could be extended, but at the time there was no need for it, and thus the functionality was not implemented.

UART Driver

The implemented UART driver worked similar to the WiFi driver, but since UART is more primitive than WiFi, there was no need for a Connection nor Control task. It was however necessary for the UART transmitter to be synchronous with the receiver. Otherwise, the transmitter might keep sending data even when the receiver is not ready to accept it. The implementation of the synchronization was done as explained in section 4.7.1. The UART driver is visualized in figure 7.15.

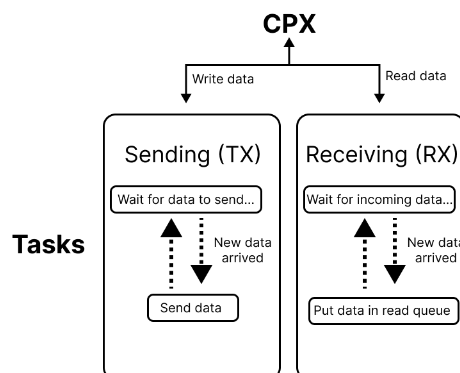


Figure 7.15: UART driver overview.

7.2.2 Crazyflie

Two modifications were made to the firmware of the STM32 on the Crazyflie: 1) A deck driver. 2) An extension to the CPX protocol. The code compilation and flashing process was done with the help of Bitcrazes development tools.

Slamdeck Deck Driver

In order for the Crazyflie to identify and communicate with the prototype deck, a deck driver had to be written and flashed into the firmware of the STM32. When the Crazyflie boots, it checks the 1-wire memory pin for available decks, and if the prototype is mounted on the Crazyflie and its 1-wire memory is identified, then the Slamdeck deck driver would be initialized. However, since the 1-wire memory had not been soldered yet, the identification of the memory was skipped and the deck driver was instead forced to initialize, with the help of a compile flag. This way, the Crazyflie would always initialize the Slamdeck deck driver on startup.

This deck driver would be responsible for resetting the ESP32-S3, and initializing a UART driver on the STM32, which would be used to communicate with the ESP32-S3. Both microcontrollers would then use CPX over UART to communicate. Resetting the ESP32-S3 was done by pulling its reset pin low, and then releasing it again. It was important that the UART was initialized *after* the reset pin was pulled low, and *before* it was released, because the two MCUs had to be in sync for the UART communication to work properly. Once this was done, the Crazyflie and the prototype deck would be able to communicate.

CPX over CRTP

Like mentioned in section 4.7, at the time of project initialization, there were only three transport methods available for CPX: UART, SPI and WiFi. Thus, in order to send data from the ESP32-S3, with CPX through the nRF radio, the CPX protocol had to be extended. An implementation of this was made, which allowed the ESP32-S3 to send data to a GCS, where the data was first sent to the STM32 through UART, which then sent it to the nRF51822 through SPI, which would finally send the data to the GCS through the radio, using the CRTP protocol. To extend the CPX protocol, an additional `Target` was added: `HOST_NRF`, in order to differentiate between sending packets to the GCS through WiFi and nRF.

7.2.3 Working Example

Figure 7.16 gives a visual example of how the process looks when a GCS wants to read distance data from the sensors. The figure is simplified to avoid (too much) confusion, but should still be sufficient to give an idea of how the firmware architecture is suppose to work. Note that in the example, the deck is not connected to the Crazyflie, and thus WiFi is used as the data link. However, this could be changed, and if the deck was mounted on the Crazyflie, it could use the nRF radio link instead.

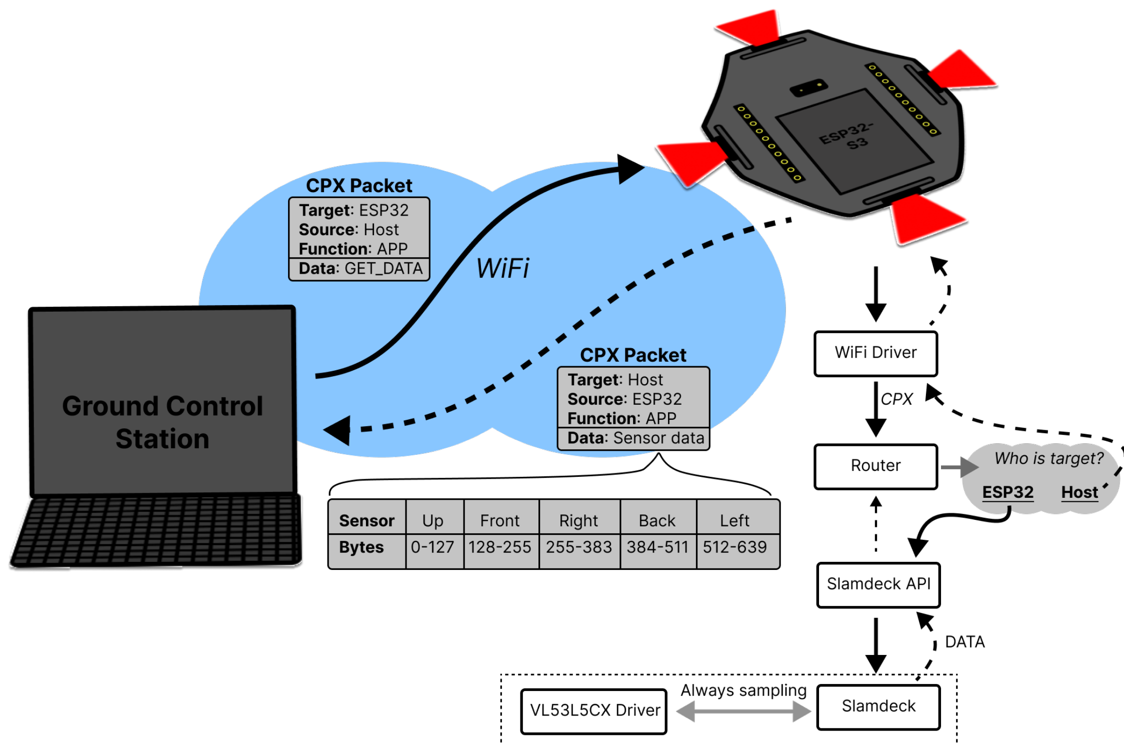


Figure 7.16: Working example of the firmware when a ground control station wants to read distance data from the sensors. The solid black lines indicate that the data goes *from* the GCS, and dashed black lines indicate that the data goes *towards* the GCS.

The steps in figure 7.16 can be summarized as follows:

1. GCS Sends a CPX packet over TCP through WiFi to the ESP32-S3. This packet contains the Slamdeck API command `GET_DATA`.

2. The WiFi driver on the ESP32-S3 receives the CPX packet from the TCP connection, and gives it to the router, which ensures that the Slamdeck API gets the packet.

3. Slamdeck API inspects the command in the data field of the CPX packet, which is `GET_DATA` and then gets the sensor distance data from the Slamdeck module.

4. Slamdeck API creates a new CPX packet to the GCS with the sensor distance data and hands the packet to the router.

5. The router ensures that the CPX packet is given to the WiFi driver, which finally sends the data packet to the GCS.

Now that a firmware architecture had been developed, appropriate software had to be written in order to communicate with the deck.

7.3 Software

A simple GUI was created, in order to communicate with the deck. It was developed in Python with PyQt, because Cfclient (see section 4.5) was also built with this. This way, if it is desired, it would be possible to extend the Cfclient with parts from this project in the future. The GUI was decided to have the following functionalities:

- Choose what data link to use (nRF, WiFi etc).
- Connect/disconnect to the deck.
- Read distance data from the deck.
- Start/stop streaming data from the deck.
- Get/set sensor settings.
- Visualize the distance data.

In order to choose what underlying data link to use when communicating with the deck, a backend abstraction was created. This abstraction allowed the frontend to treat the backend the same, regardless of what data link was used. The GUI was built around the Model View Controller (MVC) architecture. Figure 7.17 gives an overview of the different modules of the GUI.

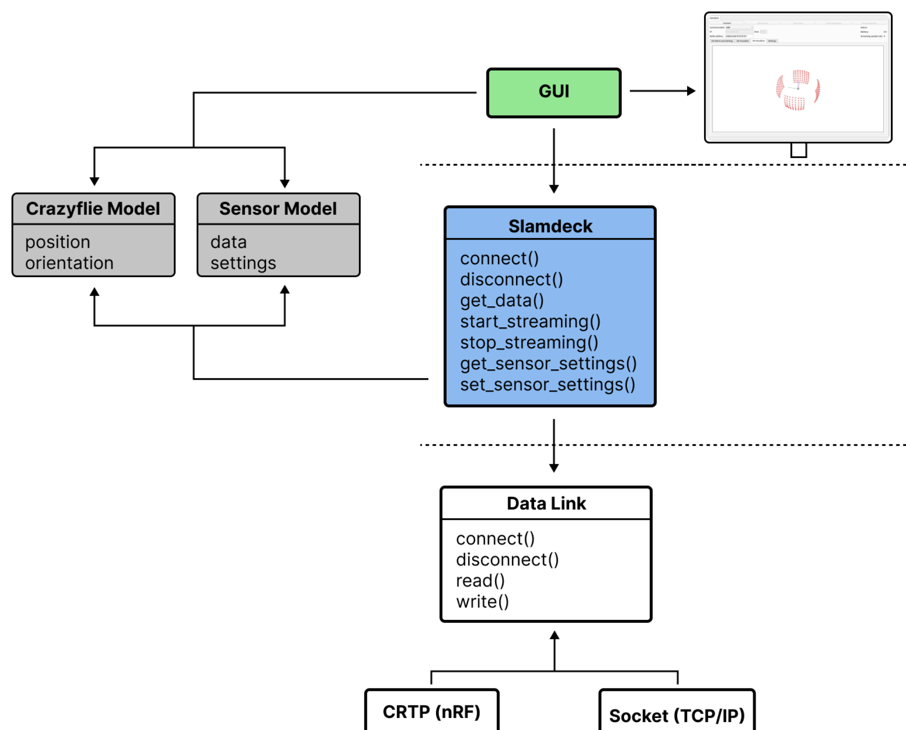


Figure 7.17: GUI Overview. Each box represents a separate software module. Referring to the MVC architecture; the *Models* are the gray boxes to the left, the *Controller* is the blue box in the middle, and the *View* is the green box at the top.

The models has the following responsibilities:

Models (gray boxes) - Holds data that the GUI can visualize. There is a `Sensor model`, for the VL53L5CX sensors, and a `Crazyflie model`, which represents the Crazyflie drone. The reason for having a Crazyflie model as well, is that the position and orientation of the drone can be helpful when visualizing the distance data.

Controller (blue box) - Is responsible for the communication with the deck, as well as updating the data of the models.

View (green box) - Displays the appropriate data on the GUI and handles user interaction, like button clicks. The view updates the User Interface (UI) at specific frequencies with the model data. This way, the view is completely independent of the backend and the underlying data link.

In order to get the position and the orientation of the Crazyflie, Bitcrazes Python library Cflib was used, which has an extensive logging framework that makes it possible to log parameters from the Crazyflie. Once most functionalities of the GUI had been developed, it was time to create data visualization methods.

7.3.1 Data Visualization

In order to visualize and better interpret the distance data, three different visualization methods have been implemented. As a reminder; the distance data returned from the VL53L5CX sensor is flipped vertically and horizontally as explained in figure 5.3, which the visualization methods has to compensate for.

2D Visualization - Matrix

In order to visualize the data in two dimensions, a simple two-dimensional matrix was constructed from the data. The size of the matrix depended on the resolution of the sensor (4x4 or 8x8). The distance data was then converted into a color, which represented how far the distance was (0-4000 mm). Figure 7.18 shows a simplified example of how this would look if a hand was put in front of the sensor that covered half of the FoV.

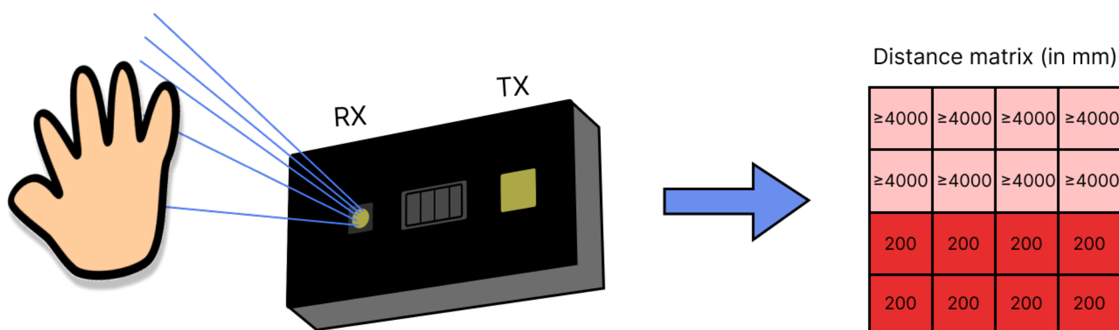


Figure 7.18: Simplified example of 2D matrix visualization. A hand is covering half of the sensors FoV. The darker red in the 2D matrix means that an object is closer to the sensor.

An implementation was created where the 2D matrix is visualized for a single sensor, and the user can select which sensor to use. Another implementation which visualizes the 2D matrix for all five sensors at the same time was also implemented, which is illustrated in figure 7.19.

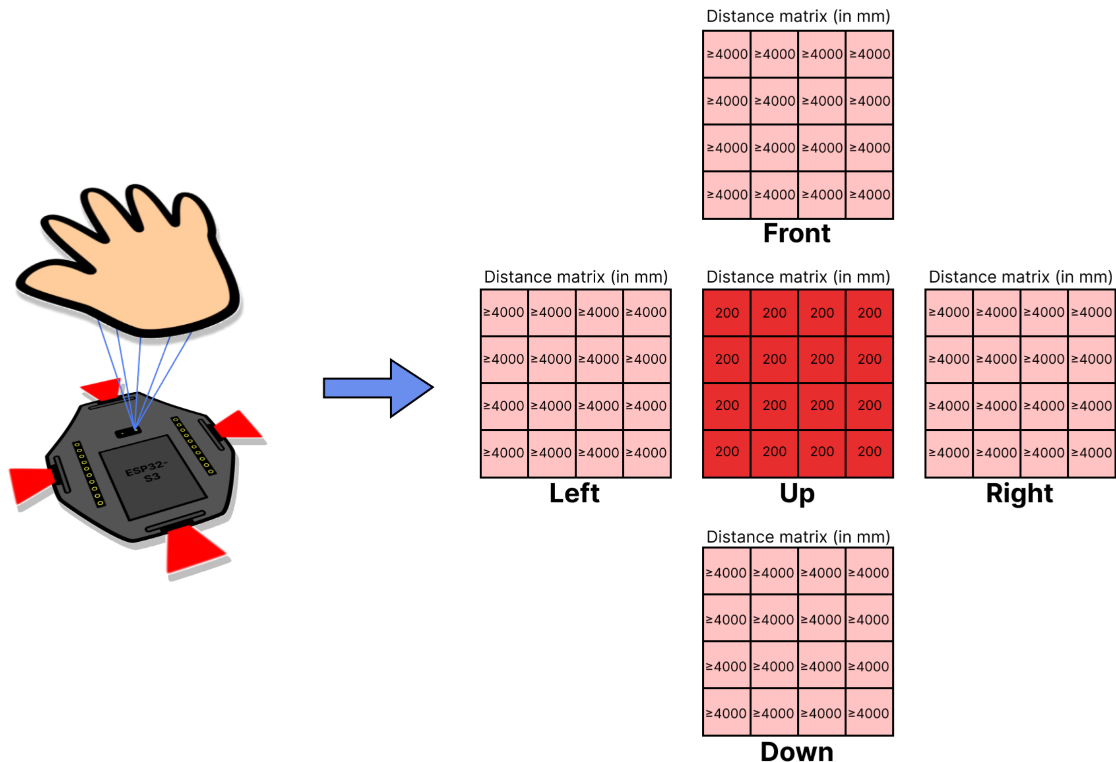


Figure 7.19: Simplified example of 2D matrix visualization with all five sensors. The hand is completely covering the FoV of the sensor facing up, and the other sensors has not detected any objects at all.

2D Visualization - Point Cloud

Another way to visualize the data in two dimensions could be to create a 2D point cloud, where the distance data for each zone is represented as a single point. The camera of the scene (the observer) would be looking at the deck from above, and thus, the distance data from the sensor facing up would not be used. However, one must decide which of the rows from the output matrix that should be used in order create these points. For a resolution of 4x4, there are a total of 4 rows that can be used. The results of the 2D point cloud might be significantly different if data from the top row is used, instead of the bottom row. This is illustrated in figure 7.20.

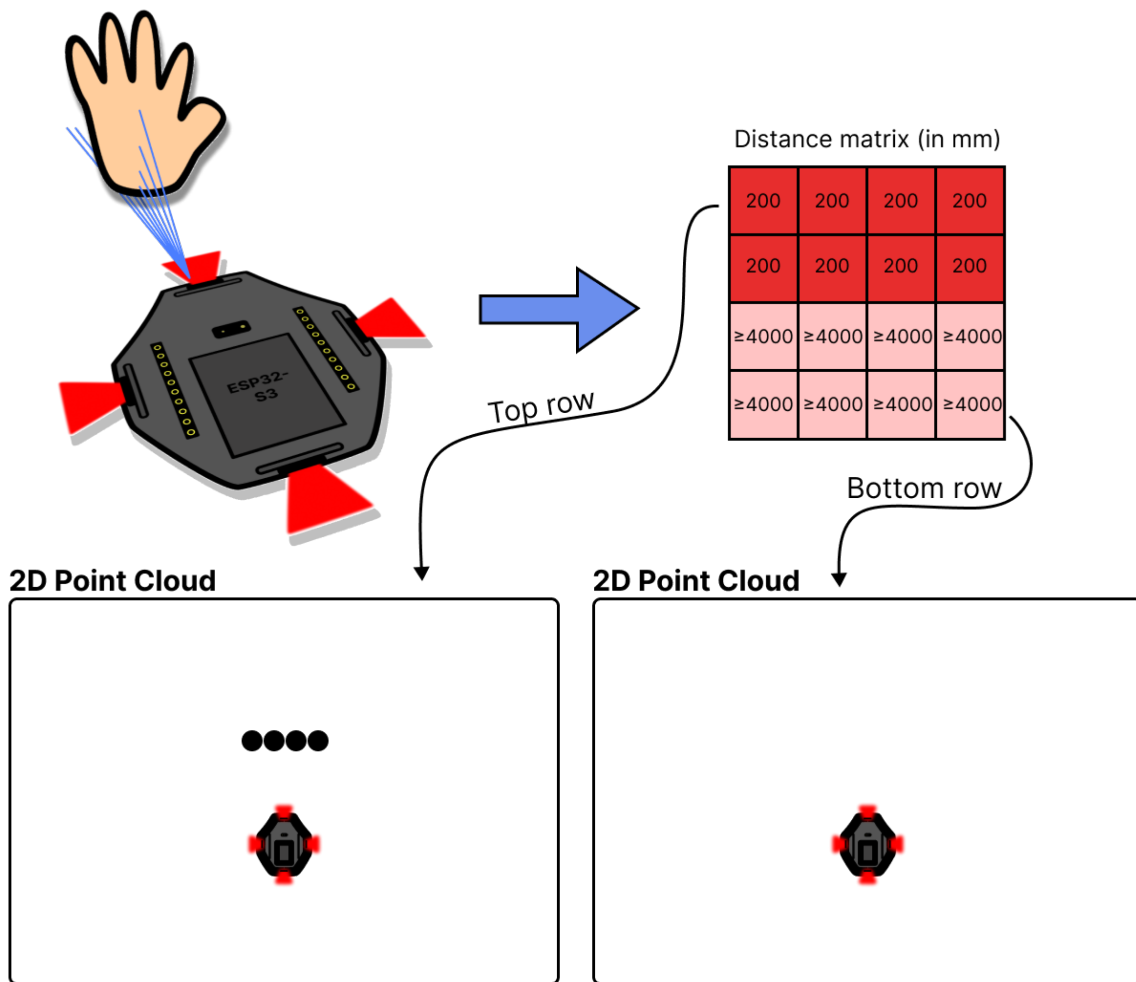


Figure 7.20: 2D Point cloud, using only the top row of the distance data (left), or using only the bottom row of the distance data (right).

Which of the rows should be used could depend on the situation. For instance, assume that the Slamdeck is mounted on the Crazyflie and one wants to detect walls around the drone. If it is flying very low, then the bottom row(s) might detect the floor, due to the sensors wide FoV. In this scenario, it might be more appropriate to only use the upper rows. Thus, an implementation was created that allows the user to choose which of the rows of the 2D matrix that should be used, in order to create the 2D point cloud. If multiple rows are used, then the distances are averaged for each column, which is illustrated in figure 7.21.

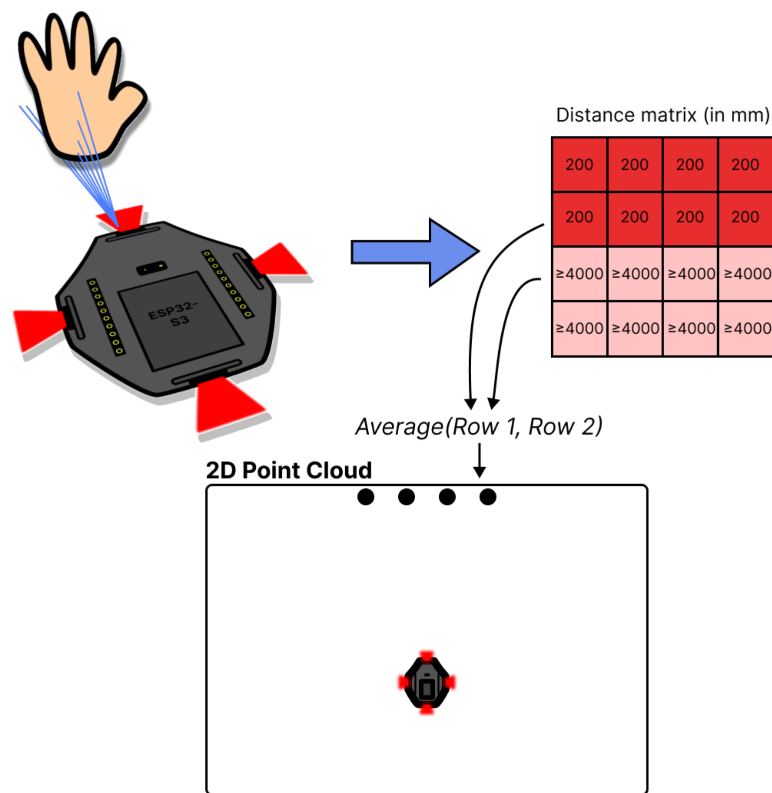


Figure 7.21: 2D Point cloud, using row 1 and 2. The distance data is averaged between the rows, for each column.

If the orientation of the Crazyflie is not taken into account, the point cloud will always be around the same coordinate system, and not follow the Crazyflies orientation. Thus, it would be necessary to know the yaw of the Crazyflie, as well as its x and y position, in order to create a 2D point cloud around the drone. This was implemented and thus, the 2D point cloud is fixed around the Crazyflie, and follows its position and orientation.

3D Visualization - Point Cloud

Another way to visualize the data could be to use three dimensions. The benefit of this, compared to the 2D visualizations mentioned above, is that it might be easier to interpret that distances in all directions. However, the downside of this is that it is slightly more complicated to implement.

A primitive 3D visualization was implemented, with the help of VisPy, which is a Python library that makes it easier to create and develop 2D and/or 3D visualizations. VisPy uses OpenGL to render the graphics [49], and it can render graphics on any OpenGL scene, and thus it was easy to plug into PyQt.

The 3D visualization created a point cloud, but in three dimensions instead of two. This meant that distance data from all five sensors could be visualized at the same time. To create a 3D point cloud, the distance measured from a single zone of a sensor, first had to be transformed to the *sensors* local coordinate system, which then had to be transformed to the *Crazyflies* local coordinate system, and at last, it had to be transformed to the *global* coordinate system (see section 4.2). Thus, some linear algebra was necessary. As a quick math

reminder; recall that to rotate a matrix in 3D, the matrices are multiplied, and to move a matrix in 3D, the matrices are added.

The following steps were used in order to project distances from a single sensor on to the 3D scene (the deck is assumed to be mounted on the Crazyflie, and thus have the same position and orientation as the Crazyflie):

1. Convert distance data into a 2D matrix.
2. Convert each zone in the matrix to a 3D vector by setting its third dimension to 1, as in figure 7.22. This results in the distances to be projected in 3D, in the sensors local coordinate system.

The output from this step should be an array of 3D vectors that represents the distances in the local coordinate system of the sensor.

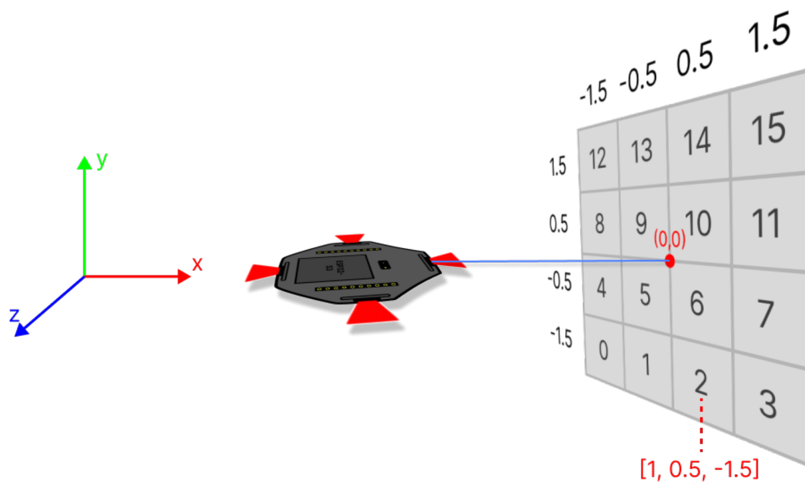


Figure 7.22: 2D distance data matrix projected to 3D around the sensors local coordinate system. The third dimension is given value 1 for each cell in the matrix. Note that the origin (0,0) is in the center of the matrix, thus the y and z components of the zones are not integers.

The following is then done for each 3D vector:

3. Scale the vector by the distance in the zone.
4. Scale the vector by a constant to fit the scene.
5. Rotate the vector with a rotation matrix that corresponds to the orientation for the given sensor. This rotation transforms the distances from the sensors coordinate system to the Crazyflies local coordinate system. As an example, matrix 7.1 shows the rotation matrix for the sensor facing up.

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{7.1}$$

If this is done for each sensor, the result should look like in figure 7.23.

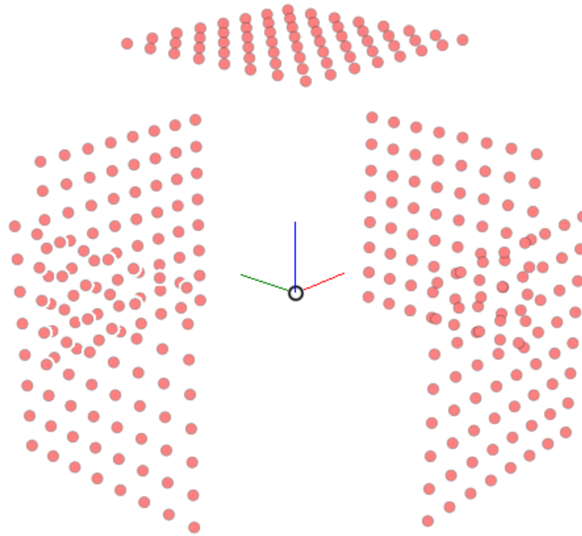


Figure 7.23: 3D point cloud visualization in VisPy. All points are projected in the Crazyflies local coordinate system.

However, the points are currently independent of the position and attitude of the Crazyflie. Thus, to make the points follow the Crazyflies position and attitude, they must be transformed to the global coordinate system. This can be done with the following steps:

6. Convert roll, pitch and yaw of the Crazyflie into a rotation matrix, with equation 7.2.

$$\begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\ \sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \quad (7.2)$$

   Roll, pitch and yaw are represented as γ, β and α respectively [52]. While an explanation of this matrix is out of scope for this thesis, the take-away is that it converts the attitude of the Crazyflie into a rotation matrix.

7. Rotate each vector by the rotation matrix, calculated from equation 7.2.

8. Move each vector to the position of the Crazyflie.

This results in a 3D point cloud that follows the position and orientation of the Crazyflie, where each point represents the distance data, from a single zone in the sensors 2D output.

A simple flight controller was also added to the GUI which made it possible to fly and steer the Crazyflie from it. This code was based on existing code from Bitcraze, which used Cflib to communicate with the drone.

8 Prototype Evaluation

Once the prototype deck had been built, firmware had been developed, and software had been created to communicate with the deck and visualize the distance data, it was time for evaluation. This chapter corresponds to final phase of the methodology, phase four, and is structured as follows: First, the complete functionality of the prototype is summarized. Second, the results of the prototype are presented. Third, the thesis questions presented in section 1.2 are answered, and at last, a discussion of the results is given.

8.1 Functionality

Assuming that the prototype deck is mounted on the Crazyflie, the functionality of the prototype can be summarized as follows:

1. Once the Crazyflie is turned on, it reads the 1-wire memory attached on the deck in order to identify it, and initializes the correct deck driver for it. Note; The 1-wire memory was never soldered on the deck, and thus the correct deck driver is instead simply forced to initialize.
2. The deck driver on the Crazyflie resets the ESP32-S3, by pulling its reset pin low, initializes a UART driver, and then releases the reset pin, which makes the ESP32-S3 start booting.
3. The ESP32-S3 initializes I^2C, and then initializes the VL53L5CX sensors one by one, assigns them a unique I^2C address, and sets default sensor settings.
4. The ESP32-S3 starts the sensors and continuously collects data from them.
5. The ESP32-S3 waits for incoming commands through CPX.
6. Once a command has been received from the CPX router, it is given to the Slamdeck API, which decodes the command and executes the appropriate action.

Assume that a GCS sends the command `START_STREAMING`.

7. The Slamdeck API starts sending distance data, collected from the sensors, through CPX to the GCS. This is done either by WiFi, or through the nRF radio, via the Crazyflie, depending on the `Target` field of the CPX packet.

8. The GCS receives the packets and decodes them to get the distance data.

Assume that the GUI is running.

9. The Controller updates the sensor models with the distance data.

10. The View continuously updates the UI with the data of sensor models, and visualizes the data with any of the visualization methods that were presented in section 7.3.1.

Thus, the implemented functionality allows distance data to be collected from all five VL53L5CX sensors by the ESP32-S3 microcontroller, which can then send the data to a ground control station through WiFi, or the nRF radio. This distance data can then be visualized in three different ways on the GCS.

8.2 Results

The source code of the project is open-sourced and freely available on Github at `https://github.com/victorhook/slamdeck`, with GNU GPLv3 license.

The functionality of the prototype was tested with, and without being mounted on the Crazyflie. In both scenarios, the microcontroller was able to collect the distance data from the sensors, and send it to a ground control station. Several flight tests were also performed, where the prototype was mounted on the Crazyflie while it was flying, which was made possible with the flight controller that was added to the GUI. During flight, the Flowdeck (see section 4.6.2) was mounted to the bottom of the Crazyflie, which helped the drone estimate its position. During testing, several results were found which are described below.

8.2.1 Initialization Time

A single sensor took approximately 1.4 seconds to initialize and another ~0.5 second to start ranging, which meant that all sensors took ~9.5 seconds to initialize and start.

8.2.2 Operating Temperature

While it was not tested during flight, when the prototype was standing still it was noticed that the sensors were getting very hot at high ranging frequencies. Especially the front sensor, which was placed close to the power-chain and the USB connection. The sensors shutdown after reaching 110° C, which actually happened when ranging with resolution of 4x4 at 60 hz, but the sensor temperatures decreased at lower ranging frequencies. The temperature was monitored by reading the sensors inner silicon temperature, which was a part of the Ranging result (see section 5.1.2), and it was also tested with an IR-camera, which measured almost the same temperatures.

8.2.3 Distance Data

While no strict distance experiments were performed, the distance data received from the sensors seemed to be accurate at short distances, but for distances above ~2 meters it seemed to be less reliable. It was also noticed that if there was no clear object in front of the sensor, for instance if it was facing an open room with furnitures, it would take some time for the distance values to converge to match the objects in the room.

Furthermore, the distance data was rather noisy, and the filter that was implemented in section 7.2.1 had the result that zones were often assigned the maximum distance (4000 mm) if there was no clear object within ~2 meters of the sensor.

8.2.4 Power

It was noticed that when the deck was mounted on the Crazyflie, and the drone started to take off, the ESP32-S3 would sometimes reset itself. However, this only occurred when the battery was not fully charged.

8.2.5 Data Rates

The prototype used a total of three different data links: UART, WiFi and nRF. Whether a CPX packet should be sent through WiFi, or through nRF via the Crazyflie, is decided by the `Target` field of the packet. The maximum data rate that was necessary to send the distance data, which had one Targets per zone, resolution of 8x8, and a ranging frequency of 15 hz, was 9.6 kB/s. Thus, all data links needed to handle at least this data rate.

All data links could indeed handle this data rate, and were sufficient for the distance data to be sent to a GCS. However, it was noticed that the data was sometimes slightly delayed when it reached the GCS. The results of the three data links can be summarized as:

UART - Used between the ESP32-S3 and the STM32 on the Crazyflie.

The UART was configured to 115.2 kbit/s, and even though CPX adds three bytes of overhead per packet (see section 4.7.1), UART was sufficient to send the distance data at the necessary data rate.

WiFi - Used between the ESP32-S3 and the GCS.

WiFi supports high data rates, and thus sending 9.6 kB/s was no problem. However, WiFi proved to sometimes have high latency issues, especially in noisy environments. This meant that there were sometimes long delays, up to one second, between communications. However, this latency was only causing an issue when sending data back and forth between the GCS and the ESP32-S3. If streaming mode was used instead, so the ESP32-S3 only sent data, and the GCS only received data, the latency was no longer an issue.

nRF - Used between the ESP32-S3 and the GCS, via the Crazyflie.

As explained in 7.2.2, an extension to the existing CPX protocol was implemented, which enabled CPX packets to be sent with CRTP, over the nRF radio. The measured

data rates of this implementation were approximately 10 kB/s, which was sufficient to send the distance data at the necessary data rate.

8.2.6 GUI & Data Visualization

The GUI that was developed made it possible to connect to the ESP32-S3 through the nRF radio via the Crazyflie, or directly through WiFi. Once connected, it was possible to get/set sensor settings, start/stop streaming data, or read the distance data from the sensors, as well as fly and steer the Crazyflie. The data could then be visualized in three different ways:

1. 2D Matrix
2. 2D Point cloud
3. 3D Point cloud

Figure 8.1 shows a screenshot of the entire GUI when the computer was connected to the ESP32-S3 through the nRF radio with the Crazyradio PA dongle, and reading distance data from the deck.

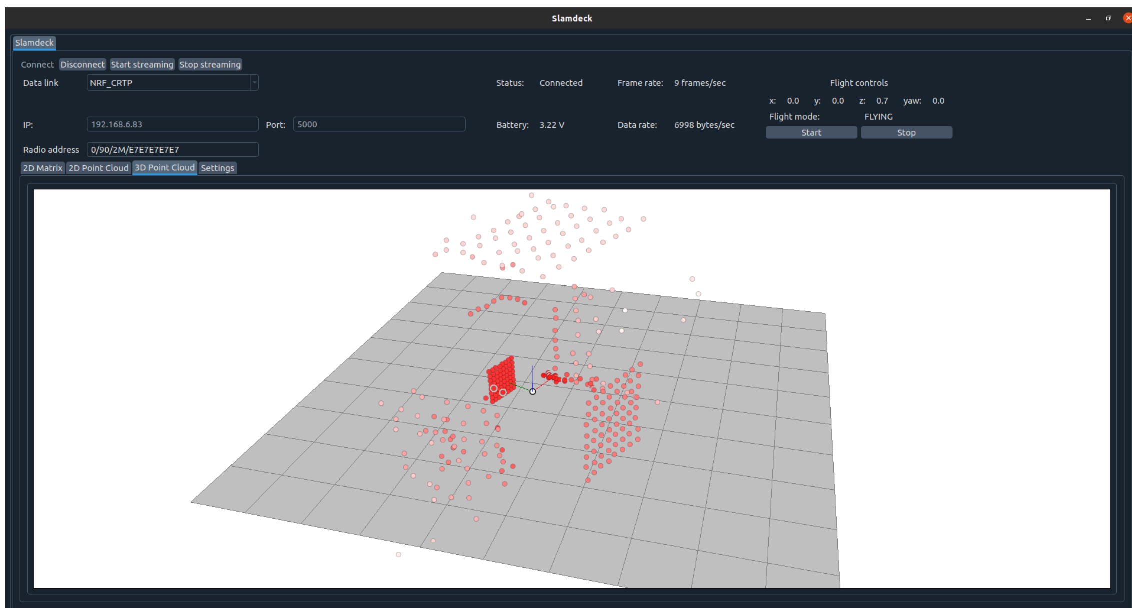
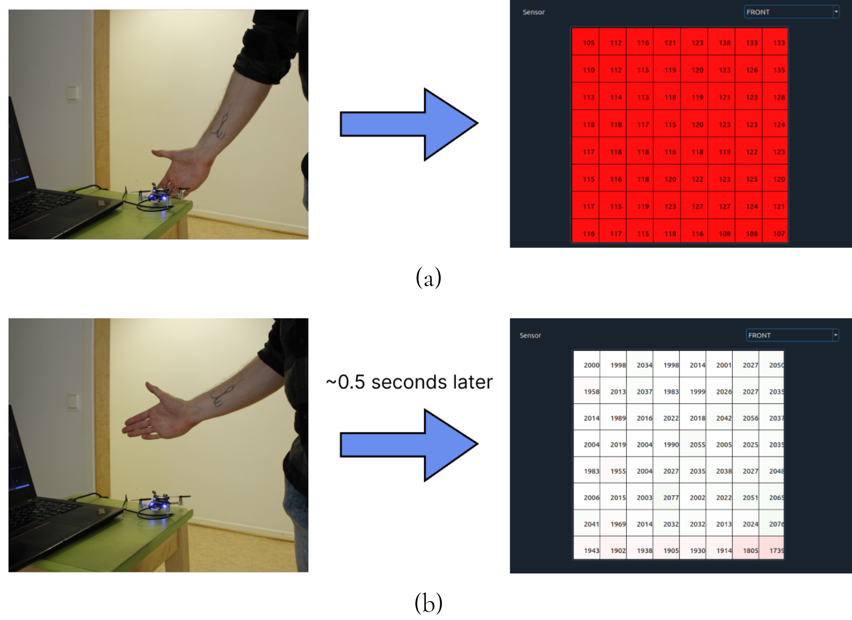


Figure 8.1: Screenshot of GUI, visualizing the distance data that is read from the prototype deck, which is mounted on the Crazyflie.

2D Matrix

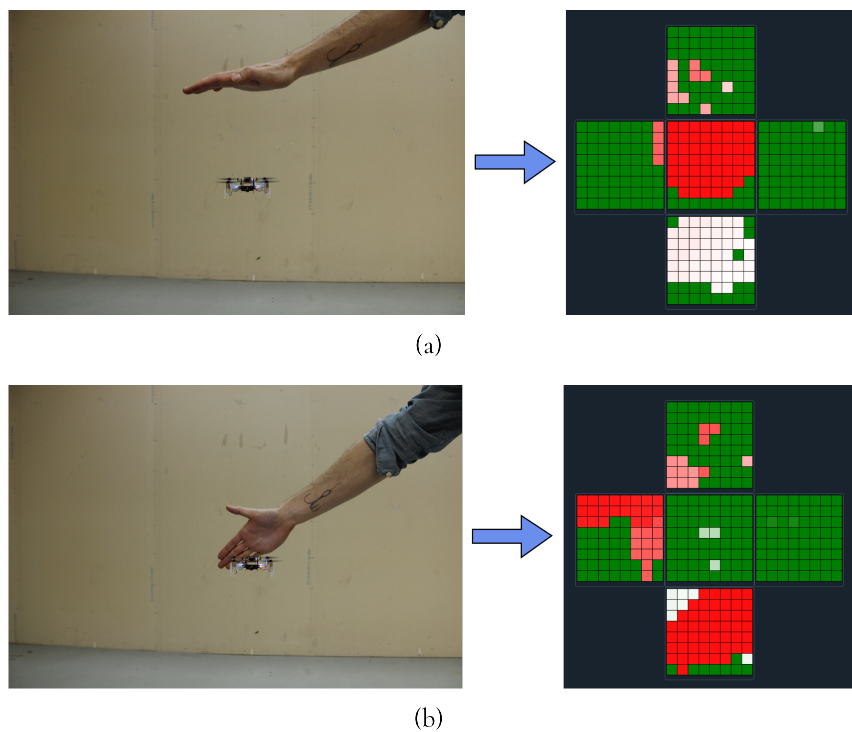
Figure 8.2 shows the 2D matrix visualization for a single sensor, where the delay of the distance data that was noticed is highlighted. In figure 8.3, the 2D matrix visualization with all five sensors is illustrated.



(a)

(b)

Figure 8.2: 2D matrix visualization with the sensor facing front. The values in the grids are the distances in millimeters. (a) A hand is placed close to the front sensor. (b) The hand is moved away from the sensors FoV and the values takes some time to update to match the distances to the wall.



(a)

(b)

Figure 8.3: 2D Matrix visualization result for all five sensors. Red grids indicates shorter distances. (a) A hand is put on top of the deck. (b) A hand is put to the back of the deck.

2D Point Cloud

Figure 8.4 illustrates the 2D point cloud visualization.

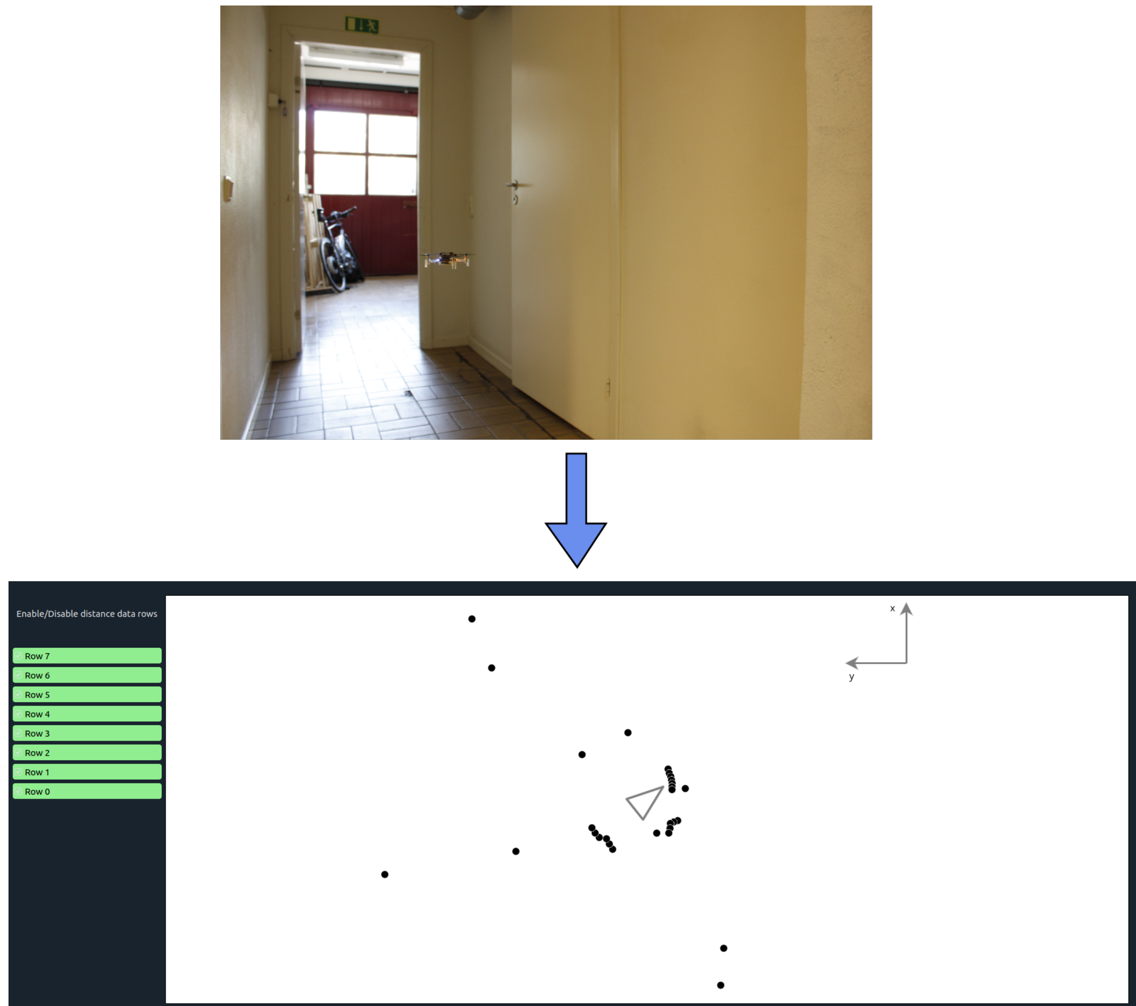
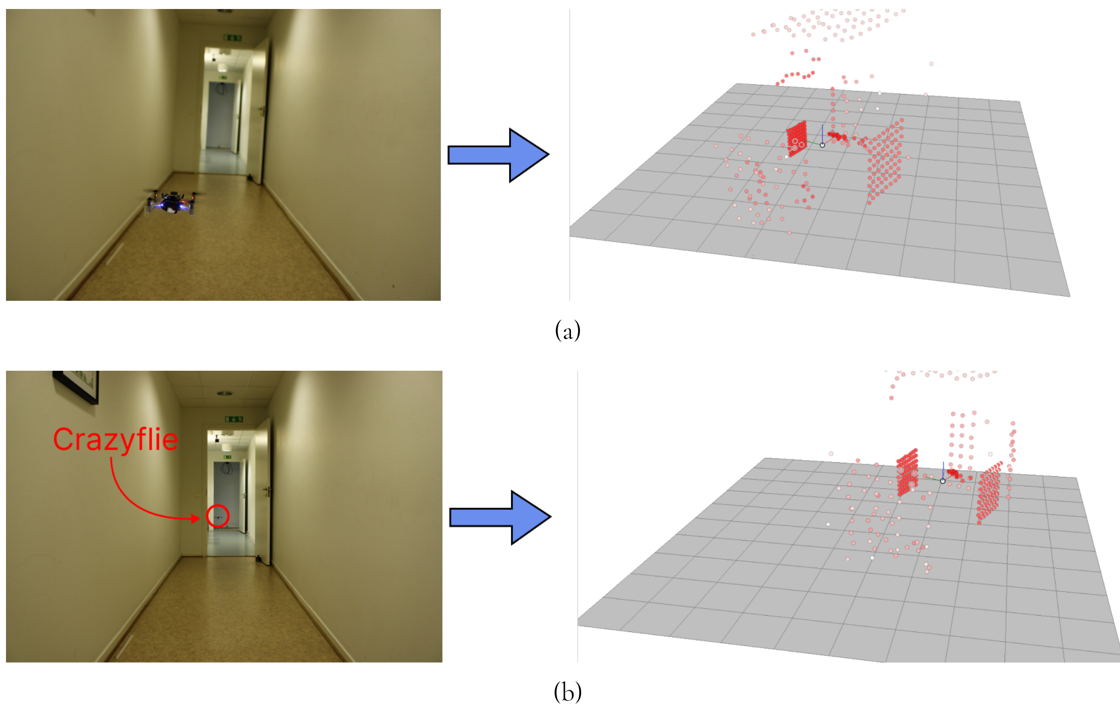


Figure 8.4: 2D point cloud visualization result. With the menu to the left it is possible to select which of the rows of distance data that should be displayed, as explained in section 7.3.1.

3D Point Cloud

Figure 8.5 illustrates the 3D visualization of the GUI when the Crazyflie is flying along a corridor.



(a)

(b)

Figure 8.5: 3D point cloud visualization result. (a) The Crazyflie is flying away from the camera. (b) The Crazyflie has flown away from the camera, and entered another room.

Even though lack of data filtering and the constrained distance detection range of the sensors, it was still possible to see walls and shapes around the Crazyflie. This is illustrated in figure 8.6, where the Crazyflie is facing a corner and detects a bulge on a wall.

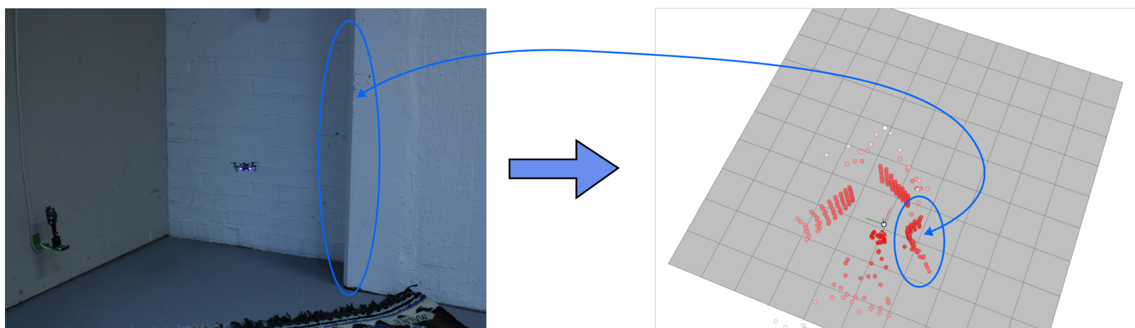


Figure 8.6: 3D point cloud visualization when the Crazyflie is facing a corner. Notice that the bulge on the wall to the right of the Crazyflie can be seen in the 3D visualization. However, the corner to the left of the drone is *not* seen in the visualization, due to the sensors having a FoV of 45°, and thus missing this area.

8.2.7 Price

Table 8.1 shows the approximate price of all components on the prototype. The total price was approximately $100, which includes shipping.

Table 8.1: Approximate price of all components on the prototype deck. Note that only the ESP32-S3 and the VL53L5CX are listed separately, which is because the other components are typically cheap, and thus grouped together. The price of shipping varies depending on the shipping time.

Component	Price (USD)
ESP32-S3	6
VL53L5CX × 5 Up	10 × 5
Other components	10
PCB	5
Shipping	25-40
Total	96-111

8.3 Answers to Thesis Questions

Now that the results have been presented, the thesis questions can be answered:

1. **What components are necessary on the deck?**

   The main components were presented and discussed in section 5, and a complete list of the required components can be found in table 6.5 and 6.6.

2. **What communication protocol should be used between the Crazyflie and the deck?**

   UART was used as the communication protocol between the Crazyflie and the deck, which was sufficient.

   While connections for SPI, I^2C and UART were made between the Crazyflie and the ESP32-S3, only UART was used to communicate between the MCUs. The reason for this was that Bitcraze preferred to use UART between the Crazyflie and decks, and they also had existing code for the transport layer in CPX for UART, which was convenient.

3. **Can the data from the SSLs be gathered in real-time and sent to a ground control station?**

   Yes; this is possible, and has been proven in this project.

   However, it is worth mentioning that if more of the Ranging result would be desired from the sensors, there could be data rate limitations on the I^2C bus between the ESP32-S3 and the sensors. The reason for this, is that if all Ranging result in table 5.1 are desired, the low-level C driver that communicates with the VL53L5CX sensor

would need to read over 3000 bytes of data, during each sample, for each sensor. At a Ranging frequency of 15 hz this equals 360 kbit/s, which for 5 sensors equals 1.8 Mbit/s. Unfortunately, the VL53L5CX supports a maximum I^2C data rate of 1 Mbit/s, so it would be necessary to split the I^2C bus to two different buses, allowing for theoretical data rate of 2 Mbit/s. Reading all available Ranging result from the sensors could also cause a problem for the UART between the ESP32-S3 and the STM32, which is typically limited to ~1 Mbit/s. In this case, it might be necessary to use the SPI between the ESP32-S3 and the STM32 instead of the UART.

4. **Can the data from the SSLs be sent through the existing radio link between the Crazyflie and the ground control station, or is it necessary to use another data link, such as WiFi?**

   Yes; it is possible to send the data through the existing nRF radio link and/or via WiFi.

   Though it is worth mentioning that the measurements of the nRF data rate were performed when the Crazyflie was not flying, and thus, the data rate could potentially decrease, as more data is typically sent when the drone is flying. However, since CRTP allows for approximately ~30 kB/s (see section 4.4), the bandwidth of the nRF radio link should not be a bottleneck for sending the distance data from the ESP32-S3 to a GCS. Though, if more of the Ranging result are desired, like explained in the previous question, the nRF might not be sufficient anymore and thus it might be necessary to send the data through WiFi.

8.4 Discussion

In this section the results are discussed, and several suggestions for improvements of the prototype are made.

Initialization time

The initialization process of the VL53L5CX required about 84 kB of firmware to be sent to the sensor, so it seems reasonable that this would take some time. However, 1.4 seconds seems very long and thus maybe this could be improved. The most likely cause of the long initialization time is that there are some unnecessary delays in the VL53L5CX driver, or the low-level C driver (see section 7.2.1), but future investigation would need to confirm this. Note that initialization time could not be improved my splitting up the initialization process in different RTOS tasks, because they would all require the same I^2C bus.

However, the Crazyflie usually takes some time to initialize and start, and is probably not flying within 10 seconds after power-up. Thus, having an initialization time of ~10 seconds might not be a *big* issue, but is definitely worth investigating.

Heat Dissipation

The deck suffered from high temperatures when the sensors were ranging at high frequencies, which could be caused by poor heat dissipation on the PCB, and could most certainly

be improved. For instance, the size of the ground pad of the VL53L5CX sensor could be increased, and vias could be added to it, which could help dissipate the heat faster. Also, using thicker traces for power and ground could potentially help. An obvious solution might be to put more space between the components, but since the size of the deck is very limited, this is most likely not an option.

However, since the temperatures were only tested when the prototype was standing still, it is worth noting that when the Crazyflie is flying, the wind from the propellers are going to cool down the entire PCB, and thus, the poor heat dissipation will probably not cause any issues when the drone is flying. Though, this would need to be tested in order to confirm it.

Distance Data

The distance data seemed to be less reliable after approximately 2 meters, and while more testing should be done in order to confirm this, similar results have been found by a research group at ETH Zurich [24]. This puts some limitations of what the deck is capable of, since distances to objects that are further than 2 meters away might not be reliable, and objects that are more than 4 meters away from the sensor will not be detected at all.

While the implemented distance data filter improved the interpretation of the data in some ways, for instance by setting unknown distances to their maximum (4000mm), it was far from perfect. Unfortunately, there was no time for further improvements of this filter, but what can be learnt from this is that a proper filter is certainly required for most applications in order to get more reliable distance data. The Target status field of the Ranging result, which gives a measurement of the validity of the distance data, can be used to create this filter.

Since the sensors on the deck was mounted in orientations: front, right, back, left and up, and covered a FoV of 45° each, they did not cover a full 360° FoV. What impact this has, will depend on what the deck should be used for. The empty gaps between the sensors that are not covered could probably be estimated, and if the drone were to rotate slightly, it might make up for the empty gaps to cover a full 360° FoV. Thus it might not be necessary to cover 360° entirely with sensors, which with 45° FoV would require a total of 8 sensors, which would increase the cost, complexity and size of the deck.

Power

The ESP32-S3 only experienced power issues when the battery was not fully charged, and thus it was most likely caused by high voltage drops of the battery, which can occur when the motors of the Crazyflie starts. To prevent this, the motors could be started more gradually, and stronger capacitors could be placed close to the power pins of the ESP32-S3, which could increase the stability of its supply voltage.

However, the power issues could also simply have been caused by poor solder connections for some of the components and further investigation would be necessary to confirm the cause of the power issues.

Data Rates

As mentioned in the section 8.3, all data links were sufficient for the ESP32-S3 to collect and send the distance data to the GCS. However, this assumes that the required data rate

is not higher than 9.6 kB/s, which is the maximum data rate necessary to send the distance data, with sensor settings: Targets per zone of one, resolution of 8x8, and a ranging frequency of 15 hz. If the resolution or ranging frequency changes, it does not increase the necessary data rate, and even if the Targets per zone increase, for instance to two (which doubles the required data rate), measuring more than one target per zone is probably not necessary for most applications. Thus, this combination of settings seemed sufficient to determine the minimum required data rate for most use-cases.

Since even the nRF radio link was sufficient to send the distance data, it might have been possible to skip the on-board microcontroller ESP32-S3, and instead make the STM32 on the Crazyflie directly collect the data from the sensors. This is how it is done on the Multi-ranger deck (see section 4.6.1), and it has the benefit of reducing the cost, complexity and size of the deck. Also, as mentioned in section 8.2.5, the data that arrived to the GCS was sometimes slightly delayed, which could have been caused by the fact that there were many different steps that the data had to pass before it reached the GCS. Between each step, there might be some delays, and each step might also use internal buffers that could cause additional delays, since the buffers must first be filled before the data can be sent. Collecting the data directly from the STM32, might decrease the delays, as the data would have to travel fewer steps.

However, if more of the Ranging result from the sensors are desired, an on-board microcontroller might still be necessary, since more Ranging result might put too much stress on the I^2C bus and/or on the nRF radio on the Crazyflie. Though, it is worth mentioning that the data is probably highly compressible, meaning it should be possible to compress the data, without loosing much information. For instance, the distance data is returned from the sensors in millimeters, as two bytes, which could be represented as a single byte instead. Furthermore, the distances does probably not change much between frames, and thus, only the differences between frames could be sent instead of the pure distance data. While these are just some examples, further investigation of the distance data would be required to find a suiting compression algorithm.

GUI & Data Visualization

The GUI made it convenient to control the deck and visualize the data, and even though the data visualization methods barely scratched the surface of what is possible, it demonstrates some of the things that the deck is capable of. Since it was developed with PyQt, it should also be easy to embed it into the Cfclient, which could be useful if the deck becomes a final product in the future. Furthermore, some of the figures of the data visualization deserves some discussion:

Figure 8.3 (2D matrix) - Some of grids in the matrices are red, even though no hand was placed there. The distances detected to the left of the drone was probably caused by the person standing there holding out the hand, and the distances detected by the front sensor was probably caused by the camera that took the photo.

Figures 8.4-8.6 (2D and 3D point clouds) - The impact of the primitive filter is visible, which sets the maximum distance to any zones whose target were not detected at all (see section 7.2.1). The effect of this can be observed by several points that are found at random positions, which probably corresponds to distances that are in reality further than four meters away, and thus, the target of that particular zone is not detected.

While not visible in the images, the lack of filtering was also noticeable by the fact that the points in the point clouds could look like they were vibrating, even though the sensor was completely still. This was probably caused by noise. In order to improve the data visualization methods, several adjustments could be done, such as:

- Experiment with different color maps that represents the distance data, in order to find one that represents distances between 0-4000 millimeters the best.
- Display more than one sample of data at the time, and perhaps decrease the opacity of the older samples.
- Only display data that is within the detection range.
- Further improvements of the GUI could include enabling data storing, which would make it possible to save the distance data from the sensors, as well as the position and attitude of the Crazyflie to disk, and then process it later, perhaps by some well-developed SLAM algorithm.

Price

As shown in section 8.2.7, the price of the components for the deck was approximately $100 in total. This shows that it is not necessary to spend hundreds, or even thousands of dollars to build circuit boards with multiple solid-state lidars, that covers a wide FoV, are small and lightweight, and can be mounted on a tiny drone such as the Crazyflie. However, in order for a company to manufacture and sell a product like this, they most certainly need to increase the price to create a profit.

Now that all four phases has been completed, the thesis questions has been answered, and a discussion of the results has been presented, it is time to give a final conclusion of the thesis.

9 Conclusion

This thesis has presented the development process of building a prototype circuit board, that has five VL53L5CX simultaneous multizone solid-state lidar sensors, an on-board ESP32-S3 microcontroller, and can be mounted on the Crazyflie. Both the VL53L5CX and the ESP32-S3 uses state of the art technologies and were new on the market at the time of the project. This project has showed that it was possible to collect the distance data from the sensors, and send it to a ground control station, which would then visualize the data.

This project has also showed that it is not necessary to use mechanically rotating lidars to measure distances in three dimensions around a small vehicle. This can be useful for small UAVs and robots that does not have enough space or lift capability to have bigger sensors. Thus, while the circuit board built during this project was only a prototype, it can serve as a proof of concept, which can lead to future products and new discoveries. Having small circuit boards with multiple multizone SSLs that together covers a wide FoV, can enable researchers and developers to better experiment with algorithms that regards SLAM, object detection and/or object avoidance on small vehicles and robots, and thus becoming a step closer to making fully autonomous vehicles.

Also, what might be particularly interesting is that this project has showed that it is possible to build circuit boards with state of the art components, that can be used for autonomous vehicles, at a very low price. The availability of cheap high-quality components and the ease of designing and ordering custom-made PCBs, without having to spend thousands of dollars makes it easier for smaller companies, technology enthusiasts and makers to contribute to new technology breakthroughs. It can also help schools and universities to educate next generations, which in turn can lead to future technology improvements and a more advanced society.

Thus, since the developed prototype worked, and the thesis questions have been answered, this bachelor's thesis can be considered successful.

9.1 Ethical Reflection

Before a further discussion of future work is presented, it is of importance to reflect on the ethics of this thesis. Thus, a short reflection is now given on some of the social benefits and ethical dilemmas that is of relevance to this thesis.

As the circuit board built during this thesis can enable better research and development opportunities that regards SLAM and object detection/object avoidance algorithms, it could help humanity becoming a step closer to making fully autonomous vehicles and UAVs. While this can help society in multiple aspects, such as faster delivery of medical equipment through drones, and less traffic accidents with self-driving cars, having fully autonomous vehicles in society certainly provides ethical dilemmas in many situations. These dilemmas can derive from questions such as "Who is responsible for the death caused by an autonomous vehicle?" and "Should the vehicle hit an older person instead of a young child, if it must hit either one of them?". Furthermore, advancements in fields such as autonomous vehicles might be one of the biggest challenges in order to achieve AGI, which provides similar benefits and dilemmas as autonomous vehicles, but at much greater scale. While a reflection and discussion of the potential benefits and downsides with AGI deserves complete books by themselves, what can be certain is that it will lead to questions of highest importance, and will probably have a significant impact on the future of humanity.

However, as advancements in fields of autonomous vehicles and AGI can provide multiple ethical dilemmas, one might argue that it is in the nature of mankind to explore and advance these technologies. This argument may continue by pointing out that intently stopping technological advancements has probably never been done in the history of mankind, even if the advancements might lead to complications and problems in the future.

Whether one thinks that investing in technology like the one used in this thesis is right or wrong, what most people probably can agree on is that making technological advancements should not be done without ethical thinking. Thus, the future will presumably hold many important discussions and regulations that regards fields of autonomous vehicles and AI.

10

Future Work

While the prototype built during this thesis could be considered successful, it was only a prototype. Thus, the firmware and the software written for it was not complete, and has potential for several improvements. These improvements, as well as other use-cases which were not explored during this thesis work are discussed in this chapter.

Improve the Prototype

In order to improve the prototype, the following could be done:

Heat dissipation - Increase the size of the ground pad of the VL53L5CX Footprint, add vias to the pad and use thicker traces for power and ground.

Data filtering - Use the Target status of the Ranging result to help filter the distance data. Perform more ranging tests and experiments, to find an appropriate filter, that is suited for the desired application.

Voltage stability - Investigate the cause of the power issues and potentially place stronger capacitors close to the power pins of the ESP32-S3, and/or start the motors of the Crazyflie more gradually.

Exclude the USB connector - While having the USB connector on the deck was convenient, if the prototype becomes a final product, the connector could be removed, which would save some space.

(Maybe) exclude the on-board microcontroller - As mentioned in section 8.4, it might not be necessary to have an on-board microcontroller on the deck. To test if this would work, one could connect the I^2C bus on the Crazyflie to the I^2C bus of the sensors on the deck, and then test if the STM32 can sample the data from the sensors. If this works, then making a deck without the on-board ESP32-S3 might be better.

GUI & data visualization - The GUI and the data visualization methods proposed in this project could generally be improved, which could make it easier to interpret the data.

SLAM

As the name of the prototype suggests (Slamdeck), the deck could be useful for SLAM applications. For instance, if the deck is mounted on the Crazyflie, it can help the drone create a 3D map of its surrounding environment. If the Crazyflie also has another deck mounted underneath it, which estimates its position, then the drone could potentially be fully autonomous. This can be helpful for researchers, as there is no deck available on market as of today, that features multiple simultaneous multizone SSL sensors, and can be mounted on the Crazyflie. However, since the particular sensor that was used in this project could only measure distances up to 4 meters, it might be most suited for indoor SLAM applications, with more narrow environments.

AI Deck

The prototype can be used in combination with the AI deck (see section 4.6.3), which could perform on-board AI calculations and SLAM or object detection/object avoidance algorithms. The AI deck also has an on-board camera, which in combination with the Slamdeck, could allow the Crazyflie to stream both image data and distance data to a GCS. This could enable interesting use-cases, but would require experiments to test its functionality. The reason for this is that with the AI deck and the Slamdeck mounted on the Crazyflie, there would be a total of *five* different microcontrollers, and several sensors on the drone. This might cause communication or power problems, but it can be worth investigating in the future, as it could enable interesting use-cases.

Other Robots & UAVs

While the prototype was built to be mounted on the Crazyflie, it should be simple to adjust to fit other robots and UAVs as well. As long as the robots do not depend on measuring distances longer than 4 meters away, it could for instance be used for indoor vacuum cleaners, or pick-and-place robots in warehouses.

References

[1] Bitcraze, "System Overview." bitcraze.io. https://www.bitcraze.io/documentation/system/ (accessed May 2, 2022).

[2] Bitcraze, "Crazyflie 2.1." bitcraze.io. https://www.bitcraze.io/products/crazyflie-2-1/ (accessed May 2, 2022).

[3] Bitcraze, "The Coordinate System of the Crazyflie 2.x." bitcraze.io. https://www.bitcraze.io/documentation/system/platform/cf2-coordinate-system/ (accessed May 2, 2022).

[4] Bitcraze, "Crazyradio PA." bitcraze.io. https://www.bitcraze.io/products/crazyradio-pa/ (accessed May 2, 2022).

[5] Bitcraze, "Client Software Overview." bitcraze.io. https://www.bitcraze.io/documentation/system/client-and-library/ (accessed May 2, 2022).

[6] Bitcraze, "Multi-ranger deck." bitcraze.io. https://www.bitcraze.io/products/multi-ranger-deck/ (accessed May 2, 2022).

[7] Bitcraze, "Flow deck v2." bitcraze.io. https://www.bitcraze.io/products/flow-deck-v2/ (accessed May 3, 2022).

[8] Bitcraze, "Ai deck 1.1." bitcraze.io. https://www.bitcraze.io/products/ai-deck/ (accessed May 2, 2022).

[9] Bitcraze, "Expansion decks of the Crazyflie 2.x." bitcraze.io. https://www.bitcraze.io/documentation/system/platform/cf2-expansiondecks/ (accessed May 2, 2022).

[10] Bitcraze, "Cpx - Crazyflie Packet eXchange." bitcraze.io. https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/cpx/ (accessed May 2, 2022).

[11] Bitcraze, "Crazyflie 2.0 expansion board template." github.com https://github.com/bitcraze/crazyflie2-exp-template-electronics (accessed Apr. 25, 2022) commit: d407886d21958346fb94a4e9fc28c5b41eb0f408.

[12] M. V. Rajasekhar and A. K. Jaswal, "Autonomous vehicles: The future of automobiles," in *2015 IEEE International Transportation Electrification Conference (ITEC)*, pp. 1–6.

[13] L. Štěpánek, F. Habarta, I. Malá, and L. Marek, ““great in, great out” is the new “garbage in, garbage out”: subsampling from data with no response variable using various approaches, including unsupervised learning,” in *2021 International Conference on Computing, Computational Modelling and Applications (ICCMA)*, pp. 122–129.

[14] Y. Li and J. Ibanez-Guzman, “Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems,” vol. 37, no. 4, pp. 50–61. Conference Name: IEEE Signal Processing Magazine.

[15] D. V. Nam and K. Gon-Woo, “Solid-state LiDAR based-SLAM: A concise review and application,” in *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp. 302–305. ISSN: 2375-9356.

[16] STMicroelectronics, “VL53L5CX - Time-of-Flight 8x8 multizone ranging sensor with wide field of view.” Ver. 5, 2021.

[17] ams OSRAM, “TMF8820/21/28 Multizone Time-of-Flight Sensor.” ver. 4.00, 2021.

[18] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part i,” vol. 13, no. 2, pp. 99–110. Conference Name: IEEE Robotics Automation Magazine.

[19] Everdrone, “Our service.” everdrone.com `https://everdrone.com/our-service/` (accessed May 1, 2022).

[20] STMicroelectronics, “VL53L1X - A new generation, long distance ranging Time-of-Flight sensor based on ST’s FlightSense™ technology.” Rev. 5, 2022.

[21] Terabee, “Terabee - High Performance Sensing Solutions.” terabee.com `https://www.terabee.com/` (accessed May. 10, 2022).

[22] Terabee, “TeraRanger Evo 60m.” 2017.

[23] Terabee, “TeraRanger Hub Evo.” 2021.

[24] O. I. P. T. M. M. B. L. Niculescu Vlad, Müller Hanna, “Towards a multi-pixel time-of-flight indoor navigation system for nano-drone applications,” 2022.

[25] R. Murch, “The Software Development Lifecycle - A Complete Guide.” Richard Murch, 2012.

[26] Bitcraze, “Ai-deck.” Rev. C, 2020.

[27] Bitcraze, “Crazyflie 2.1.” Rev. B1, 2021.

[28] Bitcraze, “Crazyradio PA 2.4 GHz USB dongle.” Rev. C, 2020.

[29] Bitcraze, “Flow deck v2.” Rev. C, 2020.

[30] Bitcraze, “Flow deck v2.” Rev. C, 2020.

[31] Bitcraze, “Multi-ranger deck.” Rev. E, 2020.

[32] E. Systems, “ESP32-S3, ESP-IDF Programming Guide v4.4.” Rev. 1, 2022.

[33] E. Systems, “ESP32-S3-MINI-1.” Pre-release v0.6, 2021.

[34] A. W. Services, “The FreeRTOS™ Reference Manual, v10.0.0.” 2018.

[35] T. Instruments, “TPS62A01/TPS62A01A, 1-A High-Efficiency Synchronous Buck Converter in a SOT-5X3 Package.” Rev. 1, 2022.

[36] N. Semiconductor, “nRF24L01.” Rev. 2.0, 2007.

[37] N. Semiconductor, “nRF51822.” Rev. 3.1, 2014.

[38] STMicroelectronics, “A guide to using the VL53L5CX multizone Time-of-Flight ranging sensor with wide field of view Ultra Lite Driver (ULD).” Ver. 2, 2021.

[39] F. Martin, P. Mellot, A. Caley, B. Rae, C. Campbell, D. Hall, and S. Pellegrini, “An all-in-one 64-zone SPAD-based direct-time-of-flight ranging sensor with embedded illumination,” in *2021 IEEE Sensors*, pp. 1–4. ISSN: 2168-9229.

[40] K. Li, C. Chase, Y. Rao, and C. J. Chang-Hasnain, “Widely tunable 1060-nm high-contrast grating VCSEL,” in *2016 Compound Semiconductor Week (CSW) [Includes 28th International Conference on Indium Phosphide Related Materials (IPRM) 43rd International Symposium on Compound Semiconductors (ISCS)*, pp. 1–2.

[41] I. I. b. Jamaludin and H. b. Hassan, “Design and analysis of serial peripheral interface for automotive controller,” in *2020 IEEE Student Conference on Research and Development (SCOReD)*, pp. 498–501. ISSN: 2643-2447.

[42] S. Panich, “A mobile robot with a inter-integrated circuit system,” in *Robotics and Vision 2008 10th International Conference on Control, Automation*, pp. 2010–2014.

[43] A. K. Gupta, A. Raman, N. Kumar, and R. Ranjan, “Design and implementation of high-speed universal asynchronous receiver and transmitter (UART),” in *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 295–300. ISSN: 2688-769X.

[44] Y.-y. Fang and X.-j. Chen, “Design and simulation of UART serial communication module based on VHDL,” in *2011 3rd International Workshop on Intelligent Systems and Applications*, pp. 1–4.

[45] F. Zappa, S. Tisa, A. Tosi, and S. Cova, “Principles and features of single-photon avalanche diode arrays,” vol. 140, no. 1, pp. 103–112.

[46] S. M. Paul Scherz, “Practical Electronics For Inventers.” The McGraw-Hill Companies, 2013.

[47] M. L. M. Larry A. Coldren, Scott W. Corzine, “Diode Lasers and Photonic Integrated Circuits.” John Wiley & Sons, Inc, 2012.

[48] Espressif, “About Espressif.” espressif.com. `https://www.espressif.com/en/company/about-espressif` (accessed Apr. 4, 2022).

[49] VisPy, “Home - VisPy.” vispy.org https://vispy.org/index.html (accessed Apr. 7, 2022).

[50] KiCad, “About KiCad.” kicad.org https://www.kicad.org/about/kicad/ (accessed Apr. 19, 2022).

[51] Bitcraze, “CRTP - Communication with the Crazyflie.” bitcraze.io. https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2022.01/functional-areas/crtp/ Accessed: 2022-04-21.

[52] S. M. LaValle, “Yaw, pitch, and roll rotations.” planning.cs.uiuc.edu http://planning.cs.uiuc.edu/node102.html (accessed Apr. 27, 2022).

[53] E. Systems, “Espressif IoT Development Framework.” github.com https://github.com/espressif/esp-idf/blob/1329b19fe494500aeb79d19b27cfd99b40c37aec/docs/en/get-started/vscode-setup.rst (accessed Apr. 25, 2022) commit: ce916db7ed4ada6cb814dde8f7ff3a554fce3094.

[54] openscopeproject, “Interactive HTML BOM plugin for KiCad.” github.com https://github.com/openscopeproject/InteractiveHtmlBom (accessed Apr. 25, 2022) commit: f26a8285dc6e91e98ec12bc5eabb36363cd96417.

Appendices

A

VL53L5CX Settings

This appendix describes the available settings of the VL53L5CX sensor, which is highly configurable with the help of the low-level C driver. Some of these settings can be set during run-time, through I^2C, while others must be set on compile-time. All settings that are described below is based on datasheet of the VL53L5CX sensor [16], and the user manual for the low-level C driver [38], both provided by ST.

A.1 Run-Time Configurable Settings

The settings below are possible to change at run-time, through I^2C.

Resolution - The resolution is the number of zones that is outputted by the sensor. This can be 4x4 or 8x8.

Ranging frequency - How often a measurement is taken. The maximum ranging frequency depends on the resolution, and the minimum and maximum values are summarized in table A.1

Table A.1: Maximum and minimum ranging frequencies, at different resolutions.

Resolution	Minimum ranging frequency (Hz)	Maximum ranging frequency (Hz)
4x4	1	60
8x8	1	15

Ranging mode - The sensor supports two different ranging modes:

- **Continuous** (high performance) - The VCSEL is enabled all the time, which improves ranging distance, and ambient immunity.

- **Autonomous** (low power) - The VCSEL is enabled only during a period of time that is defined by the user. This time period is called the *integration time*. Since the VCSEL is not always enabled, the power consumption is reduced.

Integration time - Used to determine the period of time that the VCSEL is used when the device is in Autonomous mode. Resolution 4x4 is composed of a single integration time, and the resolution 8x8 is composed of four integration times. There is an overhead of around 1 ms between each measurement, and the sum of all integration times (+ 1 ms) must be lower than the measurement period. Otherwise the ranging frequency will be changed so that the period matches it. This is visualized in figure A.1 and A.2.

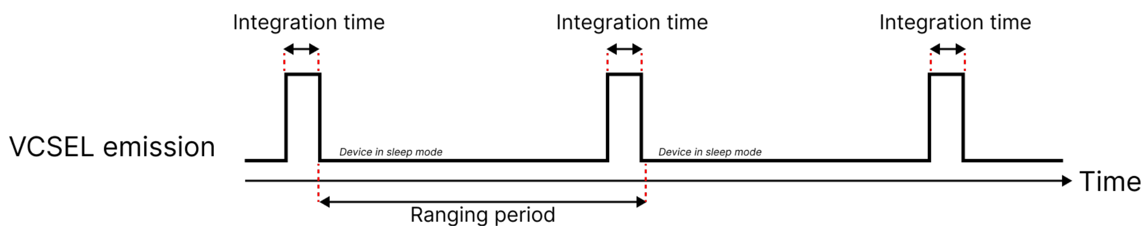


Figure A.1: VCSEL emission timeline. Resolution is 4x4, and ranging mode is Autonomous.

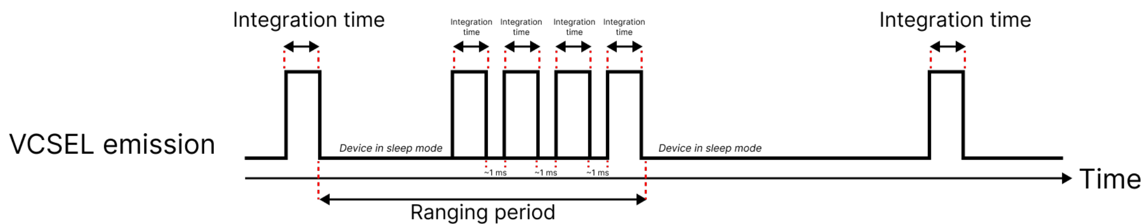


Figure A.2: VCSEL emission timeline. Resolution is 8x8, and ranging mode is Autonomous.

Power modes - The VL53L5CX supports two different power modes:

- **Wake-up** (high power) - The device is waiting for instructions.
- **Sleep** (low power) - The device cannot be used until it is set in Wake-up mode.

Sharpener - The light that is reflected from a target, is not a clean pulse with sharp edges. This is because the edges may slope away, which can affect the distances reported in adjacent zones. The sharpener setting is used to remove some, or all of the signal that is caused by this. Figure A.3 shows an example of the result of different sharpener values on a given scene.

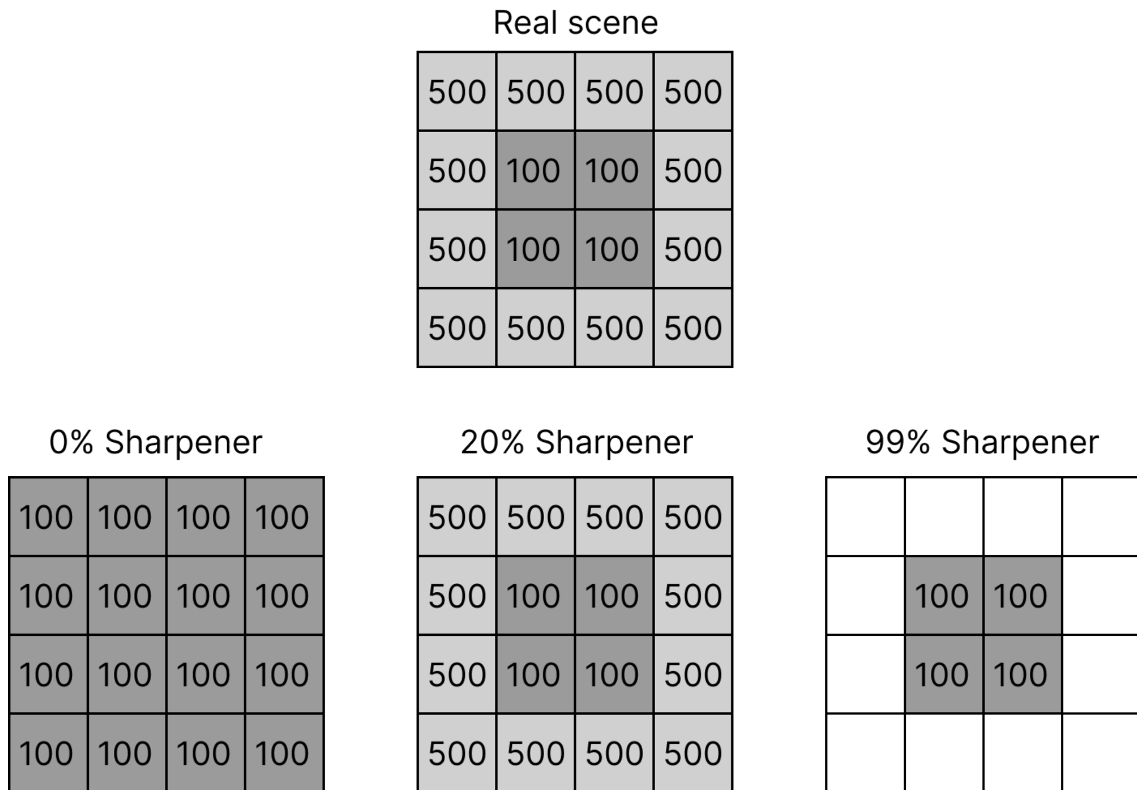


Figure A.3: Example result of different sharpener settings with a given scene. The resolution is 4x4. One target is centered in the FoV at distance 100 mm, and another target further is 500 mm behind it.

Target order - The VL53L5CX can measure several targets per zone and thanks to the histogram processing, one can choose the order of reported targets. There are two options:

- **Closest** - The closest target is reported first.
- **Strongest** - The strongest target is reported first.

A.2 Compile-Time Configurable Settings

The VL53L5CX driver also supports several settings that must be set on compile-time. Some of these settings, referred to as *plugins*, requires an additional source file to be included and compiled in the firmware.

Multiple targets per zone - The VL53L5CX can measure distances of up to four Targets per zone. The order of the detected target(s) are set with the Target order setting.

Xtalk margin (plugin) - When a coverglass is present on top of the sensor, this margin is used to change the detection threshold. The threshold can even be increased to ensure that the coverglass is never detected. The workings of Xtalk margin is illustrated in figure A.4

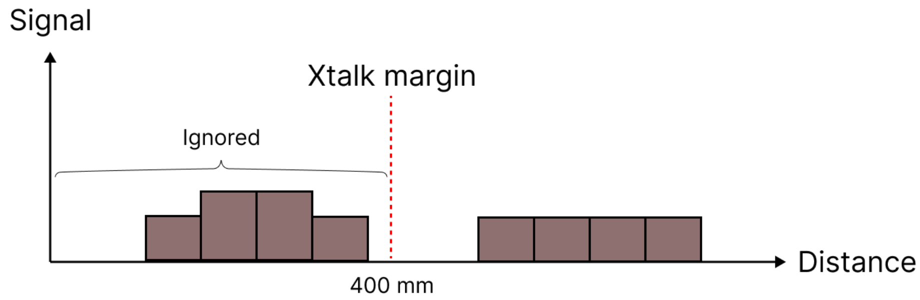


Figure A.4: Example of Xtalk margin setting. The brown blocks represents objects in front of the sensor. The margin is set to 400 mm, which means that the sensor will ignore distances that are below 400 mm.

Detection thresholds (plugin) - The sensor can be programmed to raise an interrupt on pin A3 (INT), when a specific criteria is met.

Motion indicator (plugin) - This setting can be used to track motion in front of the sensor. The sensor does this by comparing sequential frames and then computes a value of motion intensity which corresponds to "how much motion" there is in front of the sensor.

B
Firmware

This appendix aims to give a more detailed description the Slamdeck module and the Slamdeck API, which are both software modules in the firmware of the ESP32-S3.

B.1 Slamdeck

This module was given the following functionalities:

- Sensor initialization.
- Get sensor settings.
- Set sensor settings.
- Get sensor data.

Sensor Initialization

During initialization of the deck, all sensors were initialized one by one. This was done by disabling the I^2C communication on all but one sensor at the time and perform the driver initialization (write 84 kB of firmware) to each sensor. The following steps describes the entire initialization process:

1. Ensure that I^2C is initialized.
2. Assign each sensor a unique I^2C address. This address is not set yet, but simply a variable in the code.
3. Initialize the GPIO LPn pin for each sensor and pull the pin low.

The following steps were then taken for each sensor, one by one:

4. Enable I^2C communication by pulling LPn high.

5. Check if the sensor is available at its default I^2C address. This was done by sending its address, together with a write bit, and checking if an ACK bit was received back. If the sensor was *not* found on the default address, the assigned address was tried instead. If the sensor was not found on this address either, something was wrong, and the sensor was set to disabled and the next steps were not followed.

6. Initialize the sensor with the help of the VL53L5CX Driver.

7. Set the I^2C address of the sensor with the help of the VL53L5CX Driver.

8. Set default settings of the sensor with the help of the VL53L5CX Driver.

The reason for the extra complexity with address checking was necessary because once the I^2C address of the sensor has been changed, it will remain changed until either the I2C_RST (A1) pin is pulled high, or the power to the device is turned off and on again. The I2C_RST pin was not connected on the prototype due to lack of space. This meant that both of the addresses had to be checked, because if it was the first time the sensor powered on, it would have the default I^2C address 0x29, but if had already been changed, it would have its assigned address. This is because the EPS32-S3 could be technically restarted, without the sensors being restarted.

Get Sensor Settings

A data structure was made that represented the sensor settings, which could make it convenient to get and/or set the settings of the sensor. These settings were the same for all sensors, so it was not possible to change the sensor of only one sensor. This structure is explained in table B.1.

Table B.1: Data structure that represents the settings of the sensor.

Setting name	Data size (bytes)
VL53L5CX_integration_time_ms_e	4
VL53L5CX_sharpener_percent_e	1
VL53L5CX_ranging_frequency_hz_e	1
VL53L5CX_resolution_e	1
VL53L5CX_power_mode_e	1
VL53L5CX_target_order_e	1
VL53L5CX_ranging_mode_e	1
Total size	10

The *get sensor settings* function returns the data structure that represents the settings that were currently being used by all five sensors.

Set Sensor Settings

Sets the given settings for all five sensors, using the data structure described above in table B.1. If the sensors are ranging (active), they first have to be stopped, and then restarted once the settings has been changed.

Get Sensor Data

This function returns the distance data gathered from all five sensors. The distance data is represented in millimeters, and consists of two bytes. As described in appendix A.2, how many zones that are outputted from the sensors are determined by its resolution, and for each zone, one can specify how many targets should be included with the setting Targets per zone. This means that the settings might have an impact of how many bytes of data this function returns, which can span from 160 to 2560 bytes. Table B.2 shows the possible data sizes, with different settings, and table B.3 shows an example of how the data can look with a given setting.

Table B.2: Possible data lengths (in bytes) for a single data reading from the VL53L5CX sensor, with different values for the resolution, and Targets per zone.

Resolution	Targets per zone	Data length, 1 sensor	Data length 5, sensors
4x4	1	32	160
4x4	2	64	320
4x4	3	96	480
4x4	4	128	640
8x8	1	128	640
8x8	2	256	1280
8x8	3	384	1920
8x8	4	512	2560

Table B.3: Example data buffer, with resolution = 8x8, and Targets per zone = 1

Sensor	Up	Front	Right	Back	Left
Bytes	0-127	128-255	255-383	384-511	512-639

Note that while it is possible to get more Ranging result from the VL53L5CX than just distance data, this function *only* returns the distance data, and does not include any other results. However, if other results are desired in the future, it would be easy to extend this function.

B.2 Slamdeck API

The Slamdeck API works as a wrapper around the Slamdeck, which exposes its functionalities to the outside of the ESP32-S3, such as other microcontrollers or ground control stations. The API protocol works mainly as Master-Slave, but also supports *streaming*, which is a mode when the Slamdeck sends the distance data as fast as it can. The reason for this mode was to achieve as high frame rates as possible (the VL53L5CX supports up to 60 Hz ranging frequency).

A packet consists of at least 1 byte. The first byte, herein referred as the *command*, is used to indicate what action one wants from the API. This command can be any of the following:

- *GET_DATA*
- *GET_SETTINGS*
- *SET_SETTINGS*
- *START_STREAMING*
- *STOP_STREAMING*

Figure B.1 shows a simple example of a communication through the Slamdeck API.

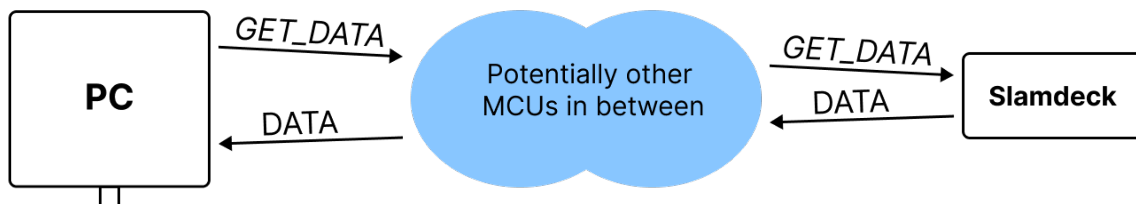


Figure B.1: Slamdeck API example. The GCS requests data through the *GET_DATA* command, and Slamdeck returns the data.

The available commands are described bellow.

GET_DATA

Returns the data, in a data format as described in appendix B.1.

GET_SETTINGS

Returns the current sensor settings, as described in appendix B.1.

SET_SETTINGS

Sets the desired sensor settings, as described in appendix B.1. Note that this packet must contain the desired setting after the command.

START_STREAMING

Make ESP32-S3 enter streaming mode. After this command is sent, the ESP32-S3 will start sending data, as if it was constantly getting *GET_DATA* commands.

STOP_STREAMING

Make ESP32-S3 stop streaming, enter idle mode, waiting for further commands.