



Master Graduation Thesis

Design of a data collector for
HEMS crew during OHCA

Yu Zhang 4363515

Table of Contents

04	1. Introduction
06	2. Concept summary
10	3. Design brief
	3.1 Overall context
	3.2 Problem definition
	3.3 Assignment
	3.4 Personal ambitions
	3.5 Method and plan
17	4. Cycle 1
	4.1 Design criteria setting
	4.2 Ideation
	4.3 Concepts
	4.4 Evaluation
	4.5 Conclusion
62	5. Cycle 2
	5.1 Electronics analysis
	5.2 Electronic prototypes building
	5.3 Data transfer system building
	5.4 Evaluation
	5.5 Conclusion
84	6. Cycle 3
	6.1 Use process analysis
	6.2 Electronics housing ideation
	6.3 Housing models building
	6.4 Evaluation
	6.5 Conclusion
102	7. Cycle 4
	7.1 Working prototype elaboration
	7.2 User tests with the working prototype
	7.3 Final concept elaboration
	7.4 Final concept evaluation
	7.5 Implementation
119	8. Reflection
121	9. References
125	10. Appendix

List of abbreviations

CA - Cardiac arrest

CPR - Cardiopulmonary resuscitation

ECPR - Extracorporeal cardiopulmonary resuscitation

ECMO - Extracorporeal membrane oxygenation

HEMS - Helicopter emergency medical service

IMU - Inertial measurement unit

OHCA - Out of hospital cardiac arrest

PEA/Asystole - 2 types of unshockable rhythms

RTC - Real-Time Clock

ROSC - Return of spontaneous circulation

VF/pVT - 2 types of shockable rhythms

01

Introduction



Introduction




The HEMS crew is planning to conduct research on implementing ECPR treatment during OHCA cases. A data collector that collects time and chest compression data is needed for the research. Thus, this graduation project is focused on the design of a data collector for HEMS crew during OHCA.

This report describes the whole process of developing the design of the data collector. The process starts with a design brief that elaborates the design assignment. Then four cycles of project development are carried out: the first cycle is focused on context analysis and exploration on possible solutions; the second cycle is focused on the electronics prototype and data transfer system building; the third cycle is focused on the housing design of the collector; the last cycle is concluded with a validated final design and recommendations on implementation.

This project collaborates with Dr. Dinis Reis Miranda from the HEMS lifeliner 2 and Erasmus MC - University Medical Center.

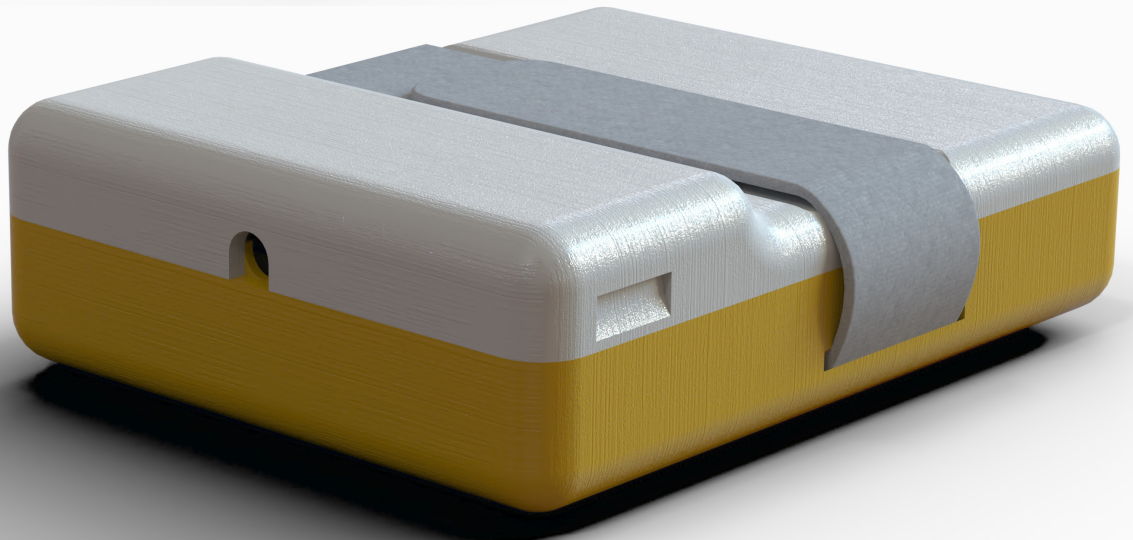
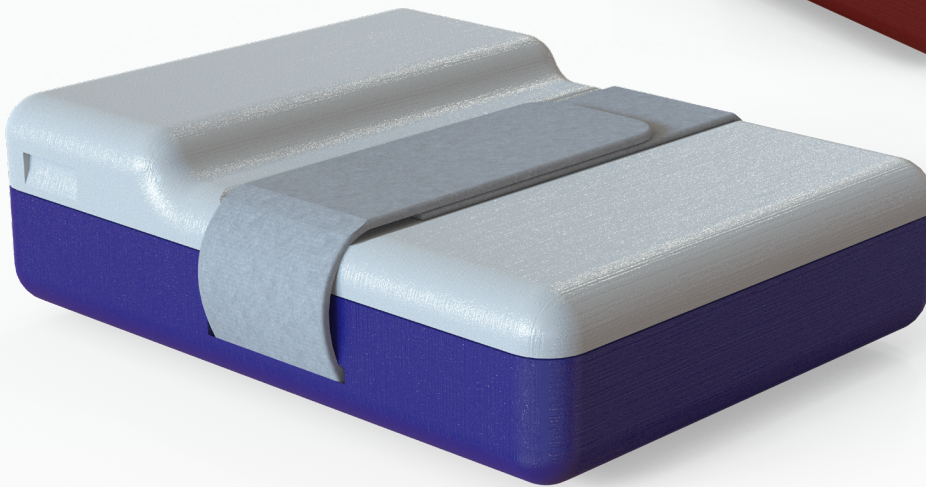
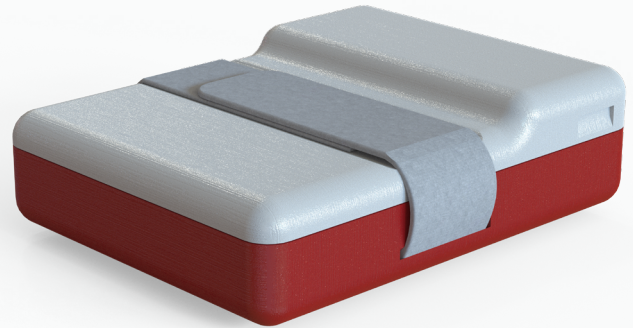
02

Concept Summary

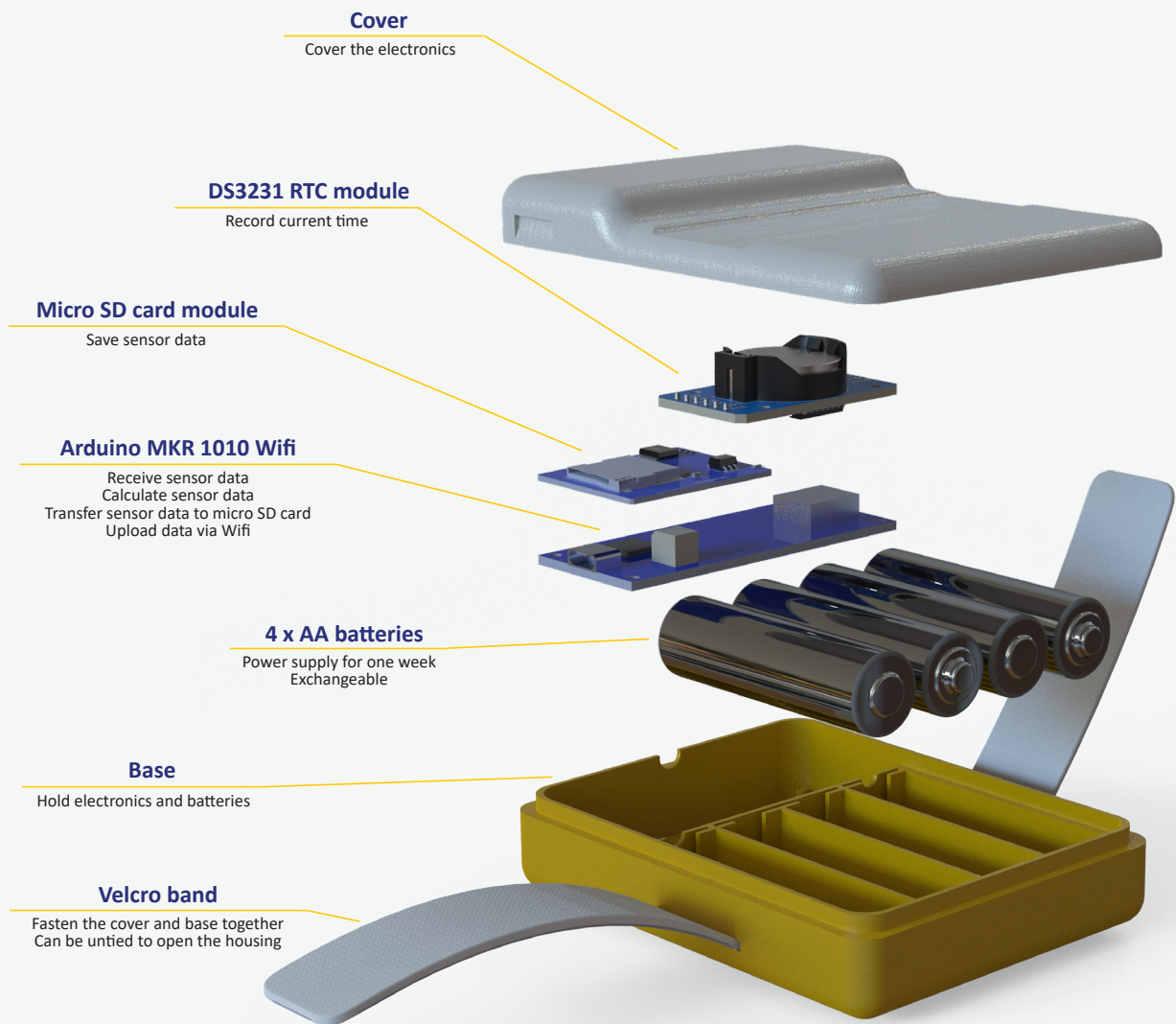
-  A91B1F - Backup
-  25237A - Automobile
-  FFC907 - Helicopter

The CPR Data Collector

The CPR data collector is designed for the HEMS team to collect the chest compression data during OHCA cases. It is connected to the CorPatch - a medical sensor brought by the HEMS team, to collect and store data, and upload data directly to Google Sheets via Wifi. For each HEMS team, three data collectors are prepared: red for backup, blue for the automobile rescue backpack, and yellow for the helicopter rescue backpack. The CPR data collector is **accurate** in data collection, **fast** to set up, **durable** in OHCA rescues.



Main Components



User scenarios



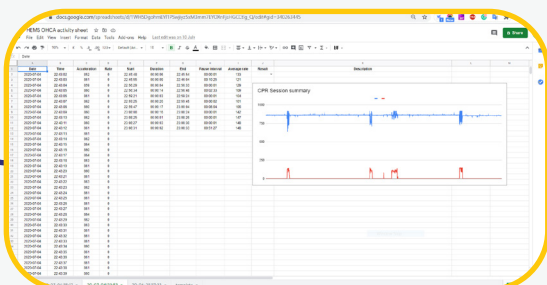
1. Take out the plug and connect with CorPatch



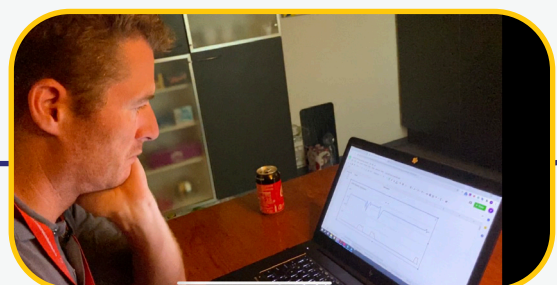
2. Place the CorPatch on Patient's center chest



3. Data collector starts collect CPR data



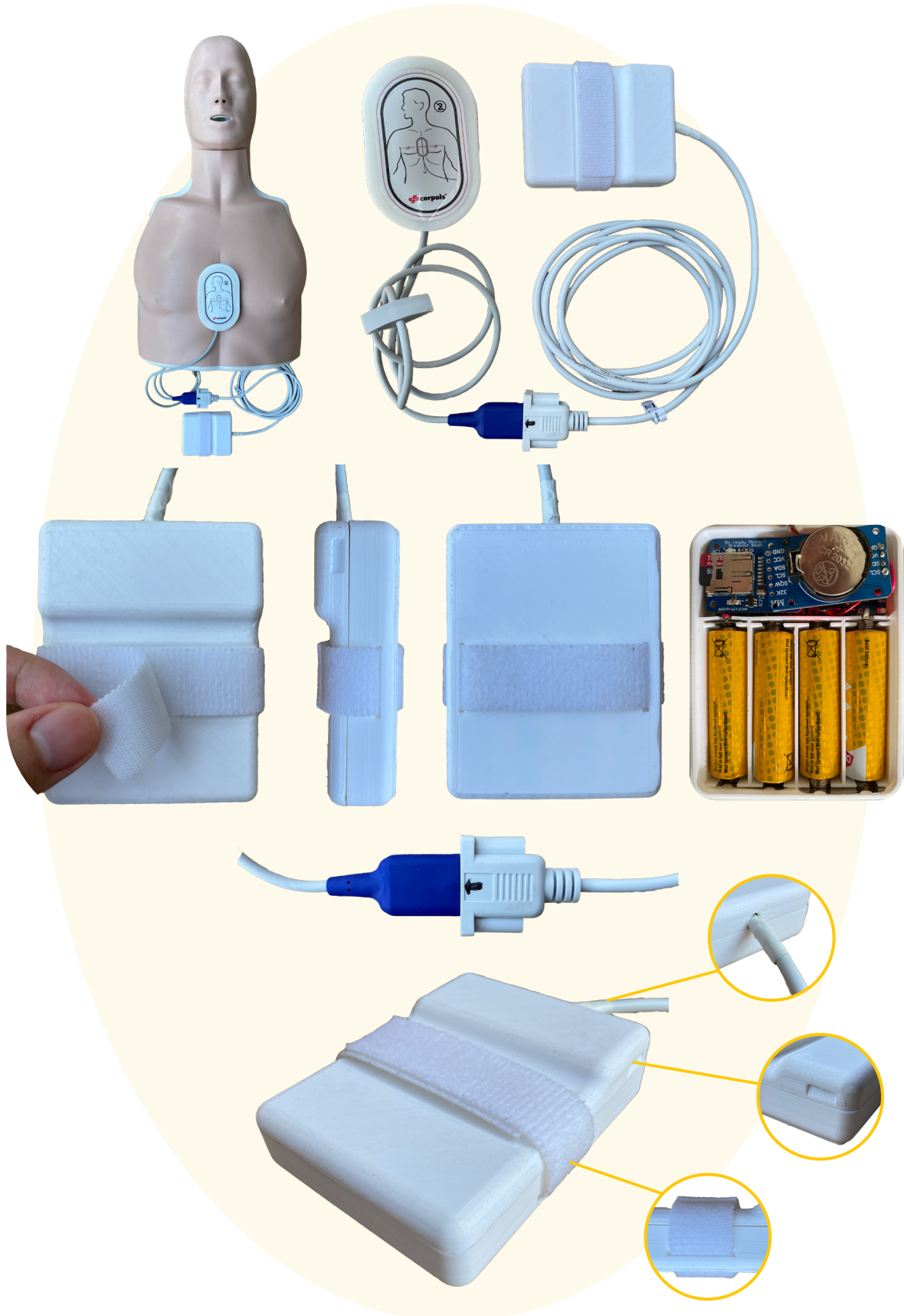
5. Data is saved as table and summarized in charts



4. Review data directly on Google Sheets

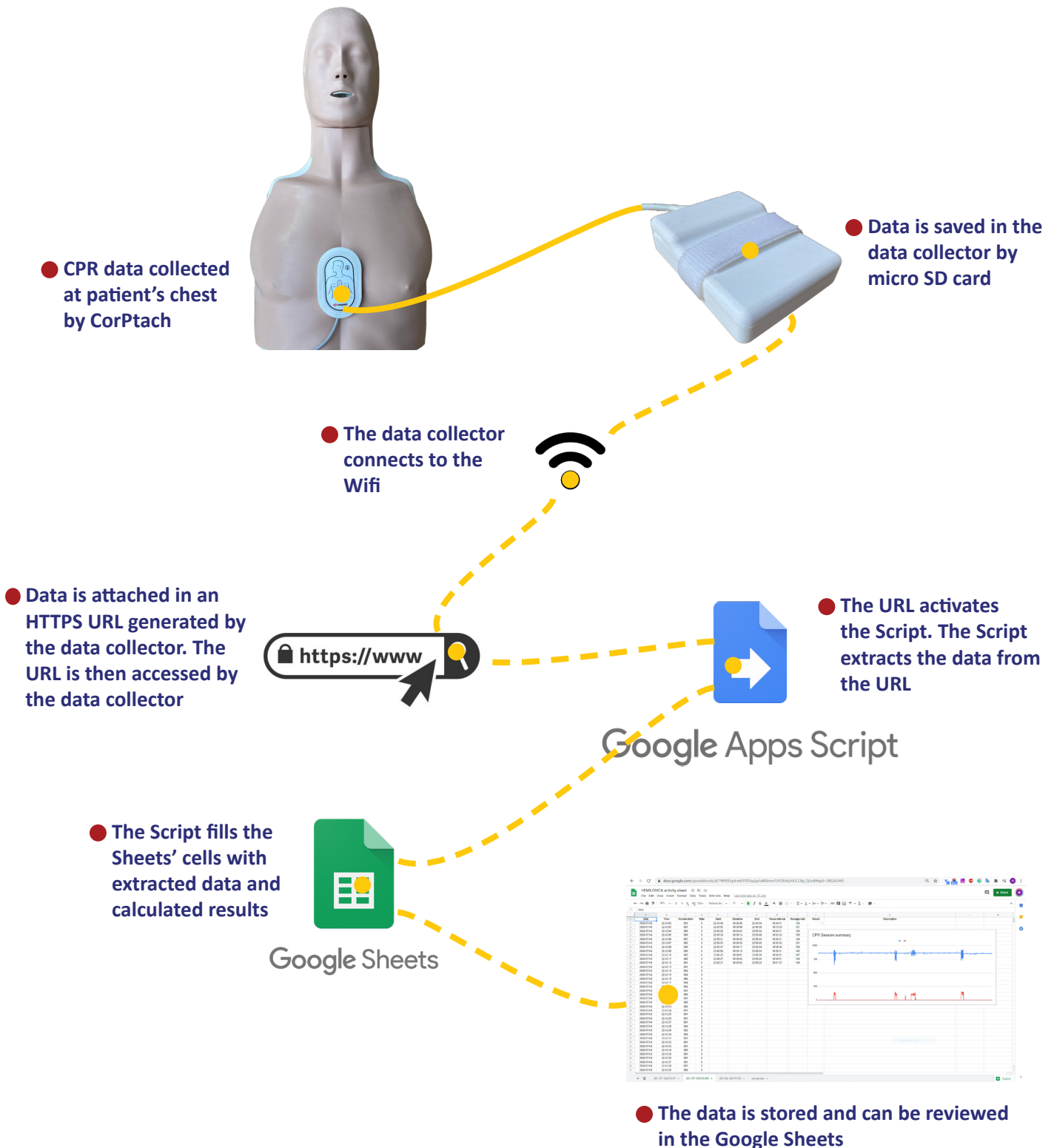
Working prototype

A working prototype is built to validate the design. It is tested with a HEMS team member on a manikin. It requires a battery exchange once a week. The whole electronics can be disassembled from the housing for maintenance and program updates.



Data transfer system

A data transfer system is built to upload the data from the collector directly to the Google Sheets. It uploads the data from the collector via HTTPS calls to the Google Script. The Google Script then transfers the data to the Google Sheets.



03

Design brief



3.1 Overall context

3.1.1 Target group: HEMS crew

The helicopter emergency medical services (HEMS) involves teams of highly specialized personnel, which are deployed very quickly. The HEMS are active over a wide area and are still able to deploy teams to the scene within 11 min of the initial emergency call. One HEMS crew is composed by one pilot, one doctor, and one HEMS crew member, as shown in figure 3.1.

3.1.2 OHCA cases and ECPR

Neurological recovery after out-of-hospital cardiac arrest (OHCA) is mainly dependent on the duration of the arrest. Some hospitals restore circulation with extracorporeal cardiopulmonary resuscitation (ECPR) administered on a patient's arrival at the hospital during refractory OHCA. Pappalardo (2017)⁶⁰ defined that ECPR as an implantation of veno-arterial extracorporeal membrane oxygenation (VA-ECMO in figure 3.2) in a patient who experienced a sudden and unexpected pulseless condition attributable to cessation of cardiac mechanical activity. In ECPR, a miniaturized cardiopulmonary bypass system (similar to that used in open-heart surgery) replaces cardiac and pulmonary function and provides full adequate circulatory support to the body. To this end, large-bore cannulas are inserted into the inguinal artery and vein and are connected to the bypass system. Transporting patients in cardiac arrest to the hospital for applying ECPR can be very time consuming, decreasing the chance of neurological recovery.

3.1.3 HEMS ECPR

As the HEMS crew are at the scene very rapidly and consist of a team of highly trained specialists, equipping the HEMS with portable ECPR - Cardiohelp ECMO (figure 3.3) may potentially significantly reduce the time till restoration of adequate circulation in patients with persistent arrest of circulation. To study this, we plan a large nationwide study covering all OHCA patients younger than 50 years by all four HEMS stations, comparing deployment of HEMS without ECPR with deployment of HEMS with ECPR capabilities.

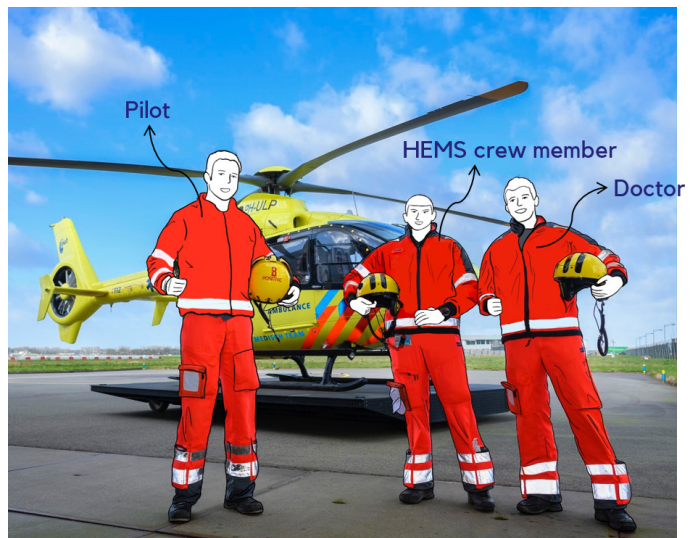


Figure 3.1 HEMS crew's composition⁸⁰

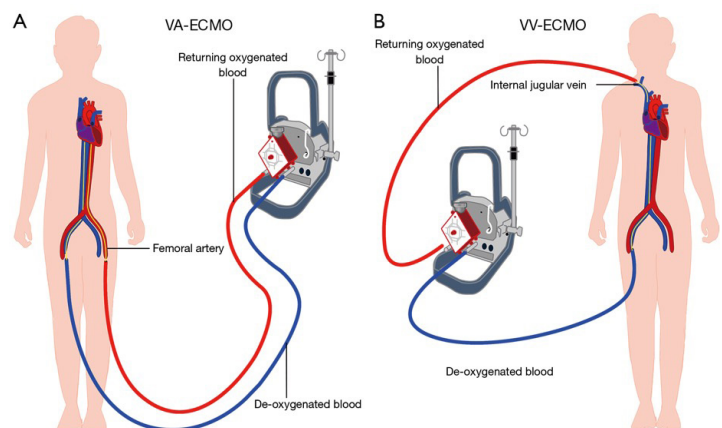


Figure 3.2 VA-ECMO and VV-ECMO⁶³



Figure 3.3 Cardiohelp ECMO³⁶

3.2 Problem definition

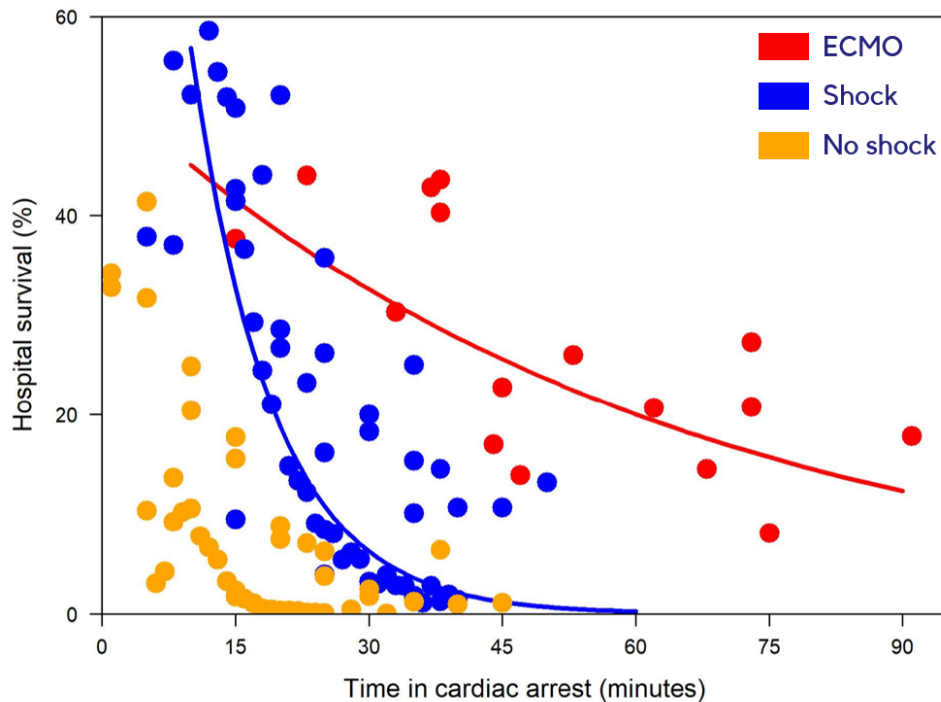


Figure 3.4 Survival rate comparing to time in cardiac arrest (Appendix B)

As described above, the neurological outcome is mainly dependent on time in cardiac arrest. As shown in figure 2, the time record on phases of OHCA and patients' situation is vital for the comparison study (ECPR vs non-ECPR). However, due to the "hectic" setting during OHCA, with a few staff, time in cardiac arrest is very difficult to measure adequately. In addition, at the end of the resuscitation, the collection of data is also difficult as the HEMS often is dispatched to another assignment. However, for the study, adequate registration of time in cardiac arrest is of paramount importance. Thus, the major problem of this project would be: How to design a data collector, that requires as little action and attention as possible (or automatic) from the HEMS crew, to record current situation and register time precisely? The challenge is not only about implementing technologies on data collectors but also about how to design the interaction between users (HEMS crew) and the device (data collector) in an emergency circumstance (OHCA). Therefore, research on the following 2 topics should be carried out:

1. What kinds of data can be retracted from HEMS crew (e.g. physical inputs) and the patient during the whole OHCA process for time registration on each OHCA phase (CPR -- shocks -- ECMO -- heartbeats/ECMO runs/death)? For this research, timelines and scenarios of each participant during OHCA should be drawn out and analyzed. Plus, an observational study on the actual OHCA case might be conducted if possible.

2. What kind of interaction should be designed for the HEMS crew while using the data collector in emergency situations? Should it be a passive automatic way, or a proactive way that requires inputs? To find out the solution, similar emergency cases from other fields (e.g. divers, skydivers, climbers) can be collected and analyzed. Based on this and results from the first research, simple prototypes can be built for user tests to seek the most suitable interaction between the HEMS crew and the data collector.

3.3 Assignment

The design assignment is formulated as following:

To design a data collector, carried by the HEMS personnel, which can register and store time in cardiac arrest in a very easy, practical, and low-time-consuming way.

The expected outcome of the project would be a design concept along with a functionally working prototype, which can be easily brought along with the HEMS crew, collect necessary medical data (mainly time of phases during OHCA), and send the data to a database (online/phone/PC). The collected data can be further retracted from the database for the HEMS team's further research.

3.4 Personal ambitions

First, I want to graduate with a Medesign project. In AED, I experienced the whole process of electronic product development. During electives, I have finished Prototype Connected Product course, which illustrates the process of prototyping connected prototypes. In the Machine learning course, I gained knowledge of using machine learning algorithms to analyze data. In the JMP project, I even went to India to conduct researches and user tests. All these experiences are prepared for me to start a Medesign project includes connected product development, data analysis and actual field tests. So this project suits me well. I have following ambitions for this project:

- In-depth knowledge on Intensive care area
- Whole process of building working connected product, from electronics to online platform or app
- Whole process of data analysis, from data collection, data transform to data analysis

3.5 Method and Plan

3.5.1 Method - Basic design cycle

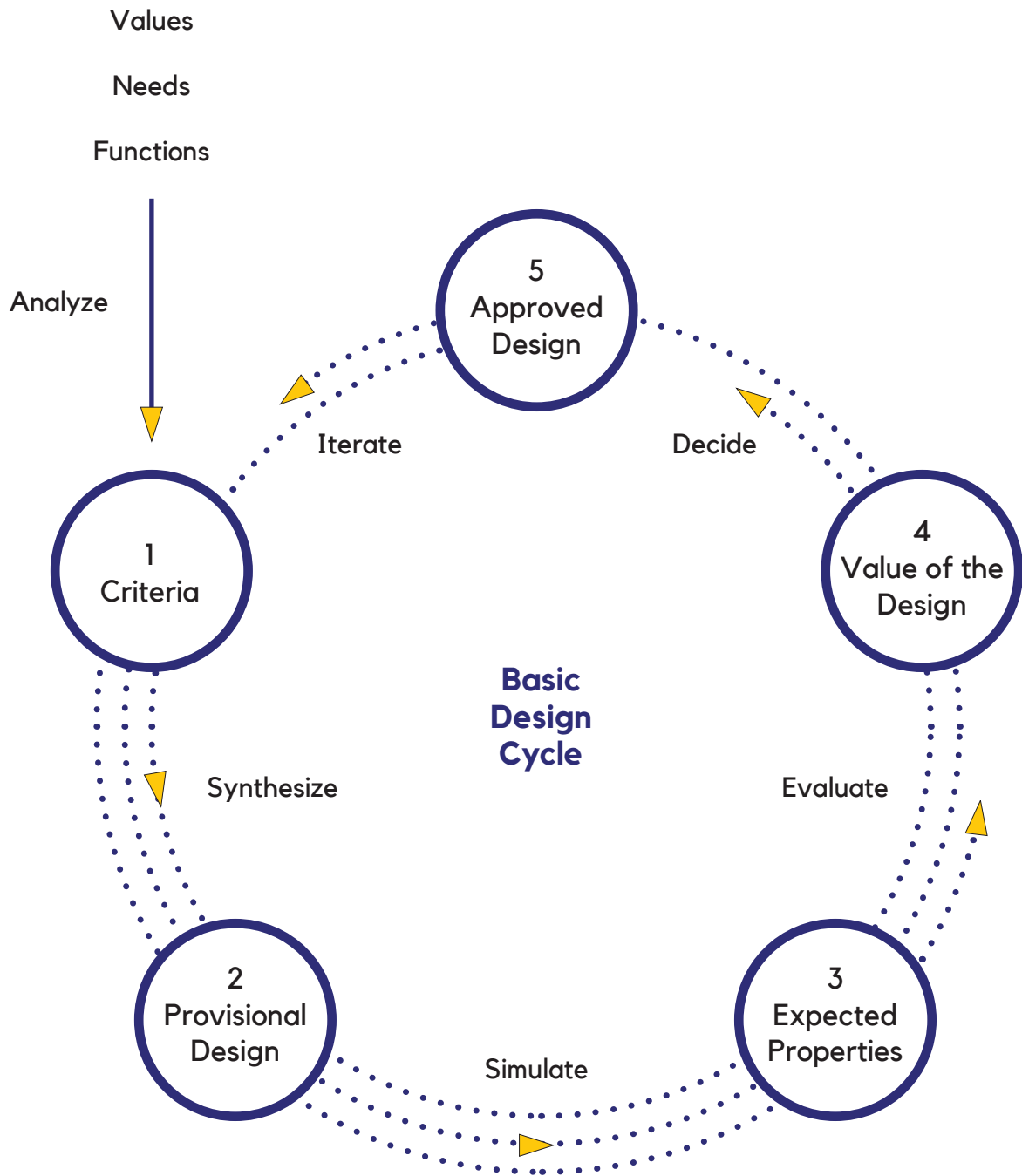


Figure 3.5 Basic design cycle ¹⁰⁶

The design process follows a basic design cycle shown in figure 3.5. Within one cycle, the designer starts with analysis on multiple aspects (e.g. values, needs, and functions) of the assignment, which leads to a list of design criteria. Then the designer moves on to the second stage and generates provisional designs. In stage 3, the designer makes simulations (e.g. drawing scenarios, clay models, etc.) of his provisional designs. Then he evaluates his designs by conducting tests based on the design criteria. At the end of the cycle, he decides what part of the design needs to be improved and starts another round of the cycle.

3.5.2 Plan - 4 cycles

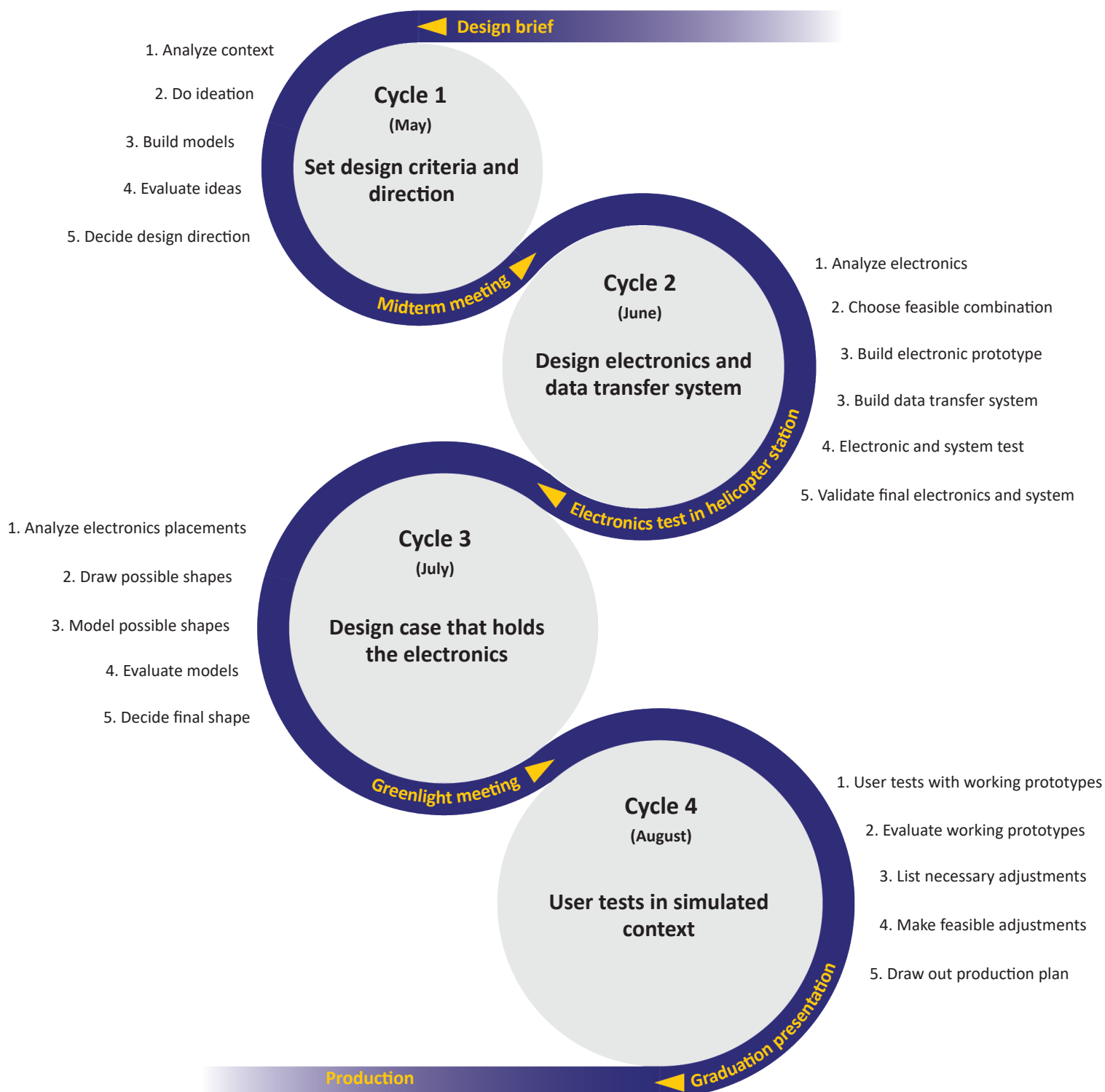


Figure 3.6 Project plan

For this 100-working-day project, 4 design cycles are planned. Each cycle takes one month and focuses on one major aspect of the design process: cycle 1 focuses on analysis on context and preliminary ideations. It ends with a list of design criteria and a clear design direction. Cycle 2 focuses on electronic design and data transfer system building. It involves more electronic prototyping and programming to build a functionally working data transfer system (both hardware and software). Cycle 3 focuses on the case design for the data collector electronics. It involves Solidworks model building and 3D printing. In Cycle 4, simulated user tests are conducted with the working prototype. The final design concept is validated and necessary elaboration is documented (e.g. renderings, photos, assembly, production methods, etc.). The whole graduation project ends with a full graduation report, a showcase, and a final presentation.

04

Cycle 1



In cycle 1, analysis of the relative topics of the project is conducted and design criteria are set up based on the analysis conclusion. Primary ideations are generated by following the design criteria. Evaluation of these ideas are taken during the midterm meeting and design direction is decided in the end.

4.1 Design criteria setting

4.1.1 Analysis overview

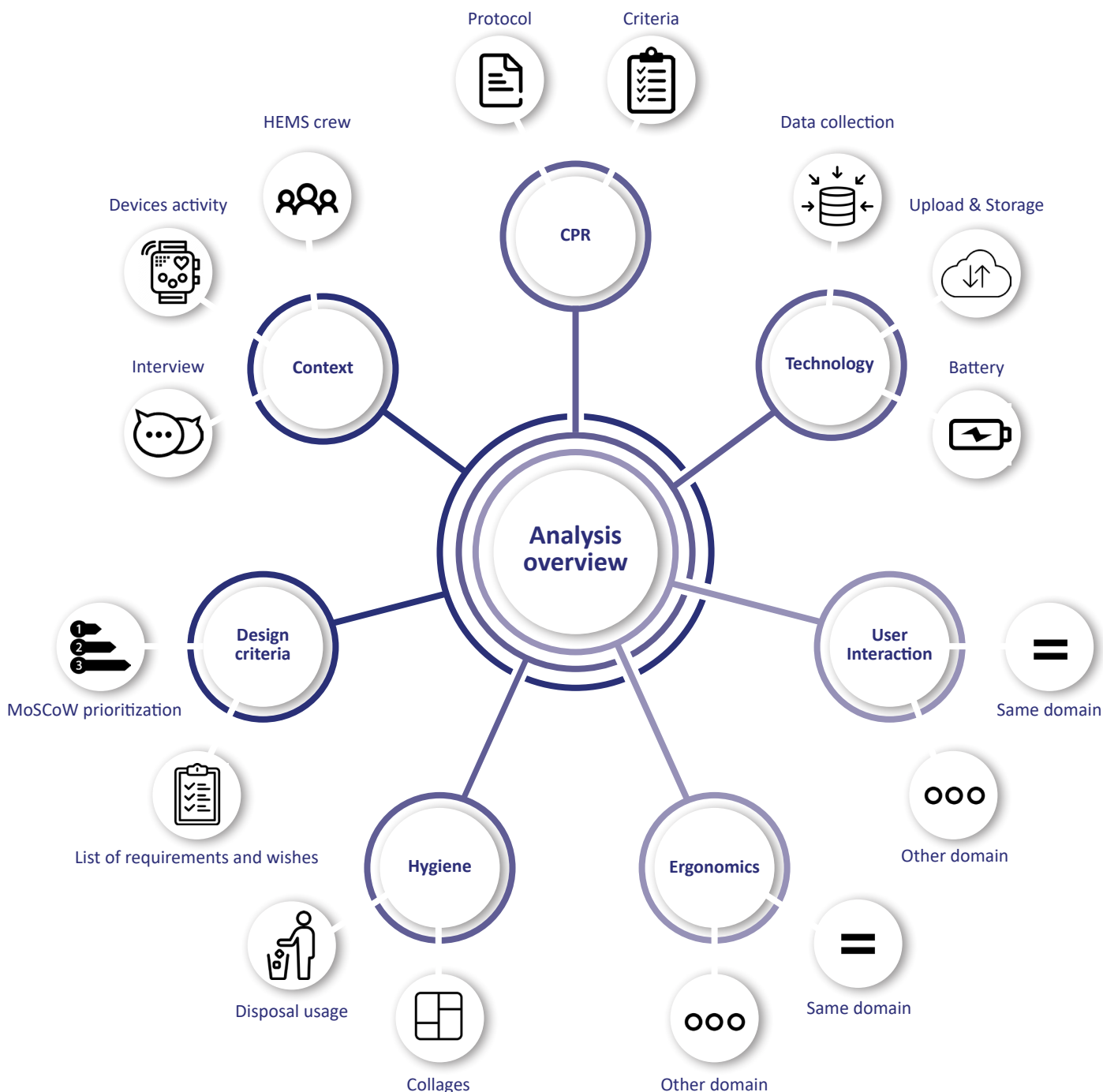


Figure 4.1 Analysis overview

An analysis overview is made to show the structure of the research topics during the analysis phase. First, the overall context of the HEMS crew in OHCA rescue cases is explored. Then the analysis perspective is zoomed into the CPR treatment process during OHCA cases, where further relative technologies, user interaction examples, ergonomics examples, and hygiene topics are covered. In the end, the analysis phase is concluded by a list of requirements and wishes on the expected data collector design, which is then prioritized based on the goal of the design assignment by following MoSCoW⁸⁷ method.

4.1.2 Context

Introduction

In this section, the overall context of the HEMS crew working in an OHCA rescue is explored. In order to find which timestamp is necessary to record during the rescue, both HEMS crew's activities and their carrying devices' activities are drawn out in timelines and scenarios. An interview with the client is taken for a better understanding of the process. In the end, this chapter is concluded that the chest compression stop time is important to record for the data collector because when chest compression stops, it normally means that the patient is either put on ECMO, or has a ROSC, or is dead.

HEMS crew activity scenarios and timeline

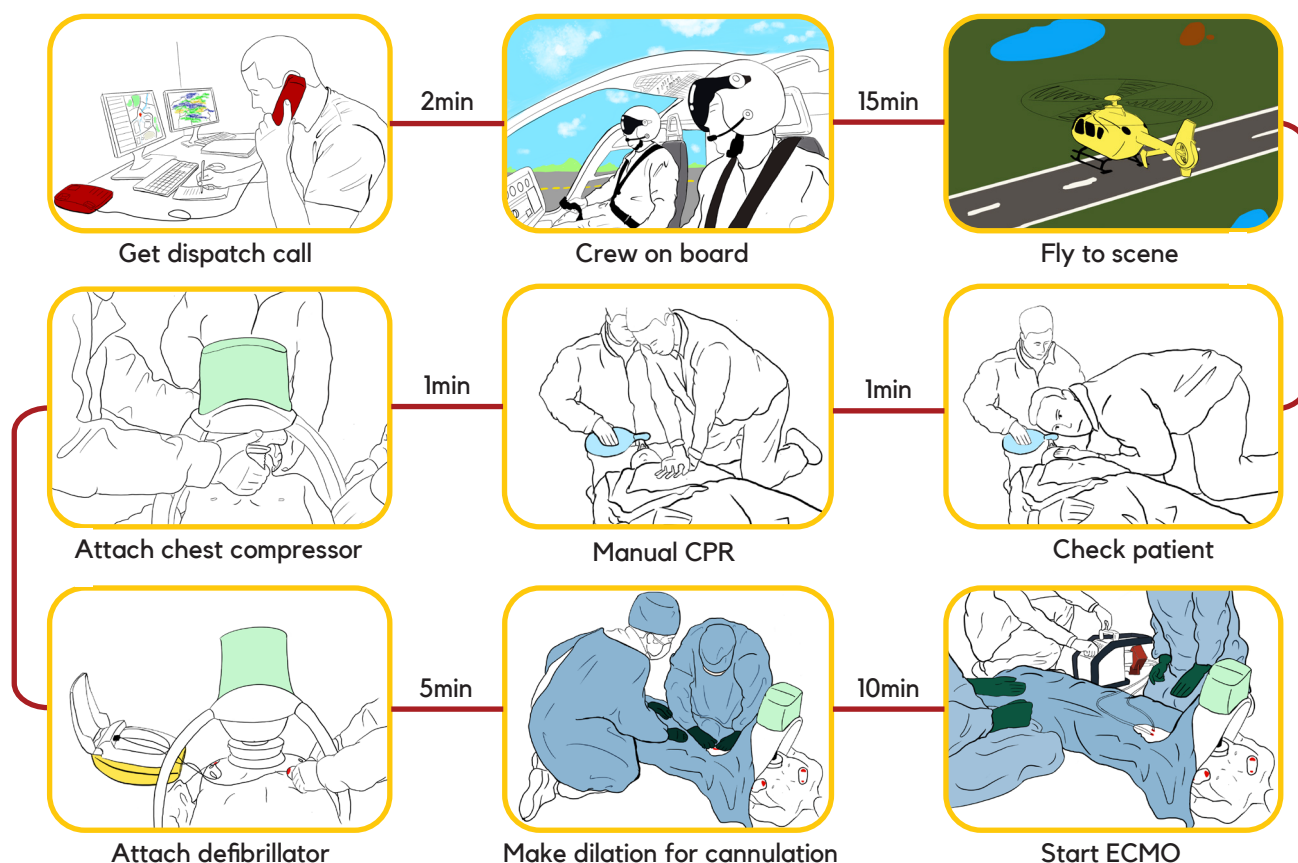
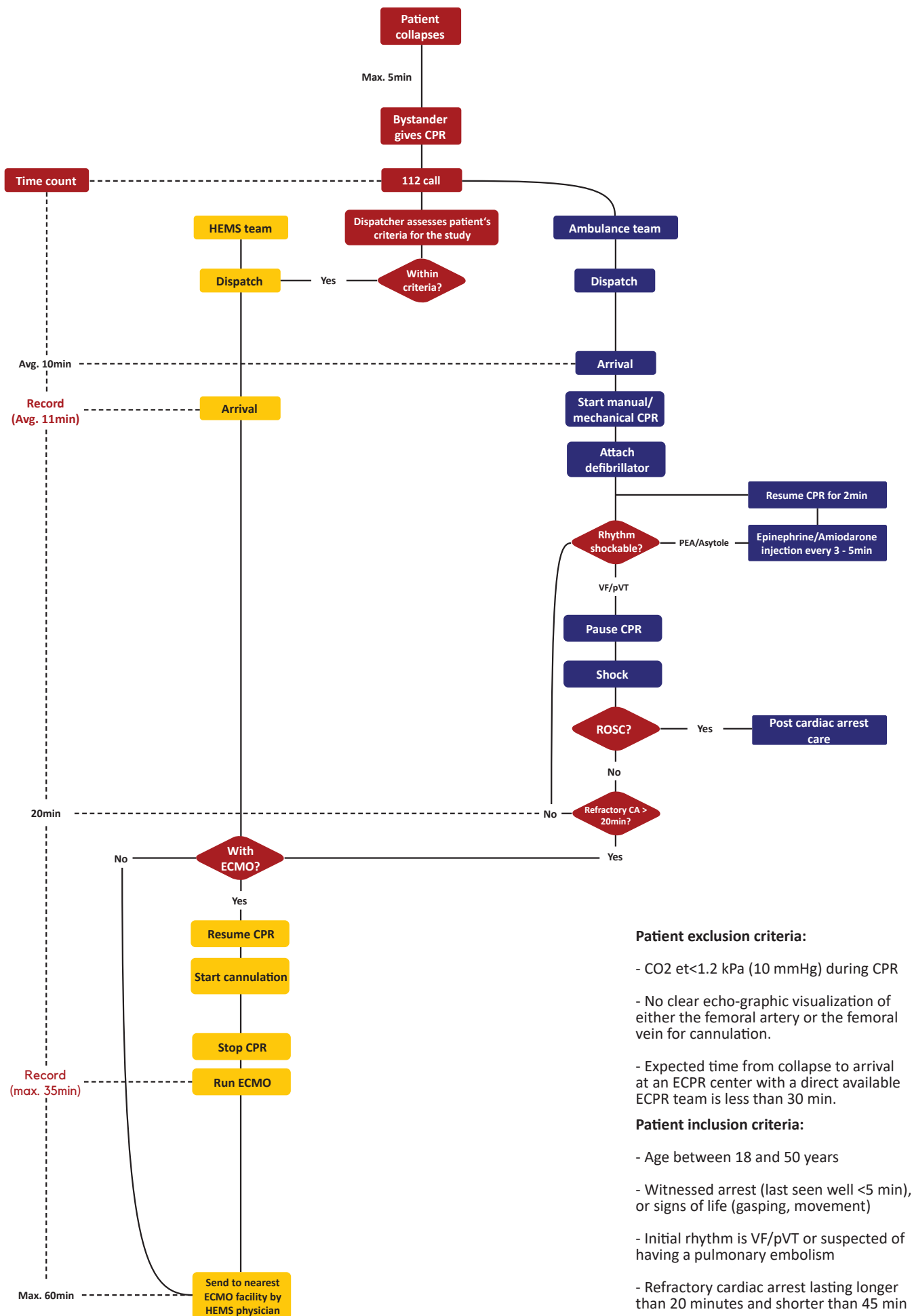


Figure 4.2 HEMS crew activity scenarios

In figure 4.2, the scenarios of the HEMS crew's activities during an OHCA rescue is given. As shown, at least 2 staff is needed for performing manual/mechanical CPR and cannulation for ECMO. Throughout the process, a bag valve mask, a defibrillator, a mechanical chest compressor, and a portable ECMO machine are used.

In the next page, figure 4.3 shows the timeline (detailed version in appendix C) corresponding to the scenarios, based on the On-Scene ECPR Study research protocol provided by Dr.Dinis (Appendix B). First, the patient should fulfill the inclusion criteria for dispatching the HEMS crew. Then the arrival time of the HEMS crew needs to be recorded. After 20min of refractory CA (no ROSC and no death), if the HEMS crew is the ECPR group which is equipped with ECMO, cannulation should start and be finished in 15min. CPR continues during the cannulation for ECMO, and stopped before running the ECMO, where the time needs also be recorded. If the HEMS crew is the control group, the patient will be sent to the nearest ECMO facility while CPR continues.



Patient exclusion criteria:

- CO2 et<1.2 kPa (10 mmHg) during CPR
- No clear echo-graphic visualization of either the femoral artery or the femoral vein for cannulation.
- Expected time from collapse to arrival at an ECPR center with a direct available ECPR team is less than 30 min.


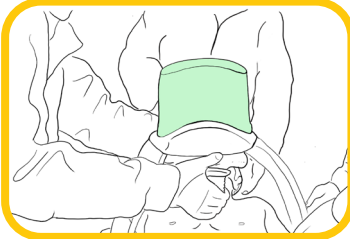
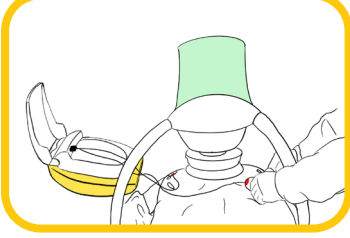
Patient inclusion criteria:

- Age between 18 and 50 years
- Witnessed arrest (last seen well <5 min), or signs of life (gaspings, movement)
- Initial rhythm is VF/pVT or suspected of having a pulmonary embolism
- Refractory cardiac arrest lasting longer than 20 minutes and shorter than 45 min

Figure 4.3 HEMS crew activity timeline

Used devices activity scenarios and timeline

Table 4.1 shows the activity scenarios and timeline of the used devices. The patient's reactions to his or her nose and chest are also included. At the preparation phase, manual CPR is first applied. Then if an automatic chest compressor is available, chest compression will be paused normally within 10s for implementing the compressor on the patient. Then CPR is resumed. During the CPR cycle phase, chest compression and ventilation are interrupted for maximal 10s every 2min because of rhythm checks and shock deliveries. After the shock delivery, chest compression immediately resumes. During the ECMO phase, chest compression continues during cannulation and stops right before ECMO starts running.

Scenarios	OHCA			Devices				Patient	
	Timeline	Phases	Actions	Bag valve mask	Automatic chest compressor	Defibrillator	ECMO	Nose	Chest
	00:00	Preparation	Start manual CPR	-	-	-	-	-	+
	00:10		Attach bag valve mask	/	-	-	-	-	+
	00:30		Attach Automatic chest compressor (If available)	+	/	-	-	+	-
	00:40		Start automatic chest compressing (If possible)	+	+	-	-	+	+
	01:00		Attach defibrillator	+	+	/	-	+	+
	01:10	CPR cycle	Stop manual CPR/automatic chest compressor	-	-	+	-	-	
	01:15		Check rhythm	-	-	+	-	-	
	01:20		Deliver shock	-	-	+	-	-	
	01:20		Resume manual CPR/automatic chest compressor	+	+	+	-	+	
	03:20		Manual CPR/automatic chest compressor for 2min	+	+	+	-	+	
	20:00	ECMO	Start cannulation	+	+	+	-	+	
	35:00		Stop manual CPR/automatic chest compressor	+	-	+	/	+	
	35:00		Run ECMO	+	-	+	+	+	

+ On - Off / Switch

Table 4.1 Used devices activity scenario's and timeline

Interview

An interview (Appendix D) is carried out with Dr. Dinis to have a deeper understanding of the actual situation during the rescue. Following is the important take-away from the interview:

Regarding their used devices:

- The HEMS teams normally don't bring defibrillator 95% of the time.
- Ambulance team is always equipped with defibrillators, but different teams use different brands.
- The defibrillator normally provides a time counting function for performing CPR cycles. However, it needs human input to reset after every cycle (2min) and does not give warning on overtime interruption (max. 10s) between cycles.
- The HEMS teams never bring automatic chest compressor. For ambulance teams, some are equipped with automatic chest compressor, some are not, and brands are also different.
- The ECPR groups of HEMS teams are equipped with Cardiohelp ECMOs and control groups do not have ECMO.

Regarding their team members:

- The pilot of the helicopter needs to guard helicopter, so normally he does not participate the rescue.

Regarding data collection:

- Time of 112 calls, arrival time of ambulance, patient arrival time in hospital, the outcome of the patient can be easily retracted from the hospital database.
- It is possible to retract CPR data from automatic chest compressors, defibrillators, and ECMO. But as said, different teams use different brands, and data retraction from these devices need to be manually conducted. Plus, these devices normally delete previous cases when new case starts, so data retraction has to be performed before a new case happens.

Conclusion

Following is a list of facts that is concluded from the context analysis, corresponding reflections on the data collector design is listed in *blue italic style*:

- The arrival time of HEMS crew to scene and the ECMO start time needs to be recorded by the data collector.

The arrival time could be recorded by the pilot, since normally pilot stands by the helicopter and has access to the transport data of the helicopter.

- Stop of chest compression means one of 3 outcomes of the patient: ROSC, ECMO on, or death.

The stop time is meaningful and important to record for the research, and the data collector should also record the patient's result from one of these three.

- The stop time of manual or mechanical chest compression could be considered as the ECMO start time.

The forces on and the movements from the patient's chest can be used to record chest compression data and define when chest compression stops.

- One CPR cycle consists of 2min chest compressions, and maximal 10s rhythm check and shock delivery.

If the chest compression stops for a short time (threshold not defined) but then resumes, that moment shouldn't be considered as ECMO start time, because it could be a long rhythm evaluation phase and does not mean ECMO starts running.

- The time counter of defibrillator does not give proper guidance to the team during the CPR process.

A time counter function that can guide the team to perform well CPR cycles is needed.

- During CPR cycles, both manual and mechanical chest compressions could be given to the patients based on the team's equipped devices.

The data collector should perform along with different devices in both manual and mechanical CPR process.

- Data retraction from currently used devices among different rescue teams is difficult and hard to organize.

If the data collector can also collect other devices data such as CPR data, ECMO data, it would be easier and more helpful for the study.

- 95% of the time HEMS crew only brings ECMO if needed, and does not bring defibrillator or automatic chest compressors.

If the HEMS crew brings the data collector to the scene, normally the HEMS team will face 2 situations: manual CPR is performing, or mechanical CPR is ongoing. So the way to set up the data collector into the CPR process while interrupt the ongoing action as little as possible needs to be considered.

4.1.3 CPR cycle

Introduction

In the previous section, an understanding of the overall context of the HEMS crew in OHCA is established. As shown from the conclusion of the context analysis, the data generated from the CPR process offers opportunities to determine the ECMO start time, and CPR data are also important for the study. So the analysis is zoomed in to focus on the CPR cycle in this section.

CPR protocol

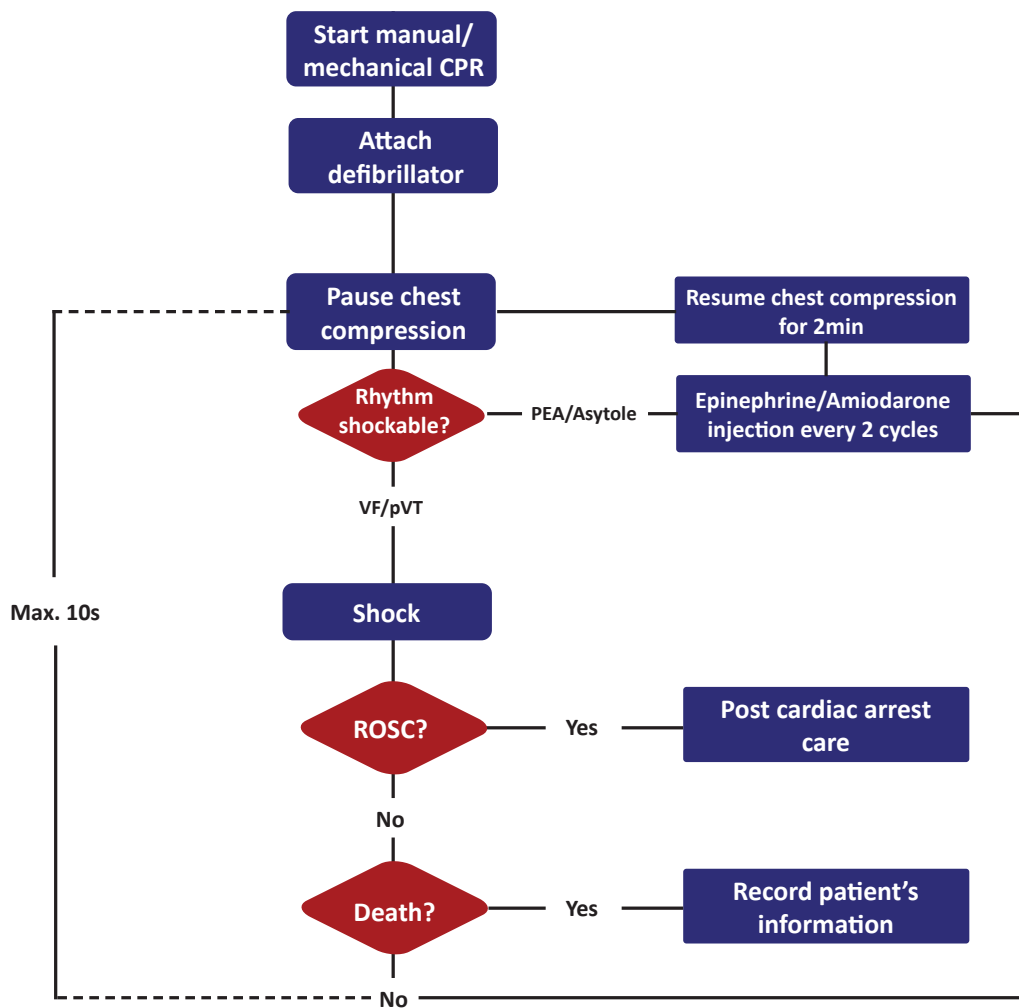
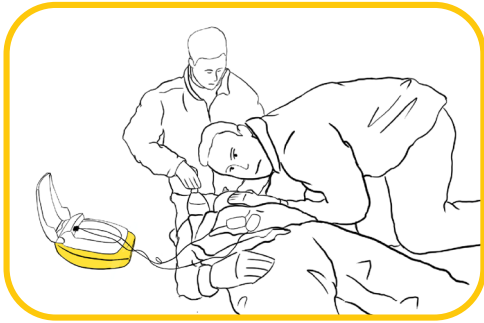
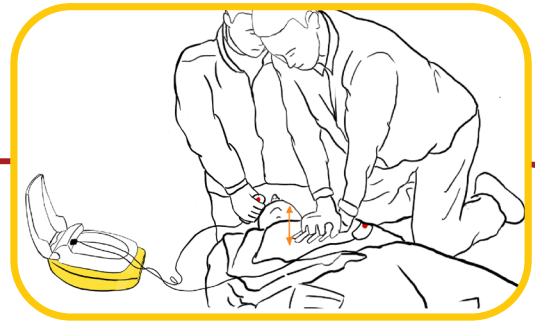


Figure 4.4 CPR protocol timeline

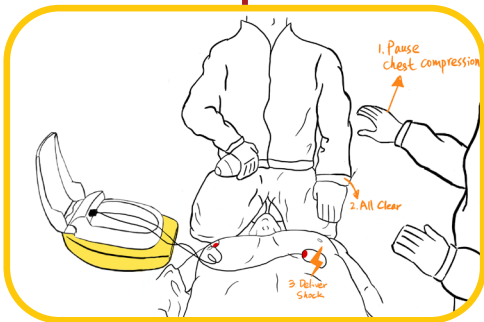
Figure 4.4 shows the CPR protocol timeline according to the European Resuscitation Council Guidelines for Resuscitation 2015⁶¹. Chest compression is paused every 2min, then there is a maximal 10s period for rhythm check and shock delivery. Chest compression is resumed immediately, if there is no sign of ROSC or death, epinephrine or amiodarone injection can be performed every 2 cycles; else if there is a sign of ROSC or death, pulse check needs to be performed.



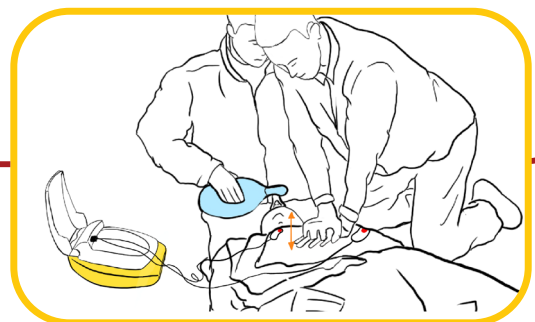
Check the patient's pulse



Start chest compression

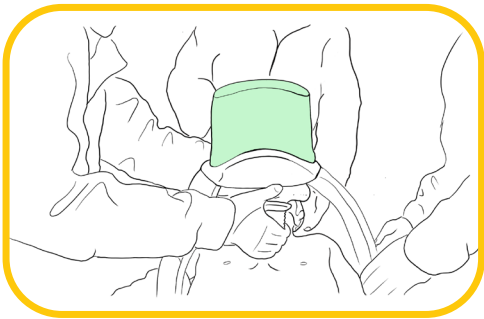


Pause chest compression, check rhythm and deliver shock

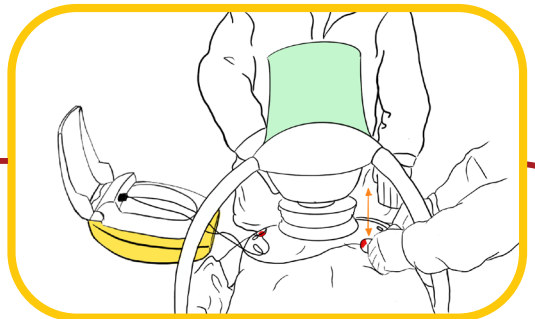


Attach defibrillator and give ventilation

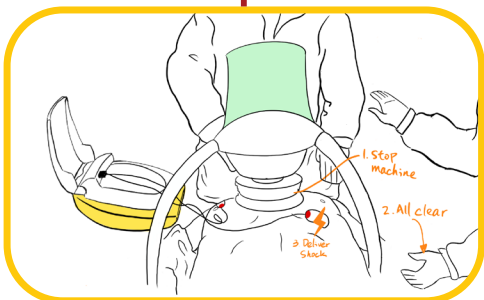
Figure 4.5 Manual CPR scenarios



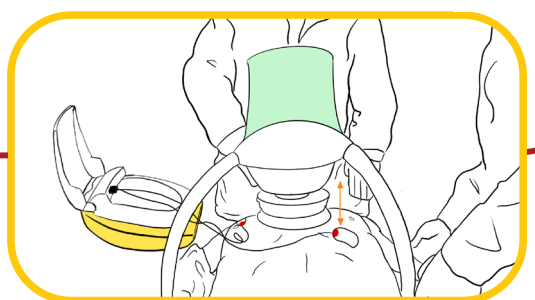
Set up the automatic chest compressor



Start chest compression



Pause chest compression, check rhythm and deliver shock



Attach defibrillator and give ventilation

Figure 4.6 Mechanical CPR scenarios

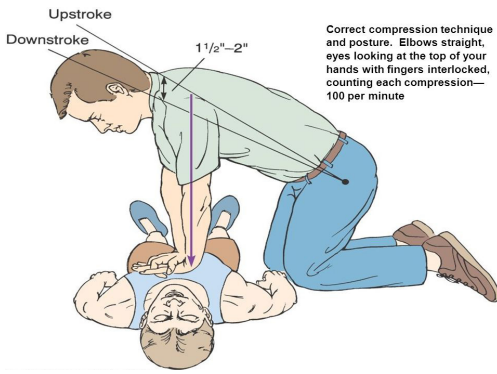


Figure 4.7 Manual chest compression posture⁵

Figure 4.5 and 4.6 shows the overall scenario's of manual and mechanical CPR. During manual CPR, rescue person needs to stack his or her hand on top of another one and lace the fingers of both hands together, then place the heel of bottom hand on the patient's central chest and push the chest with all upper body fixed and down. Figure 4.7 shows how the rescuer's upper body rotates up and down within an angle. From the scenarios, it is shown that the up and down movements of the vertical arms and hands are consistent with the patient's chest, so does the piston from the automatic chest compressor. Furthermore, during shock delivery, no person should have physical contact with the patient. The defibrillator and the chest compressor are designed specifically to withstand or insulate electric currents.

CPR performance criteria

Chest compression factors	Definition	Criteria
Depth	Compression displacement	Between 50mm to 60mm
Rate	Compression frequency per minute	Between 100 to 120 per minute
Fraction	Compression time/Total rescue time	At least 60%
Release velocity	The velocity when chest recoil from a compression	No exact threshold defined
Leaning	Rescuer exerts a constant pressure on patient's chest due to leaning, that causes incomplete recoil of the chest	Rescuer should avoid leaning and allow full recoil of patient's chest
Fatigue and injury	Rescuer encounters fatigue or feels uncomfortable to continue CPR	When basic chest compression quality is not meet, rescuer should be replaced if possible

Table 4.2 CPR performance criteria

Table 4.2 summarized the overall criteria to evaluate CPR performance, based on America Heart Association Guideline 2015⁵². Chest compression depth and rate are the most fundamental and important criteria to give a high-quality CPR. Chest compression fraction is the proportion of chest compression time during the whole rescue time, which for professional emergency teams, they always try to keep a 92% (120s/130s) proportion in each CPR cycle, according to Dr. Dinis. Higher release velocity helps chest recoil faster, study (Cheskes et al., 2015, p. 134) shows that faster release velocity corresponds to a higher survival rate in adult OHCA cases. Leaning is also another factor that causes incomplete chest recoil, and hard to be noticed by the rescuer.

Conclusion

Following is a list of facts that is concluded from the CPR cycle analysis, corresponding reflections on the data collector design is listed in *blue italic style*:

- Same from previous conclusion: One CPR cycle is 2min chest compressions, plus maximal 10s rhythm check and shock delivery.

The time counter function should be designed based on this rule.

- During chest compression, movements of the vertical arms and hands, and the pistons of the automatic chest compressor are consistent with the patient's chest.

Movement sensor that attached to the rescuer's hand or arm, or on automatic chest compressor's piston, can also record chest compression data. So the data collector could be designed as a wearable.

- During shock delivery, no person should have direct physical contact with the patient.

The data collector should withstand the electric shock if it has physical contact with the patient during shock delivery.

- CPR performance criteria include chest compression depth, rate, fraction, release velocity, leaning and rescuer's fatigue and injury.

The data collector should record depth data precisely, because rate, fraction, release velocity and leaning can be calculated based on it. Rescuer's fatigue and injury caused by chest compression is an interesting field to look up, see if there are any ergonomic solution to ease the problem.

4.1.4 Technology exploration

Introduction

The last 2 sections conclude that the data collector should record: ECMO start time (CPR stop time), the result of the patient (ROSC, ECMO, or death), chest compression depth. This section is focused on available technologies of collecting, uploading, and storing data.

Data collection

Real-Time Clock (RTC) module for time record

A real-time clock (RTC) module can keep track of the current time that starts from a setup time²⁷. To keep the RTC module running, it is necessary to keep the board powered. However, a button-sized lithium battery placed in the module as shown in figure 4.7 can keep the RTC module running for years⁵⁵. Besides, RTC can continue to operate in any sleep mode of the Arduino boards, and can also wake up the boards from sleep mode in a programmed way²⁷. This module is the only solution to keep track of time without internet access.



Figure 4.7 RTC module with cell battery

Chest compression depth

Chest compression depth can be reflected by applied force on the chest. A study indicates that the relation between compression force and depth is non-linear, as shown in figure 4.8. The stiffness of the chest differs a lot from people to people, and that makes it difficult to estimate compression depth based on compression forces. So measuring pressure on the chest is not a good solution to measure chest compression depth.

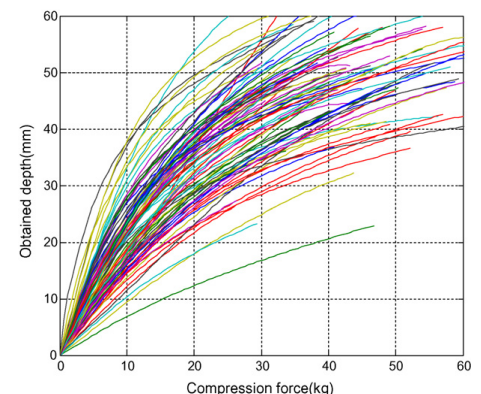


Figure 4.8 Compression force vs. Depth⁴

Normally, a double integration on the acceleration of the chest or the hands can calculate the displacement, also the depth: first integral of the acceleration over time is velocity, second integral of velocity over time is displacement (figure 4.9). However, in reality, double integration on acceleration data from an accelerometer always amplifies noises and offsets from the sensor. This makes the result drift away from the correct result. So a simple double integration on acceleration is not enough to estimate a correct displacement.

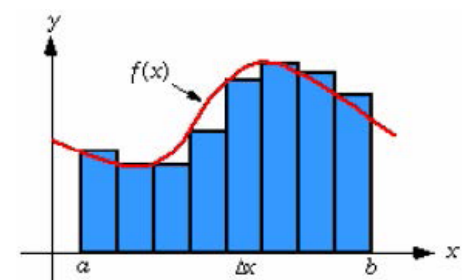


Figure 4.9 Integration of acceleration⁷²

2 studies indicates that by applying more complex algorithms and filters, the errors generated during double integration can be fixed^{38, 72}. One study applies the Trapezoidal method⁷² and another one uses Fast Fourier Transform³⁸.

Trapezoidal method

As shown in figure 4.10, the key solution of this method is to eliminate the area errors from integration by applying trapezoidal approximation, so half of the area errors can be subtracted. Plus, as shown in figure 4.11, a low pass filter can be applied to eliminate noises from the sensor. In this way, most errors from double integration are subtracted and the displacement results would be more accurate.

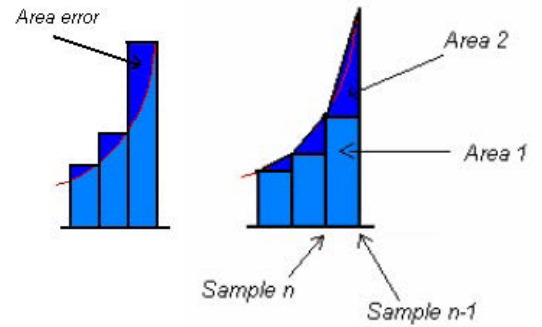


Figure 4.10 Eliminate area errors with trapezoidal approximation ⁷²

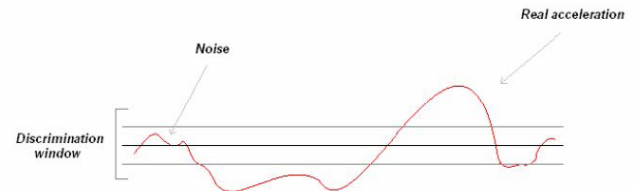


Figure 4.11 Apply low pass filter ⁷²

Fast Fourier Transform (FFT) method

Figure 4.12 shows the flowchart of the FFT algorithm. First, a time window of 2s is selected. Then a low pass filter is applied to filter out noises. After that, the acceleration-time wave is transformed into a frequency function by FFT. Finally, the frequency function is reconstructed into a sine wave that indicates the depth and rate during this 2s window. This method has been proven with actual chest compression data in its research (González-Otero et al., 2018).

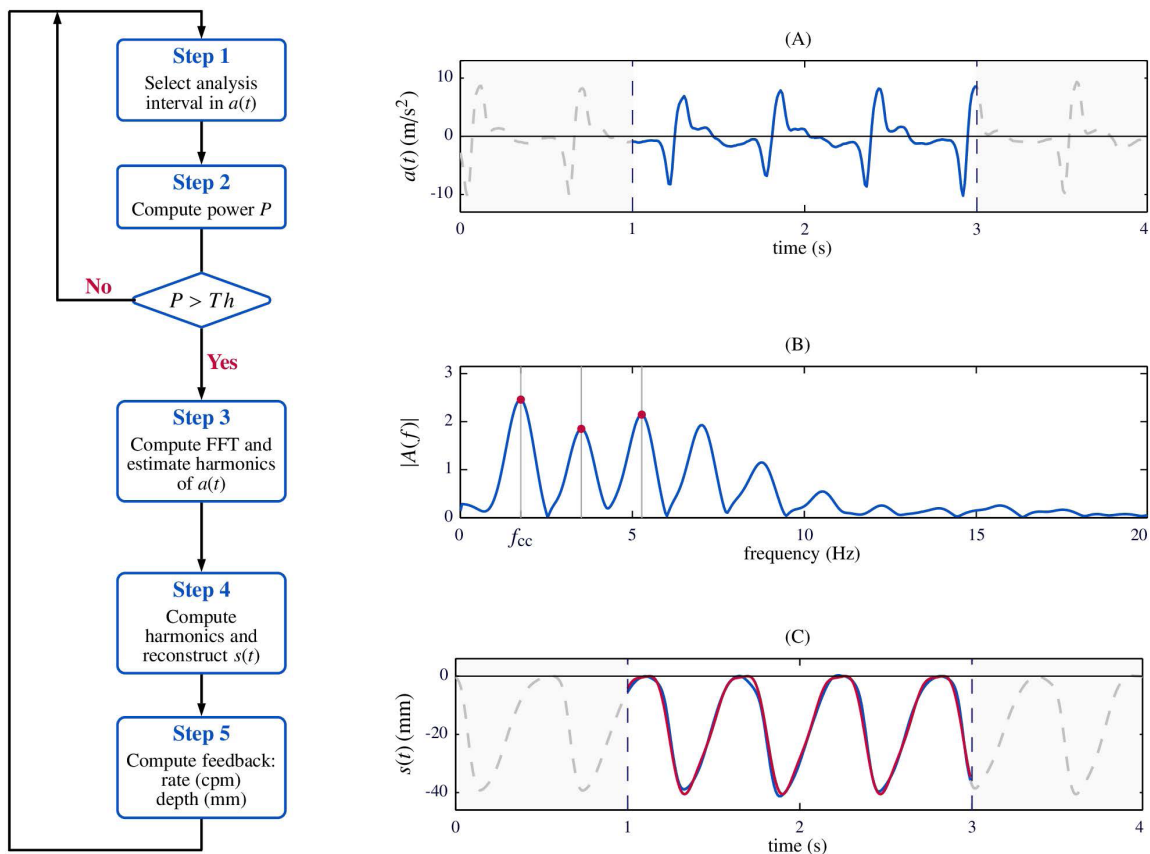


Figure 4.12 FFT application ³⁸

Data upload and storage

Data upload and storage methods

Figure 4.13 shows 4 ways to upload collected data from the data collector to the user's computer. The first method is saving the data locally to a micro SD card equipped in the data collector, then manually upload it to the user's PC. This method is conventional and requires no wireless technology, which makes it more reliable. The second method is the collector transferring the data to the user's smartphone via Bluetooth, then the smartphone will upload the data to a cloud dataset for later downloading to the PC. This method requires the device being Bluetooth connected to the phone while working, which may require extra configuration action on the phone side. The third one is implementing a 3G sim card to the data collector, so it constantly has access to the Internet to upload data to the cloud. It needs a monthly subscription for the 3G service. Instead of using 3G, the last method uses Wifi for Internet access. However, Wifi is not always available, so the device needs to save collected data locally first, and once it connects to a Wifi, it uploads the data to the cloud.

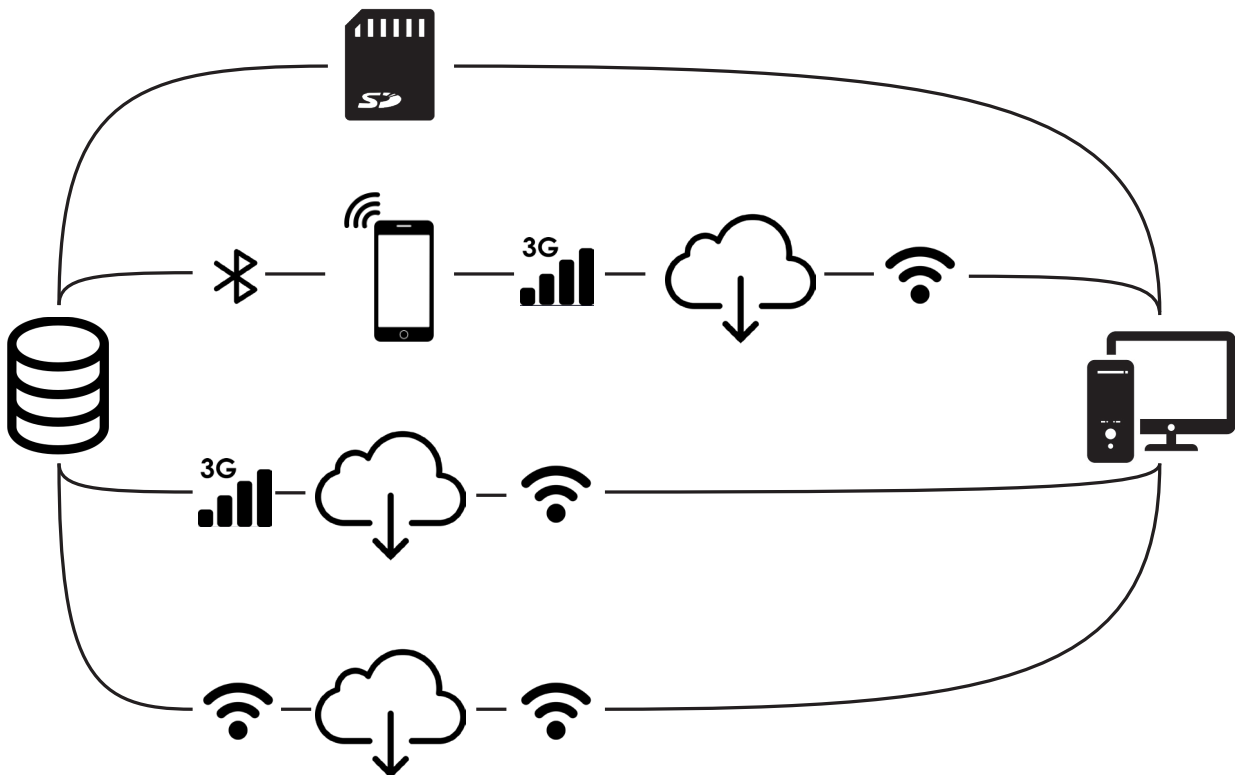


Figure 4.13 4 data upload and storage methods

All 4 ways have their own limits, either on autonomy, reliability, cost, or consistency. Combining some of their features can make up for the shortcomings among them. The micro SD card keeps the collected data in the device from possible data loss. So if any wireless technology is not working, data can still be saved in the memory card. However, the SD card has a limited memory. So it is necessary to make sure the card will be emptied for the new incoming data after all old data is uploaded. One good choice is combining the first method and the fourth one, the data stay in the micro SD card until Wifi is available for data transfer. Another good choice is combining the first and the second because smartphones can provide not only Internet access but also real-time audio and visual feedback from the data. It could provide an assistive feature for the data collector.

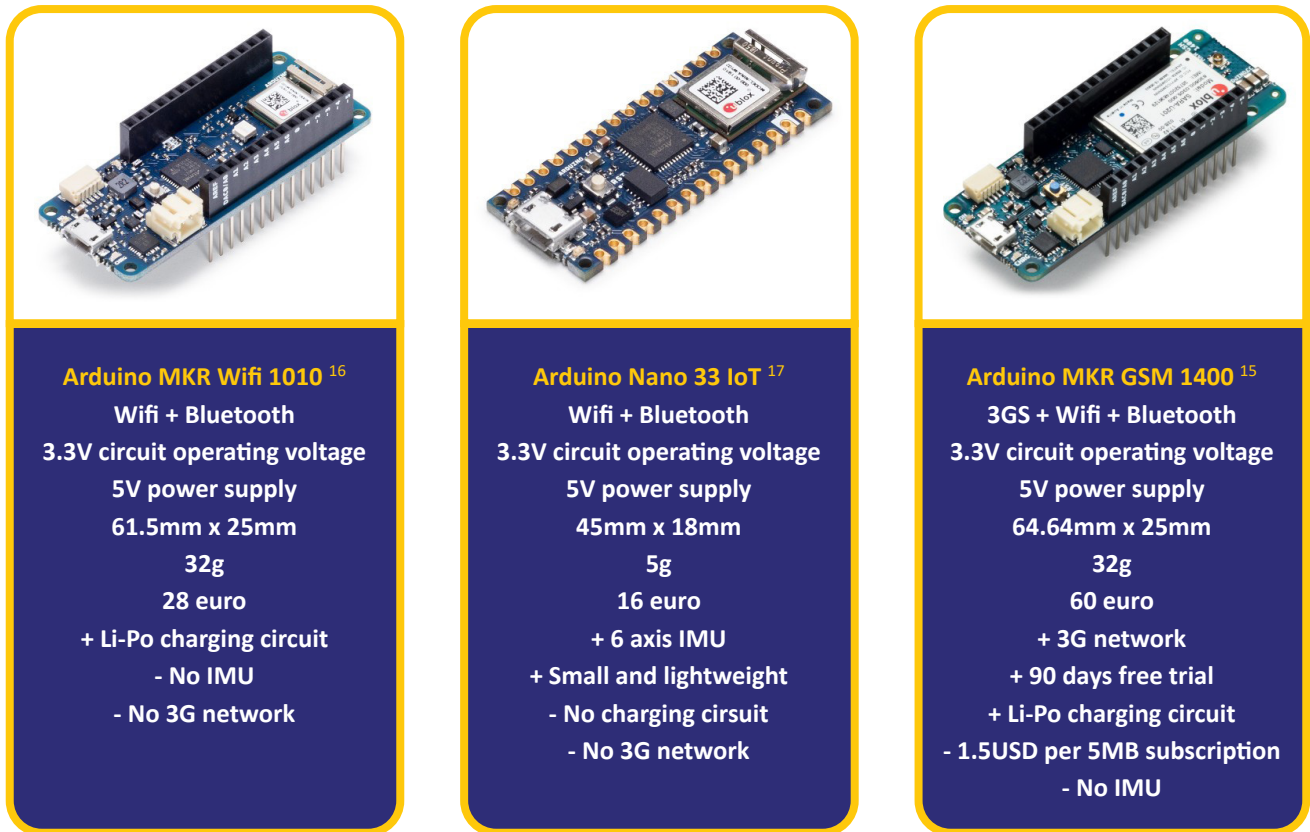


Figure 4.14 3 Arduino board choices

3 Arduino board choices

Figure 4.14 shows 3 Arduino boards ^{15, 16, 17} that have both wifi and Bluetooth features implemented on the board. Plus, all of them possess an RTC feature and can synchronize time via the Internet.

The left board is Arduino MKR Wifi 1010, despite the mentioned features, it has a Li-Po charging circuit implemented so no extra power management circuit is needed. It is middle-sized but relatively heavy. It does not have an IMU for acceleration sensing. Extra implementation of IMU is needed. Overall, it is a good function and price combination.

The middle one is Arduino Nano 33 IoT. It is cheaper, smaller and lightweight. It possesses a 6 axis IMU (3 axis accelerometer + 3 axis gyroscope), which is needed for collecting acceleration data as mentioned in the data collection topic. However, it does not have a power management circuit. So a charging circuit might be needed, which increases the size and weight. All in all, this is also a good choice, if the data collector is expected to be lightweight and small.

The last one is Arduino MKR GSM 1400. It is the largest among the three. It has a slot for a 3G sim card, so it has 3G service if there is a subscription. It also possesses a charging circuit but no IMU. This is the ideal choice if the 3G network is needed.

All 3 boards have their own advantages and shortcomings. But they provide enough functions for later ideation phase.







Name	 Arduino IoT Cloud ⁸	 Blynk ¹⁸	 Azure ³³	 Google Firebase ⁵⁸	 SMTP2GO ⁷	 IFTTT ⁷⁵
+	<ul style="list-style-type: none"> + Arduino's own platform + Examples available + Simple and clear interfaces + Associates with Google sheets 	<ul style="list-style-type: none"> + IOS and Android app available + Customizable GUI of app, showing data visualization + Free cloud data storage 	<ul style="list-style-type: none"> + First 12 months free on most service + 200USD credit + Professional interfaces + High quality services 	<ul style="list-style-type: none"> + Familiar platform + Generous free plan (10GB storage) 	<ul style="list-style-type: none"> + Email service + 1000 emails/month for free + Examples available 	<ul style="list-style-type: none"> + Brings all platform together, and can perform instructions among platforms + Voice control with Google assistant and Alexa
-	/		<ul style="list-style-type: none"> - Limited examples to learn from - Complex implementation 	/	/	<ul style="list-style-type: none"> - 200 USD per year
Conclusion	Good choice, worth a try because data will be stored as a Google sheets for download and further analysis. The price is not clear on the website.	Good choice if a real time data visualization is needed and it saves data in the phone which provides a second secure.	Not a good choice because available example is not a lot to learn from, which increase the difficulty of implementation.	Good choice, all the expected function can be fulfilled	Good choice, if receiving collected data in a form of Emails is preferred.	Not a good choice due to its high price.

Table 4.3 Available cloud platforms for data storage

Available cloud platforms for data storage

Table 4.3 shows an overview of available cloud platforms for data storage that is applicable in Arduino boards.

Different platforms have different advantages and limits. To conclude, Arduino IoT cloud would be a good choice if data stored in Google sheets is the expected outcome; Blynk is the go-to app for connecting the data connector to a smartphone (second method in data upload topic); Firebase is the least cost choice if only fundamental data transfer feature is needed; SMTP2GO is ideal if sending data in Email format is expected.

Battery supply




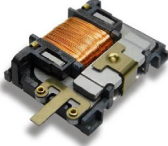
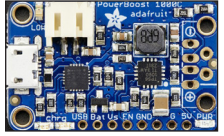
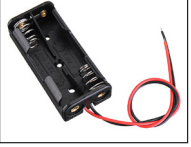
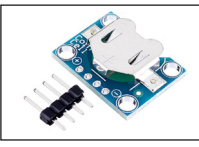
Name	 Li-Po battery	 Alkaline AAA battery	 Lithium cell battery	 Eco 200 energy harvester⁶⁹
Extra accessories	 A power boost module may be needed to manage charging	 AAA battery case is needed for connection	 Arduino cell battery holder is needed	Extra charging circuit might be needed
Parameters	1200mAh 35mm x 60mm x 5mm 24g 3V Rechargeable Around 12 euro	1000mAh ø10.5mm x 44.5 mm 15g 1.5V One time use Around 0.85 euro	200mAh ø20mm x 3.2mm 2.6g 1.5V One time use Around 0.25 euro	120 to 210 micro Joule 29.3mm x 19.5mm x 7mm N/A Output in 2V Constant use Around 6.65 euro
Pros and cons	+ Rechargeable, durable and operates in 3 - 5V - Expensive	+ Durable, one time use, lighter and cheap - Large size, need openings for change and need 2 in combination to output 3V	+ Lightest, small size, one time use and cheap - Small capacity (not durable), need openings for change and need 2 in combination to output 3V	+ Convert linear motion into electric energy - Charging efficiency is relatively low

Table 4.4. Available battery choices comparison

Table 4.4 shows an overview of available battery choices for Arduino boards.

Li-Po battery can be widely served as an external power for all types of Arduino boards, because of its compact size, lightweight and feature of recharging. Two of the three mentioned Arduino boards have already Li-Po charging circuit implemented. It is the perfect solution if a recharging feature is needed for the device.

AAA battery is mostly for one-time use. For the users, sometimes it is more convenient because they won't worry about constantly recharging the device. Preparation for backup batteries might suit certain scenarios better, such as emergency lighting,

Cell battery is the smallest and lightest choice, but with least capacity. It is also for one-time use. As mentioned, it would be enough to run the RTC module for almost a year, but not enough to run the board for a relatively long period.

Eco 200 energy harvester can convert linear motion into electric energy. It can collect the energy during the CPR process and store as extra power. However, its size is relatively large, and its efficiency needs to be further tested.

Conclusion

Following is a list of facts that is concluded from the technology exploration, corresponding reflections on the data collector design is listed in *blue italic style*:

- Real-Time Clock function is needed for time records.

All 2 mentioned Arduino boards are implemented with RTC. To keep track of real-time, constant power is needed, if the module is switched off, Internet access is needed to reset time.

- Accelerometer is a more accurate measurement than Force-sensitive resistor.

IMU (including a 3 axis accelerometer) should be implemented to measure chest compression depth. 2 algorithms are available for calculating depth.

- Micro SD card can secure the data before uploading it to the cloud/PC.

Micro SD card should be implemented to the data collector. Saved data can be cleared for new incoming data after uploading it to the cloud/PC.

- A smartphone is a good access to the Internet for the data collector.

The data collector can access the Internet by connecting to a smartphone via Bluetooth.

- Arduino MKR Wifi 1010, Nano 33 IoT, and MKR GSM 1400 have wifi, Bluetooth implemented. Each board also has its own features.

These 3 boards can be chosen for prototyping the data collector based on later developed ideas.

- Arduino IoT cloud platform can associate with Google sheets. Blynk app connects the Arduino device to the smartphone via Bluetooth. Firebase provides free data transfer and storage services.

SMTP2GO can send data in Email format to the user automatically. These 4 platforms are available for prototyping data transfer and storage features based on later developed ideas.

- Li-Po battery, AAA battery are 2 available options to power the board. Cell battery can power the RTC module for a year. Eco 200 can convert linear motion to electric energy.

Both Li-Po and AAA are good power sources for the board, but their recharging ways (recharge vs. backup battery) are different, the choice between these two is based on the user scenario's from later ideation. Cell battery could be implemented just for RTC, but it occupies space too. Eco 200 collects energy when the device is moving, but its efficiency needs to be tested, and it also requires space for implementation.

4.1.5 User interaction examples

Intuitive interaction model

Blackler and Popovic (2006, p.9) introduced a design methodology for applying intuitive interaction. As shown in figure 4.15, the interactive interaction mode guides designers to look at a certain product or system at a body reflectors, population stereotypes, familiar features from the same domain, familiar features from other domains and metaphors from other domains level respectively, and at each level, categorize collected features into function, appearance, and location. Then the collection serves as inspirations or examples to design their own interaction product or system.

Body reflectors: these are the simplest forms of intuitive interactions. Examples are: headsets, glasses, shoes, and gloves. These forms are self-evident to people.

Population stereotypes: these are at a more complex level that people learned these interactions at a young age. They are highly dependent on people's areas, professional backgrounds. For example, most Europeans won't know how to use chopsticks but east Asians normally knows it at a young age. These forms are less universal than the body reflectors.

Familiar features from the same domain: these interactions are at a more concrete level. Products tend to have similar user interaction under the same product category: smart electronics usually consists of a display section and a hand input section, two-section normally are next to each other.

Familiar features from other domains: these are at the same level as the former one, but look at other product categories. One example could be the interfaces of the calculator in smartphone adopts the similar forms of an actual calculator. Features from other relative domain can also be good examples to learn from.

Metaphors from other domains: metaphors retrieve useful analogies from memory or elements of a know situation, to explain a completely new concept or function. For example, the curvy form of the Philips Senseo coffee machine metaphors a bending servant who pours coffee to you.

The article also mentions that familiar features from the same and different domains would be the main mechanism for designers to use to apply intuitive interaction.

This model is applied in both interaction and

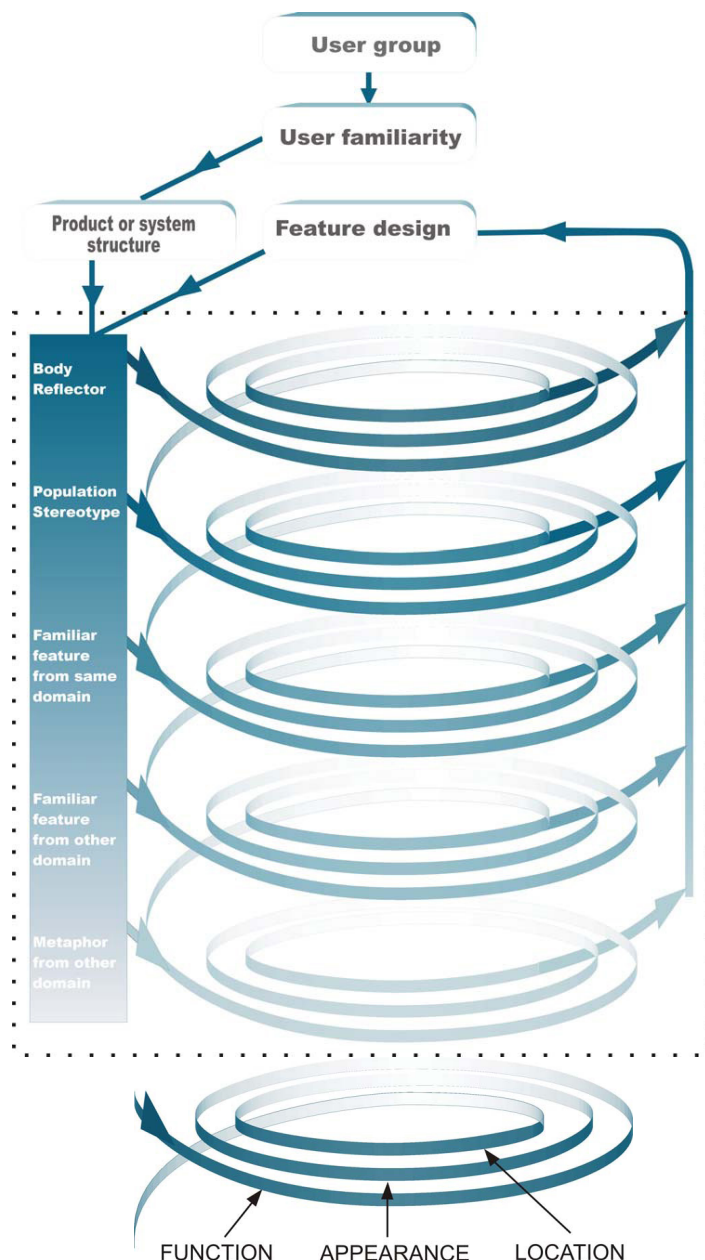


Figure 4.15 Intuitive interaction model ²³

ergonomics analysis, to generate an inspiring collection of features that later ideation phase can benefit from. Only familiar features in the same and different domains are applied as this level is the most influential one for the later ideation phase.

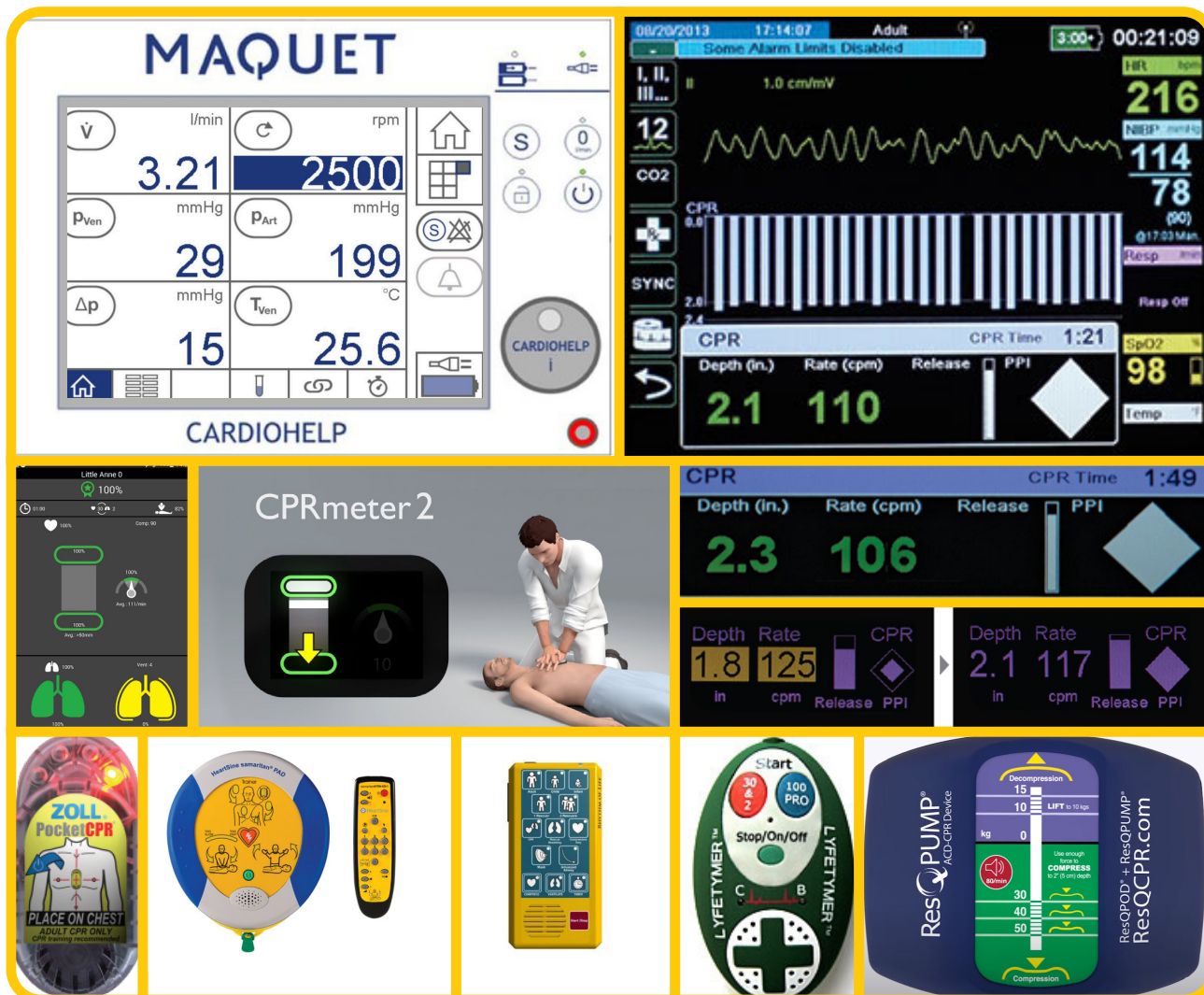


Figure 4.16 Familiar features from devices designed for OHCA

Figure 4.16 is a collage that collects different interfaces of different devices that are designed for OHCA.

Function

The major function feature is giving audio and visual feedback to the rescuers. Most feedbacks shows whether the compression depth and rate meet the standard, or simply showing the number of depth and rate. The devices are designed to passively react to rescuers' actions.

Appearance

For dynamic feedbacks, numbers are mostly used to show rate, depth, and time. Columnar shapes are used to indicate the dynamic change of depth, force, and release velocity. About the color use, green and blue are widely used. Red is only used for important buttons and indicating important signals. There is only one red button or signal in one device. Texts are used more than

icons. Drawings are used to describe procedures, not delivering feedbacks.

Location

The information placements on the display follows normal reading orders (left to right, top to bottom) based on its importance. For small devices with only one or two visual feedbacks, the important information is put in the center of the display, others information will show in a smaller size.

Familiar features from other domains



Figure 4.17 Familiar features from other domain

Figure 4.17 is a collage that collects familiar interfaces from other domains that serve for the similar interactions like the medical devices used in OHCA.

Function

The interfaces include timers, metronomes, and music games. Music game interfaces offer multiple types of visuals to guide the players' actions. This is a more proactive interaction between devices and users.

Appearance

For timers, circle is a commonly used feature, indicating a population stereotype that clock is round. For metronomes, a swinging pointer is the major feature to show the tempo. In music games, moving units towards a threshold line is applied to tell the player when to take an action.

Location

For timers, a large circular visualization indicating remaining time typically sits at the middle of the interfaces, and the exact numbers of the time are shown in the center of the circular clock or around it. In music games, the moving unit moves in either horizontal or vertical direction. If the units move in a horizontal direction, the animation is normally in perspective.

Conclusion

Following is a list of facts that is concluded from the interaction analysis, corresponding reflections on the data collector design is listed in *blue italic style*:

- The medical devices are designed to passively react to rescuers' actions, while music games proactively try to guide users to react with correct actions.

The way how music game guides the player worth to learn from for designing the interaction of CPR guidance.

- Numbers and texts are commonly used as visual feedback

For complex devices like ECMO and defibrillator, exact numbers with a clear indication of text are important for doctors to make decisions. But for simple chest compression feedback devices, it is more important to give clear instructions to guide the rescuer operating correctly. Clear visualization and audio indication could be more helpful than plain numbers and texts.

- Columnar shapes are widely used to display the dynamic change of depth, force, and release velocity.

Adapting the music game interaction to design interactive columnar shapes to guide rescuers could be an interesting direction.

- Circular shapes are usually used for showing remaining times.

Timer app interfaces are good examples to learn from to implement the time counter feature for the data collector.

- Both in the medical domain or other domains, for a small display, important information is put in the center and other information is sized down.

For the data collector, time and chest compression feedbacks are both important, how to position these 2 feedbacks in one small display needs to be thought through.

4.1.6 Ergonomic shape examples

Familiar features from same domain



Figure 4.18 Familiar ergonomic features from medical devices in OHCA

Figure 4.18 is a collage that collects familiar ergonomic features from the medical domain.

Function

There are usually hand bars on large medical devices for rescuers to carry around. 2 CPR devices are designed into ergonomic shapes that are easy to hold on for the rescuers, which changes the hand positions, but they are proved to improve the rescuer's CPR performance ^{41, 53}.

Appearance

The appearance of most medical devices gives

a reliable, robust, and durable impression. Matte plastic is widely used to establish a professional look.

Location

All the CPR devices operate on the patient's chest. Pocket-sized CPR devices stay between the chest and hands.

Familiar features from other domains



Figure 4.19 Familiar ergonomic features from other domains

Figure 4.19 is a collage that collects familiar ergonomic features from other domains.

Function

One ergonomic features of a hand plunger is similar to one CPR device, implemented with a suction cup to generate an up-forward force when retracting the compression. Another ergonomic feature is the wrist protector, which keeps the wrist movement from hyperextension⁵⁷. The push-up grip is also designed to straighten the wrist and avoid exerting too much pressure on the wrist during push-ups.

Appearance

As shown, the appearance of these products guided the users to hold in the correct ergonomic positions, in order to maximize efficiency or to protect the user's joints.

Location

The collected products are all handheld products.

Conclusion

Following is a list of facts are concluded from the ergonomics analysis, corresponding reflections on the data collector design is listed in *blue Italian style*:

- Ergonomic shapes that change the hold position during chest compression may improve the performance.

If the data collector can be designed as an ergonomic shape and might help the rescuer's CPR performance.

- Wrist protector restrain wrist movement to avoid hyperextension and potential injury. Push up bars straighten the user's wrist to avoid pressure on the wrist.

These two structures can be learned from for ideation on the shape of the data collector.

- The appearance of most medical devices gives a reliable, robust, and durable impression.

The appearance of the data collector should give the same feelings as these medical devices to the user.

- All pocket-sized CPR devices stay between the chest and hands.

The data collector should be a wearable or a handheld device, which size should not be too large to bring with. Dr.Dinis indicates that they have a bag for all equipment and the data collector should fit in.

4.1.7 Hygiene problems in scene

Introduction

This analysis area is conducted after the midterm evaluation. During the evaluation, hygiene needs such as keeping the device clean, keeping the patients' wounds away from devices, usage of disposal products are discussed. Thus, additional analysis on used devices during OHCA and how these devices are treated to keep both the patient and themselves clean is conducted.

Collages

The first collage is made to understand more how the OHCA scenes look like in reality. Some simulation scenes and real scenes are collected in the collage. As shown in figure 4.20, in most cases, the rescuers wear disposable rubber gloves when they have contact with patients. Patients normally lie on the ground or on the emergency beds. In some cases, patients may have bloodstains on their bodies or clothes. The medical devices and rescue bags are usually placed on the ground.



Figure 4.20 Collage of OHCA rescue scenes

The second collage gathers some simulated and real cannulation scenes. As mentioned in the context analysis, cannulation is a necessary treatment to implement ECMO on patients. As shown in figure 4.21, in all cases, the patients are covered with a disposable sheet with an opening at the cannulation location. All the doctors wear disposable clothes, masks, gloves, and hats, while conducting cannulation. During cannulation, the chest compression continues. Blood may splash out around the cannulation location. Surgical instruments are placed on the sheet near the surgeons.



Figure 4.21 Collage of cannulation scenes during ECMO implementation

Disposal usage

Table 4.5 summarizes some devices that are used by rescuers during OHCA. As it shows, the part of the devices which directly have skin contact with the patients are designed to be disposable or cleanable. Rest parts of the device usually can be cleaned with mild detergent.












Category	Product	Disposal part	Usage
Defibrillator	 <p>Corpuls 3 ⁴²</p>	 <p>Corpatch</p>	<ul style="list-style-type: none"> - Sense chest motion - Send compression data to defibrillator - Patch is one time use
	 <p>Zoll Defibrillator ¹⁰⁵</p>	 <p>Zoll AED plus pads ¹⁰⁸</p>	<ul style="list-style-type: none"> - Sense chest motion - Send compression data to defibrillator - Conduct electricity for defibrillation - Pad is one time use
Automatic chest compressor	 <p>Corpuls CPR ²⁶</p>	 <p>Stamp</p>	<ul style="list-style-type: none"> - Compress chest - Stamp can be sterilized and reused
	 <p>Zoll Autopulse ¹⁰⁷</p>	 <p>Liveband</p>	<ul style="list-style-type: none"> - Compress chest - Liveband is one time use - Spills with a disinfectant or bactericidal wipe to clean the body surfaces
	 <p>Lucas device ⁶²</p>	 <p>Suction cup</p>	<ul style="list-style-type: none"> - Compress chest - Suction cup is one time use - Main body can be cleaned with soft cloth and warm water with soft detergent
Manual chest compressor	 <p>Zoll ResQ ³</p>	 <p>Suction cup</p>	<ul style="list-style-type: none"> - Compress chest - Can be disposed or cleaned with mild detergent and tap water
	 <p>Zoll PocketCPR ⁶⁴</p>	<p>None</p>	<ul style="list-style-type: none"> - Compress chest - Sense chest motion - Give compression feedback - Wipe with mild detergent and water after use

Table 4.5 Summary of disposal usage of used devices during OHCA

Conclusion

Following is a list of facts are concluded from the ergonomics analysis, corresponding reflections on the data collector design is listed in *blue Italian style*:

- In most cases, the in-use medical devices and rescue bags are lay on the ground around the rescuers.
The data collector should withstand the outdoor environment.

- The part of the devices which directly has skin contact with the patients are usually designed to be disposable or cleanable.

 - In order to keep the device clean and ready for every use, the part of the data collector that has direct skin contact with the patient should also be disposable or cleanable.*

- The disposable part of the device can be easily removed and replaced with a new one for the next session.

 - The disposable part of the data collector should be also removable.*

4.1.8 Design criteria

List of requirements and wishes

As a conclusion of the analysis phase, a list of requirements and *wishes* for the data collector design is formulated, based on the checklist by Pugh¹⁰⁶.

1. Performance

1.1 The data collector needs to collect following data precisely: ECMO start time, chest compression depth and rate.

1.2 The data collector can be set up within 10s without interrupting other running devices.

1.3 The data collector can save the data locally before upload to cloud.

1.4 The data collector can upload the data to a cloud platform, Google drive, or Email automatically.

1.5 Using data collector will not affect hygiene standard during OHCA rescue.

1.6 The data collector can guide the rescuers to follow proper CPR cycles with its timer feature.

1.7 The data collector can improve rescuer's chest compression performance.

2. Environment

2.1 The data collector can function under compression forces during both the manual CPR and mechanical CPR process.

2.2 The data collector should withstand electric shock from defibrillator.

2.3 The data collector can be used both in day and night.

3. Life in service

3.1 FIX.

3.2 The data collector can be easily cleaned (by detergent) after use.

4.1 Software updates for the data collector should be available.

4. Maintenance

4.2 The data collector can be disassembled for maintenance.

5.1 The production cost and of the data collector, data upload and storage cost should be as cheap as possible.

5. Product cost

6.1 At least 3 data collectors should be made for one HEMS station.

6. Quantity

7.1 The data collector is producible based on Arduino by the

designer.

7. Production

8.1 The data collector should fit in HEMS crew's bag and be as light as possible.

8. Size and weight

9.1 The data collector should have a reliable, robust and durable look.

9. Aesthetic, appearance and finish

10.1 The HEMS team can understand how to use the data collector within a 10min demonstration session.

10. Interaction

10.2 The audio and visual feedback of the data collector guide the rescuer to perform better chest compression.

11.1 The data collector is ergonomically comfortable to use.

11. Ergonomics

11.2 The data collector can relief fatigue or the wrist pain of the rescuers during chest compression.

12.1 The data collector sends notification to user if it needs a recharge or exchange of battery.

12. Reliability

12.2 The data collector can secure collected data locally when encounter sudden powering off.

13.1 The data collector can be stored in HEMS crews' rescue bag.

13. Storage

14.1 Test on the precision of the collected data should be carried out.

14. Testing

14.2 Test on the data upload and storage to dataset (Cloud, Google drive or Email) should be carried out.

14.3 Test the performance of the data collector running on a simulated CPR scene with a manikin should be carried out.

14.4 Test the performance of the data collector running on simulated manual and mechanical CPR scenes with real person should be carried out.

15.1 The data collector does not harm the rescuers and the patients under proper usage.

15. Safety

16.1 The electronic parts of the data collector can be reused for other purposes.

16. Reuse, recycling

MoSCoW prioritization

The end goal of this project is not only to provide a concept design as a solution, but also a working prototype is expected to be tested in real OHCA cases. Within this 100-working-day project timeframe, the progress of building a working prototype is hard to predict. So it is necessary to prioritize the list of design requirements more specifically within a working prototype scenario. Hereby MoSCoW⁸⁷ prioritization method is introduced. The list of requirements is divided into 4 priorities: Must have, should have, could have, will not have, as must-have requirements will be definitely fulfilled within this project's timeframe. While should-have and could-have features will be considered to add to the working prototype if time is proficient.

Must have

Non-negotiable features that must be fulfilled within given timeframe

1.1, 1.3, 1.4

The working prototype of the data collector must be able to collect ECMO start time and chest compression rate data, store it locally, and uploads it to an online dataset.

Should have

Important features that are not vital, but add significant value

1.2, 1.5,

The working prototype should be able to be set up within 10s, not affect the hygiene standard of the scene.

Could have

Features that are nice to have but have small impact if left out

1.6, 1.7, 10.1, 10.2, 11.1, 11.2

The working prototype could be able to guide rescuers and improve their rescue performance by giving audio and visual feedback and ergonomic shapes.

Won't have

Not a priority for given timeframe

All other requirements and wishes are not a priority for working prototype development in this project's timeframe.

4.2 Ideation

4.1.1 Clay models

Before starting the ideation phase, a CPR training manikin is used to experience the chest compression maneuver. After 3 rounds of chest compression with each rounds lasting 2 minutes, sore pain on tester's palm, wrist and back is felt strongly. Based on this experience and by following requirement 11.1 and wish 11.2, 4 clay models are developed to explore if the data collector's shape can be designed to relieve the rescuer's pain from chest compression and therefore improve efficiency.



Figure 4.22 4 clay models build upon manikin

Model 1



Figure 4.23 Clay model 1

Figure 4.23 shows the first built clay model. It has an ergonomic shape that its upside is dented to follow the shape of the human's palm and wrist, and its downside's shape fills the middle chest shape. This shape fills up the hollow space between the rescuer's palm and the patient's chest to reduce the friction between them during chest compression, therefore gives a more comfortable experience to rescuers.

Model 2

Based on the model 1 and inspired by Philips Q-CPR⁶⁷ measurement and feedback tool, model 2 is built, as shown in figure 4.24. It possesses an ergonomic shape at the compression area like model 1, but gets an extensive part out of the compression area, which is a screen that gives feedback to the rescuer during chest compression, like the Q-CPR.



Figure 4.24 Clay model 2

Model 3



Figure 4.25 Clay model 3

Model 3 in figure 4.25 is a smaller circular version of the model 1. Its compact round shape is easy to fit in the rescuer's pocket. It totally lies under the palm so no visual feedback can be given.

Model 4

Model 4 in figure 4.26 is developed to relieve the wrist stress during chest compression by changing the hand position. As shown, by giving hand bars to hold on, the rescuer's wrists are not excessively bending. When the wrists are straightened, no hyperextension⁵⁷ occurs, which may cause wrist pain. This model is inspired by the push-up bars that are used for fitness. In the middle between the two bars, there are also spaces left to place visual feedback on the model for the rescuers to review during chest compression.



Figure 4.26 Clay model 4

4.1.2 Electronics

Force Sensitive Resistor (FSR)

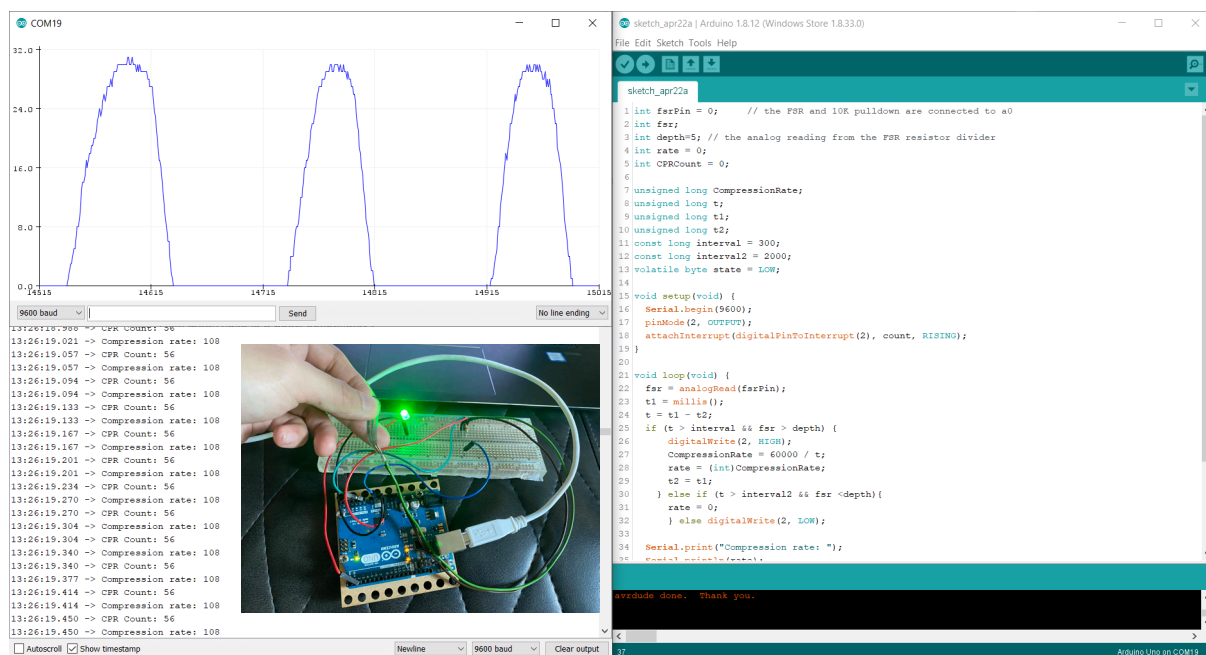


Figure 4.27 FSR on Arduino

The Force Sensitive Resistor (FSR) is first being explored on Arduino because it can measure the compression force exerted on it. As shown in figure 4.27, FSR can continuously detect how much for the fingers are pressing on it. The recorded compression force can be used to evaluate the compression rate, depth by following the CPR criteria.

CorPatch

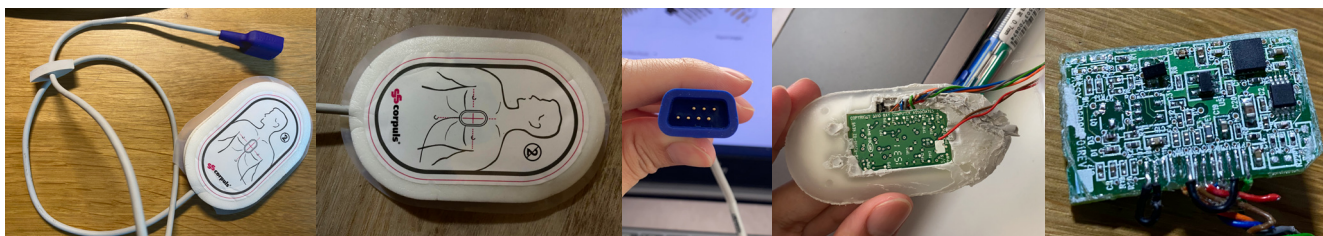


Figure 4.28 CorPatch disassemble

CorPatch²⁵ is a product used by Dr.Dinis' HEMS crew during OHCA rescue. It is a patch that one side is connected to a Corpuls defibrillator, another side is placed on the center of the patient's chest. It is used to collect chest compression data for the defibrillator. It is also disposable. As shown in figure 4.28, the CorPatch uses a 9-pins male D-Sub⁹⁴ to connect to the defibrillator. Inside the CorPatch, there is an ADXL335³⁰ 3-axis accelerometer, which is validated by reading the prints on the electronic components. The CorPatch could be a possible disposal component for the data collector. If the data collector can be connected to the CorPatch, it can stay away from the patient to keep itself clean. Then CorPatch can collect acceleration data and send them to the data collector.

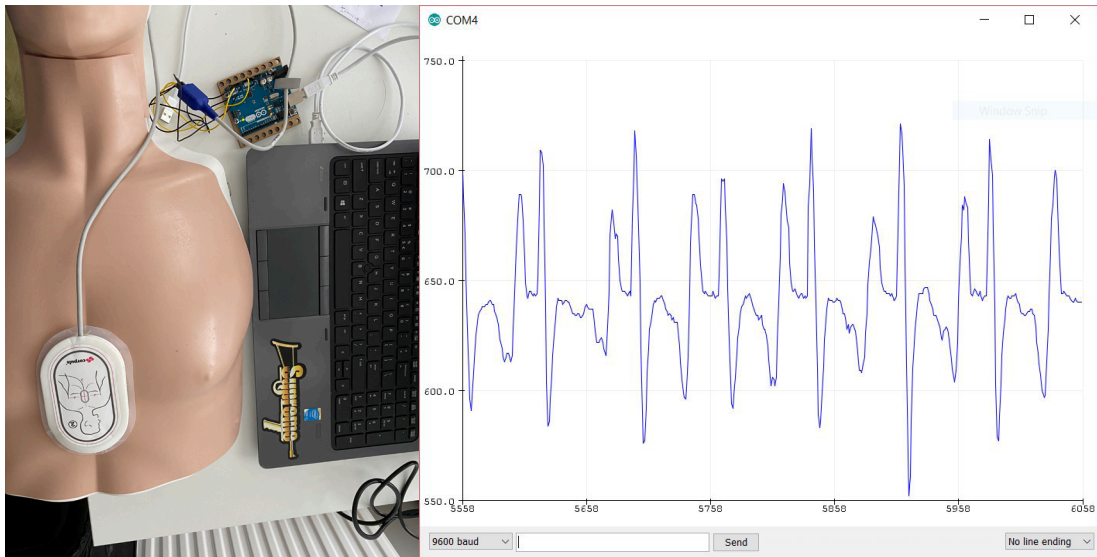


Figure 4.29 CorPatch on Arduino

Figure 4.29 shows how the CorPatch's output value can be read on Arduino. To find out which pin value outputs the acceleration data, a pinout sheet of CorPatch is provided by the company (Appendix E). However, it shows that the CorPatch uses the I2C⁹⁸ interface to transfer data and the company does not have detailed I2C addresses to read the actual acceleration value from the sensor (ADXL335). It also takes a lot of effort to reverse-engineer the chip, according to an expert from Applied Labs.

As reading values from the sensor via I2C interface is impossible, every pin from the D-Sub is tested on Arduino by connecting to A0 (analog read pin) to see what value it outputs (Appendix E). The result is shown in figure 4.30, pin 4 and 8 acts as ground and power supply pin⁹⁰, pin 6 outputs values that reacts to the movement of the CorPatch (reading in figure 4.29). Figure 4.31 shows how the values from pin 6 changes along with the turning angle of the CorPatch. The pin 6 output value is possibly associated with an angular acceleration of one axis from the sensor, but it is uncertain what this value exactly means, as again reverse-engineer it takes huge amount of time.

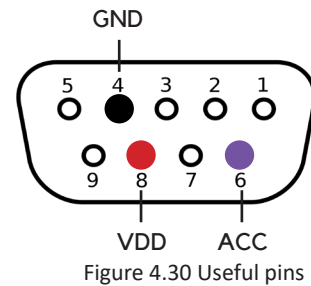


Figure 4.30 Useful pins

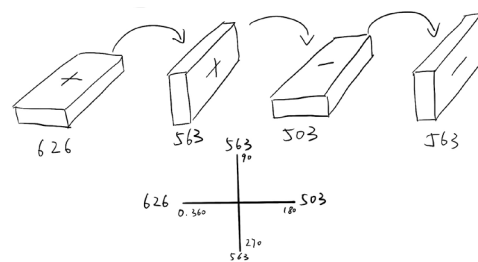


Figure 4.31 Pin 6 relation to CorPatch angles

To conclude, it is feasible to receive values from pin 6 of the CorPatch, which is related to the movement of the CorPatch. The meaning of the value is unknown. But during the test with Arduino, it is clearly shown that the value pulses along with the linear movement of the CorPatch in figure 4.29. So the pin values can still be used to calculate the chest compression rate, but not the compression depth.

Basic logic flowchart

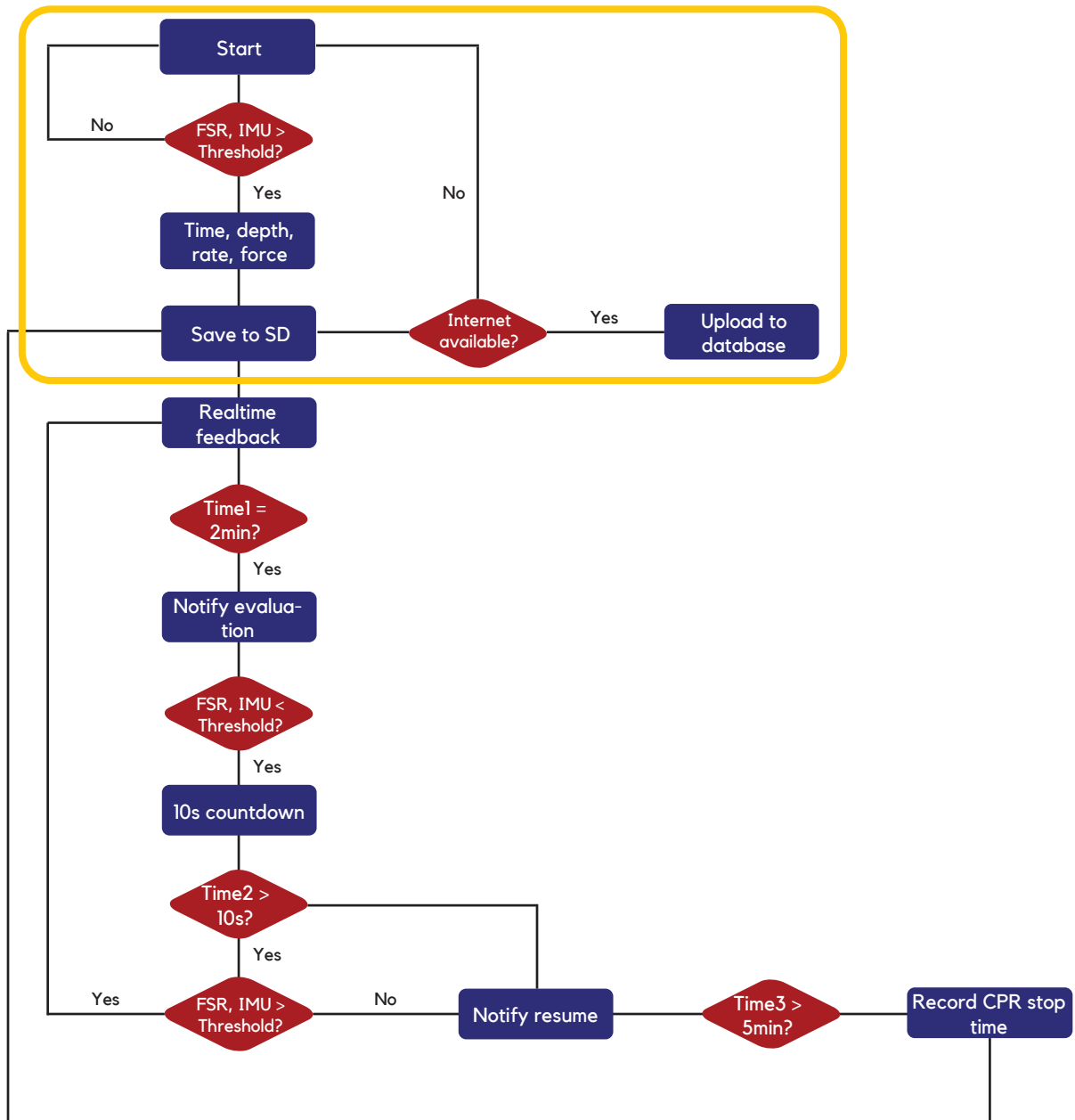


Figure 4.32 Arduino MKR Wifi 1010 to micro SD card module

A basic logic flowchart (Figure 4.32) is built for the data transfer system to sort out different duties that the codes and electronics need to finish at different stages. Also, it shows where a threshold needs to be set for stage changing. The following electronic prototype is built to fulfill the must-have duties (inside the yellow rectangle) - read data, save data to SD, and upload data to the database.

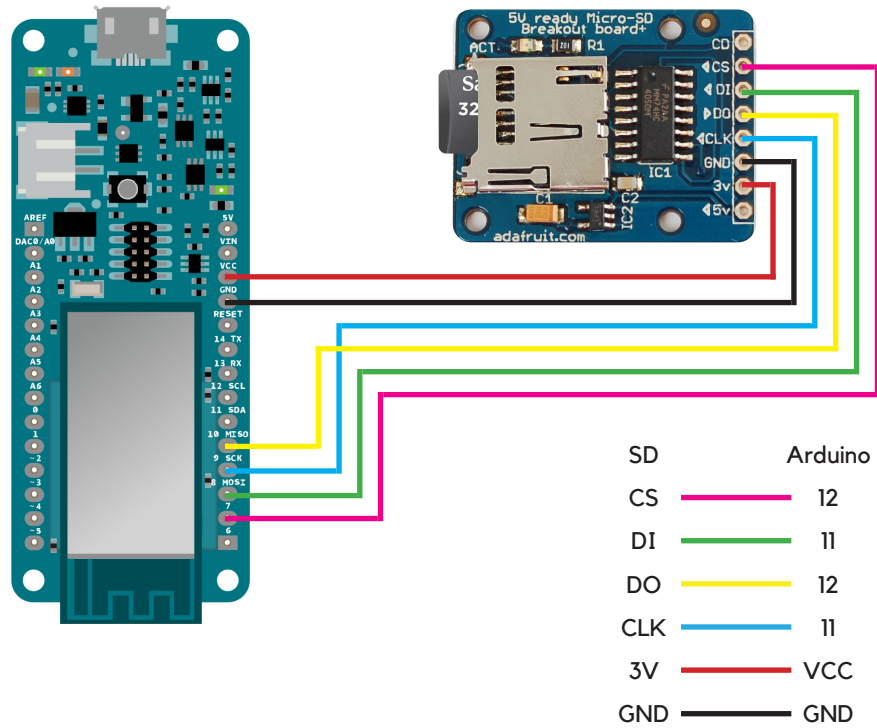


Figure 4.33 Arduino MKR Wifi 1010 to micro SD card module

From former technology analysis on data upload and storage, Arduino MKR Wifi 1010 is chosen as the prototyping board. A micro SD card module⁵⁴ is added (figure 4.32) first because the ability to save data locally is a must-have. In this electronic prototype, acceleration data from an accelerometer (MPU6050) and time data is recorded per second and saved in the micro SD card.

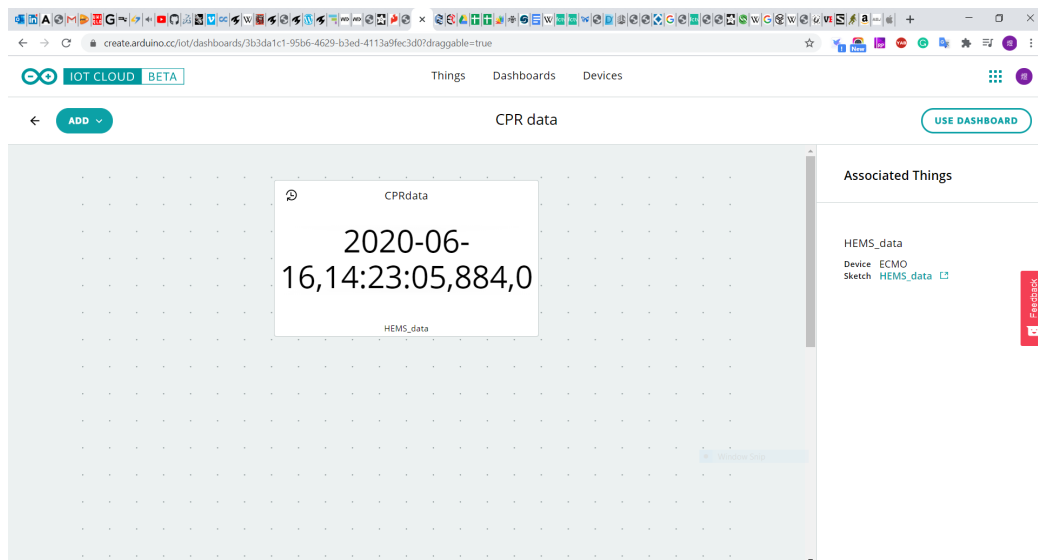


Figure 4.34 CorPatch disassemble

Arduino IOT cloud¹³ is the first explored cloud platform because it is developed for Arduino boards and offers useful examples. The Arduino MKR Wifi 1010 first accesses the Internet via Wifi. Then upload the saved data from the micro SD card to the Arduino IOT cloud (codes in appendix F). As shown in figure 4.33, the current date, time, and acceleration value is shown on the dashboard. Users can download historic data from Arduino IOT cloud as a csv file. However, for free account users, the historic data is only saved for a day in the cloud. For subscription users, the historic data is kept for a week after upload.

4.3 Concepts

Morphological chart










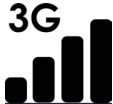









Sense	 FSR	 IMU			
Feedback	 Visual	 Audio	 Haptic		
Collect	 From patient	 From rescuer	 From device		
Upload	 USB	 3G/4G	 Bluetooth	 Wifi	
Store	 Memory card	 Smartphone	 PC	 Cloud	 Email
Power	 Rechargeable	 Disposable			

Figure 4.35 Morphological chart ¹⁰⁶

To generate principal solutions in an analytical and systematic way, the morphological chart ¹⁰⁶ method is used. As shown in figure 4.32, the main required functions of the data collector are listed on the left in the chart. The possible electronic solutions for each function are listed on the right. These solutions are generated from the former technology analysis. The chart let the designer to combine these solutions to form a complete workable concept.

The following concepts are all generated based on this morphological chart, clay models, and electronic prototypes.

Concept 1 - Steer wheel

Concept 1 - steer wheel adopts clay model 4 as the overall shape. The main electronics are placed inside the removable cylinder part, which contains the Arduino board, sensors, and batteries. This part is Bluetooth connected to the rescuer's smartphone and send collected data to the app on the phone. The phone can be placed in the middle of the steer. The App receives data from the cylinder and present them on the screen to give rescuers guidance.

This concept uses a modular design to fit both 2 scenarios: manual CPR and mechanical CPR. It changes the hand position of chest compressing to reduce the rescuer's fatigue and wrist pain. It uses phone to save data. On the other hand, it is large in size, it needs phone to show visual feedback and data storage. It needs to be designed to fit with different phones.

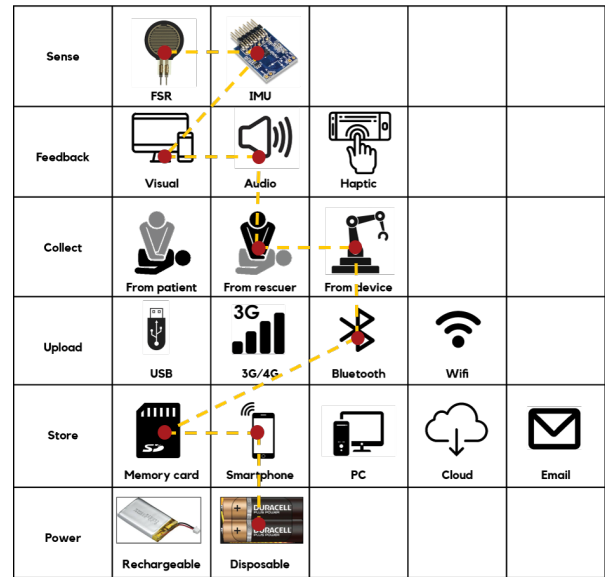


Figure 4.36 Morphological chart of concept 1

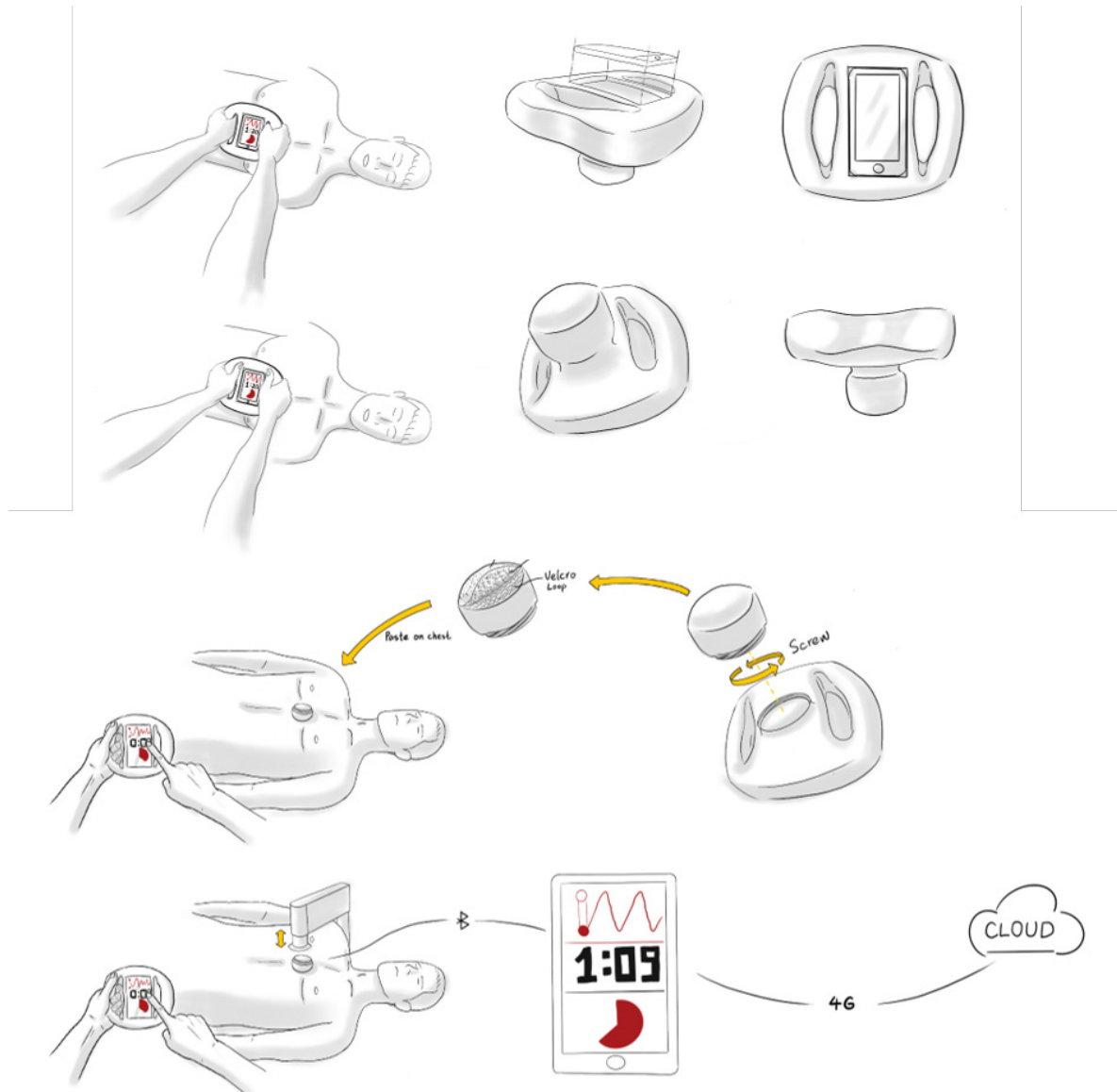


Figure 4.37 User scenarios of concept 1

Concept 2 - Phone

Concept 2 - Phone is based on clay model 2, which is inspired by Philips Q-CPR ⁶⁷. It has an ergonomic shape that fits the rescuer's palm in a compression position. It has an extensive screen to show the status of compression. It saves data to a micro SD card and uploads data to the cloud via Wifi. It is pocket-size and can be recharged.

Due to its compact size, the visual feedback it gives is limited. Uploading data is also dependent on Wifi availability.

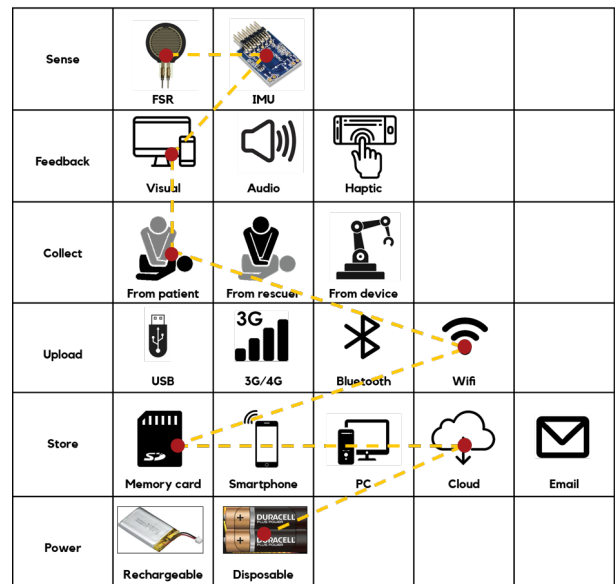


Figure 4.38 Morphological chart of concept 2

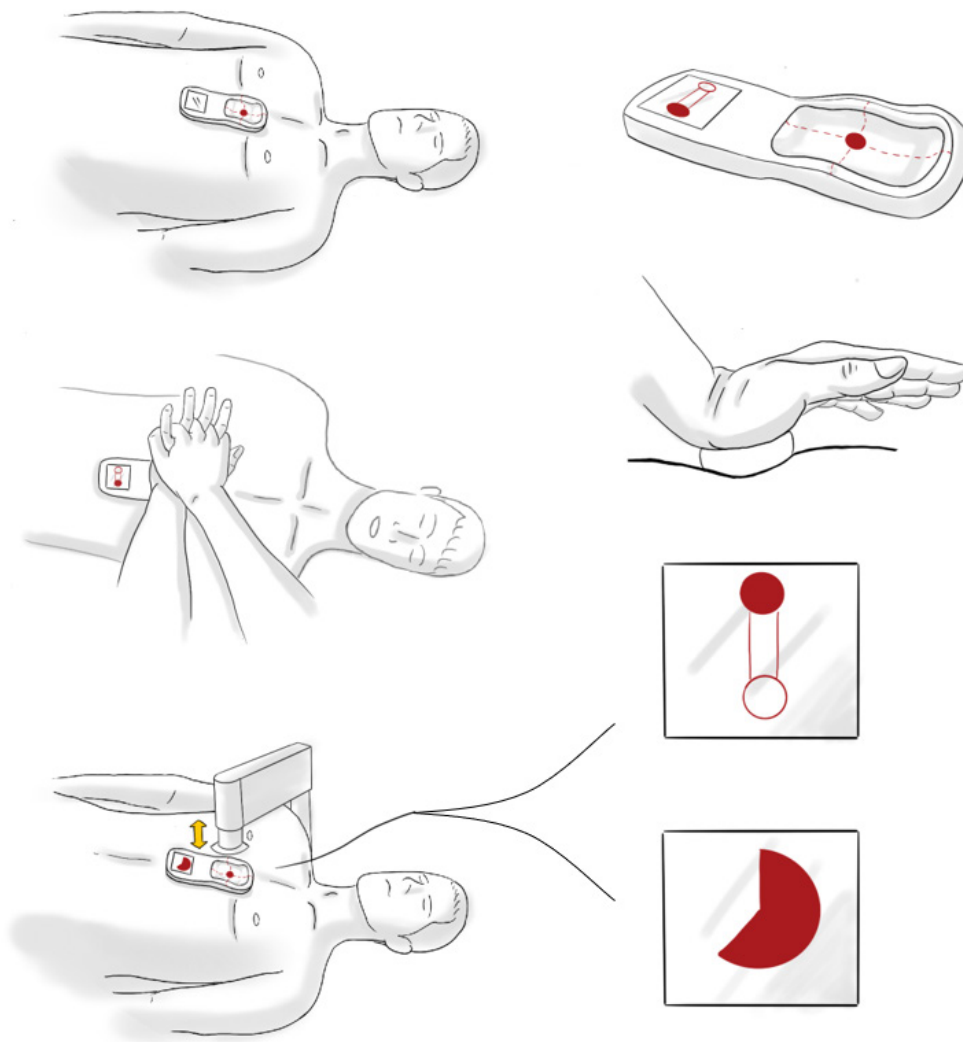


Figure 4.39 User scenarios of concept 2

Concept 3 - Pebble

Concept 3 - Pebble adopts the shape of clay model 3. This is the smallest design of all concepts. It only gives haptic (vibration) feedback to the user in a manual CPR scenario. It saves data in micro SD card and uploads data via Wifi.

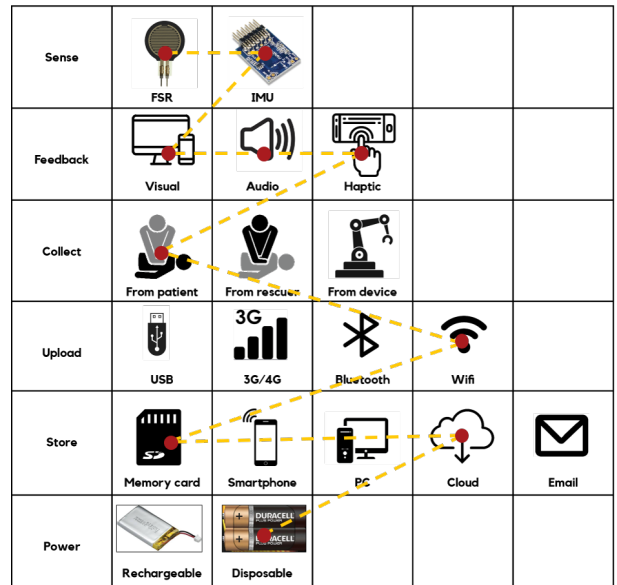


Figure 4.40 Morphological chart of concept 3

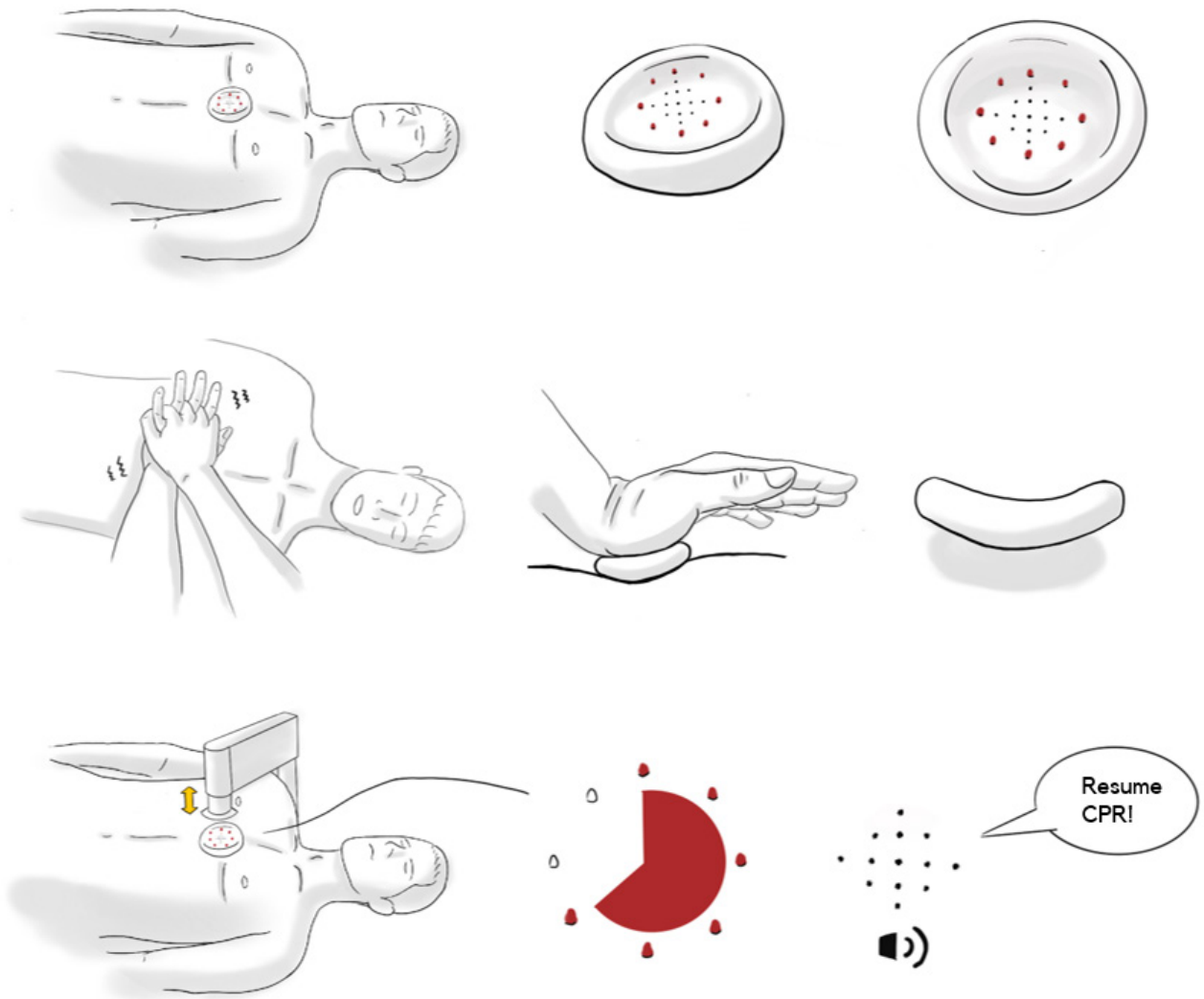


Figure 4.41 User scenarios of concept 3

Concept 4 - Watch

Concept 4 - Watch is a wearable design. According to the study from González-Otero et al. (2018)³⁸, the rescuer's Arm movement tracked by the accelerometer is also accurate to evaluate compression depth and rate. The watch can give audio and visual feedback to the rescuer during chest compression. It can also be placed on patient's chest for mechanical chest compression. It is rechargeable and can be worn by the rescuer during the on-duty period. It saves data to a micro SD card and uploads it to a smartphone via Bluetooth.

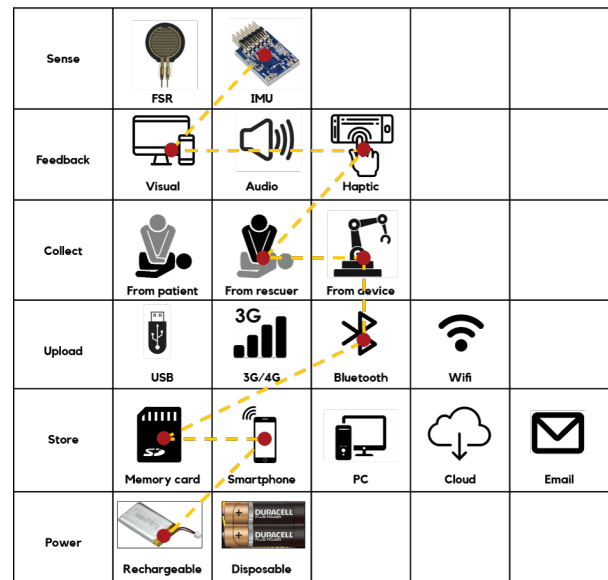


Figure 4.42 Morphological chart of concept 4

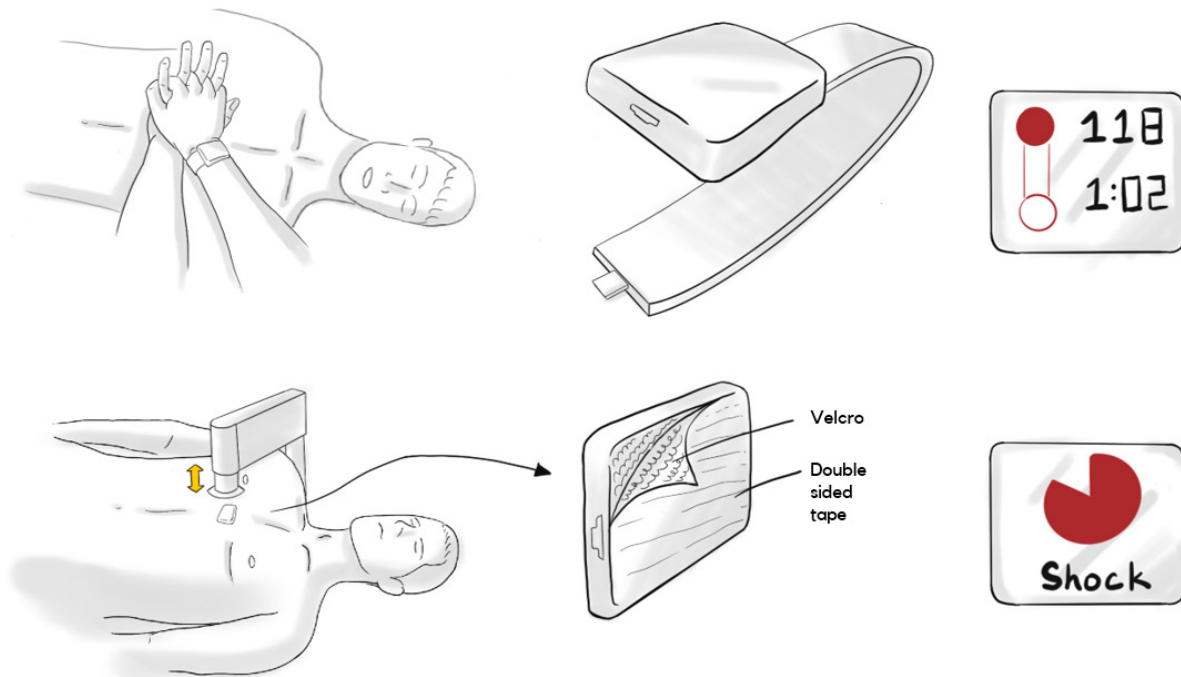


Figure 4.43 User scenarios of concept 4

Concept 5 - CorPatch combination

Concept 5 is generated after the midterm evaluation. During midterm evaluation, it is reflected that the hygiene problem is not well considered in the mentioned 4 concepts. As additional research on hygiene problems, implementing the disposable part into the design is a widely adopted solution by professional medical devices. Thus, Dr.Dinis suggests combining the CorPatch with the data collector design to solve hygiene problems.

Concept 5 demonstrates that the data collector is placed on the rescuer's arm, with a cable connected to the CorPatch, which is placed on the patient's chest. In this way, the data collector stays around the rescuers and also keeps a distance from the patient. It provides visual and audio feedback.

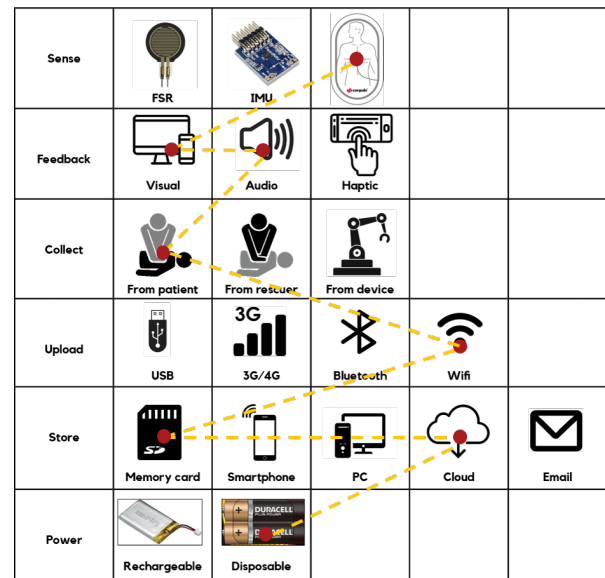


Figure 4.44 Morphological chart of concept 4

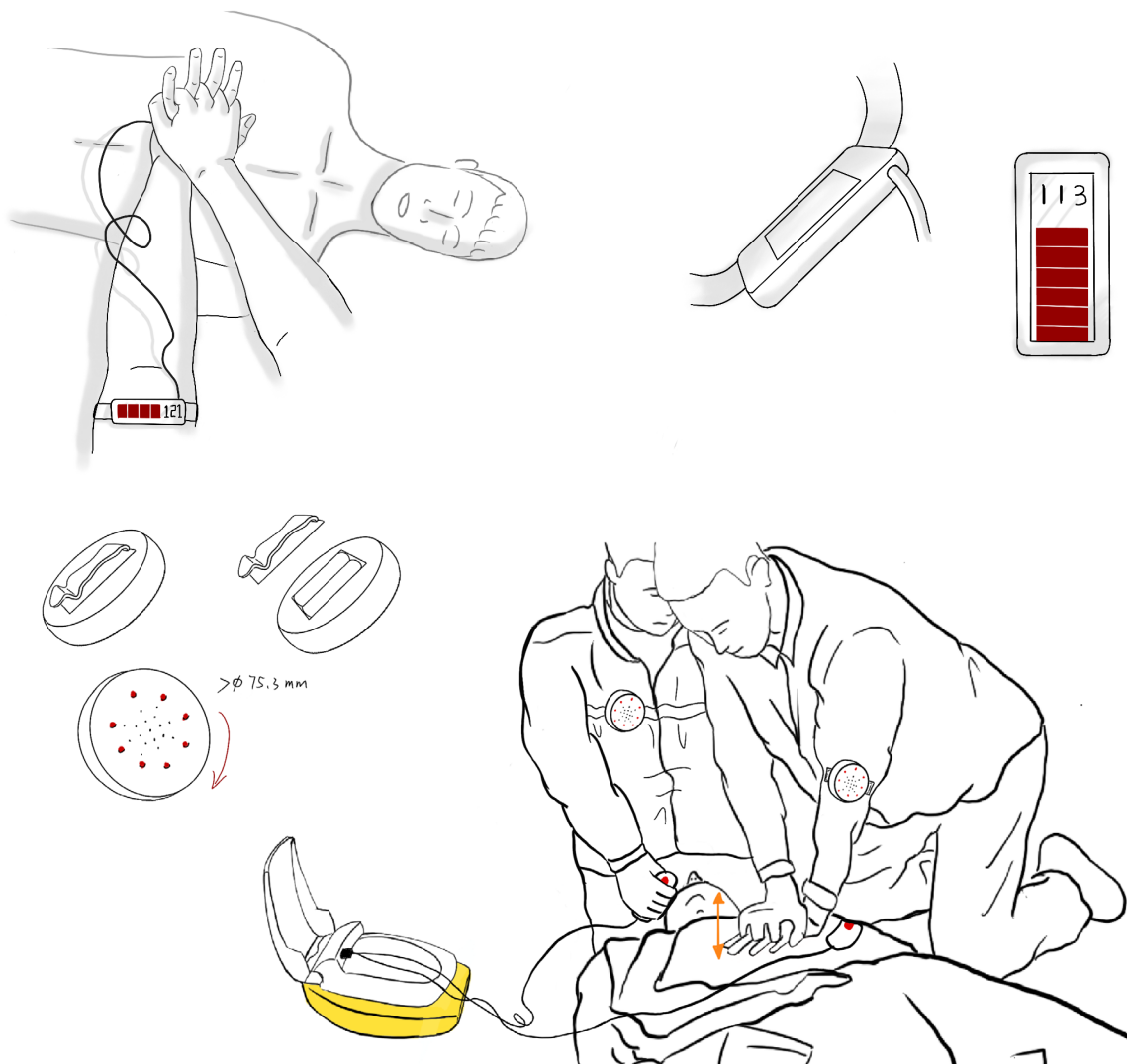


Figure 4.45 User scenarios of concept 5

4.4 Evaluation


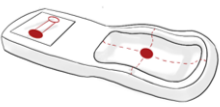
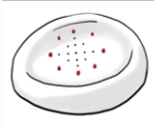

Concept Criteria	 Steer wheel	 Phone	 Pebble	 Watch
Data Collection (x 0.2)	5	5	5	4
Data Transfer (x 0.2)	5	5	5	5
Time Guidance (x 0.15)	5	5	4	5
CPR Feedbacks (x 0.1)	5	4	3	4
Comfort (x 0.1)	5	4	4	3
Feasibility (x 0.05)	3	4	5	3
Total	3.90	3.75	3.55	3.40

Table 4.6 Evaluation table of first 4 concepts

Table 4.6 shows the result of the evaluation of the first 4 concepts before midterm by using weighted objectives method¹⁰⁶. The evaluation criteria are defined based on the result of the MoSCoW prioritization. The steer wheel concept scores the highest among 4. However, as discussed in the midterm meeting, hygiene problems are not well considered with these concepts. Concept 5 is also discussed with Dr.Dinis after midterm that placing data collector on rescuers is also not ideal, as the collector is also connected with the CorPatch, which is fixed on the patient, and this will restrict rescuer's movement.

4.5 Conclusion

Electronics

As demonstrated in the midterm meeting, the Arduino IOT cloud works well with Arduino MKR Wifi 1010, the data from SD card can be all uploaded to the cloud via Wifi. However, the Arduino IOT cloud only keeps uploaded data for a week even for subscribed users (7 USD per month), which means every week the HEMS team should download historic data to their local computers. Plus, the given data storage on the cloud with subscription is 200MB, which is minimal comparing to other platforms such as Google Drive (10GB).

Shapes

The generated shapes and their interaction ways and setups around the user are not ideal, as concluded from the midterm meeting. As for now, what is validated as a feasible idea is using CorPatch as a disposable part and connecting it to the data collector. The data collector therefore just receives output values from the CorPatch, stores them, and uploads them to the cloud via Wifi. However, using CorPatch as the sensor to collect compression data means that the data cannot be used to calculate compression depth, only rates, as explained in the CorPatch section.

Next cycle

The next cycle will be focused on the electronics development and data transfer system building. Because once the electronics setup is defined, the shape of the housing design can follow. The electronics will be built based on the CorPatch. More methods to upload and store data on the cloud platforms will be explored.

05

Cycle 2



In cycle 2, the focus is moved to building electronics prototype and the data transfer system. Extended from the former cycle, a detailed analysis of potentially suited electronics choices is conducted. Cloud platforms are tested with Arduino. At the end of this cycle, the electronic setup for the data collector is validated and a working data transfer system is built.

5.1 Electronics analysis

5.1.1 CorPatch

Circuit

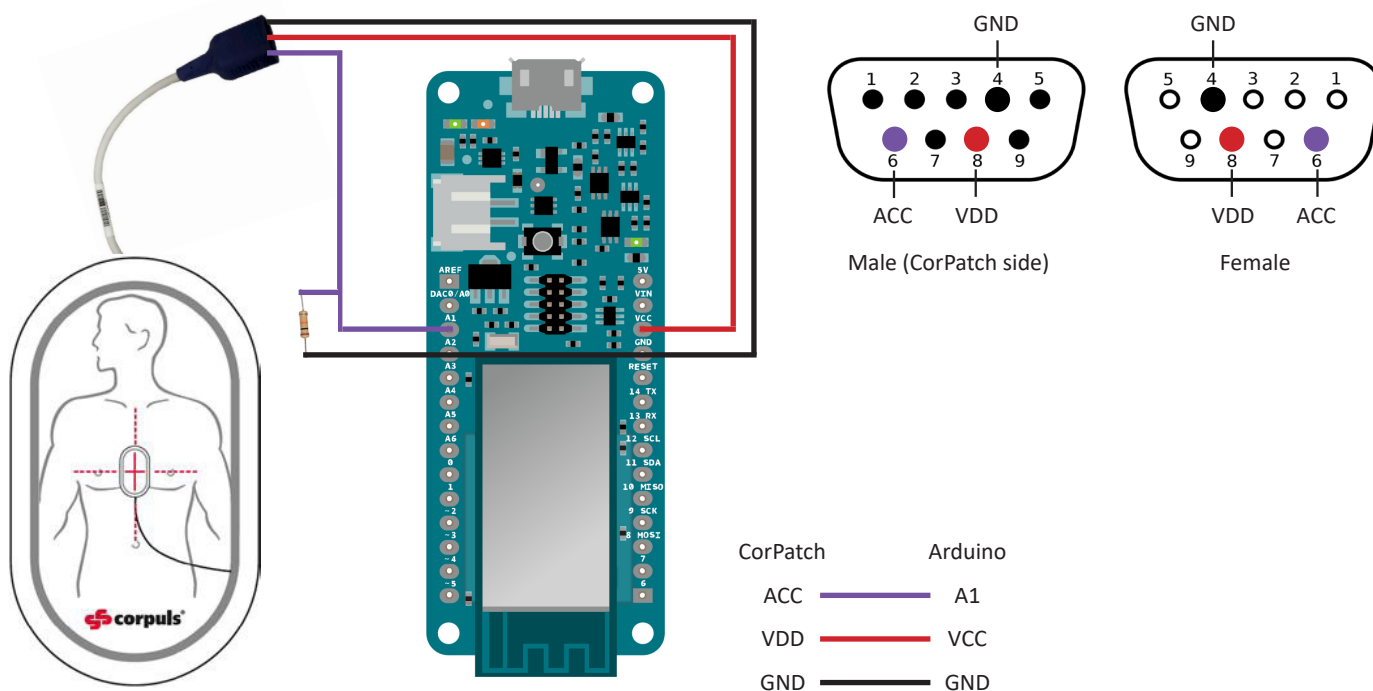


Figure 5.1 CorPatch to Arduino circuit

In cycle 1, CorPatch’s pinout is studied. As shown in figure 5.1, the D-Sub male plug from the CorPatch needs to connect to the GND (ground), VDD (3.3V power supply), and A1(analog read 1) of the Arduino MKR Wifi 1010. The ACC pin of the CorPatch outputs reactive values to its movements and is connected to A1 for the Arduino to read the values. Plus, the ACC pin of the CorPatch is connected in parallel with a 10k Ohm resistor to A1. This resistor in parallel is used to identify if the CorPatch is plugged in or not. When the CorPatch is not connected to the Arduino, A1 gets values below 10 because it reads analog voltage values from the resistor. If CorPatch is plugged in, the value of CorPatch will be over 600, which this threshold will be explained in the next paragraph. If this parallel resistor is not implemented in the circuit, the A1 reading will be random between 0 and 1023 when CorPatch is not plugged in.

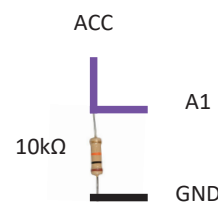


Figure 5.2 Parallel connected resistor

Reading values from ACC

As mentioned in cycle 1, every pin is tested on Arduino with analog read. ACC pin (pin 6) is the only responsive pin to CorPatch movement. Figure 5.3 illustrates the relationship between the ACC pin value and the placement angle of CorPatch when the CorPatch is connected to 5V power supply from Arduino Uno. When the CorPatch lies on the ground (0 degree), the value maintains at around 626 (± 5). At 90 degrees, the value drops to 563 ± 5 . When it is flipped down, the value is at its lowest 503 ± 5 .

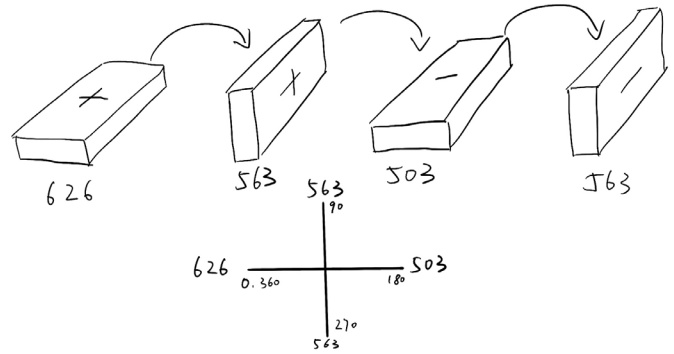


Figure 5.3 ACC pin values in 5V supply regarding to placement angle

But how the value changes in a chest compression scenario? In figure 5.4, the CorPatch is placed on the CPR manikin and tested on Arduino Uno with a 5V power supply. On the right of the figure, a plot shows the CorPatch's readings during the chest compression maneuver. The sample window is 300 milliseconds. At the

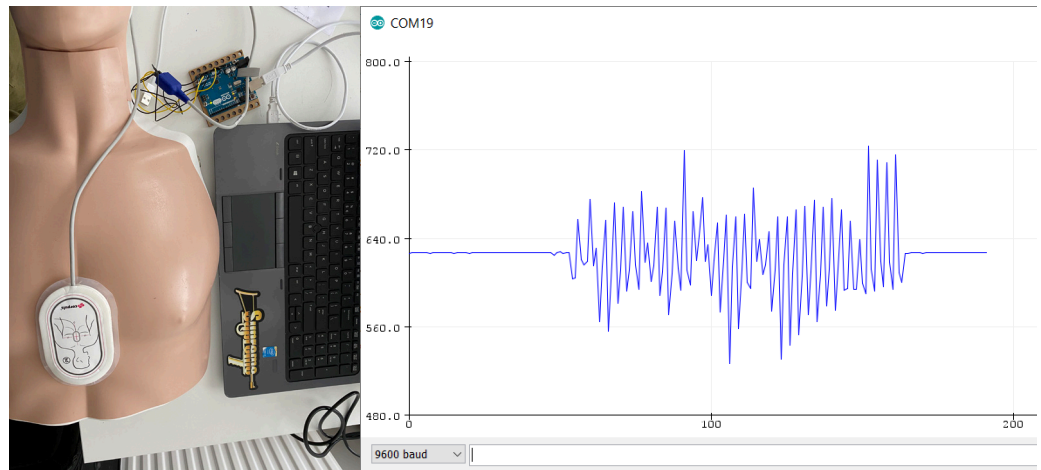


Figure 5.4 ACC pin values plot with 300ms window

beginning of the plot, when the CorPatch is placed on the center chest of the manikin, the value stays at around 626, same as tested before. When the compression starts, the value fluctuates between 720 and 500. The value stays at 626 again when the compression stops. This shows that the minimum value is around 500, same as tested before. The maximum value can go up to around 720. From this plot, it is concluded that the peaks of the value could be used to calculate the compression rate, and the range of the value is fixed.

5.1.2 Micro SD card module

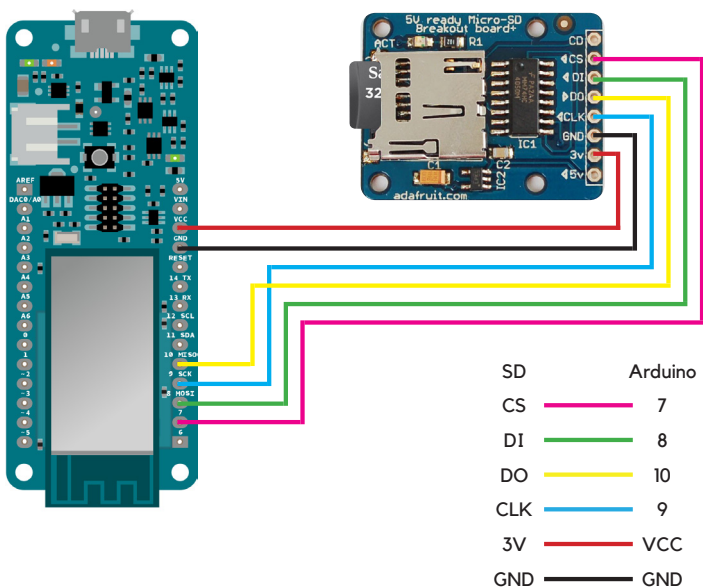


Figure 5.5 Micro SD card module to Arduino circuit

A micro SD card module⁵⁴ bridges the micro SD card and the Arduino. Figure 5.5 shows a circuit that connects the module to an Arduino MKR Wifi 1010. The module communicates with the Arduino via Serial Peripheral Interface (SPI)¹⁰⁴. With this module, Arduino can save the data as CSV (comma-separated values)⁹² file collected from the sensor to an inserted micro SD card, and also read the files from the card.

5.1.3 Real-Time Clock (RTC) module

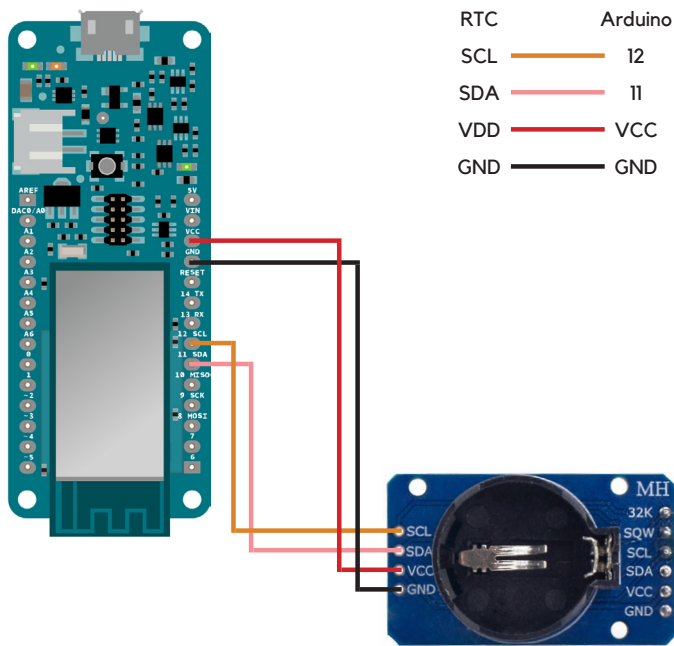


Figure 5.6 Real-Time Clock module to Arduino circuit

As researched in cycle 1, an RTC module⁵⁵ is used to record the current time in seconds. It communicates with the Arduino via Inter-Integrated Circuit (I2C) interface. The RTC module needs a cell battery to power so it can still record the current time when the Arduino board is turned off.

5.1.4 NeoPixel LED ring

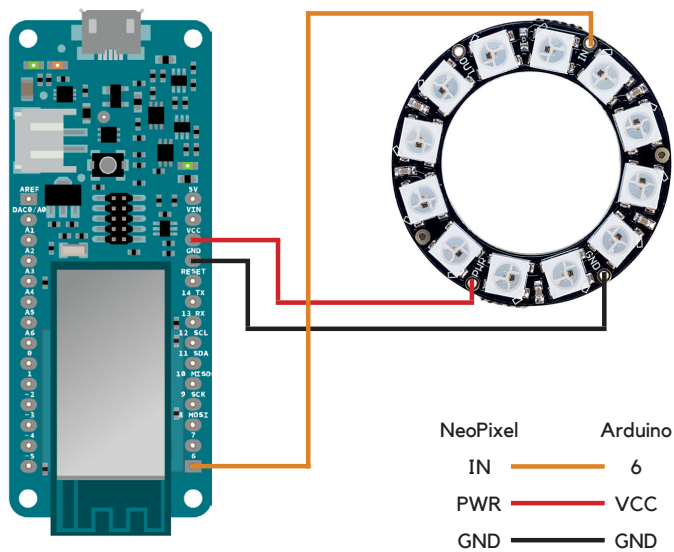


Figure 5.7 NeoPixel ring to Arduino circuit

A NeoPixel LED ring² is chosen to explore the visual feedback of the data collector because it is easy to implement to Arduino comparing to a screen, which normally needs an SPI connection. As shown in figure 5.7, besides regular ground and power connection, one digital pin connecting to the NeoPixel's input is able to control all 12 LEDs on the ring. Figure 5.8 shows the visual feedback given by the NeoPixel in different stages. During the chest compression stage, the NeoPixel blink 2 halves of the ring back and forth in green in 100bpm (beats per minute) to show the correct compression rate to the rescuers. 100bpm compression rate is validated as a suitable rate for chest compression in cycle 1. When the chest compression stops, the NeoPixel switches to 10s countdown, by showing 10 LEDs turning off second by second. After the 10s countdown, if chest compression does not resume, the NeoPixel will start to blink in red to warn the rescuers it is time to resume compression. (Arduino code in appendix G)

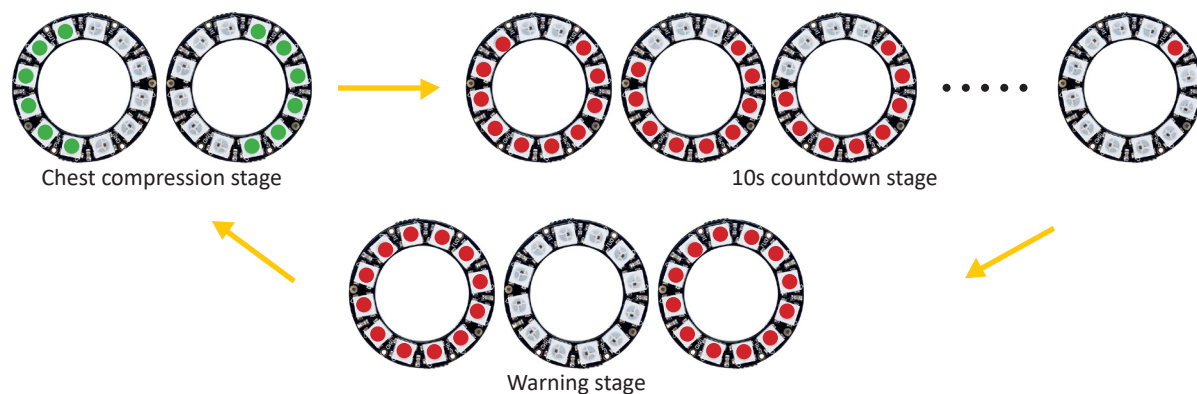


Figure 5.8 Visual feedback in 3 stages

5.1.5 Buzzer

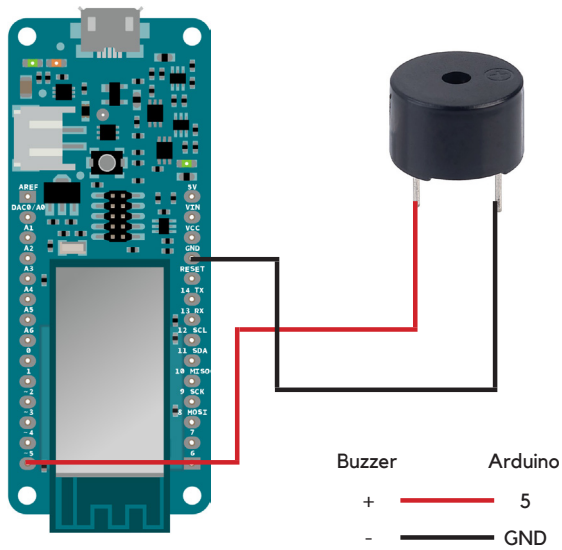


Figure 5.9 Buzzer to Arduino circuit

A piezo buzzer⁸³ is firstly chosen to prototype with Arduino in exploring audio feedback of the data collector. It is simple to implement and it can produce beep sound in different notes (frequencies) and form a melody. In this electronic prototype, the buzzer beeps in note C4 at 100bpm pace during the chest compression stage. In the 10s countdown stage, it beeps in C5 once per second for 10s. In the warning stage, it beeps in C6 on per second. The buzzer produces 3 different tones in 2 different rates to let the rescuers identify which stage they are in now. (Arduino code in appendix G)

5.1.6 Mini speaker + amplifier

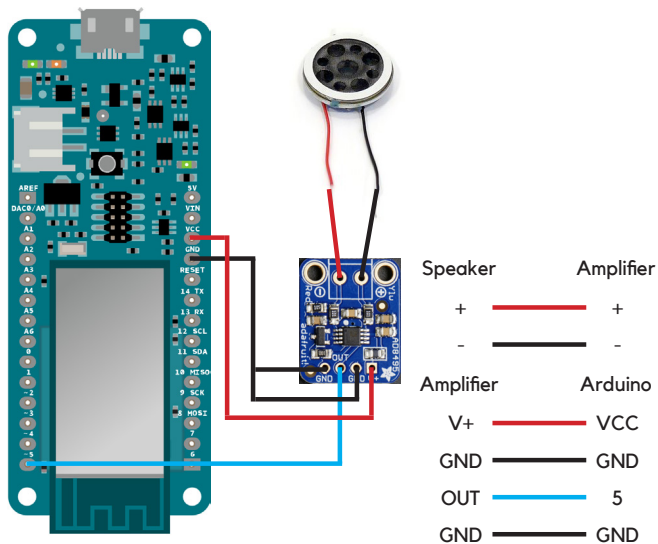


Figure 5.10 Speaker to amplifier to Arduino circuit

Because buzzer only beeps and cannot play audio files. A mini speaker⁸⁶ is chosen for playing audio files on Arduino. An amplifier⁴⁵ is used to amplify the sound of the mini speaker up to around 80 dB (decibels). In the chest compression stage, the speaker beeps in C4 at 100 bpm pace to indicated the ideal compression rate. In the 10s countdown stage, the speaker plays an audio file that counts 10 seconds down in the human voice. This audio file is kept in the micro SD card. In the warning stage, the speaker repeatedly plays “Time to resume CPR!” until compression is resumed. (Arduino code in appendix G)

5.1.7 Battery

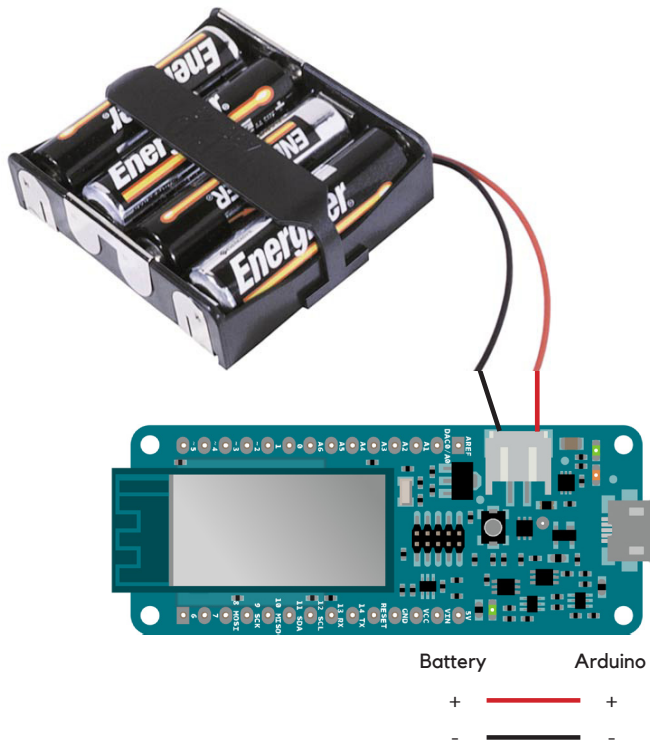


Figure 5.11 Speaker to amplifier to Arduino circuit

According to Arduino ¹⁶, MKR 1010 Wifi requires at least 3.7V power supply. For disposal batteries such as AAA and AA ⁹⁷, each piece provides 1.5V. So in principle, three AAA or AA batteries in series connection can provide 4.5V which should be enough to run MKR 1010. As interviewed with Dr.Dinis, The HEMS team routinely changes the batteries of all devices once a week. This means that the data collector should stand by at least a week and can still be used for OHCA cases, which normally takes one to two hours. The MKR 1010 operates in a 3.3V or 5V circuit. Here 3.3V is chosen because less operation voltage consumes less energy ⁹¹. As the circuit and electronic parts are not defined yet, Arduino Uno in 5V circuit is taken as a reference here, which draws about 50mA ³⁵ when it runs an empty sketch (doing nothing). This means it needs 8400mAh (50mA x 24h x 7d) of battery capacity to maintain working for a week. An AA battery normally holds 2400 - 2500 mAh capacity²⁰ with a compact size (ø7mm x 50mm). Thus, 4 AA batteries in series setup with 9600 to 10000mAh in total is enough for powering the circuit for a week.

5.1.8 Google script app



Figure 5.12 Google App script

In cycle 1, the Arduino IOT cloud platform is used as an online data storage method. But it limits user's storage size and upload times, even with a subscription. In this cycle, during the research on the possibility of combining Arduino IOT cloud and Google sheets in one tutorial ¹⁴, Google Apps Script shows the potential to handle data transfer and pre-analysis with Google sheets. In another tutorial ⁷³, it suggests using an API (Pushingbox⁶ to upload data from Arduino directly to Google sheet. If data can be uploaded and stored directly in Google sheets, the free given storage by the Google drive is 10GB, which is more than enough for this study. However, pushingbox still has limitations on 1000 uploading times per day. From another github page ²⁹, a way to directly send data to Google sheet via Google Script App³² is found. It uses HTTPS ⁸⁹ request from Arduino to upload data to Google sheets, with no API needed. Along with this github example²⁹ and http request example ¹¹ from Arduino, this method becomes feasible and it solves both the storage size and upload times limitations. More details will be explained in the next section.

5.1.9 Conclusion

In this analysis section of cycle 2, different electronic parts for the expected functions of the data collector are explored and prototyped. A new method that can associate Arduino directly with Google sheets is found. The next step is combining these outcomes and testing its feasibility and functions.

5.2 Electronic prototype building

5.2.1 Circuit

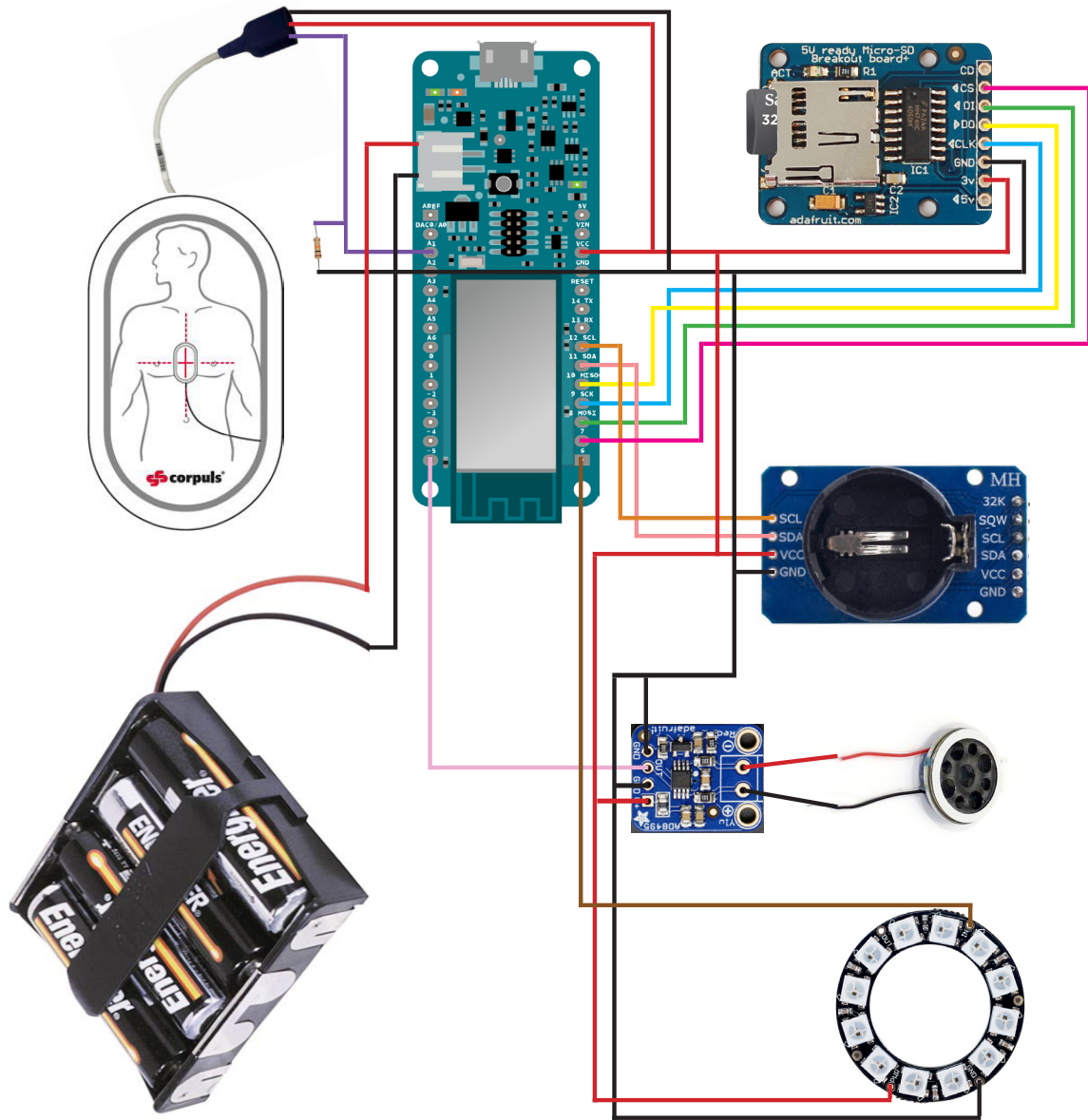


Figure 5.13 Full circuit setup

Figure 5.13 shows the full circuit of the built electronic prototype that includes Arduino MKR 1010 Wifi (processor and Wifi connection), CorPatch (Accelerometer sensor), micro SD card module (data storage and upload), RTC module (real-time record), mini speaker and its amplifier (audio feedback), NeoPixel ring (visual feedback), and battery (power supply). At the bottom is the photo of the whole setup. This prototype not only fulfills the must-have functions (upload and collect data) from the MoSCoW list, but also realizes one should-have (set up within 10s) and some could-have functions (audio and visual feedback).

5.2.2 Code explanation

Overview

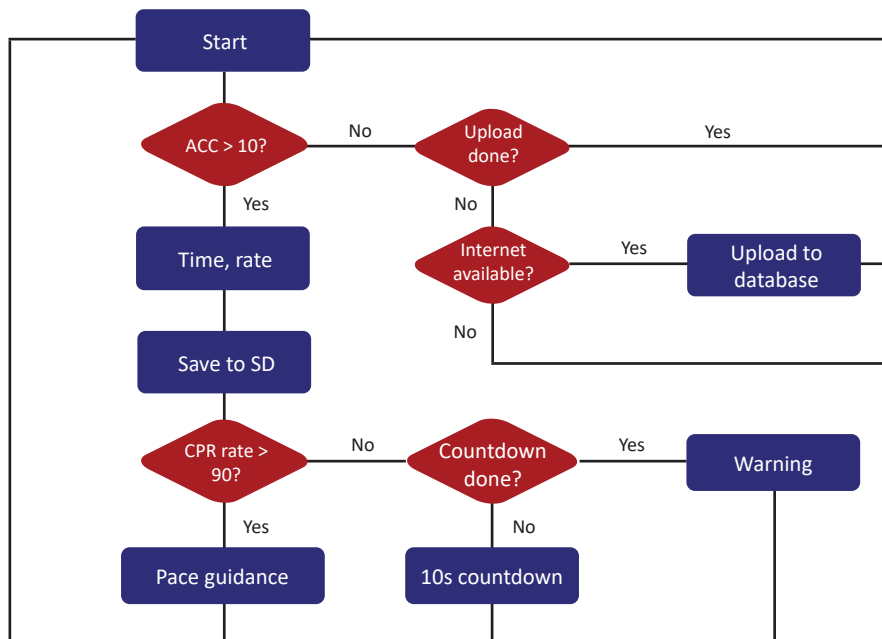


Figure 5.14 Logic flowchart

Figure 5.14 illustrates the logic flow of the coding of the Arduino setup. First, Arduino as a microcontroller⁹³, after being turned on, runs first the setup function once, then loops the loop function until it is turned off. This means Arduino only runs a single thread⁹⁵ operation, and it cannot run multiple functions at the same time. So as shown in the flowchart, after starting the program, there are multiple conditions that need to be defined to execute certain functions, after execution, everything starts from the beginning again. (Full Arduino code in Appendix G)

Start data collection

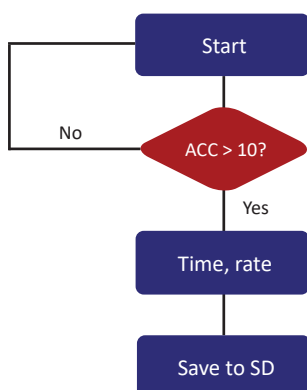


Figure 5.15 Start condition

As mentioned in 5.1, by adding a resistor parallel to the CorPatch ACC pin, when CorPatch is not plugged in, the ACC value is smaller than 10, and the Arduino will do nothing but keeps checking the ACC value. Once the ACC value is over 10, the loop starts.

```

void acceleration() {
  acc = analogRead(accPin);
  t1 = millis();
  t = t1 - t2;
  if (acc > maxThreshold) {
    if (t > interval) { // calculate compression rate every 300ms
      compressionRate = 60000 / t; //so maximal rate is 60s/0.3s=200pushes/min
      t2 = t1;
    }
  }
  else if (t > interval2 && acc < minThreshold) { // if no force sensed longer than 2s, rate is 0
    compressionRate = 0;
  }
}
}
////////////////////////////////////////////////////////////////////ACCELERATION//////////////////////////////////////////////////////////////////

```

Figure 5.16 Calculate the compression rate

As researched, the ACC value changes in reaction to its own motion in a range. Thus, a maximum threshold and a minimum threshold is set based on this range. For every 300 milliseconds, when the value exceeds the maximum threshold, it counts as a valid compression and its corresponding time is recorded. Then if another valid compression time is also recorded, the in-between period is a subtraction between this two timestamps. This period is then divided by 60 (seconds) to calculate the current compression rate. If the ACC value is smaller than the minimum threshold for more than 2 seconds, the compression rate is set to be 0.

```

void currtime()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
  dateString = "20" + String(year, DEC) + "-" + monString + "-" + dayString;
  hourString = String(hour, DEC);
  timeString = hourString + ":" + minString + ":" + secString;
}

```

Figure 5.17 Calculate current time

Current time data is calculated by a function provided by an RTC library. It is then formatted and combined to one string of text.

```

void savedata() {
  if (filenamed == false) {
    sheetname = monString + dayString + hourString + minString + ".csv";
    filenamed = true;
    sdfile = SD.open(sheetname, FILE_WRITE);
    if (sdfile) {
      sdfile.println(title);
      sdfile.close();
    }
    else {
      Serial.println("fail to write");
    }
  }
  dateString = dateString + "," + timeString + "," + String(acc) + "," + String(compressionRate) + ","; // convert to CSV
  sensorData = SD.open(sheetname, FILE_WRITE);
  if (sensorData) {
    if (p < 99) { //100lines
      sensorData.print(dataString);
      p++;
    }
    else {
      sensorData.println(dataString);
      p = 0;
    }
  }
  sensorData.close();
}
else {
  Serial.println("fail to write");
}
}

```

Figure 5.18 Save data to SD

The compression rate and the current time is calculated per second. After the first calculation, the Arduino will create a new CSV file named as the first recorded timestamp in the SD card. Then all the data will be combined into one string line and wrote to the CSV file. Incoming data lines will be added to the CSV file from then on until the CorPatch is unplugged, and in the end, an "END" string will be added to the last line of the file as a remark. In the CSV file, per 100-line of incoming data (100s of data) form a row in the CSV file in order to upload the data to Google sheets in a 100-line batch per time. This speeds up the uploading process significantly.

Figure 5.19 CSV file for uploading and backup

The top of figure 5.19 shows the CSV file created for uploading the data. As it shows, it combines 100 lines of recorded data in one row for later faster uploading to Google sheets. On the other hand, the bottom shows a backup CSV file that writes the collected data row by row. With specification of date, timestamp, acceleration, and rate in each column. This CSV file is named CPR_DATA and every recorded data is also saved in here, in case if the upload process does not succeed, the SD card still has the complete data listed in a formatted way.

Pace guidance

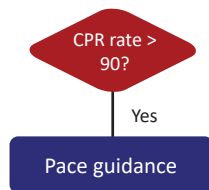


Figure 5.20 Pace guidance

After data is saved on SD card, if the calculated chest compression rate is over 90, which is considered as a suitable rate (research in cycle 1), the pace guidance begins. The pace guidance includes audio feedback that the speaker beeps at 100bpm, and visual feedback that the NeoPixel blinks in green, as discussed in section 5.1.

```

void compress() {
  unsigned long current = millis();
  if ((state == HIGH) && (current - previous) > onTime) {
    state = LOW;
    previous = current;
    if (state2 == LOW) {
      state2 = HIGH;
      for (int i = 0; i < 6; i++) { // For each pixel...
        pixels.setPixelColor(i, pixels.Color(0, 150, 0));
      }
      for (int i = 6; i < 12; i++) { // For each pixel...
        pixels.setPixelColor(i, pixels.Color(0, 0, 0));
      }
    }
    else {
      state2 = LOW;
      for (int i = 0; i < 6; i++) { // For each pixel...
        pixels.setPixelColor(i, pixels.Color(0, 0, 0));
      }
      for (int i = 6; i < 12; i++) { // For each pixel...
        pixels.setPixelColor(i, pixels.Color(0, 150, 0));
      }
    }
    AudioZero.begin(16000);
    audioFile = SD.open("compress.wav");
    pixels.show(); // Send the updated pixel colors to the hardware.
    AudioZero.play(audioFile);
  }
  else if ((state == LOW) && (current - previous) > offTime) {
    state = HIGH;
    previous = current;
    for (int i = 0; i < 12; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(0, 0, 0));
    }
    pixels.show();
    //AudioZero.end();
  }
}
  
```

Figure 5.21 Pace guidance code

Figure 5.21 shows the code that realizes the audio and visual feedback of the pace guidance. The LEDs are first set to green and blinks one half on while one half off for 500 milliseconds, then switch side. The audio file "compress.wav" plays a metronome sound in 100bpm for audio guidance.

Countdown and warning

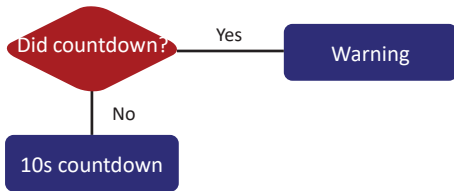


Figure 5.22 10s countdown and warning

```

void countdown() {
  unsigned long current = millis();
  if (j > 1) {
    if ((state == LOW) && (current - previous) > onTime2) {
      state = HIGH;
      previous = current;
      for (int i = 1; i < j; i++) {
        pixels.setPixelColor(i, pixels.Color(150, 0, 0));
      }
      if (j == 10 or j == 9) {
        AudioZero.begin(16000);
        audioFile = SD.open("resu.wav");
        AudioZero.play(audioFile);
      }
      else if (j == 8 or j == 7) {
        AudioZero.begin(16000);
        audioFile = SD.open("87.wav");
        AudioZero.play(audioFile);
      }
      else if (j == 6 or j == 5) {
        AudioZero.begin(16000);
        audioFile = SD.open("65.wav");
        AudioZero.play(audioFile);
      }
      else if (j == 4 or j == 3) {
        AudioZero.begin(16000);
        audioFile = SD.open("43.wav");
        AudioZero.play(audioFile);
      }
      else {
        AudioZero.begin(16000);
        audioFile = SD.open("21.wav");
        AudioZero.play(audioFile);
      }
    }
    pixels.show();
    j = j - 1;
  }
}
  
```

Figure 5.23 10s countdown code

If the compression rate is smaller than 90, which shows a lack of compression, the Arduino first checks the state of counting down. If counting down is not conducted yet, it starts to play the count down audio file: “CPR resumes in 8, 7 1”. After the countdown, if the rate is still not over 90, warning in “Resume CPR now!” will loop.

Because audio file lasts for 10 seconds, which interrupts the loop. If the compression is already resumed within 10 seconds, the program still needs to wait for the audio file to finish to move on. To solve this interruption, the audio file is cut into 5 pieces with 2 seconds each. So 2 seconds of audio is played, the condition of compression rate is rechecked, if the rate goes back to over 90 again, the countdown will stop and the pace guidance will start immediately.

Data upload

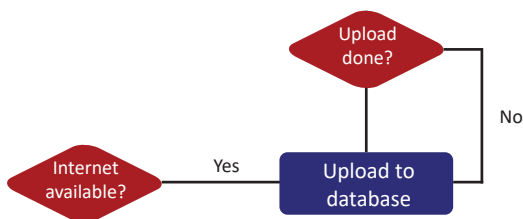


Figure 5.24 Data upload

After one session ends, Arduino starts to find available Wifi to connect to the Internet until it succeeds. After connecting the Wifi, it starts to upload the whole newly created CSV file in batches to Google sheets. When it finishes uploading, it goes back to the standby mode of checking ACC value.


```

void readdata() {
  sensorData = SD.open(sheetname);
  if (sensorData) {
    while (sensorData.available()) {
      CPRdata = sensorData.readStringUntil('\n');
      Serial.println(CPRdata);
      if (client.connectSSL(server, 443)) {
        Serial.println("connected to server");
        client.println("GET https://script.google.com/macros/s/AKfycbw2VNula-d2qICRNFSq0CgYL3DwjHHyvu9rY4PlPYy9038aIQ0_exec?CPRdata=" + CPRdata);
        client.println("Host: script.google.com");
        client.println("Connection: close");
        client.println();
      }
      if (CPRdata == "END") {
        uploaded = true;
        //LowPower.sleep();
        break;
      }
    }
  } else Serial.println("fail");
}
////////////////////////////////////SD////////////////////////////////////

```

Figure 5.25 Data upload code

Figure 5.25 shows the code that uploads the data to Google sheets directly. Arduino first opens the CSV file and retract the first row of data, which includes 100 lines of data. Then it assigns the data as one string to a variable CPRdata as highlighted in the blue frame. After that, Arduino tries to establish a connection to the server and sends a request with an Http address. This Http address contains a specific ID of the Google sheet as highlighted in the yellow frame. At the end of the address, the CPRdata variable which contains the data is added. As this Http address is requested, the Google Script App will read through it and add all the data that the CPRdata variable brings to the sheet.

5.2.3 Conclusion

The electronic prototype realizes must-have functions (data collection and upload), and offers an option for the could-have function (audio and visual feedback). It will be further tested with the HEMS team to check its feasibility.

5.3 Data transfer system building

5.3.1 Google sheet

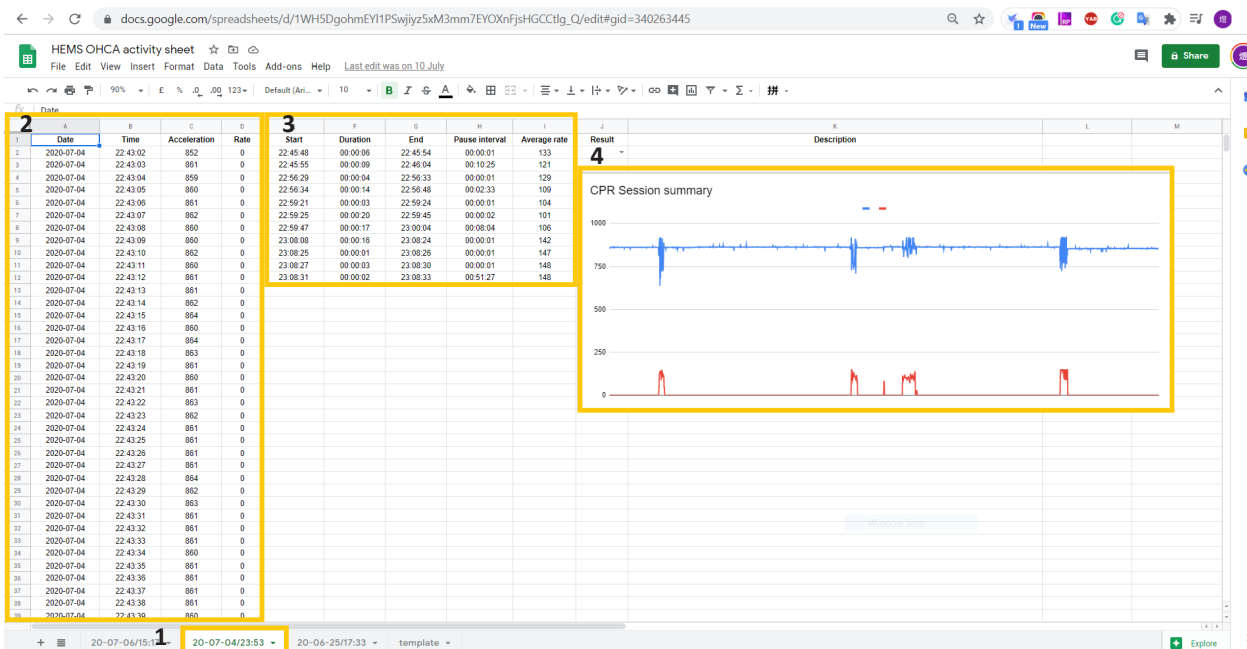


Figure 5.26 Google sheets overview

Figure 5.26 shows how the Google sheet looks like after the data is uploaded. As mentioned in the last section, the Arduino requests the Google sheet's Http address to upload the data. When it starts request, the Google Script App name a new sheet with the current time, as shown in step 1 in the figure. Then the incoming data is formatted as step 2 shows. When the last line of the data which is an "END" text is uploaded, the Script App starts to read the data line by line and calculate the start time, duration, end time, pause interval, and average rate in step 3. At last, a CPR summary plot is generated in step 4. This plot shows the changes in the ACC value and the compression rate throughout the whole OHCA session. (Full code in Appendix H)

5.3.2 Code explanation

How to automate this whole process? Google Script App provides the access to manipulate the Google sheets with a piece of code running in the background. In this section, the Javascript ⁴⁶ code that runs in the Script App is explained. But first, the whole process of the data transfer needs to be explained:

First, Arduino saves the data as a variable CPRdata as formatted:

"date, time, ACC, rate,date,time, ACC,rate,..... date, time, ACC, rate, /n" with 100 lines of date, time, ACC and rate combined in one row. So the CPRdata variable looks like this in Arduino:

CPRdata = "2020-08-06,12:53:21, 907, 98, 2020-08-06, 12:53:22, 558, 95,..... 2020-08-06, 13:15:06, 670, 0, /n"

Then Arduino tries to access following Https ⁸⁹ URL ¹⁰⁰ with specific script ID and CPRdata at the end:

[https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPy9038aIQ0/exec?CPRdata="](https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPy9038aIQ0/exec?CPRdata=) + CPRdata

Combined together, the Https URL would be like:

[https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPy9038aIQ0/exec?CPRdata=2020-08-06,12:53:21, 907, 98, 2020-08-06, 12:53:22, 558, 95,..... 2020-08-06, 13:15:06, 670, 0,](https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPy9038aIQ0/exec?CPRdata=2020-08-06,12:53:21, 907, 98, 2020-08-06, 12:53:22, 558, 95,..... 2020-08-06, 13:15:06, 670, 0)

The Google Script App identify the Https request based on the ID in it:

“AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHYvu9rY4PIPYy9038aIQ0”

This ID represents the specific script that is being used to associate with Google sheet at the moment.

The linked Google sheet that should be manipulated is identified in the first line of the code in the script. This ID is retracted from the Google sheet’s own Https URL, as shown in figure 5.27.

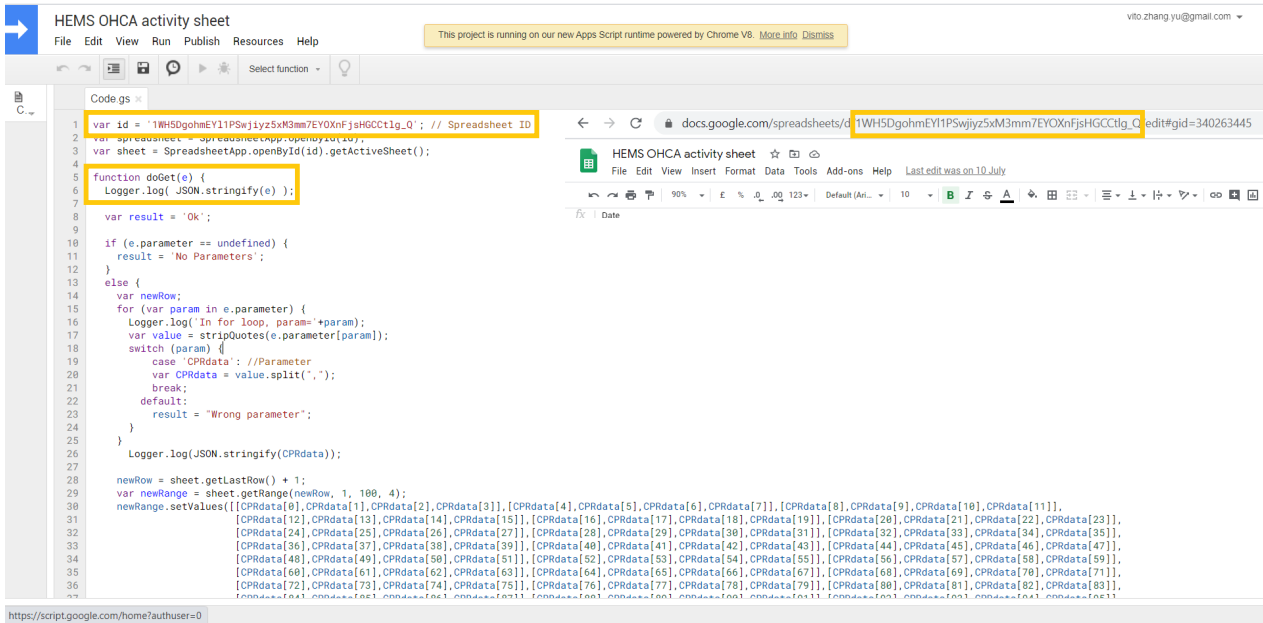


Figure 5.27 Google Script App

After verification, the script retracts all the blue part after “CPRdata=” from the Http address with the function doGet(e) in line 5 of figure 5.27:

<https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHYvu9rY4PIPYy9038aIQ0/exec?CPRdata=2020-08-06,12:53:21,907,98,2020-08-06,12:53:22,558,95,.....2020-08-06,13:15:06,670,0>,

So the 100 lines of data has successfully been transferred to the script as a string variable ‘e’.

After that, as shown in lines 15 to line 23, the variable ‘e’ that holds the whole data is split by comma and assigned into a variable CPRdata. This CPRdata variable is an array variable. Then the CPRdata array will be assigned to the sheet’s cells in batches:

CPRdata[0]=2020-08-06, CPRdata[1]= 12:53:21, CPRdata[2]= 907, CPRdata[3]= 98

	A		C	D
1	Date	Time	Acceleration	Rate
2	2020-07-04	22:43:02	852	0
3	2020-07-04	22:43:03	861	0
4	2020-07-04	22:43:04	859	0
5	2020-07-04	22:43:05	860	0
6	2020-07-04	22:43:06	861	0
7	2020-07-04	22:43:07	862	0
8	2020-07-04	22:43:08	860	0
9	2020-07-04	22:43:09	860	0
10	2020-07-04	22:43:10	862	0
11	2020-07-04	22:43:11	860	0

Figure 5.28 Fill cells

As shown in line 30 to 35, the 100 lines of data is split into 400 pieces because each line contains 4 pieces of cell data (date, time, ACC, rate). All these 400 pieces are assigned to the CPRdata array as mentioned. To fill in all 400 pieces in each of their own cells as shown in figure 5.28, a loop function will take too long to finish. Fortunately, Google sheet can set values to multiple cells all at once. So In the code, by manually setting values to each cell rather than using loops, the upload speed is 20 times faster (using loops to add each row per loop takes about 5 seconds, now 100 rows are added in 5 seconds).


```

function setName(){
//var name = JSON.stringify(getDate()).concat(JSON.stringify(getHours())).concat(JSON.stringify(getMinutes()));
var name = Utilities.formatDate(new Date(), "CET", "yy-MM-dd/HH:mm");
sheet.setName(name);
}

function newChart() {
// Generate a chart representing the data in the range of A1:B15.
// var ss = SpreadsheetApp.getActiveSpreadsheet();
// var sheet = ss.getActiveSheet();
var lastRow = sheet.getLastRow()-2;
var chart = sheet.newChart()
.setChartType(Charts.ChartType.LINE)
.addRange(sheet.getRange(2,3,lastRow,2))
.setPosition(4, 10, 2, 2)
.setOption('title', 'CPR Session summary')
.setOption('width', 1000)
.setOption('height', 400);
sheet.insertChart(chart.build());
}

```

Figure 5.31 Set sheet name and generate plot

Two functions are presented in figure 5.31. The setName() function is for step 1 to set the sheet name to the current time. When the HEMS team tries to find historic data, they can identify the sheets by the uploading time. Function newChart() is for step 4 to generate the plot of the ACC value and compression rate during the whole session.

```

function newSheet(){
var template = spreadsheet.getSheetByName('template');
var options = {template: template};
spreadsheet.insertSheet(0,options);
}

function sendEmail () {
var numSheets = spreadsheet.getSheets().length;
if (numSheets == 200){
DriveApp.getFileById(spreadsheet.getId()).makeCopy();
var address = 'vito.zhang.yu@hotmail.com';
var subject = 'Data of 199 CPR sessions are collected';
var body = 'Hi Dinis, the data of 199 CPR sessions are collected and saved in the HEMS folder';
MailApp.sendEmail(address,subject,body);
for (i = 0;i<199; i++){
spreadsheet.deleteActiveSheet(); //delete all tabs except template tab
}
}
}

```

Figure 5.32 Create new sample sheet and send Email

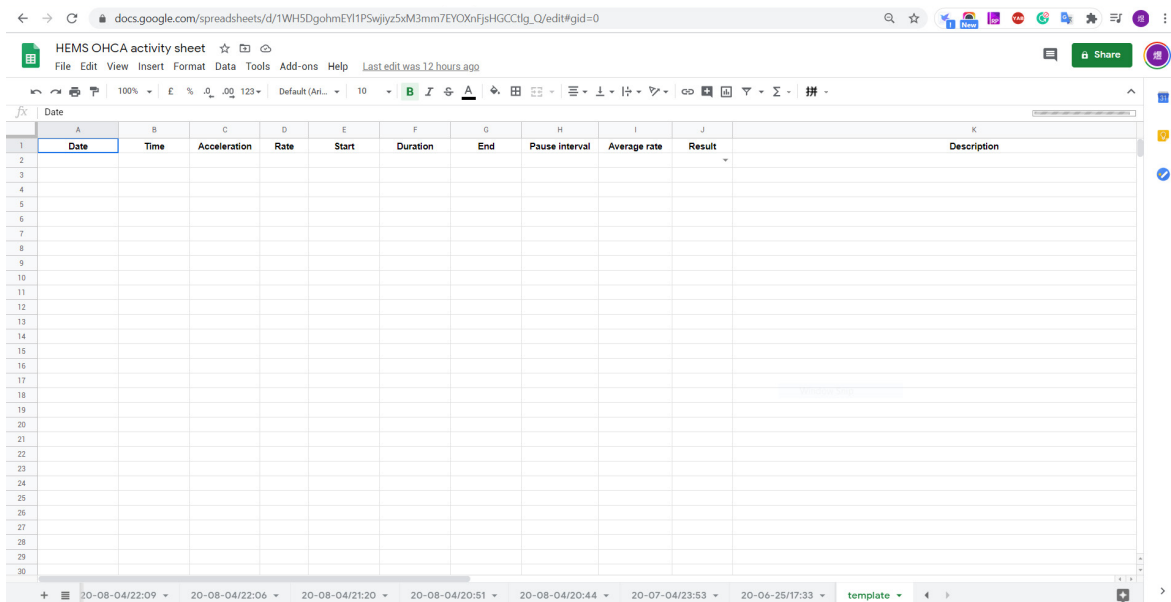


Figure 5.33 Template sheet

In figure 5.32, a `newSheet()` function is called after the whole steps of the current sheet is finished, a new template sheet is generated and ready for the next upload. Figure 5.33 shows the sample sheet, it is set as 9999 rows which can contain 9999 seconds of data. This size is big enough to hold data generated by a 2.5-hour OHCA case. Also in the template sheet, all titles are formatted.

In figure 5.32, a `sendEmail()` function is written to prevent potential system collapse. There are certain limitations for the Google sheets ⁷⁴, one of them is that one spreadsheet holds maximal 200 sheets, which means this spreadsheet can be used for up to 199 OHCA cases (the last one is for the template sheet). As mentioned in former paragraphs, the script only associates with the defined Google sheet ID in the code. If the current spreadsheet is full and a new spreadsheet is created for future cases, a new script that associates with it needs to be created too, and the ID from the Http address that Arduino requests also needs to be changed. In order to avoid all of these renewal processes, those 199 sheets in the current spreadsheet should be deleted so another round of collection can start over in the same spreadsheet.

In the program, when 199 OHCA cases are uploaded to the spreadsheet, this whole spreadsheet would be copied inside the same Google Drive account. Then an email will be sent to the HEMS team informing them that the 199 OHCA cases have been collected. At last, these 199 OHCA cases are deleted so this spreadsheet is ready for more cases.

5.3.3 Conclusion

The data transfer system is built based on using Google Script App as a bridge to connect the Arduino and Google sheets. It offers 10GB free storage and requires no in-between API. It can analyze the uploaded data and visualize them automatically. It also arranges the storage of the data. The users can access and work on their data directly in Google sheets. Overall, this system fulfills the must-have functions and also brings extra functions like pre-analysis. It needs to be tested in the HEMS station to see if the networks there suits the system.

5.4 Evaluation

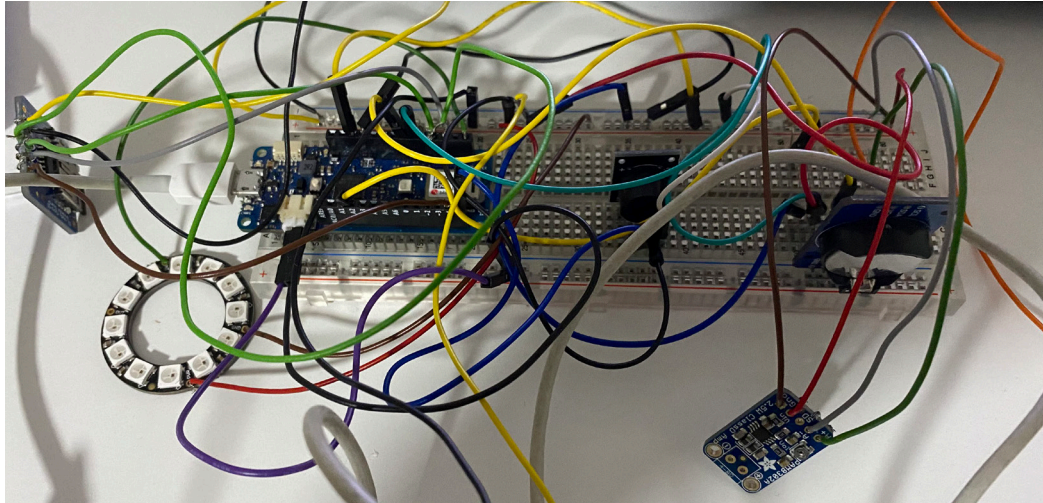


Figure 5.34 Electronic prototype

Figure 5.34 shows the electronic prototype built based on Arduino MKR 1010 Wifi. This prototype is built on a breadboard and it connects to the CorPatch via female jumpers. In this section, multiple tests are conducted to evaluate the electronic setup and the data transfer system.

5.4.1 Tests on manikin

CorPatch placement test



Figure 5.35 4 placements of CorPatch on manikin

During weekly feedback with Dr.Dinis, it is mentioned that in some cases the CorPatch may not be able to be placed on the center chest of the patient as it should be, due to that the patient might have wound on his chest. So it is necessary to test if the ACC value from CorPatch is still valid to track motion if it is placed on other parts of the body during chest compression. Here a CPR training manikin is used to test with the CorPatch. As shown in figure 5.35, the CorPatch is placed in different areas around the center chest in 4 tests, and the tester compress the chest with no CorPatch under his palm. The result is that the CorPatch can still output values that are reactive to the motion. As studied in section 5.1, the ACC value is reactive to the angle of placement of the CorPatch. When the CorPatch is placed around the center chest during compression, it still have placement angle changes due to the deformation of the center chest. Thus, as long as the placement of CorPatch is around the center chest area, its output values should be valid for later calculation.

Uploading speed test

```

2020-08-04,20:50:43,860,0,
2020-08-04,20:50:44,860,0,
2020-08-04,20:50:45,860,0,
2020-08-04,20:50:46,861,0,
Connecting to wifi...
Connecting to wifi...
Connected to wifi
2020-08-04,20:48:04,738,0,2
connected to server
2020-08-04,20:49:44,859,0,2
connected to server
END
connected to server
2
2
2

```

Figure 5.36 Serial COM of Arduino during test

Originally the Arduino and Google sheets are programmed to upload data line by line, but during testing, it shows that the averagely it takes 5 seconds for the Google sheet to update one Http request from the Arduino. It means that for one second of data (one line) it needs 5 seconds to upload to the Google sheets if the line by line uploading method is used. To speed up the uploading speed, the 100-line batch upload method is introduced as mentioned in the last section. This speed test is conducted to check how much time is used at each phase of uploading during the whole process. In figure 5.36, the serial COM of Arduino is used to inform the tester what mission the Arduino is on at the moment, and the tester is holding a timer to mark timestamps for each uploading phase.

There are 3 phases during the whole process: 1. Connecting to Wifi, 2. Uploading to Google sheets, 3. Pre-analysis in Google sheet.

Data size	1834s	3653s
Wifi connection	4.07s	4.26s
Upload to Google	73.05s	143.79s
Pre-analysis	209.63s	463.02s

Figure 5.37 Speed tests result (Appendix J)

Two tests are conducted. The first test collects in total 1834 seconds of data, around half an hour. The second one collects 3653 seconds of data. Table 5.1 shows the result of the tests. It takes around 4 seconds for the Arduino to connect to a WPA2⁹⁹ protected Wifi network. It uploads averagely 25 lines/seconds of data per second to the Google sheets, which is 125 times faster than the line-by-line method. For the pre-analysis phase, it handles around 8 lines/seconds of data per second.

Actually a third test with a 2-hours of data is also conducted but unfortunately, the original record is lost. The test result is close to the former 2 during the Wifi-connection and Upload to Google phase, but one problem is discovered during the pre-analysis. As calculated in the last paragraph, a 2-hours (7200s) of data could cost the script 900 seconds (7200/8) to go through. However, this time the pre-analysis is not finished and the system also stopped. After digging in, the main reason is found: Google limits the Script App to execute a function for no more than 6 minutes⁶⁸ and the setValue() function needs costs 15 minutes (900 seconds) to go through every line of the data and calculate the values. The reason that the setValue() function runs so slow in the loop is not only because the amount of data it needs to handle, but also because within one loop, setValue() and getValue() function are called at least once for each, and these two functions require service calls to Google App, which takes a lot of time to execute. A possible solution⁴⁰ is using an array to assign values all in one batch. This would make the loop goes much faster and possibly finish the loop within 6 minutes.

5.4.2 Tests with HEMS team

Electronic prototype

The electronic prototype is also shown to the HEMS team for their opinions.

First, they like that the device does not require any action to turn it on. All they need to do is plug the CorPatch in, and the device just starts to collect data from then on until the CorPatch is unplugged.

For the audio feedback, the main problem is that the sound is not loud enough. Even the mini speaker is applied with an amplifier, it can only go up to 75dB, which is not enough to draw the rescuer's attention during the real scene. For the audio itself, they prefer a human voice rather than a buzzer with tones.

For the visual feedback, they find the NeoPixel shows the countdown and warning notification in a universally understandable way. For the pace guidance part, they think it is also understandable if being taught before use. However, they are not sure if every rescuer will notice or look at the NeoPixel ring in the emergency scenes, as they also need to pay attention to the defibrillator and the mechanical chest compressor. Plus, they are not sure this notification send out by NeoPixel can be understandable with other different rescue teams like the ambulance team.

Data transfer system

The workflow of how Arduino sends data directly to Google sheets is also shown to the team.

At the very beginning of the test, one problem is revealed when Arduino tries to connect to the station's Wifi. In the station, there are three Wifi networks: the Erasmus MC, the Eduroam, and one for the Google Map tracking. The first two are using WPA2 enterprise protection, which asks for a user name and password to log in. A personal WPA2 network requires only the Wifi's password. The electronic prototype can only connect to a personal WPA2 Wifi signal at the moment. The last Wifi network for Google Map is built for emergency lines and requires permission to use. In the end, the data transfer system is shown with connection to a hotspot from a smartphone.

After the test, they like that they can directly access the data in Google sheets, which is a simple and familiar platform. Plus, Google Drive provides enough storage (10GB) for the whole project.

For the Wifi connection, Arduino is able to connect to WPA2 enterprise network¹⁰. However, as tested with TU Delft' Eduroam, the success rate is quite low (two out of ten connections made). Detailed causes of the low success rate need to be explored. Another choice is connecting to the Google Map Wifi which uses personal WPA2. If this is also not possible, one more choice is to use a Wifi repeater³⁷ as a bridge to transfer a WPA2 enterprise network to a WPA2 personal one.

Data transfer system

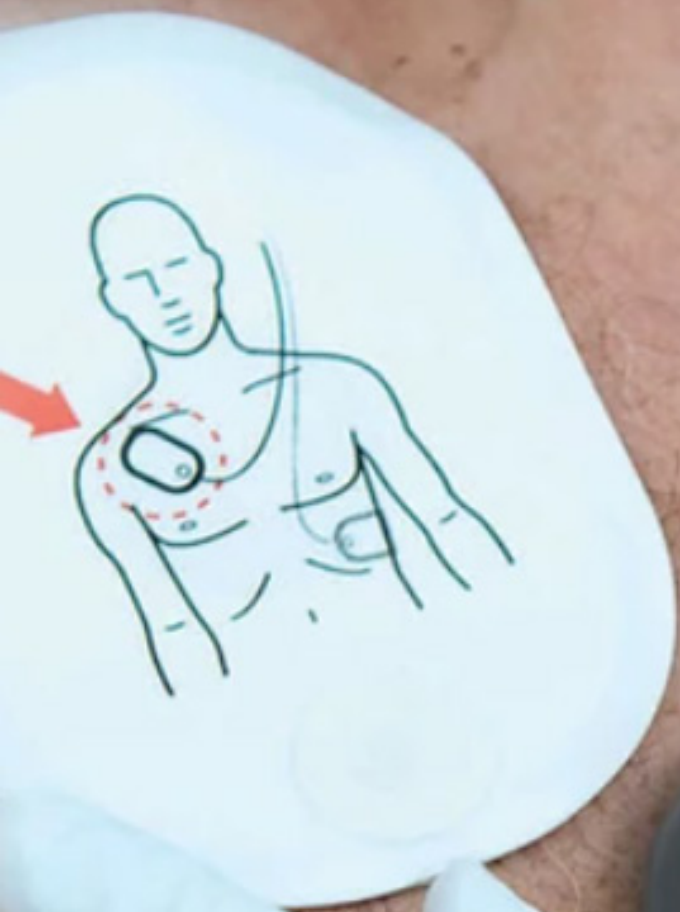
The data transfer system also fulfills its must-have function of storing data in an online dataset. Moreover, it offers pre-analysis and online editing functions. Two major problems are the time limit of executing functions of Google Script and the WPA2 enterprise Wifi connection. The first problem may be solved by better coding. The second one needs more tests in the HEMS station to generate a working solution.

Next cycle

The next cycle will be focused on the housing design of the data collector based on the electronics setup confirmed in this cycle.

06

Cycle 3



In cycle 3, the focus is on designing the housing for the electronics based on the electronic circuit defined in the former cycle. An exploration of the electronic placement is first conducted. After validation of the electronic placement, ideation on the housing design is carried out. Then a multiple models are 3D printed for testing. At the end of this cycle, a final model is chosen and a working prototype of the data collector is built.

6.1 Use process analysis

6.1.1 Use Process tree

A process tree¹⁰⁶ is first made to schematically check the activities that the data collector will encounter during its life cycle based on the defined electronic setup and data transfer system. This process tree helps to define what functions the housing design needs to provide for the expected design. It contains a production process, a use during OHCA process, a battery exchange process, and a repair process because the user needs to open the housing to perform these four processes, which affects the housing design. Following shows each process:

Production

- Step 1. Upload the program to the Arduino MKR 1010 Wifi
- Step 2. Solder all electronic parts together based on the defined circuit with wires
- Step 3. 3D print the housing parts
- Step 4. Put the electronic parts onto one housing part
- Step 5. Close the housing with another part

Use during OHCA

- Step 1. Open the rescue backpack
- Step 2. Take the plug of the data collector out of the backpack
- Step 3. Take the plug of the CorPatch out of the backpack
- Step 4. Push the plugs together
- Step 5. Take the package of the CorPatch off
- Step 6. Take the sticker on the CorPatch off
- Step 7. Place the CorPatch on the center chest of the patient
- Step 8. Unplug the CorPatch
- Step 9. Put the plug of the data collector back into the backpack
- Step 10. Close the rescue backpack

Battery exchange

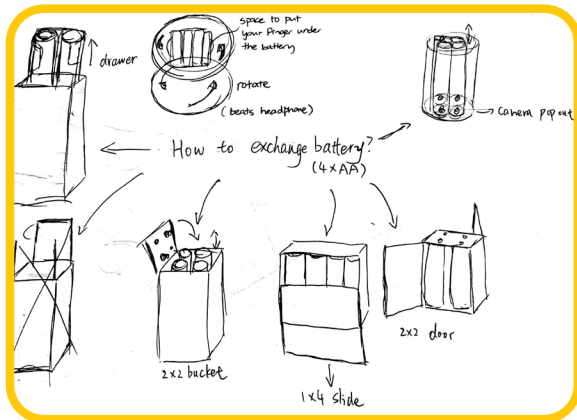
- Step 1. Open the rescue backpack
- Step 2. Take the data collector out
- Step 3. Open the lid of the data collector
- Step 4. Take the old 4 batteries out
- Step 5. Put new 4 batteries in
- Step 6. Check if one red light of the electronic parts is on
- Step 7. Put the lid back
- Step 8. Put the data collector back to the backpack
- Step 9. Close the rescue backpack

Repair

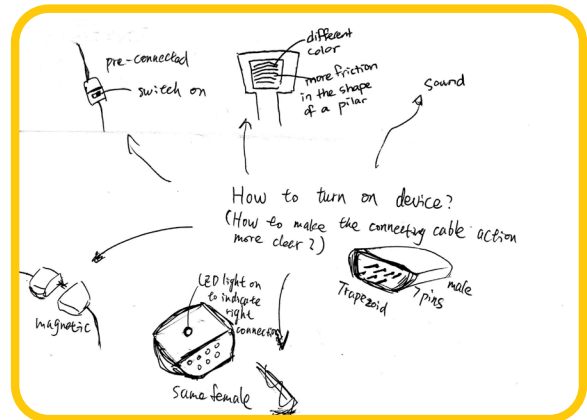
- Step 1. Open the lid of the data collector
- Step 2. Take the batteries out of the housing
- Step 3. Take whole electronic parts out of the housing
- Step 4. Connect the Arduino MKR 1010 Wifi to a laptop with a micro USB cable
- Step 9. Check possible problems from the electronic parts, the Arduino program and the Google sheets
- Step 10. Fix problems
- Step 11. Place the Electronics back to the housing
- Step 12. Place the batteries back to the housing
- Step 13. Put the lid back on the data collector
- Step 13. Test the data collector with connection to CorPatch
- Step 14. Put the data collector back to rescue backpack

In the production process, steps in blue show that the housing design should be 3D printable and leave an opening for the electronic parts to be assembled on the housing and it also should cover the whole electronic parts inside. In the highlighted part of the use during OHCA process, it indicates that the housing design should let the rescuer easily find the plug of the data collector inside the rescue backpack. In the battery exchange process, the housing is able to be opened and closed for battery exchange, and offer LED feedback to the user if batteries are successfully exchanged. In the repair process, the assembled electronic parts can also be taken out of the housing and put back onto it. All in all, these four processes require the housing design to be 3D printable, easily disassembled and reassembled, covering the electronics tight and well, and showing indication during battery exchange.

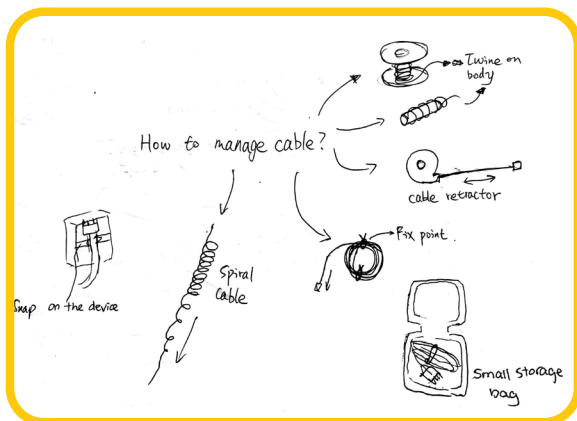
6.1.2 How to's brainstorm



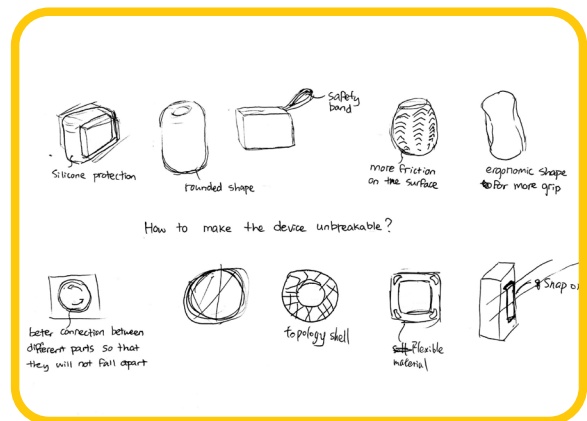
How to exchange battery?



How to plug the device with CorPatch?



How to manage the cable?



How to make the device drop-proof?

Figure 6.1 4 How-to's brainstorm

Based on the process tree, a how-to's ¹⁰⁶ brainstorm is brought out. In a how-to's brainstorm, how-to problem statements are written as shown below for idea generation. These four how-to's statements are focused on: battery exchange (battery exchange process), plugs connection, cable management, and drop-proof feature (use during OHCA process). Drop-proof feature is added because the data collector needs to be taken out and placed on the ground in some cases (e.g. limited space for placing rescue bag), according to Dr.Dinis. Because in this cycle a working housing model is also expected to be built, useful examples on these 4 topics collected and analyzed.

6.1.3 Battery exchange

Battery cases

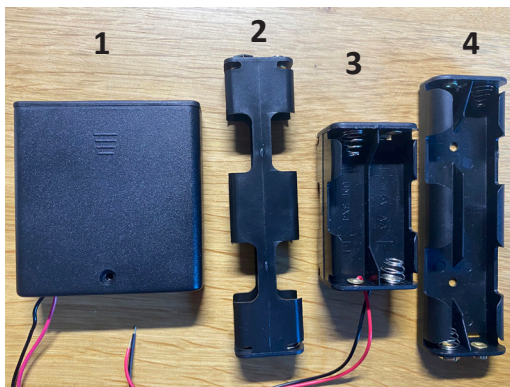


Figure 6.2 4 types of battery cases

Four types of 4xAA battery cases are first studied. It not only provides existing case designs but also can be directly implemented into the housing of the data collector if feasible. As shown in figure 6.1, 4 types of 4xAA battery cases are chosen because of their hand-held sizes. As the HEMS team needs to exchange the battery of the data collector once a week (chapter 5, cycle 2), the exchanging battery maneuver becomes one important user interaction for the data collector design. Thus, a battery exchanging speed test is carried out.

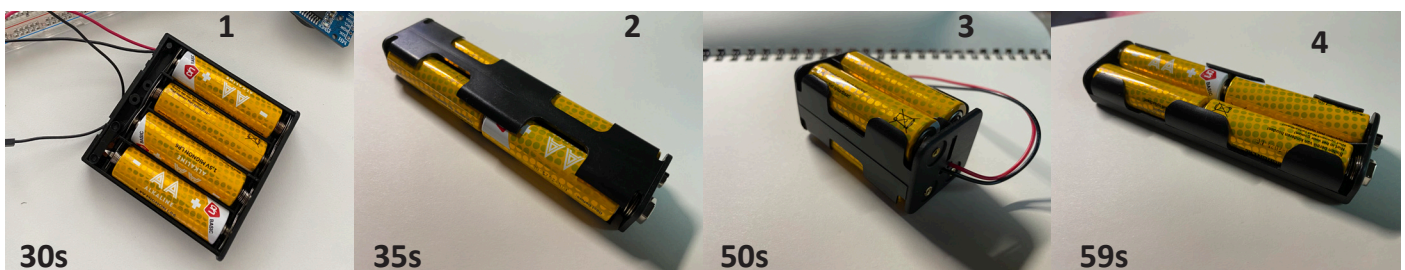


Figure 6.3 Battery exchanging speed test result

In this test, the tester performs the battery changing maneuver, by taking all 4 AA batteries out of the case and then place 4 AA batteries back. The whole process is measured by a timer. Figure 6.2 shows the result of the test. Type 1 case needs the least time to exchange battery, while type 4 takes almost twice the time to finish the maneuver. Type 2 also performs well with only 35s needed for battery exchange. However, both type 2 and 3 have 2 sides of openings for the batteries to pop out. This will give difficulties in design the data collector since 2 sides need to be left open for battery exchange.

The reason that type 4 takes the longest time is because, in its battery placement, two batteries in one slot can easily pop out in the middle, as shown in figure 6.3. To solve this, a snap cover is added in the middle of the case, which restrain the batteries from popping out. But it is also harder to take batteries out.



Figure 6.4 Type 4 battery case

The type 1 case in figure 6.5 is a classic setup with 4 batteries in a row. It requires much less time to exchange because it does not contain any snap cover constructions, unlike the other 3. Its way to hold batteries tight is by using a slide cover. At the opening side of the slide cover, there are two snap-fit constructions which can be easily pushed to open.



Figure 6.5 Type 1 battery case

Snap-fits

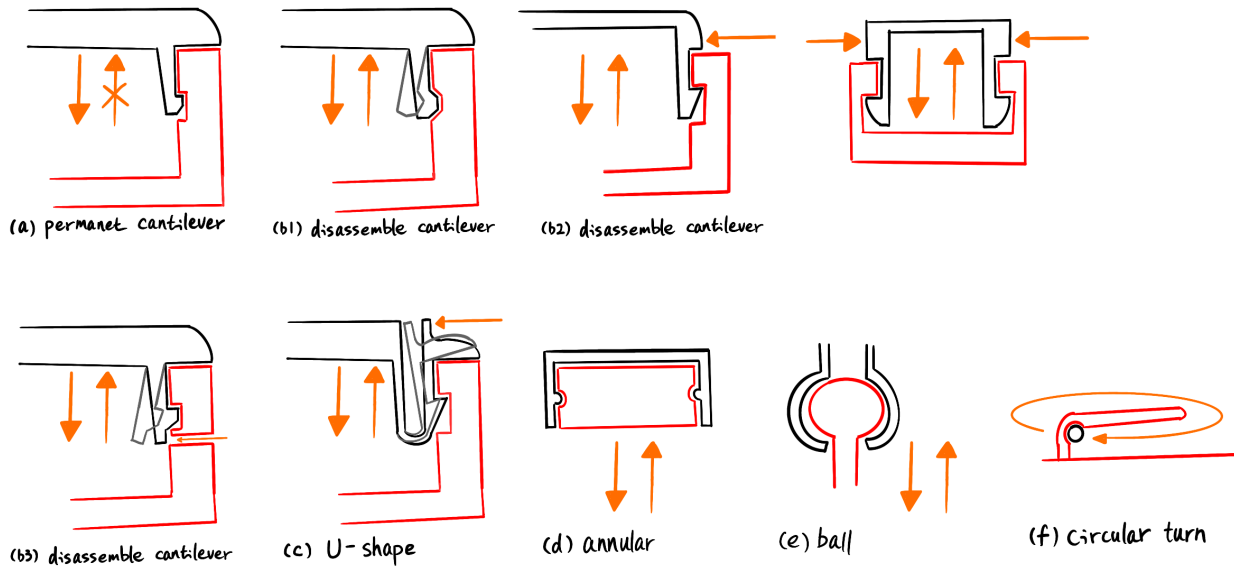


Figure 6.6 8 types of snap-fit joints

In the study on battery cases, snap-fit construction plays an important role to hold batteries while offers ways to take the batteries off. So more types of snap-fit joints are studied.

“Snap-fit joints are a simple and rapid way of joining two different components. All types of snap joints have in common the principle that a protruding part of one component (e.g. a hook, stud, or bead) is deflected briefly during the joining operation and catches in a depression in the mating component. After the joining operation, the snap-fit features should return to a stress-free condition. The joint may be separable or inseparable depending on the design” (Bayer Material Science, n.d., pp. 1–3) ²¹.

In figure 6.6, 8 types of snap-fit joints are concluded, which can be reassembled after joining. Except for the first joint (a), others are all two-way joints which can be opened and closed repeatedly. These joints are later used as references during ideation.

6.1.4 Plug connection

CorPatch plug

The CorPatch plug has been analyzed in former cycles, but they are more about its electronic properties. In this cycle, the focus is on its shape. As shown in the left of figure 6.7, because the plug is a 9-pin D-sub type, its pin layout is in a trapezoid shape (5 pins on the long base side, 4 on the short base side), so does its overall shape. This means its female connector should also be trapezoidal. In the middle of



Figure 6.7 The plug of CorPatch

figure 6.7, it shows a grip texture on the sides of the CorPatch. This is designed to add friction between the thumbs and the plug during the connecting process. On the right of figure 6.7, an arrow in the highlighted yellow circle is added on the edge of the longer base side (5 pins side) of the CorPatch to indicate that this side is the longer base side. This should help the user to identify the base side of the trapezoidal plug during the connecting process. But the arrow is not really noticeable. In emergency cases, rescuers will not pay too much attention during plugging. Fortunately, the trapezoidal shape itself can already be felt when the user pinches it. It might be enough for the user to notice which side should be matched. The female connector from the data collector should also have the same indication so the user can match the base sides of two plugs easily.

Apple MagSafe plug

One good example would be the Apple MagSafe⁹⁶ plug. It uses magnetic attraction to connect the plug and the socket. In this way, even the user does not precisely align the plug to the socket, the magnetic force will do it for him. Plus, it blinks LED lights to indicate that the plug is successfully connected. The MagSafe plug is symmetric, unlike the CorPatch plug, which means its side does not need to match the socket.



Figure 6.8 The plug of CorPatch

6.1.5 Cable management

The CorPatch comes with an 0.8 meters long cable. According to Dr.Dinis, the connected cables between the CorPatch and the data collector should be around 2 meters long because this is the usual distance between the rescue bag and the patient. This distance is meant to keep the rescue bag clean. Thus, the cable from the data collector side (female plug) should be at least 1.2 meters long. These two cables are kept inside the rescue bag during transfer. At the rescue scene, the rescuer needs to find and connect the plugs while all the cables need to be sorted out. This process might cause chaos in the bag since there is a 1.2 meters long cable wrapped inside. Thus, analysis on cable management is necessary to carry out to see if there are better solutions available to make the process easier for the rescuers.

Cable retractor

Figure 6.9 shows a USB cable retractor³⁹. A cable retractor uses elastic force generated by a circular spring to retract the cable automatically. This structure is widely used in tape measure, vacuum, etc. It is really convenient because it stores the cable in one place and retracts the cable automatically.



Figure 6.9 Cable retractor

SuperCalla

Figure 6.10 shows a SuperCalla⁷⁹ cable. SuperCalla is a USB cable that can be shaped with the magnetic rings on it. These magnetic rings are fixed on the cable. They snap to each other to shape the cable into a shape the user want. In this way, the cable will not be twined in the bag and stay stable during usage.

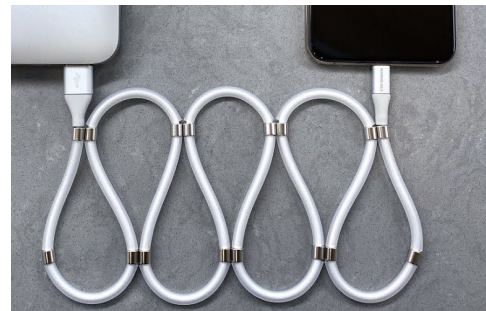


Figure 6.10 SuperCalla

Fuse Reel

Figure 6.11 shows the lineup products from Fuse Reel³⁴, which is a company focuses on cable management tools. In their lineup, the cables are all wrapped around an object for compact storage. It offers snap-fit joints or spinning caps to release the wrapped cable for use.



Figure 6.11 Fuse Reel lineup

Velcro

Figure 6.12 shows how a simple Velcro¹⁰¹ band can be used to manage cable. A Velcro band has one hook side and one loop side. The hook side and the loop side can be stuck together and separated by hand. Both sides provide considerable force to stick to each other. Velcro material is useful when assemble and disassemble is both often operated on a product.

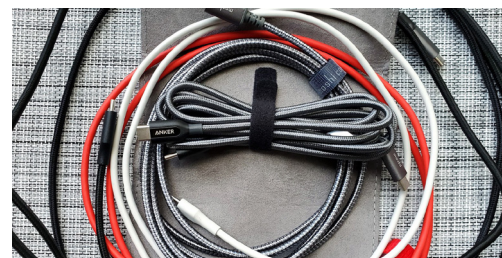


Figure 6.12 Velcro band on cables

6.1.6 Drop-proof

In most OHCA cases, the data collector is expected to be placed inside the rescue bag, while its cable and the connected CorPatch is taken out to collect data. However, there are also cases that the rescue scene is located in a rather small space where the rescuers cannot bring the bags in and the distance between the bag and the patient is more than 2 meters. In these cases, the data collector will be taken out of the bag along with the CorPatch and be placed on the ground.

Phone case

In figure 6.13, a phone case is made to protect the cellphone from being scratched and damaged after dropping on the ground. It is usually made by soft elastomer material (e.g. silicone rubber¹⁰² with a snap-fit construction. It mainly wraps around a cellphone's frame to absorb shocks and endure scratches.



Figure 6.13 Cable retractor

Table corner protector

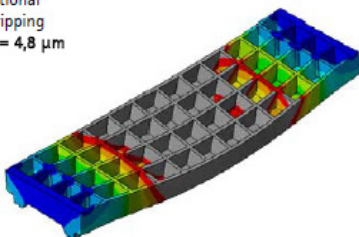
Figure 6.14 shows the table corner protectors. They are used to cover the solid table corners with soft material, so the kids will not get injured by bumping into the table corners. They are cheap, replaceable, and can be applied to other types of corner protection too.



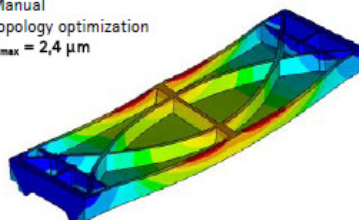
Figure 6.14 Table corner protectors

Topology optimized structure

Traditional
box ripping
 $u_{max} = 4,8 \mu m$



Manual
topology optimization
 $u_{max} = 2,4 \mu m$



Automatic
topology optimization
 $u_{max} = 1,7 \mu m$

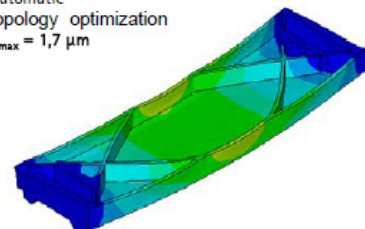


Figure 6.15 Topology optimization application⁴⁴

Figure 6.15 shows how topology optimization⁴⁴ changes the strain of the object under a defined condition by optimizing its inner structure. Topology optimization can be used to reduce weight and material while keeping the whole structure under deformation limits.

6.1.7 Conclusion of analysis

In this analysis phase, 4 areas have been researched and useful examples are collected. For the battery exchange process, the type 1 4xAA battery case is considered as the best setup and snap-fit joints are the ideal construction to make a two-way housing. For the plug, indications of the side of the plug should be designed. For the cable management and drop-proof feature, the examples provide different solutions that can be applied in housing design and prototype making.

6.2 Electronics housing ideation

After analysis of the use process, ideation on the shape of the housing is carried out. During this phase, the electronics placement with a chosen battery case is explored first, so a basic shape of the housing can be built based on that. The ideation of the detailed construction of the housing is illustrated. At the end of this phase, one electronics placement and one basic housing shape is chosen for working prototype building.

6.2.1 Type 1 battery case ideation

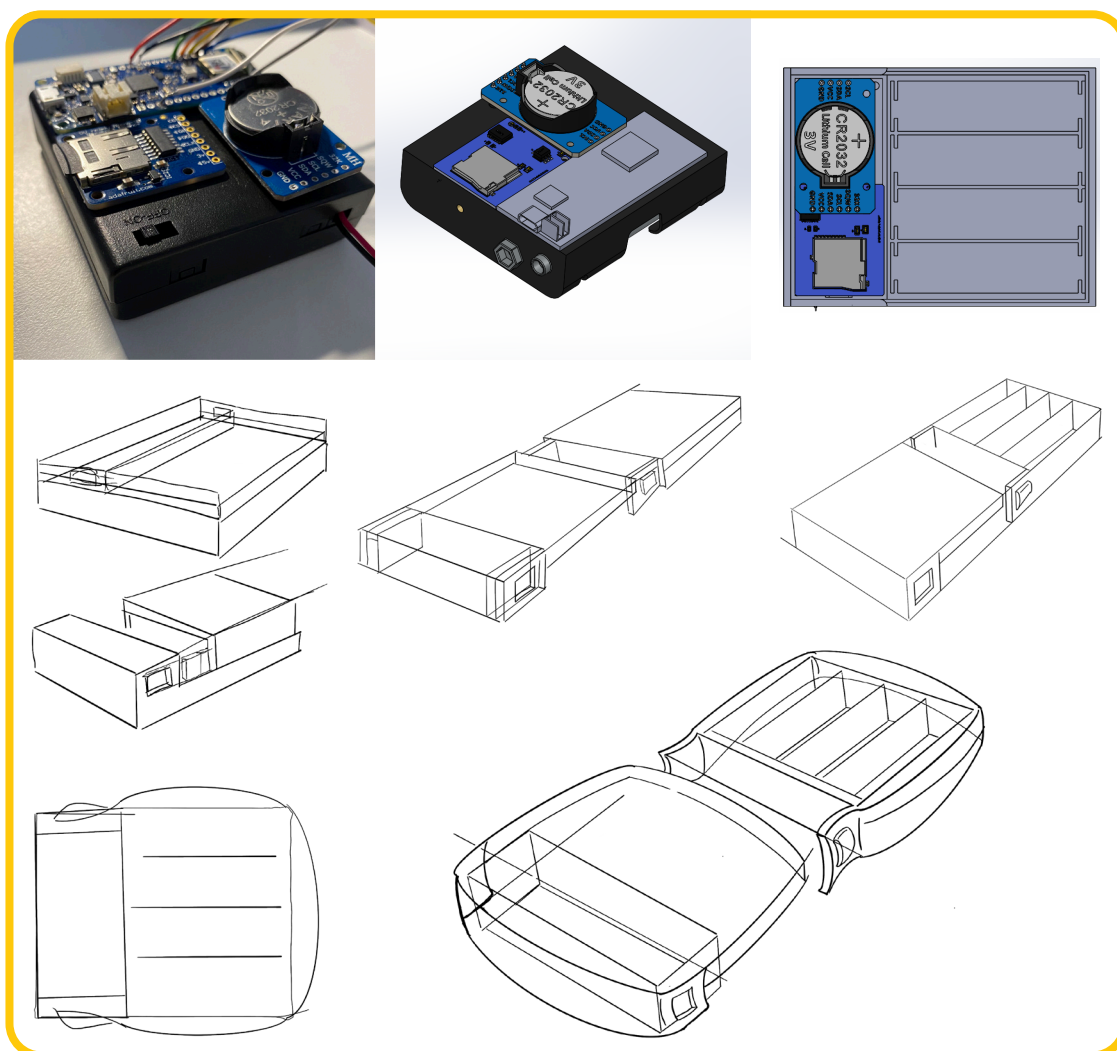


Figure 6.16 Housing ideation on type 1

Figure 6.16 shows the ideations based on the type 1 battery case. On the left of the first row, the electronic components of the defined circuit are placed around the type 1 battery case to explore possible setups. This setup is then modeled in SolidWorks as shown in the middle. All the electronic components are modeled following the actual measurements. A new setup is then generated in SolidWorks as shown on the right, all the electronic components are stacked together and placed on the side of the battery case. It reduces the thickness of the housing and separates the battery and the electronics. In this way, the LED light indication can also be shown directly to the user during battery exchange.

Based on this setup, sketches are made to explore possible detailed shape of the housing in figure 6.16. The illustrated idea implements the snap-fit joints into the housing construction. Curves are added around the edges of the shape to absorb falling shocks and make it more comfortable lying on hand. However, this slider plus snap-fit combination brings difficulty in actual 3D printing based prototyping. First, the electronic components are hard to be inserted and fixed in the lid part because the opening is small. Second, the battery case needs to be dragged out for battery exchange, but it is also connected to the electronics for powering via wires. So in this setup, the battery wires need to be long enough for the battery case to be dragged out.

6.2.2 Type 2 battery case ideation

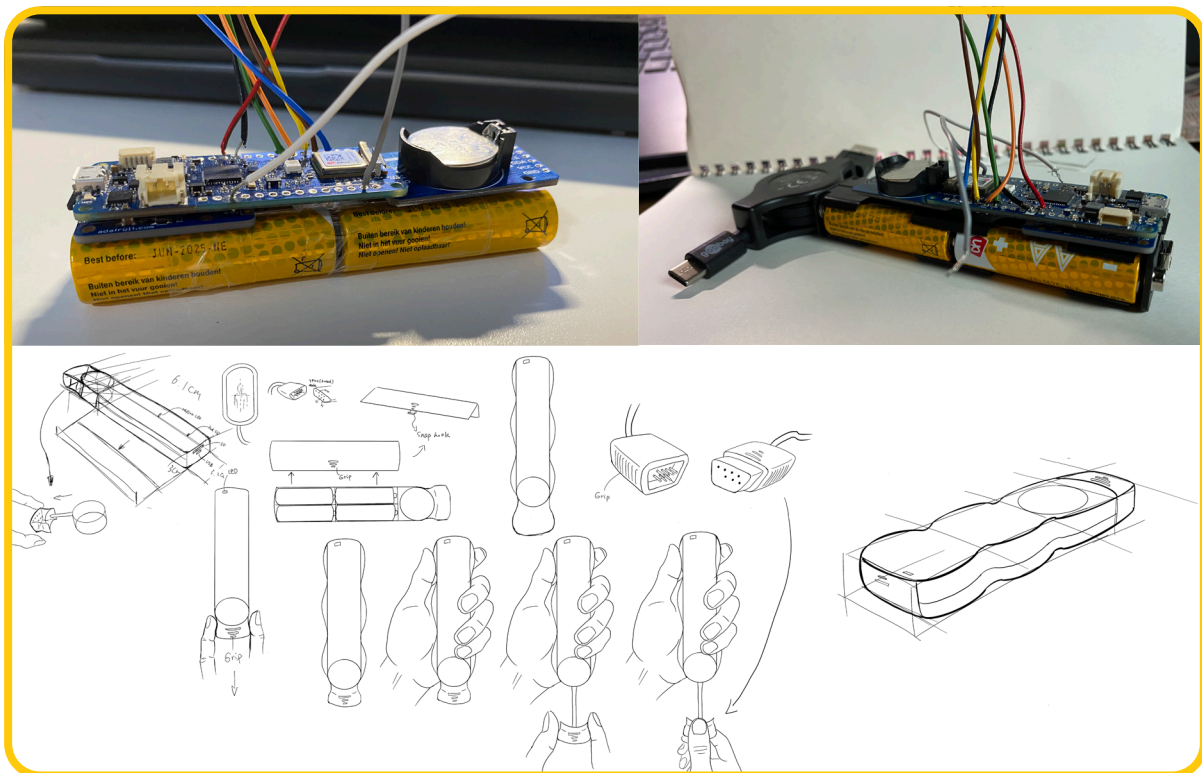


Figure 6.17 Housing ideation on type 2

The second idea is generated based on the type 2 battery case. As shown in figure 6.17, the electronic components are placed on the batteries in a narrow way, so the whole setup is slim and long. At the end of the setup, a cable retractor mechanism is added to manage cable. In the sketch, the batteries are covered by a lid that can be slid out for battery exchange. The lid can be snapped back to the housing. The body of the housing is wavy so it gives the user more grip when dragging the cable out. The female plug has grip textures on the broadside, and the narrow side is curvy, so both provide enough grip when the user drags the plug.

6.2.3 Type 3 battery case ideation

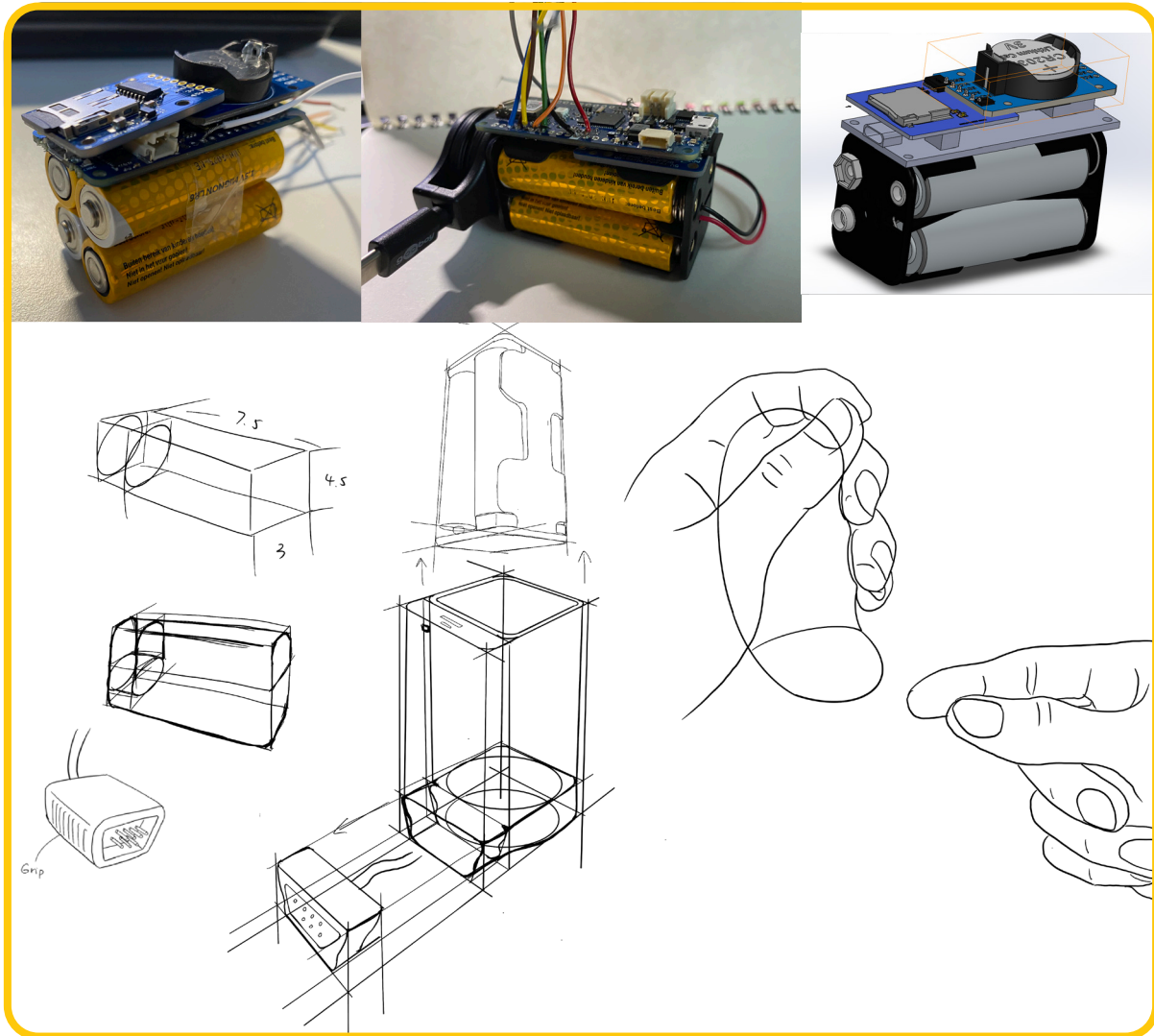


Figure 6.18 Housing ideation on type 3

As shown in figure 6.18, the third ideation is built based on the type 3 battery case. The electronic components are piled up on the batteries and at the bottom, a cable retractor mechanism is added, like ideation 2. Here the housing acts like a bucket for the battery. During battery exchange, the user takes out the whole battery case to access batteries. The overall shape of this ideation is more ergonomic, that lies in the hands comfortable, like an egg. The user can drag the cable out at the bottom.

6.2.4 Conclusion of ideation

At this ending phase of the project, and the end goal is building a working prototype, the feasibility of building a working prototype by 3D printing is the decisive factor. With this standard, ideation 1 that builds on type 1 battery case wins because its overall shape is more suitable for 3D printing (less structural support needed during printing), and the type 1 case is also the fastest in battery exchange. The next step in modeling the housing is based on ideation 1.

6.3 Housing models building

In this section, the process of building the housing model is shown. Because the 3D printers are available in the faculty, during this process, Solidworks models are made and printed daily for evaluation, then adjustments will be made for the next day printing. This daily cycle of printing - evaluating - adjusting - printing improves the model efficiently. At the end of the process, a housing model for the working prototype is made.

6.3.1 Model set 1



Figure 6.19 Model set 1

Figure 6.19 shows the first set of models that are built based on ideation 1. All the 4 models borrow the construction from the type 1 battery case as shown on the left of the figure. They adopt the slide and snap-fit features

from the battery case, and extend an extra room for placing the electronics, like ideation 1 did. For details of how each model is evaluated and adjusted, please check appendix J.

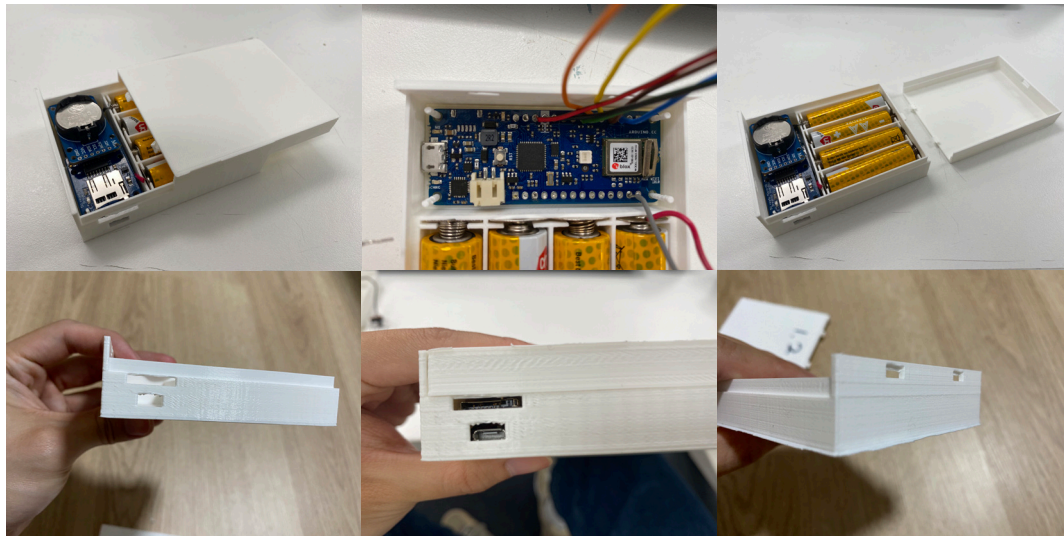


Figure 6.20 Details of model 1.4

Model 1.4 is the most developed model in this set, as shown in figure 6.20. The electronic components fit well inside the designed slots and are fixed by the sticks. Batteries can be placed inside tightly. There are two openings left to connect to the micro USB cable and insert the micro SD card.

However, the printed snap-fit joints in all four models are not usable. Figure 6.21 shows the failed printed snap hooks (left) and lock (right) from model 1.4. This snap-fit construction fails in printing with support/no support, and also in printing with 0.15mm accuracy (normally is 0.2mm). The reason could be that the tip of the hook is only 0.25mm wide, which



Figure 6.21 Failed printed snap-fit joints

requires more accuracy from the 3D printer, while the printing nozzle from the 3D printer only outputs 0.15mm. Also, in the case of the support structure being generated for the overhang snap joints, the support structure needs to be taken off after printing. But the snap joints are too thin so in model 1.3, they also get taken off along with the support structure.

6.3.2 Model set 2



Figure 6.22 Model set 2

Since the snap hook is not compatible with 3D printing, another type of snap joints is implemented and 4 new models are printed as shown in figure 6.22. Model 2.1 on the left is the basic model that uses a disassemble cantilever structure (chapter 6.1.3 - snap-fits). Model 2.2 and 2.3 are all failed because of same problem as model 1.3 and 1.4, the joints are damaged during taking off the support structure.

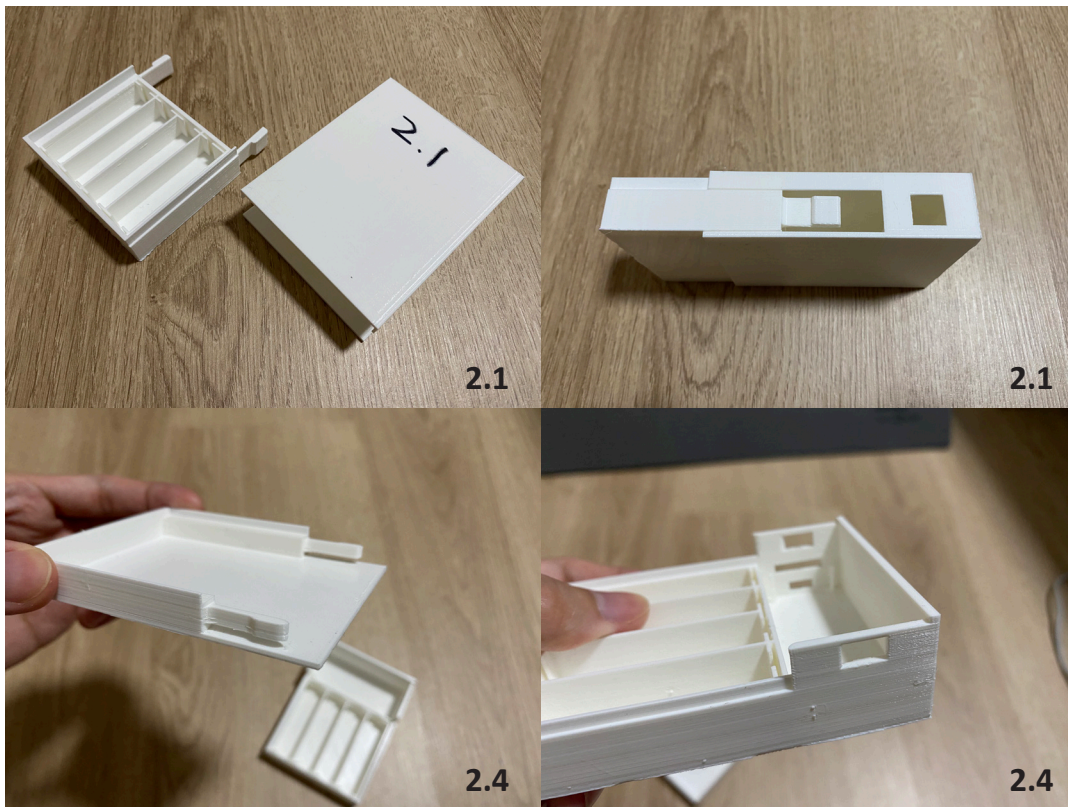


Figure 6.23 Model 2.1 and 2.4

Figure 6.23 shows two survived models - model 2.1 at the top and model 2.4 at the bottom. The snap cantilevers of model 2.1 can snap tightly to combine two housing parts. But it also takes some effort to open the housing. Also as shown, the battery housing part is separated from the electronic housing part, this causes trouble during battery exchange because there are wires between them. So model 2.4 is developed to put the electronic housing and battery housing together in one part. The other part is a sliding lid with the snap cantilevers. They are also designed to be thinner and longer so it needs less force to deform them. However, slimmer design also makes it less durable, it can be easily damaged during the taking off support procedure, and also if the user is not careful when pressing the cantilever, he might break them.

6.3.3 Model 3

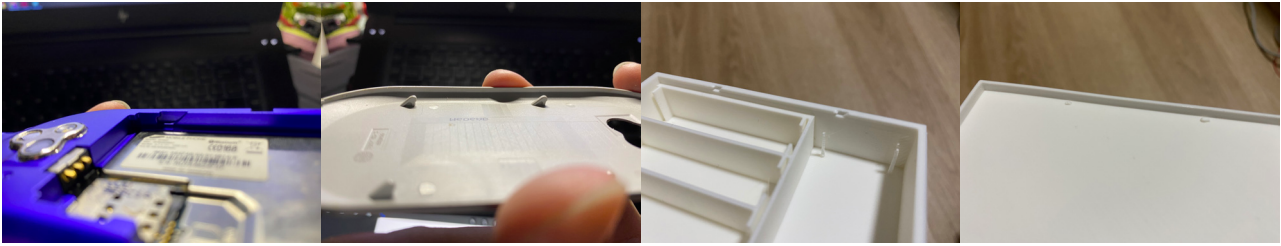


Figure 6.24 Model 3

Figure 6.24 shows model 3, which adopts the snap structure from a battery cover of an old cellphone. This snap structure does not require any support during printing because there is no overhang structure. However, when testing the printed model, the snap feature requires high accuracy on matching to each other that the printer cannot offer. So the snap feature does not help fix the lid on the box.

6.3.4 Model 4

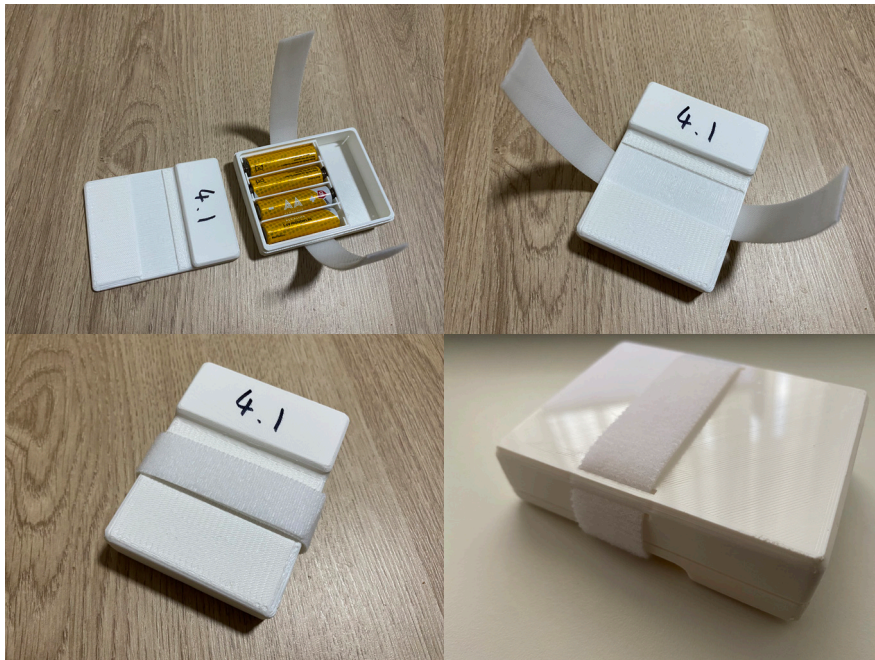


Figure 6.25 Model 4.1

After trying 3 types of snap-fit joints with 9 models printed, no models can ideally fulfill the requirements. The reason might be that two-way snap-fit joints not only require high accuracy in production, but also materials with high strain, which the PLA 3D printer can hardly fulfill. Thus, the fourth model is not built on a snap-fit feature. Instead, the Velcro band is chosen as a method to open and close the housing parts, because of its easiness in implementation.

Figure 6.25 shows model 4.1 which is built with a Velcro band wrapped around. The Velcro band is a two-sided band, with one hook side and one loop side. So it can stick its one side to another side and make a wrap. It goes through the body of one housing part so when it is released, it stays on the housing. The Velcro band wraps the two parts tightly and it can be easily opened for battery exchange. So this model is decided as the final shape for the working prototype.

6.3.5 Female plug



Figure 6.26 Printed Plugs and purchased plug

As shown in figure 6.26, multiple female plugs are modeled and printed to test if they are compatible with the CorPatch male plug. Metal conductors also need to be placed inside the printed plug for transmitting the electricity. On the other hand, a 9-pin female D-Sub plug with 2 meters long cable is also purchased for testing.

The 3D printed plug follows the measurement of the CorPatch plug as shown in figure 6.27. It leaves holes through its body for the placement of female jumpers. However, it is difficult to fix the jumpers inside the holes because a closing structure needs

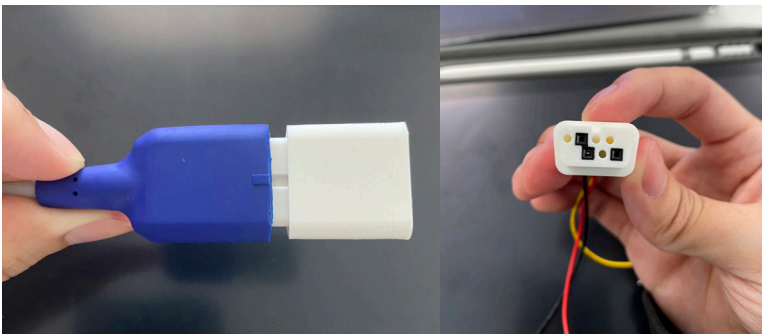


Figure 6.27 Printed plug with jumper wires

to hold the jumpers from moving out. Besides this drawback from the printed plugs, they offer possibilities to add usecues⁴⁸ on the plug.

The purchased plug shown in figure 6.28 fits with the CorPatch tightly. The plug itself is bulky and symmetric, so user really needs to look at the pin layout on the plug to identify which side is matching to the CorPatch plug during connection. However, as inspired from the arrow shape indication from CorPatch, the usecues can also be drawn on the purchased plug to notify the user which side he is facing, as shown in the figure, a matching arrow is drawn on the plug.

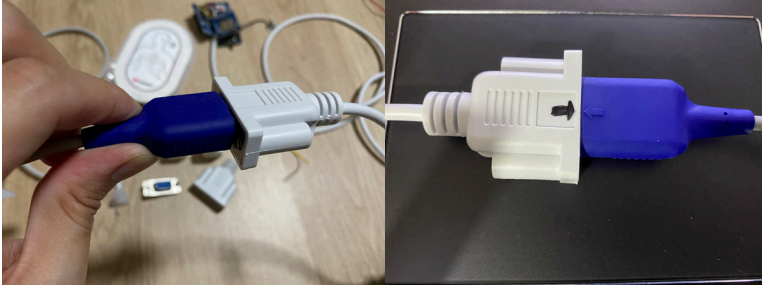


Figure 6.28 Purchased plug

6.4 Evaluation on model 4

6.4.1 Velcro band tightness test

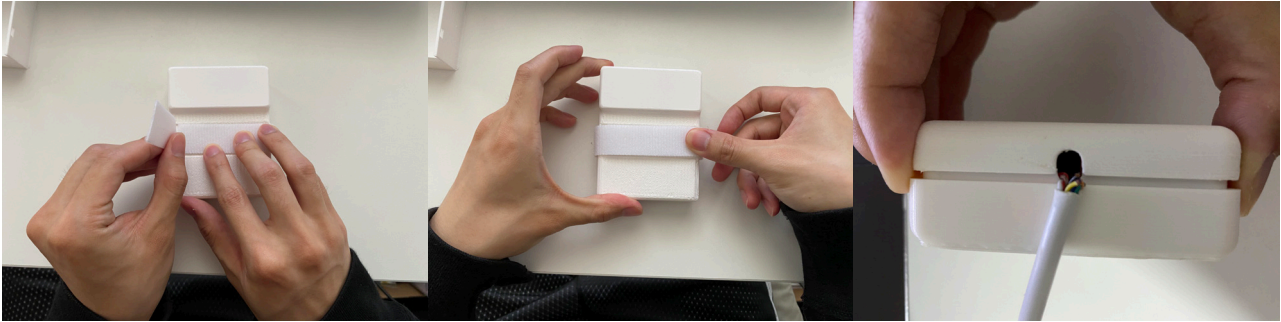


Figure 6.29 Velcro tightness test

How tight the velcro can wrap the housing parts together is important because the electronics should be fully covered when the data collector is used outdoor. As shown in figure 6.29, the velcro wraps the housing parts with a normal tightness. When the tester forces to split the housing parts, an inner wall shows in the gap. This 3 millimeters high inner wall is added following the edge of the housing. It provides several functions: First, it adds friction between two housing parts, so they will not easily be split. Second, It prevents the housing part from moving off because the wall is blocking. Third, when there is a gap between the two parts, it fills the gap to cover the inner electronics. For the Velcro band, in a normal tightness condition, it can already hold two parts tight enough that even external force cannot split them, and also no electronics are exposed.

6.4.2 Drop test



Figure 6.30 Drop test 1

In figure 6.30, a drop test is carried out to see how drop-proof the housing design is. The test is first performed from dropping at 50 centimeters height. The housing survives with only scratches on the surface. Then a one-meter dropping is performed, the outside wall of the cover part is cracked, but only a slit is shown, so the housing is still usable. Then the drop height increases to around 1.8 meters high as shown in the figure. The whole side wall of the cracking cover part is fallen off and a visible gap is shown, so the device is not usable and needs a replacement of the cover part.



Figure 6.31 Bumper version and drop test 2

On the second drop test, 8 bumpers are added on the 8 corners of the housing, as shown in figure 6.31. These bumpers are made of soft rubber and are used initially to cover the corners of the furniture. They protect the housing from being scratched and broken even from a 2-meter falling. However, the bumpers add up the size of the housing, and they easily fall from the housing because they are attached to the housing only by double-sided tapes.

Overall speaking, the Velcro band performs well on wrapping the housing. The housing is quite durable in the OHCA cases because in 90% of the cases, according to Dr. Dinis, the data collector only needs to stay inside the bag, while for the rest of the cases it is also rare to drop it from more than one meter high because the rescue bag is normally placed on the ground. For the plug, the purchased plug suits well with the CorPatch plug and comes with a 2 meters long cable. Usecases can also be added by drawing and stickers. For a working prototype, the purchased plug is good enough.

6.5 Conclusion

At the end of this cycle, the housing design is validated and the plug is also decided to use the market available one as shown in figure 6.32. Until this point, a working prototype of the data collector is built.

In the next and last cycle, this working prototype will be elaborated and tested with simulated manikin with a HEMS team member. Then a final design of the data collector will be carried out and explained. Recommendations on further development of the project will be listed at the end.

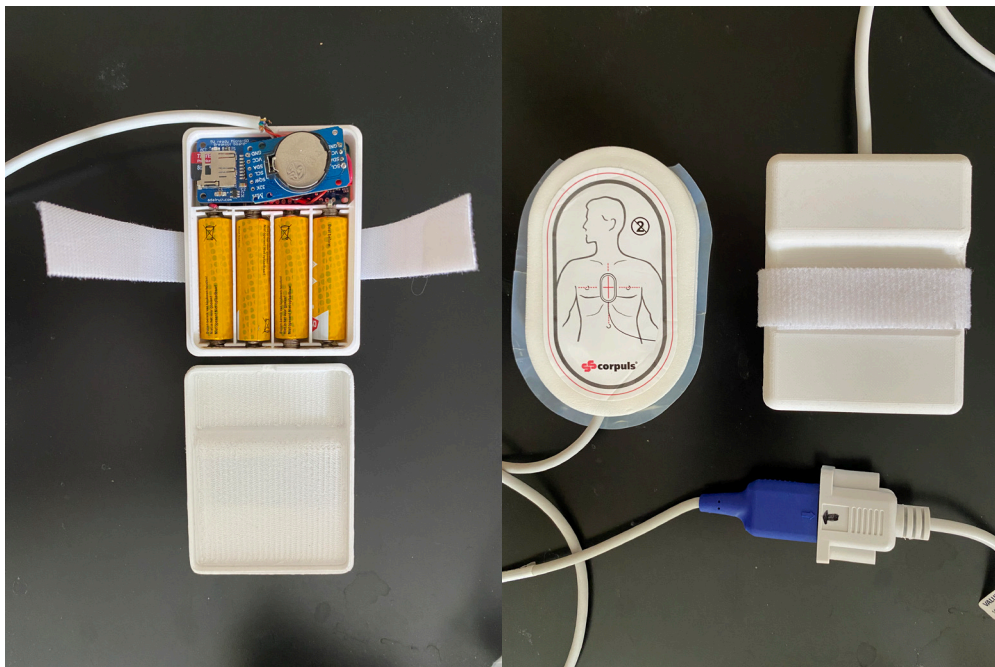


Figure 6.32 The working prototype setup



07

Cycle 4

In this last cycle, the working prototype is elaborated and tested by me and the HEMS team. Then a final design of the data collector including its appearance, its functions, user scenarios, production method, and assembly process, will be elaborated. At last, recommendations on further development of the project will be listed.

7.1 Working prototype elaboration

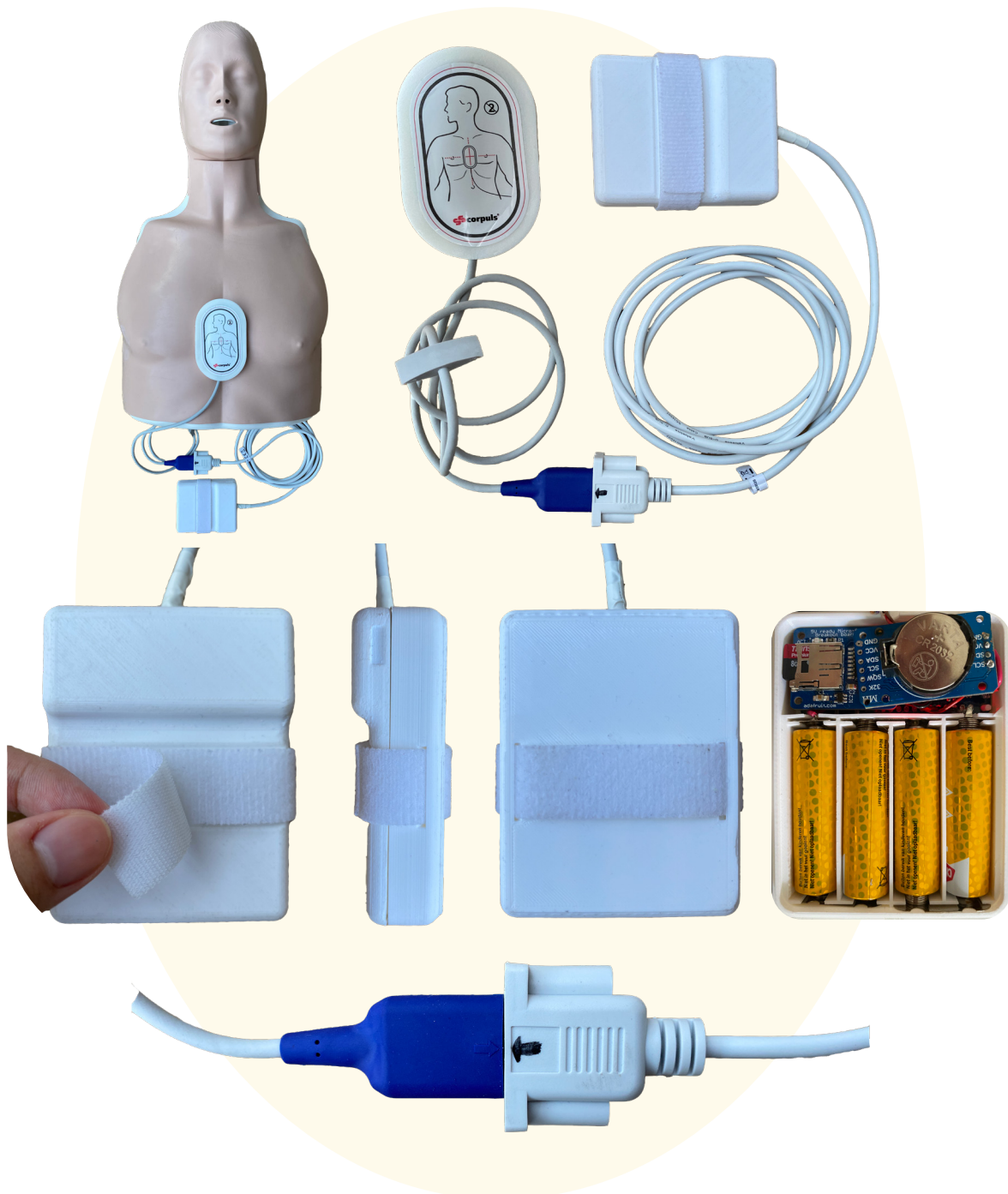


Figure 7.1 The working prototype setup

Figure 7.1 shows the whole setup of the working prototype of the data collector. The CorPatch is placed on the patient's center chest for data collection. It is connected to the data collector by a 2-meter long cable. The data collector is fastened by a Velcro band and can be opened for battery exchange and maintenance.

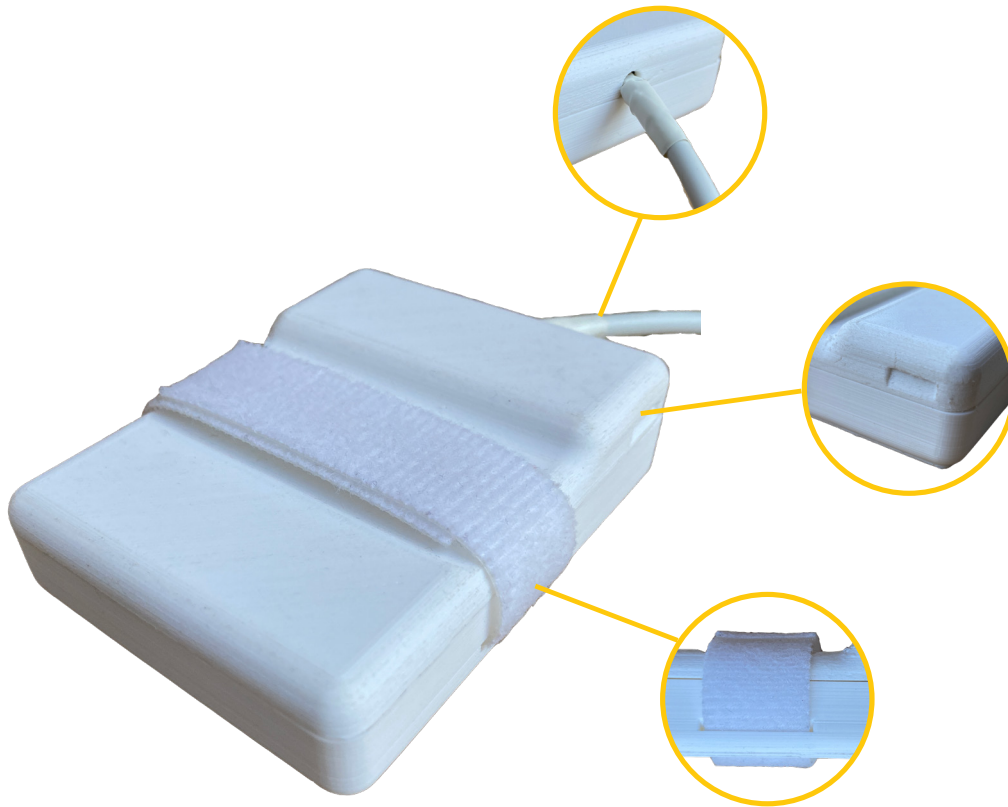


Figure 7.2 3 details of the working prototype

Figure 7.2 shows 3 details of the working prototype. First, white duct tape is wrapped at the connection point of the housing and the cable. The duct tape is durable and waterproof, it fills up the gap between the cable and the housing and also provides protection at this connection point, which is usually the most vulnerable part of a cable. Second, at each side of the raised housing part, a groove is added to make it easier to buckle off the housing. Third, the Velcro band passes through the housing so it stays attached to the housing when it is unfastened.

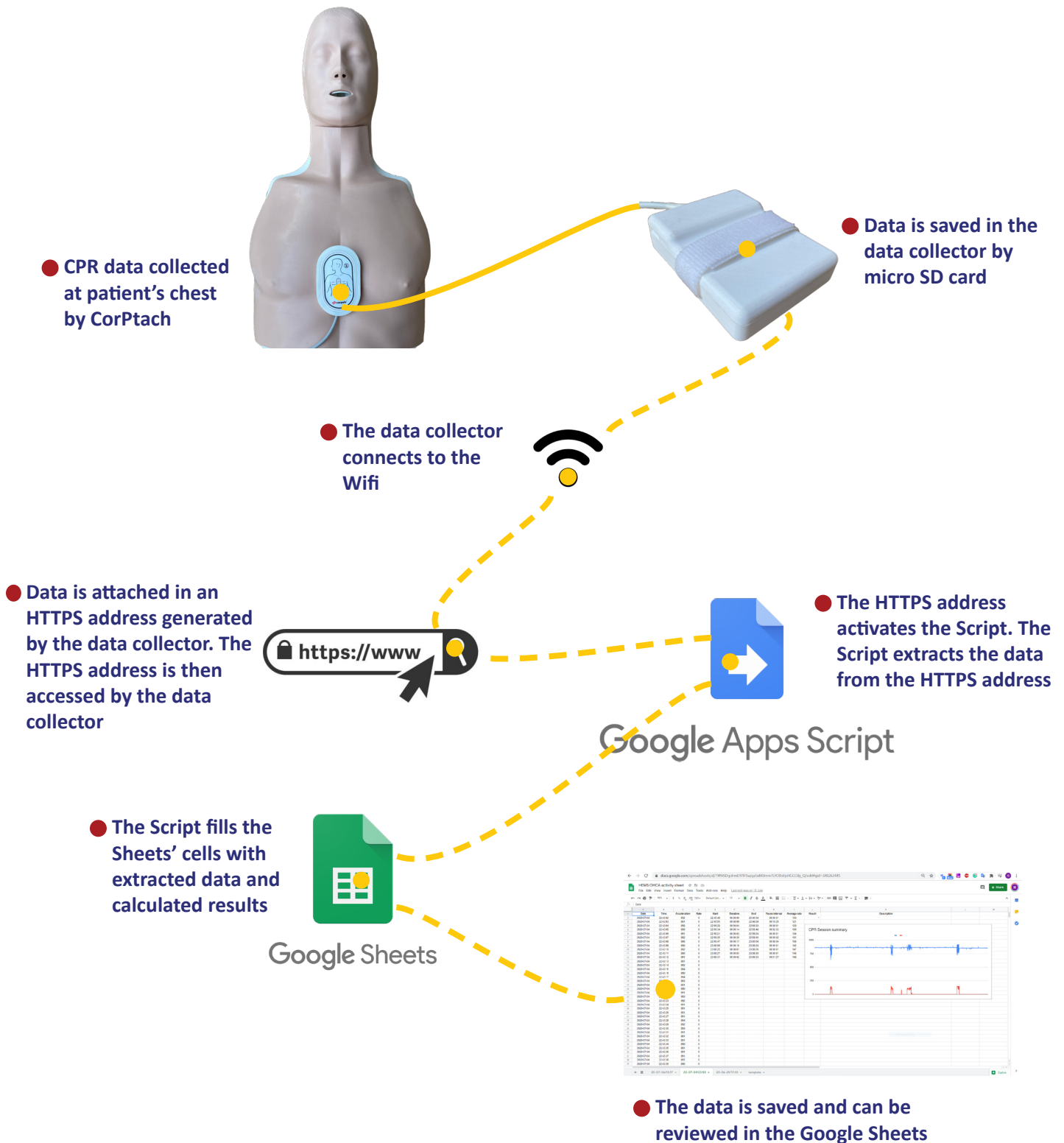


Figure 7.3 Data transfer route of the working prototype

Figure 7.3 shows the data transfer route of the working prototype. The CPR data is first generated by the chest compression motion that the CorPatch follows. Then the data collector reads the sensor values from the CorPatch and saves them with their corresponding time in the micro SD card. When the data collector connects to Wifi, it generates an HTTPS address with the data attached at the end and call that address on the Internet. The script is then activated because the address always contains its ID. The script extracts the data from the address and fills them inside it corresponded Google sheet. Based on the uploaded raw data, the script calculates the analysis results and generates the summary chart for this rescue case.

7.2 User tests with the working prototype

7.2.1 Pilot test

The pilot test is conducted by the designer in an outdoor environment. The test videos can be found in the appendix FIXME. Two user scenarios are tested: usage during the OHCA case and battery exchange, because these two are the only scenarios that the user has direct interaction with the data collector.

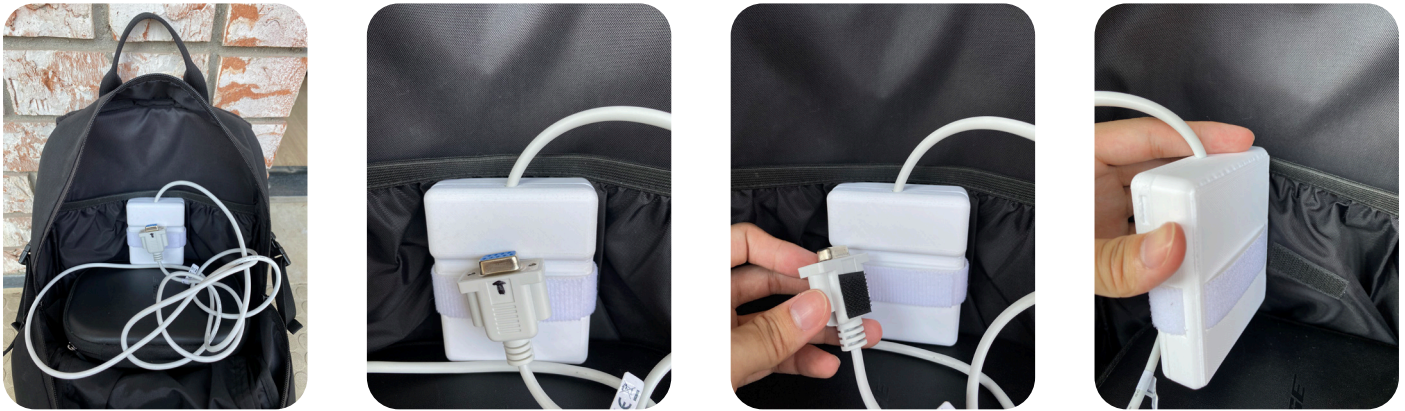


Figure 7.4 Velcro band usage in the rescue backpack

The result of the pilot test is positive. The prototype collects accurate chest compression data and uploads it to the Google Sheets via home Wifi (WPA2 personal). The setup time of the data collector during OHCA rescue is within 10 seconds. One outcome is that in order to improve the cable management inside the rescue backpack to shorten the setup time, Velcro bands can be implemented on the plug and the inner wall of the backpack, as shown in figure 7.4. So the rescuer does not need to find the plug in the backpack, instead, he can directly take the plug off from the collector after opening the backpack, while the collector is still fixed on the inner wall.

7.2.2 User test



Figure 7.5 User test in two simulated scenarios

The user test is carried out with a HEMS team member in the hangar of the Rotterdam HEMS station. A manikin for CPR training is used as a replacement of a patient. Two scenarios (rescue and battery exchange) are performed by the user with the working prototype.



Figure 7.6 Rescue backpack and the data collector placement

The result of the user test is also positive. The prototype collects data correctly during OHCA cases, it cannot upload data to Google Sheets because the available Wifi in the station are WPA2 enterprise networks. But it can successfully upload data via smartphone's hotspot. It proves that if the Internet is connected, the data transfer can be realized. The setup time of the data collector is also within 10 seconds.

During this test, the placement of the data collector is also decided, as shown in figure 7.6. On the side of the rescue backpack, there is a net bag that is used to hold small accessories available for placing the collector. It holds the data collector tight. The cable of the collector can be placed on the top of the four inner bags (green, blue, red, and black) so it is available when the backpack is opened.

Besides the network problem, another 2 questions are raised by the user. First is the durability of the housing of the data collector. Another question is about the waterproofness of the data collector because when there is rain and the rescue occurs outdoor, the rain will splash on the opened backpack. Both two questions are reflected in the recommendations section in the implementation chapter.

7.3 Final concept elaboration

7.3.1 Appearance

Figure 7.7 shows the renderings of the final concept - **CPR data collector**, and the color code of each model. According to Dr. Dinis, each HEMS station should possess three data collectors, one for the helicopter rescue backpack, one for the automobile rescue backpack, and the last one serves as the backup. The main colors on the emergency vehicles used by the HEMS team are yellow (body), blue (pattern), and red (pattern). The main color yellow assigns to the helicopter backpack because the HEMS team's preferred vehicle is the helicopter. Red assigns as backup because it shows the meaning of warning and tells the team that they are using a backup device, which means the original one is broken or being repaired. Then blue is assigned to the automobile backpack. Detailed measurements of the CPR data collector can be found in appendix J.

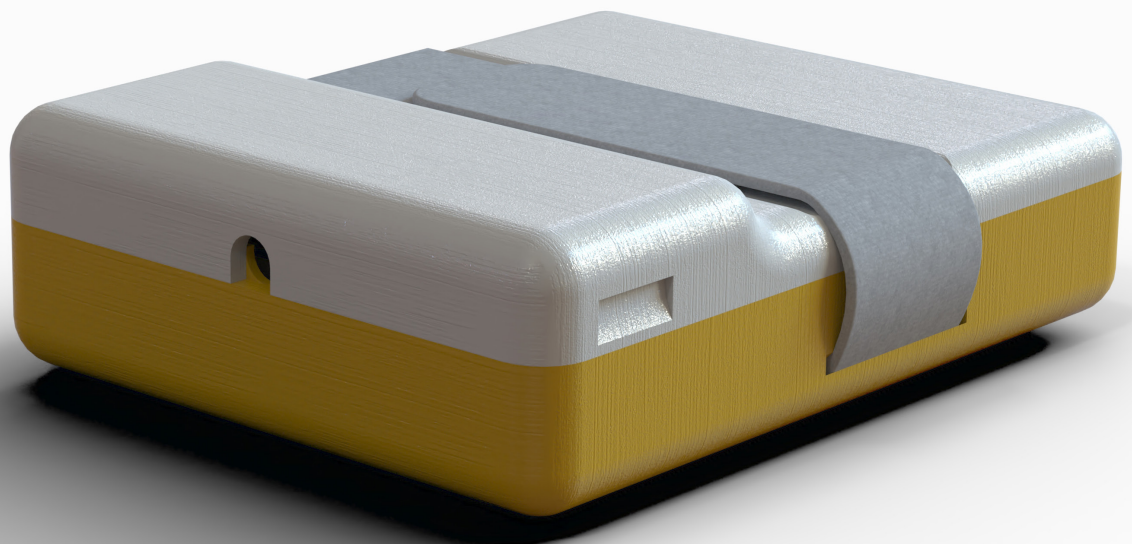
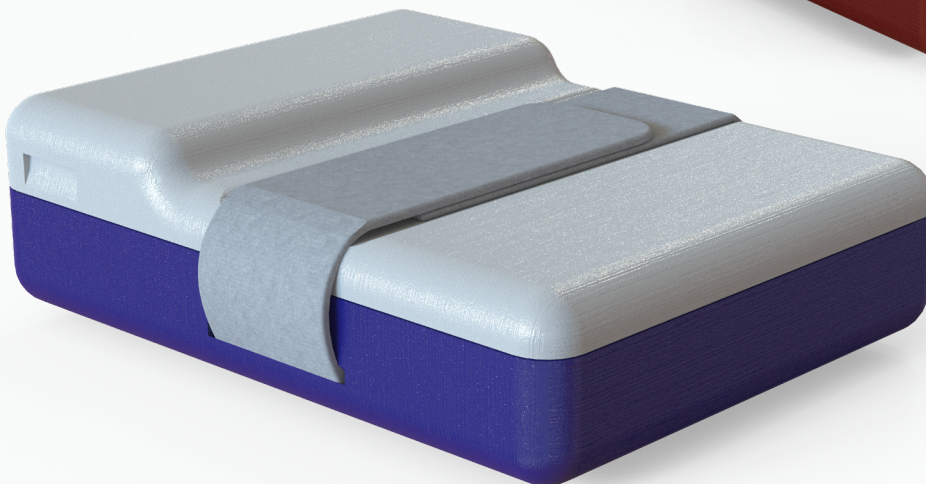
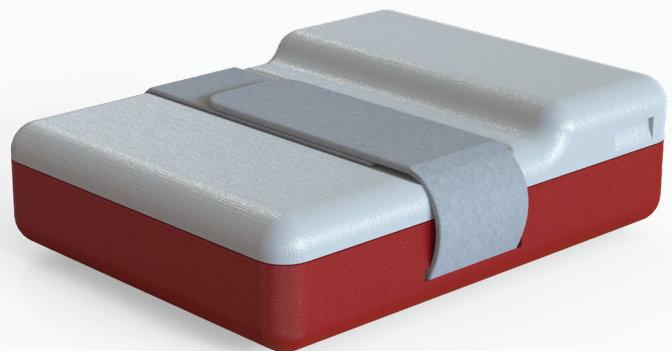
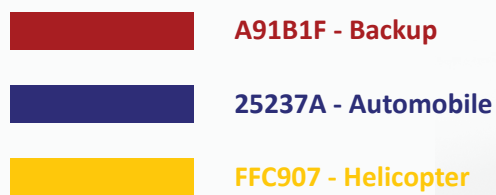


Figure 7.7 Final concept renderings

7.3.2 Main components

Figure 7.8 shows the main components of the CPR data collector.

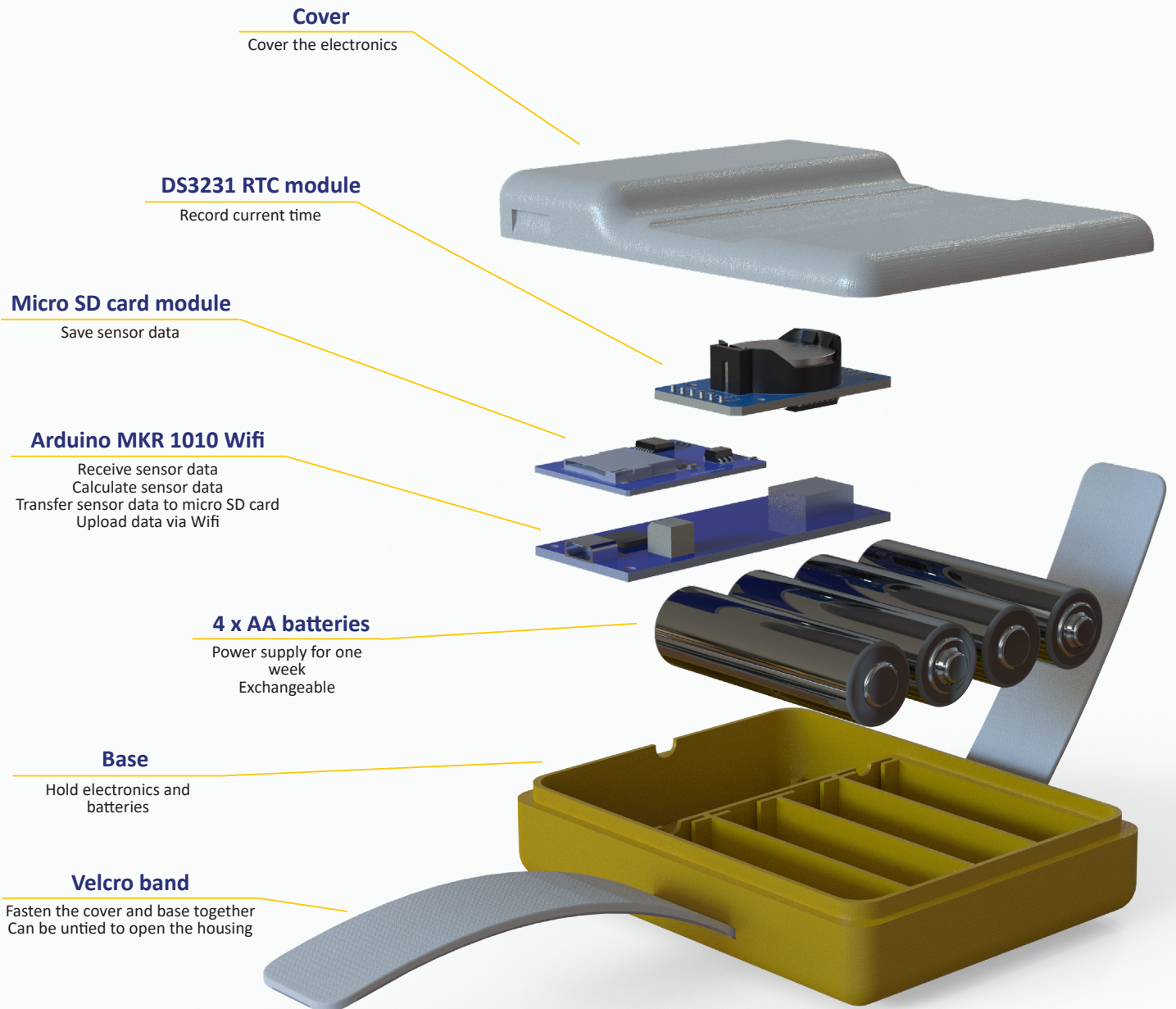


Figure 7.8 Exploded view

7.3.3 User scenarios

During OHCA rescue



1. Check the condition of the patient



2. Open the rescue backpack



3. Plug the CorPatch with the data collector



4. Take out the CorPatch



5. Place the CorPatch on the patient's center chest



6. Conduct chest compression



7. Give ventilation



8. After rescue, unplug the CorPatch



9. Take the data collector's plug back to backpack



10. Put the CorPatch back to backpack



11. Close the backpack



12. Ready for next task

Battery exchange & data review



1. Open the backpack



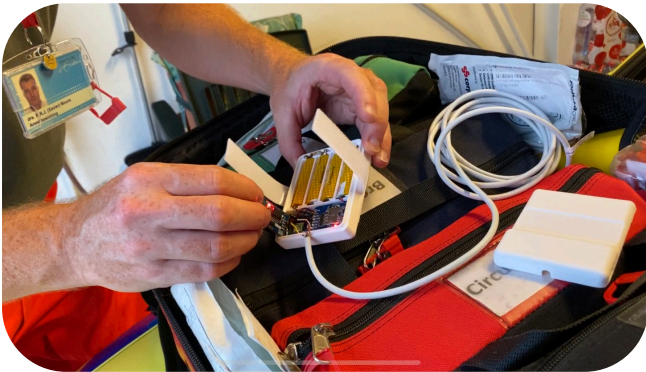
2. Take out the data collector



3. Open the lid



4. Exchange the batteries



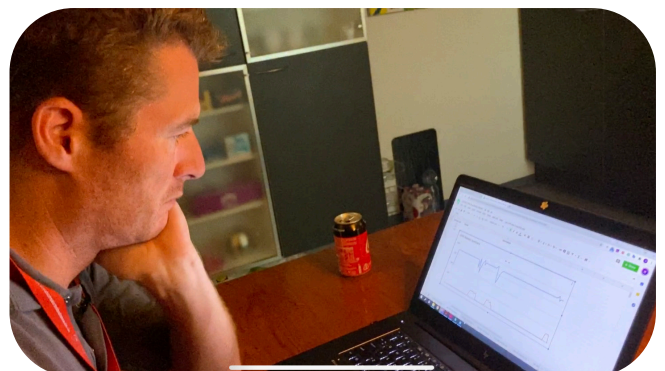
5. Check if the LED blinks



6. Close the housing of the device

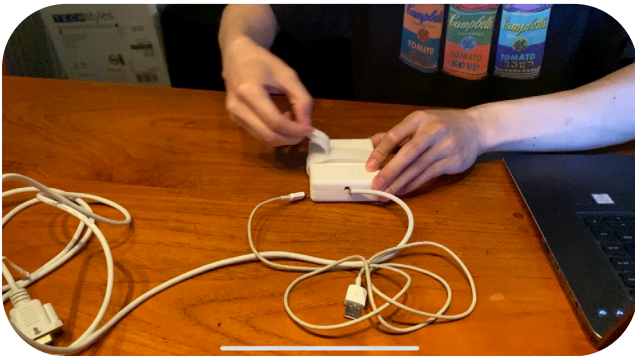


7. Put the data collector back to the rescue backpack

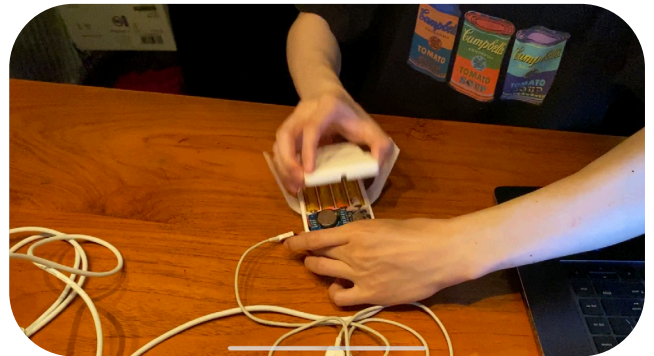


8. Review collected data in Google Sheets

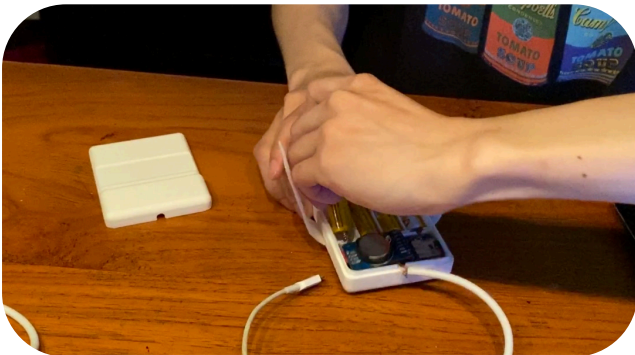
Maintenance



1. Take off the velcro band



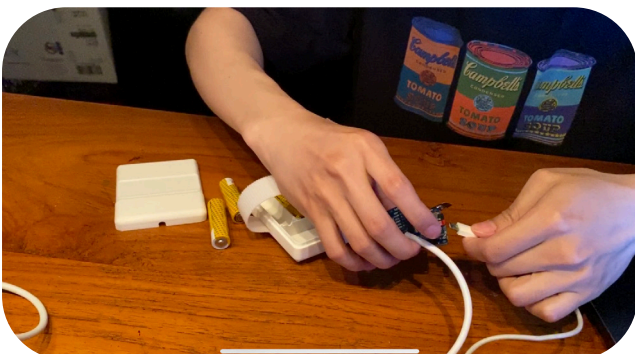
2. Take off the lid



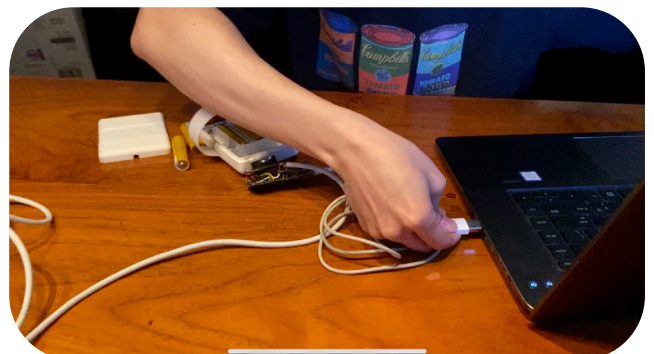
3. Take out the batteries



4. Take out the electronics



5. Connect the Arduino to a micro USB cable



6. Connect the cable to PC and check the electronics in Arduino IDE

7.3.4 Manufacture

Cost estimation

Table 7.1 shows the material cost estimation of the CPR data collector. The seven main components of a CPR data collector shown before in the exploded view are a base, a cover, a DS3231 RTC module, a micro SD card module, an Arduino MKR 1010 Wifi, four AA batteries, a Velcro band. Besides these seven, there are six more components needed to produce a fully functional device: a micro SD card, a 10k Ω resistor, a CR2032 cell battery, a 2-meter long 9-pin D-Sub cable, wires in multiple colors for circuit connection, and eight contact plates for the AA batteries.

According to Dr. Dinis, the batch size for production is 15 CPR data collectors. With this small production batch size, the base and cover can be printed by 3D printers. Based on the working prototype building, Ultimaker 2+ ⁸² printing with PLA ¹⁰³ would be a cost-effective choice. Because the printer rent price and PLA filament price is lower compared to other choices (appendix K). Plus, PLA is recyclable ¹⁰³ and still proved to be durable in the drop tests. Other parts of the device can be purchased directly from the market.

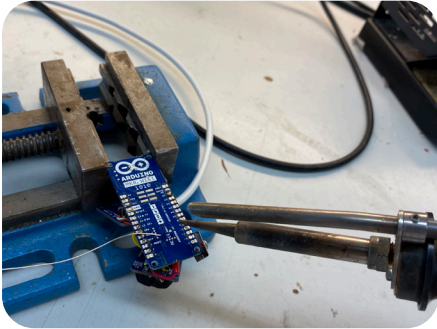
All in all, the material price to build one CPR data collector is 65.84 euros. To see the details of the calculation of the cost estimation and the purchase links of the listed components, please check appendix K.

Item NO.	Name	QTY. per device	Batch size	Total price	Bulk price
1	Base	1	15 pieces	€75	€5
2	Cover	1	15 pieces	€75	€5
3	DS3231 RTC module	1	15 pieces	€74.25	€4.95
4	Micro SD card module	1	15 pieces	€127.50	€8.50
5	SanDisk Micro SD card 16GB	1	15 pieces	€119.85	€7.99
6	Arduino MKR 1010 Wifi	1	15 pieces	€334.80	€22.32
7	10k Ω resistor	1	2 packs (20 pieces)	€2	€1
8	AA battery	4	1 pack (72 pieces)	€37.99	€2.11
9	CR2032 cell battery	1	2 packs (20 pieces)	€14	€0.70
10	Velcro band	25cm	500cm	€7.29	€0.36
11	9 pins D-Sub cable 2m	1	15 pieces	€91.20	€6.08
12	Wires	40cm	1 pack (6000cm)	€13.95	€0.01
13	Contact plates	8	3 packs (150 pieces)	€34.14	€1.82
Total material price					65.84

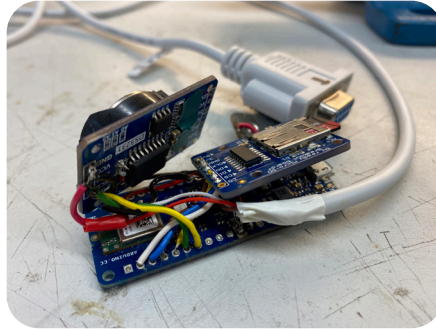
Table 7.1 Material cost estimation

Assembly

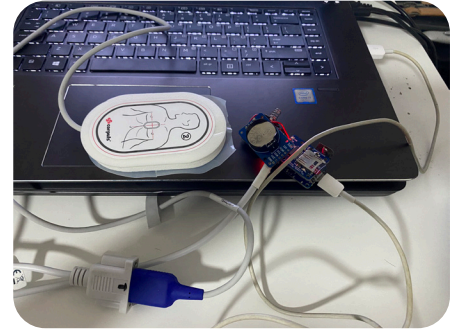
Following figures shows the assembly process of the CPR data collector.



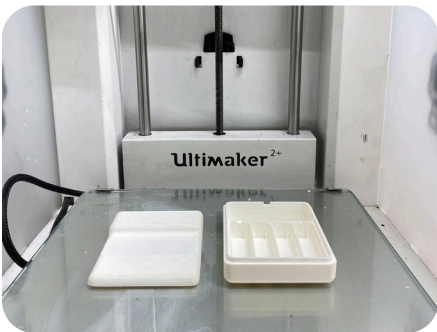
1. Solder the electronic parts together based on the designed circuit



2. Tape the exposing areas of the wires to avoid electric contact between wires



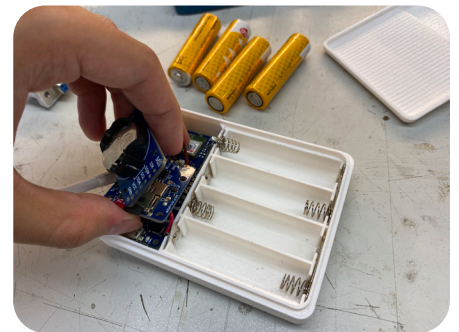
3. Connect the electronics to the computer to upload the Arduino code.



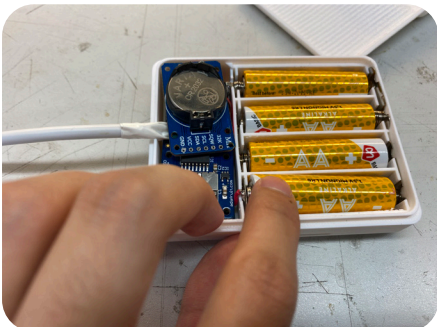
4. 3D print the housing parts



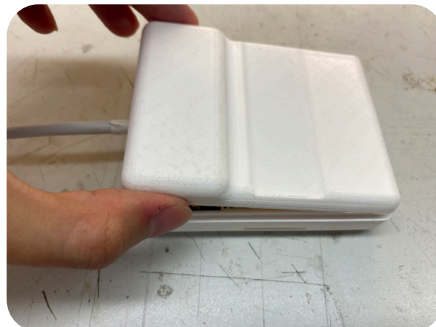
5. Insert the contact plates for batteries



6. Place the electronics inside the base housing



7. Place the AA batteries inside the base housing



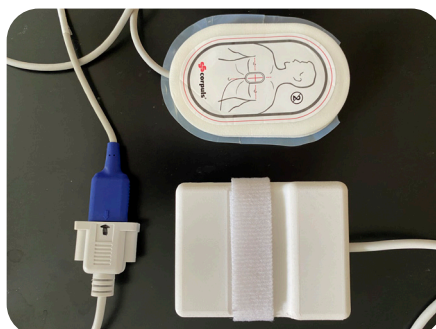
8. Close the housing with the cover



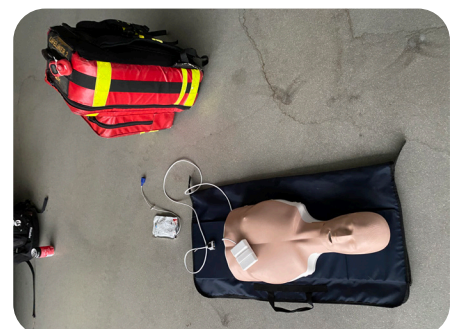
9. Insert the velcro band on the housing



10. Fasten the housing with the velcro band



11. Connect the CorPatch sample with the plug and test if the data transfer system is working



12. Conduct a simulation test in the helicopter station to ensure the device is ready to use

7.4 Final concept evaluation

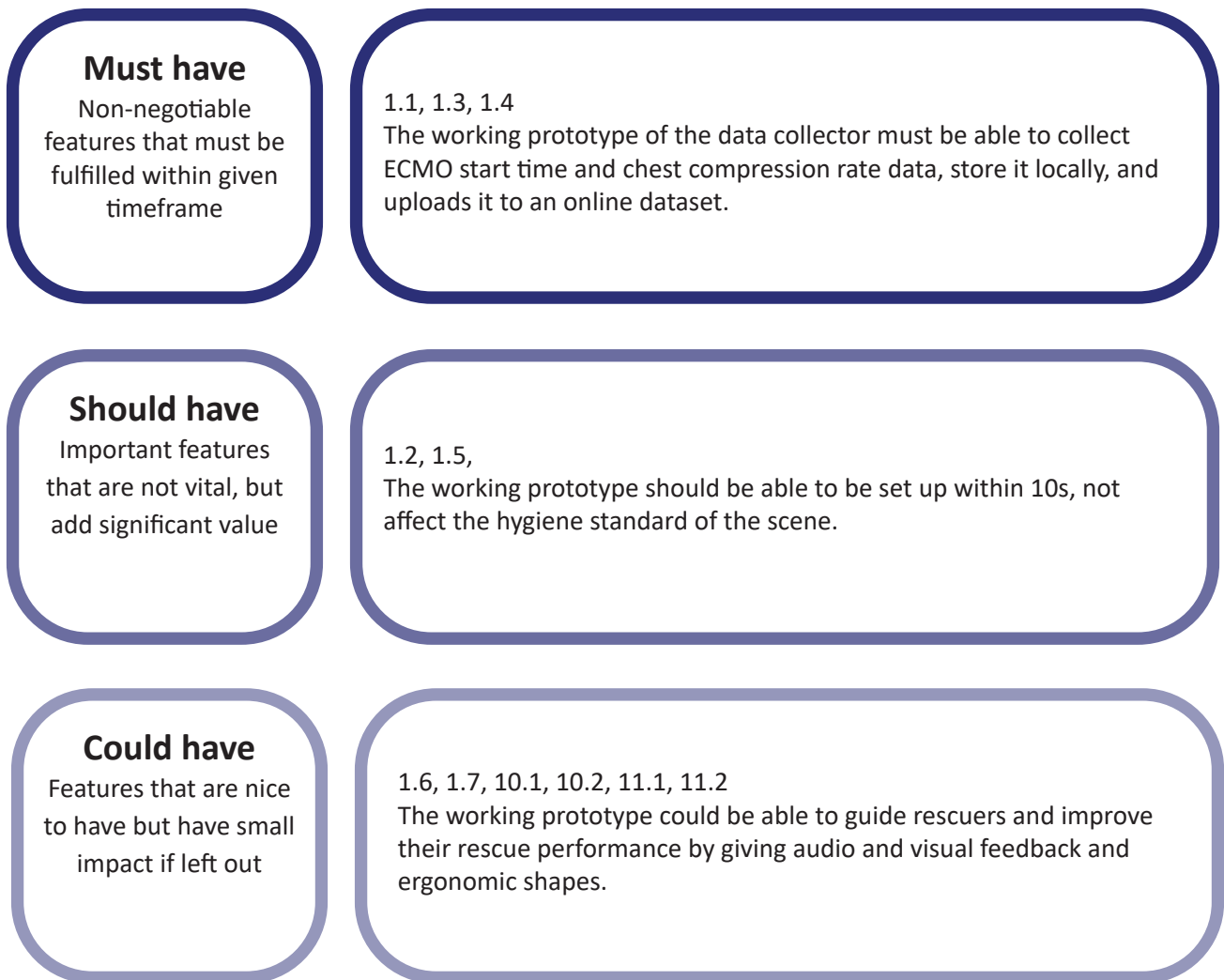


Figure 7.9 MoSCoW prioritization

To evaluate the final concept, the MoSCoW prioritization made in cycle 1 is set as the standard. As shown in figure 7.9, the final concept and its working prototype fulfills the must-have requirements. It can collect the required data, store it locally in the collector, and upload it to an online dataset - the Google Sheets. It also fulfills the should-have requirements, that it can be set up within ten seconds, its 2-meter long cable ensures that it can keep a distance from the rescue scene and will not affect the hygiene of the scene. It does not fulfill the could-have requirements because of the limitation of the hardware, as mentioned in cycle 2. In general, the CPR data collector is proved to be accurate in data collection, fast to set up, durable in OHCA rescues.

7.5 Implementation

7.5.1 Recommendations

In this section, recommendations on improving the current design of the CPR data collector are listed and elaborated. The feasibility of these recommendations needs to be verified by prototyping and testing.

Wifi connection in HEMS station

As tested in cycle 2, the Arduino cannot connect the Wifi networks in the HEMS station. Two of the networks (Eduroam and ErasmusMC) use WPA2 enterprise network, which the Arduino cannot connect to due to hardware reasons. Another one for GPS tracking uses WPA2 personal network, which can be connected. But as confirmed by Dr. Dinis, the CPR data collector is not allowed to connect due to security reasons.

In order to create a WPA2 personal Wifi network for the CPR data collector, three methods are suggested:

- One method is to create a guest Wifi ¹ network from the current router in the HEMS station. Currently, most routers can create a guest Wifi network that runs separately from the primary network. In this way, information from the main network is secured and the device connected to the guest network can also have access to the Internet.

- Another method is using a Wifi repeater ⁸⁸. A Wifi repeater can extend the Wifi signal from a Wifi router, and it is also possible that the extended Wifi signal can be set as a WPA2 personal network, which is usable for the collector. However, transferring the WPA2 enterprise signal to a less safe personal one might violate the network security regulation.

- The third method is the VirtualWifi ⁷¹ feature from the Windows system. It takes a PC as a Wifi repeater, and other devices can connect to the Wifi that the PC distributes.

It is necessary to consult the IT department for the feasibility of each method and specific implementation should also be conducted under its guidance.

Google Script function execution time limit

As mentioned in cycle 2, Google Script has a 6-minute limitation ⁶⁸ on function execution time. However, when the collected data is more than 3600 lines (1-hour data), one function that calculates the CPR cycles (CPR start, end time, period, pause, and average rate) normally needs more than six minutes to execute. When the execution is timeout, the Script will return a timeout error and stops working. There are three ways to deal with this limitation:

- The easiest way is to apply for a G Suite for Education ²² account. G Suite for Education is free and extends the function execution time to 30 minutes, which lets the function handle around 5-hour data, while the data collector normally collects around 2-hour data in an OHCA case. If G Suite for Education account is not accessible, a G Suite for Business account ²² costs 12 dollars a month and also extends the execution time to 30 minutes.

- Batching ⁴⁰. The reason that the function takes so long time to finish is because when the function extracts and fills the cells in Google Sheets, it calls the Google AppService method, which takes a longer time than other methods. During the calculation, instead of calling AppService, saving the results in an array will be much faster principally. Then call the AppService only for one time to assign all values from the array to the Google Sheets. This will save a lot of time in the function execution. So the function can calculate much more data while not breaking the time limitation

- Loop counter ⁴⁰. It is also possible to set up a loop counter in the code, to automatically recall the function every six minutes, so it will not exceed the time limitation.

Wifi RTC usage in Arduino MKR 1010 Wifi

The Arduino MKR 1010 Wifi also has an internal Real-Time Clock ¹². However, as the device may be turned off from time to time, it needs the Internet to sync its RTC, that is why an external RTC module is added to the circuit. The final concept elaborates that the device will be always on until battery exchange, which also happens in the HEMS station. So the collector can be set as syncing the time via Wifi after it is turned on, and uses the internal RTC to keep track of current time. In this way, the external RTC can be eliminated from the circuit and saves a lot of space for the device. The only drawback of this change is that a constant time tracking feature is also eliminated, and the collector cannot sync time if the Internet access is not available.

Data upload speed

Data upload speed can be further improved. According to the quotas ⁶⁸ from the Google Script, the maximum URL Fetch POST size is 50MB per call. If the URL ¹⁰⁰ call is handled with POST, not the current GET request, it has no limits on the length of the URL ⁴³, which means the URL can be as long as the Arduino's SRAM ¹⁶ memory can hold, which is 32kB ⁹. So as tested in cycle 2, one call URL call takes around five seconds, theoretically, the maximum uploading speed would be around 6.4kB/s. As calculated, one second of data is 27 bytes, so one hour of data can be uploaded within 16 seconds, if the maximum upload speed is achieved.

Interruption during data upload

What if the data upload is interrupted for some reason? The current working prototype cannot re-upload the rest of the data after Wifi is back, due to that the code does not track the upload position of the data. It can only recognize if the upload is done or not. There two conditions that interrupt the upload. One is that the device is on all the time, but the Wifi is disconnected. In this condition, it is needed to add a variable that tracks the position of the last uploaded data inside the upload function loop. Another condition is that the device is turned off and it needs to re-upload the rest of the data after being turned back on. After being turned back on, the Arduino needs to check if it has finished the former upload (state) yet, if not, it goes to check the last upload position (mark) and starts re-uploading. All these state and mark needs to be saved in the micro SD card before turned off, so the Arduino can access these data when it is on. This does make the code really complex but it is possible to realize.

2 or multiple OHCA cases in a row

What if there are 2 or more OHCA cases happens in a row? The current working prototype will record all the data during the periods that the CorPatch is plugged in. But it only uploads the last period (OHCA case) to Google Sheets, because the current code does not label saved files. To make sure the Arduino uploads all the files to Google Sheets, the code needs to mark the saved files as 'not uploaded' state, and then change it to 'uploaded' state after upload. However, this means that the code needs to check all the saved files' states in the SD card before each upload. This also adds up complexity in the code.

3D printed bumpers for anti-drop and waterproof features

As tested in cycle 3, the bumper can protect the device and enhance the device’s durability. It is cheap and exchangeable. However, its size does not match with the housing completely. Figure 7.X on the left shows a simple sketch that a 3D printed bumper not only protects the housing, but also replaces the Velcro band to hold the housing parts together. The bumper can be printed with TPU⁸¹ filament so it is flexible and can be taken on or off anytime. The bumper can also be designed to wrap the whole sides of the housing as shown on the right of figure 7.10, in this way, the gap of the housing is covered by the bumper so the waterproofness of the collector is improved.

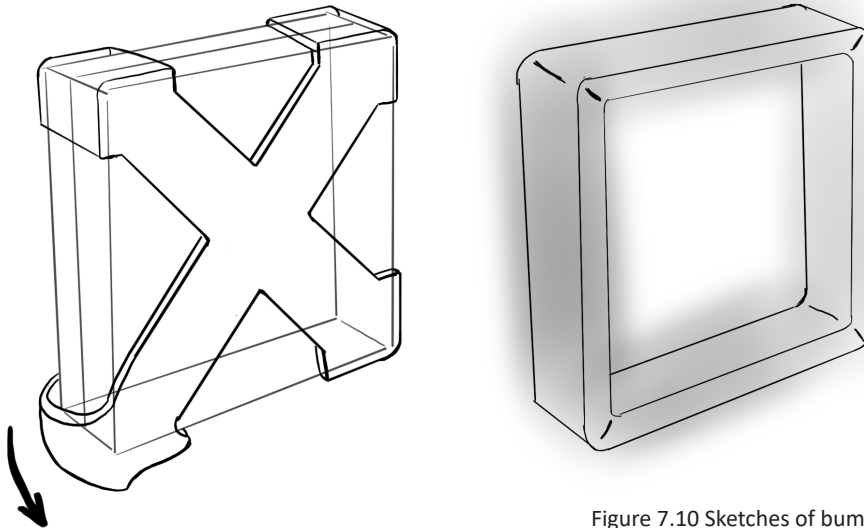


Figure 7.10 Sketches of bumper shapes

7.5.2 Implementation plan

Figure 7.11 shows the implementation plan of the CPR data collector. According to the ECPR research plan, the first research starts in Q4 of 2020 and it is conducted by HEMS team 1. Then every 2 quarters, a new HEMS team will join in. So at least three current working prototypes (version 1) of the CPR data collector should be produced, tested, and got ready before Q4 for the first HEMS team to use in the OHCA cases. Drawbacks of the device and code bugs will be collected and fixed in the version 2 of the collector. The version 2 should operate the whole data transfer system stably. So six version 2 collector will be produced and used by both HEMS team 1 and 2 during the second period. Then minor improvements can be applied to the version 3 and 4 if necessary, but only produced for the HEMS team 3 and 4. In total, 15 CPR data collectors will be produced throughout the research project.

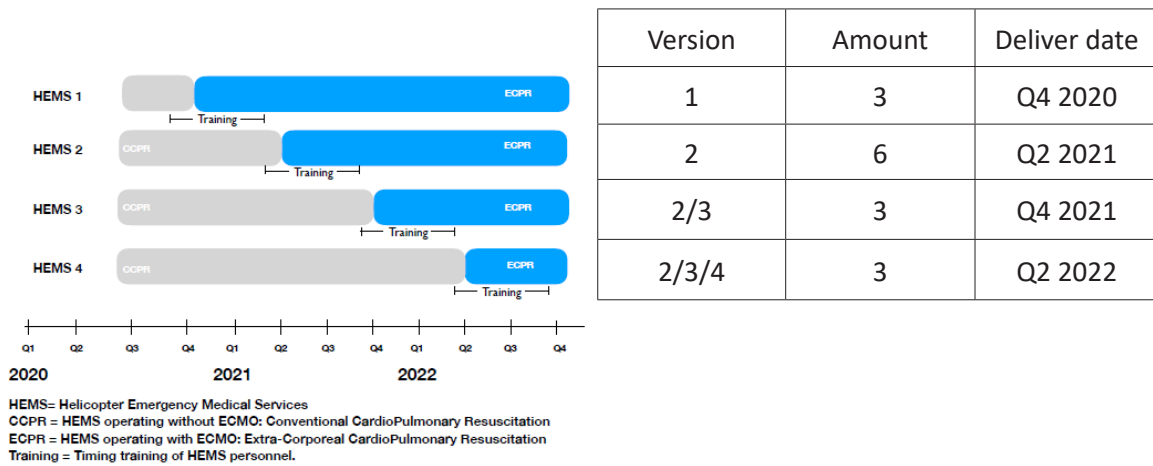


Figure 7.11 Implementation plan

08

Reflection

The reflection is written based on the rubric of the IDE master graduation project.

Knowledge (Collect and analysis, generate and evaluate)

Knowledge over OHCA rescue, HEMS team, relative technologies and examples are collected and reflected as design criteria. For the context analysis, the rescue process of the HEMS team during OHCA is analyzed. However, the midterm meeting revealed that the hygiene problem is not considered in former analysis topics. This is mainly because all the scenarios of the context are generated based on the videos and images from the documentary, TV shows or simulated scenes, where some actual scenes are hidden from the viewers. The designer overlooked that the actual scenes could be more bloody, dirty and messy. Due to the Covid situation, field observation is also not available, the designer should have asked raw video clips (Dr. Dinis once showed in the first meeting) to do more observation on the context. Fortunately during the midterm meeting, this fault is pointed out so extra research on hygiene problem is conducted. All the research topics are reflected and selected to form the preliminary design criteria.

Methods (Use of methods and tools, dealing with project complexity)

In general, the basic design cycle is the main guidance in this project. Four cycles are carried out in total. Each cycle contains a major assignment/theme, which is a core part of the whole design process. In each cycle, different methods are used to conduct analysis, ideation, prototyping and evaluation. As the end goal of this project is building a working prototype of the data collector, the complexity of this project is high. So the complexity is broken into 4 cycles (context, electronics, shapes, finalize), and each cycles' result is built on the former cycle. In this way, the designer is more focused on one aspect of problem in each cycle, so the complexity decreases.

Project result (Feasibility, desirability, viability)

The feasibility of the project result is high, as the working prototype has proved. The Wifi problem from the station is possible to be solved. The production serie is low so it can be produced with products in the market. Even though the working prototype cannot react to all different situations, as mentioned in recommendations, but the current model can do the basic jobs in actual fields, such as data collection, data upload (if Wifi is fixed). The final result meets the user's values and needs. It is a pity that the audio and visual feedback and time counter features are not implemented in the final concept. But it fulfills the fundamental need - data collection and data storage in online dataset. Plus, the use of Google Script to upload data and make the Google Sheets as the online dataset is not a common solution, but it make the later processes such as data mangement and analysis easier, more user friendly and direct.

Communication (Academic level, connecting to stakeholders)

The project proposed an different way to upload and store data directly in the Google Sheets. It also introduces using the Google Script to manage and analyze data in Google Sheets automatically. The data collector can be used to collect chest compression data with time, which provides important information to the HEMS team to evaluate their rescue performance and the effectiveness of implementing ECPR treatment in OHCA cases. Before the midterm, weekly report is handed for updating with the mentors and the clients. However, after midterm, it is pointed out that more communication is needed. After the midterm, weekly Zoom meeting with the client and the mentor is carried out, while week report is also handed out for feedbacks. This is proved to be much more effective than only updating with the week report. A lot of valuable and effective decisions are made during these meetings. Plus, after midterm,

the designer also visited the HEMS station twice for discussion on the prototype and user tests.

Project management and planning (Planning, autonomy & initiative, response to feedback, time spent)

According to the original plan, the graduation date should be around 20th of August, at present, it is one week delayed. When look back to whole process, the cycle 1 (context analysis) takes almost half of the project time to conduct, which is too long, comparing to the rest of cycles (each cycle takes one month). One of the reason is because the Covid situation limits the possibility to do prototype. Another one is because in cycle 1, some analysis topic is not focused on data collection but more on how improve chest compression, and that leads to ergonomic study, which is not a necessary feature for the data collector, neither it is the needs from the client. Thus, a method that evaluates the analysis direction for the designers could be useful in a time-tight project to save time and guide the designer to focus on more related topics. The initiative to communicate with the stakeholders is higher after the midterm, because the designer at that time has a common agreement with the client on the overall form of the design. Then the faculty is reopened for graduates, the designer becomes more engaged with prototyping and therefore would like to have more feedback of his work, the communication between the designer and the mentor increases. The designer is reactive to the feedback from the supervisory team and the client. Feedbacks are evaluated and adopted to the prototype instantly. There are certain delays comparing to the original plan, but overall the time spent is around 100 working days.

Personal ambition

When I look back the whole process of my graduation project, I am satisfied with the project result. I have overcome multiple problems that I think I am not able to solve at the beginning. I am satisfied that I deepened my knowledge on intensive care domain, developed my prototyping skills on connected products throughout the project. I fulfilled my personal ambitions that I mentioned in the project brief. I sincerely thanks the supervisory team - Prof. Goossens, Prof. van Heur and Dr. Reis Miranda for their assistance, guidance, all the open discussion over the project, and the caring. I expect to grow up to a more mature and professional designer in the future and make more contribution to the design community and society.

24/08/2020
Yu Zhang

09

References

1. A. (2019a, December 11). Guest WiFi: What is it? How do I set it up? | Learn More. Actiontec.Com. <https://www.actiontec.com/wifihelp/guest-wifi-what-is-it-why-do-i-want-it-how-to-use-it-how-to-get-it/>
2. Adafruit NeoPixel Überguide. (2013, August 30). Adafruit NeoPixel Überguide. <https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use>
3. Advanced Circulatory. (n.d.). ResQCPR System Instruction For Use. Advanced Circulatory.
4. A.E. Tomlinson, J. Nysaether, et al. (2006). Compression force—depth relationship during out-of-hospital cardiopulmonary resuscitation.
5. American Red Cross Training Services. (2020). How to Perform Hands-Only CPR | Red Cross. Red Cross Training & Certification, and Store. <https://www.redcross.org/take-a-class/cpr/performing-cpr/hands-only-cpr>
6. API. (n.d.). Pushingbox. <https://www.pushingbox.com/api.php>
7. Arduino. (2020). SMTP2GO. <https://www.smtp2go.com/setupguide/arduino/>
8. Arduino - HomePage. (2020). Arduino. <https://www.arduino.cc/en/loT/HomePage>
9. Arduino - Memory. (n.d.). Arduino. <https://www.arduino.cc/en/tutorial/memory>
10. Arduino - WiFinINABeginEnterprise. (n.d.). Arduino. <https://www.arduino.cc/en/Reference/WiFinINABeginEnterprise>
11. Arduino - WiFinINAWiFiWebClient. (2018). Arduino. <https://www.arduino.cc/en/Tutorial/WiFinINAWiFiWebClient>
12. Arduino - WiFiRTC. (n.d.). Arduino. <https://www.arduino.cc/en/Tutorial/WiFiRTC>
13. Arduino IoT Cloud. (2020). Arduino. <https://www.arduino.cc/en/loT/HomePage>
14. Arduino IoT Cloud Google Sheets Integration. (n.d.). Arduino Project Hub. https://create.arduino.cc/projecthub/Arduino_Genuino/arduino-iot-cloud-google-sheets-integration-71b6bc
15. Arduino MKR GSM 1400. (2020). Arduino. <https://store.arduino.cc/arduino-mkr-gsm-1400-1415>
16. Arduino MKR WiFi 1010 | Arduino Official Store. (2020). Arduino. <https://store.arduino.cc/arduino-mkr-wifi-1010>
17. Arduino Nano 33 IoT | Arduino Official Store. (2020). Arduino. <https://store.arduino.cc/arduino-nano-33-iot>
18. Arduino_Genuino. (2019, April 3). Control Two Relays Over the Internet. Hackster.io. https://www.hackster.io/Arduino_Genuino/control-two-relays-over-the-internet-751138
19. AZDelivery Real Time Clock RTC DS3231 I2C Real time klok voor Arduino met eBook: Amazon.nl. (n.d.). Amazon.Nl. https://www.amazon.nl/gp/product/B01M2B7HQB/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1
20. Battery Capacity. (n.d.). Techlib. <http://www.techlib.com/reference/batteries.html>
21. Bayer Material Science. (n.d.). Snap-fit joints for plastics. http://fab.cba.mit.edu/classes/S62.12/people/vernelle.noel/Plastic_Snap_fit_design.pdf
22. Betalingsplannen | G Suite. (n.d.). Google. <https://gsuite.google.com/pricing.html>
23. Blackler, A., & Popovic, V. (2015). Towards Intuitive Interaction Theory. *Interacting with Computers*, 27(3), 203–209. <https://doi.org/10.1093/iwc/iwv011>
24. Cheskes, S., Schmicker, R. H., Rea, T., Powell, J., Drennan, I. R., Kudenchuk, P., Vaillancourt, C., Conway, W., Stiell, I., Stub, D., Davis, D., Alexander, N., & Christenson, J. (2015). Chest compression fraction: A time dependent variable of survival in shockable out-of-hospital cardiac arrest. *Resuscitation*, 97, 129–135. <https://doi.org/10.1016/j.resuscitation.2015.07.003>
25. Corpatch - CorPatch is the solution that increases the quality of bystander CPR in case of an out of hospital cardiac arrest. (2020, May 6). Corpatch. <https://corpatch.com/>
26. CORPULS CPR. (n.d.). <https://Corpuls.World/En/Products/Corpuls-Cpr/>. <https://corpuls.world/en/products/corpuls-cpr/>
27. D. (2019b, December 15). Arduino and DS3231 Real Time Clock Tutorial. *HowToMechatronics*. <https://howtomechatronics.com/tutorials/arduino/arduino-ds3231-real-time-clock-tutorial/>
28. Duracell batterij 72 pack AA. (n.d.). Bol.Com. https://www.bol.com/nl/p/duracell-batterij-72-pack-aa/9200000074443115/?Referer=ADVNLGOO002018-G-58263947517-S-493987207082-9200000074443115&gclid=Cj0KCQjw-O35BRDVARIsAJU5mQXBfA8yNOgvndlxoUct4dugEVn16sWYt8vuOUcybdNWKqzj8Mq-bFMaAsUXEALw_wcB

29. E. (2019c). Embedotronics/Attendance-System-with-storing-Data-on-Google-Spreadsheet-using-RFID-and-Arduino-Ethernet-Shield. GitHub. <https://github.com/Embedotronics/Attendance-System-with-storing-Data-on-Google-Spreadsheet-using-RFID-and-Arduino-Ethernet-Shield>
30. Engineers, L. M. (2019, December 13). How Accelerometer works? Interface ADXL335 with Arduino. Last Minute Engineers. <https://lastminuteengineers.com/adxl335-accelerometer-arduino-tutorial/>
31. Executing Functions using the Apps Script API | . (n.d.). Google Developers. <https://developers.google.com/apps-script/api/how-tos/execute>
32. Extending Google Sheets | Apps Script | . (2019). Google Developers. <https://developers.google.com/apps-script/guides/sheets>
33. F. (2018a). firedog1024/mkr1000-iotc. GitHub. <https://github.com/firedog1024/mkr1000-iotc>
34. Fuse Reel Innovative Tools For Tech and Cable Management. (n.d.). Fuse Reels. <https://fusereel.com/>
35. Gammon Forum : Electronics : Microprocessors : Power saving techniques for microprocessors. (2012). Gammon. <https://www.gammon.com.au/power>
36. GETINGE. (2019, July). Cardiohelp System Extracorporeal life support wherever you need it. Maquet Cardiopulmonary GmbH. https://www.getinge.com/dam/hospital/documents/marketing-sales/brochures/english/cardiohelp_system_brochure-en-non_us_japan.pdf
37. GL.iNet GL-AR150-3 Mini Travel Router, wifi-converter, OpenWrt pre-installed, Repeater Bridge, 150 Mbps High Performance, OpenVPN, Wireguard, Programable IoT Gateway: Amazon.nl. (n.d.). Amazon. https://www.amazon.nl/GL-iNet-wifi-converter-pre-installed-Performance-Programable/dp/B015CYDVG8/ref=asc_df_B015CYDVG8/?tag=nlshogostdde-21&linkCode=df0&hvadid=430533494283&hvpos=&hvnetw=g&hvrand=4626464473184909334&hvpone=&hvptwo=&hvqmt=&hvdc=c&hvdvcmld=&hvllocint=&hvllocphy=1010704&hvtargid=pla-298850372584&pvc=1
38. González-Otero, D. M., Ruiz, J. M., Ruiz de Gauna, S., Gutiérrez, J. J., Daya, M., Russell, J. K., Azcarate, I., & Leturiondo, M. (2018). Monitoring chest compression quality during cardiopulmonary resuscitation: Proof-of-concept of a single accelerometer-based feedback algorithm. *PLOS ONE*, 13(2), e0192810. <https://doi.org/10.1371/journal.pone.0192810>
39. Goobay 45743 USB-kabel 1 m 2.0 USB A USB C Zwart. (n.d.). Bol. https://www.bol.com/nl/p/goobay-45743-usb-kabel-1-m-2-0-usb-a-usb-c-zwart/9200000104843023/?bltgh=ldBeO-iX6dkT5S95ZypFfQ.1_4.5.ProductTitle
40. Google app script timeout ~ 5 minutes? (2013a, January 22). Stack Overflow. <https://stackoverflow.com/questions/14450819/google-app-script-timeout-5-minutes>
41. Gruber, J., Stumpf, D., Zapletal, B., Neuhold, S., & Fischer, H. (2012). Real-time feedback systems in CPR. *Trends in Anaesthesia and Critical Care*, 2(6), 287–294. <https://doi.org/10.1016/j.tacc.2012.09.004>
42. GS Elektromedizinische Geräte G. Stemple GmbH. (2020). User Manual corpuls3. GS Elektromedizinische Geräte G. Stemple GmbH.
43. HTTP Methods GET vs POST. (n.d.). W3schools. https://www.w3schools.com/tags/ref_httpmethods.asp
44. I. (2011, May 27). Topology Optimization Guide | Your source for Topology Optimization world | Page 6. Topology-Opt. <http://www.topology-opt.com/page/6/>
45. Industries, A. (n.d.). Adafruit Mono 2.5W Class D Audio Amplifier - PAM8302. Adafruit. <https://www.adafruit.com/product/2130>
46. JavaScript. (2020, July 9). MDN Web Docs. [https://developer.mozilla.org/en-US/docs/Web/JavaScript#:~:text=JavaScript%20\(JS\)%20is%20a%20lightweight,Apache%20CouchDB%20and%20Adobe%20Acrobat.](https://developer.mozilla.org/en-US/docs/Web/JavaScript#:~:text=JavaScript%20(JS)%20is%20a%20lightweight,Apache%20CouchDB%20and%20Adobe%20Acrobat.)
47. KabelDirekt - klittenband kabelbinders hersluitbaar - 20 mm x 5 m - (rol voor kabels, vrij op maat te snijden & herbruikbaar, wit): Amazon.nl. (n.d.). Amazon.Nl. https://www.amazon.nl/gp/product/B07BYLY15R/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&pvc=1
48. Kanis, H., Rooden, MJ., & Green, WS. (2000). Usecases in the Delft design course. In P. T. McCabe, M. A. Hanson, & S. A. Robertson (Eds.), *Contemporary ergonomics 2000* (pp. 365-369). Taylor & Francis.
49. Kiwi Electronics. (n.d.-a). Arduino MKR WiFi 1010. <https://www.kiwi-electronics.nl/arduino-mkr-wifi-1010?search=MKR1010&description=true>
50. Kiwi Electronics. (n.d.-b). MicroSD card breakout board+. <https://www.kiwi-electronics.nl/microsd-card-breakout-board-plus?search=micro%20sd&description=true>
51. Kiwi Electronics. (n.d.-c). Weerstand 10K Ohm - 1/4 watt - 5% - 10 stuks. <https://www.kiwi-electronics.nl/Weerstand-10K-ohm-1-4-watt-5-procent-10-stuks?search=resistor%2010k&description=true>
52. Kleinman ME, Brennan EE, Goldberger ZD, Swor RA, Terry M, Bobrow BJ, et al. (2015). Part 5: adult basic life support and cardiopulmonary resuscitation quality: 2015 American Heart Association guidelines update for cardiopulmonary resuscitation and emergency cardiovascular care. *Circulation* 2015;132(18 Suppl. 2):S414–35.
53. Kovic, I., Lulic, D., & Lulic, I. (2013). CPR PRO® Device Reduces Rescuer Fatigue during Continuous Chest Compression Cardiopulmonary Resuscitation: A Randomized Crossover Trial Using a Manikin Model. *The Journal of Emergency Medicine*, 45(4), 570–577. <https://doi.org/10.1016/j.jemermed.2013.04.021>
54. Lady ada. (2020). Micro SD Card Breakout Board Tutorial. Learn.Adafruit.Com. <https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial/look-out>

55. LastMinuteEngineers. (2019a, December 13). Interface DS3231 Precision RTC Module with Arduino. LastMinuteEngineers. <https://lastminuteengineers.com/ds3231-rtc-arduino-tutorial/#:%7E:text=Assuming%20a%20fully%20charged%20CR2032,an%20external%205V%20power%20supply>.
56. Last Minute Engineers. (2019b, December 13). How Accelerometer works? Interface ADXL335 with Arduino. Last Minute Engineers. <https://lastminuteengineers.com/adxl335-accelerometer-arduino-tutorial/>
57. LevelGloves. (2017). Biomex Protection. LevelGloves. <https://www.levelgloves.com/biomex-protection/>
58. M. (2018b). mobizt/Firebase-Arduino-WiFiNINA. GitHub. <https://github.com/mobizt/Firebase-Arduino-WiFiNINA>
59. Oberhaching, D. K. G. E. |. |. (2020). EnOcean for Europe: 868 MHz - ECO 200. EnOcean. https://www.enocean.com/en/products/enocean_modules/eco-200/
60. Pappalardo, F., & Montisci, A. (2017). What is extracorporeal cardiopulmonary resuscitation? *Journal of Thoracic Disease*, 9(6), 1415–1419. <https://doi.org/10.21037/jtd.2017.05.33>
61. Perkins GD, Handley AJ, Koster RW, Castrén M, Smyth MA, Olasveengen T, et al. (2015). European Resuscitation Council Guidelines for Resuscitation 2015: Section 2. Adult basic life support and automated external defibrillation. *Resuscitation* 2015;95:81–99.
62. Physio Control. (2020). LUCAS 3 Chest Compression System Instruction for Use (Physio Control ed.). Physio Control.
63. Pillai, A. K., Bhatti, Z., Bosserman, A. J., Mathew, M. C., Vaidehi, K., & Kalva, S. P. (2018). Management of vascular complications of extra-corporeal membrane oxygenation. *Cardiovascular Diagnosis and Therapy*, 8(3), 372–377. <https://doi.org/10.21037/cdt.2018.01.11>
64. PocketCPR. (n.d.). [www.Baycomp.Com](https://baycomp.com/pocketcpr/). <https://baycomp.com/pocketcpr/>
65. Premium seriële RS232 kabel 9-pins SUB-D (m) - 9-pins SUB-D (m) / gegoten connectoren - 2 meter - 9-pin SUB-D (RS232/RS485) - SUB-D (9p/15p/25p) - Computer | Onlinekabelshop.nl. (n.d.). Onlinekabelshop. <https://www.onlinekabelshop.nl/seriele-kabel-9pins-sub-d-mannelijk-9pins-sub-d-mannelijk-2-meter>
66. Prijzen – Goedkoop 3D printer huren. (n.d.). [Goedkoop3dprinterhuren](http://goedkoop3dprinterhuren.nl/prijzen/). <http://goedkoop3dprinterhuren.nl/prijzen/>
67. Q-CPR measurement and feedback tool CPR meter | Philips Healthcare. (2005). Philips. <https://www.usa.philips.com/healthcare/product/HCNOCTN89/qcpr-measurement-and-feedback-tool-cpr-meter>
68. Quotas for Google Services | Apps Script |. (n.d.). Google Developers. <https://developers.google.com/apps-script/guides/services/quotas>
69. Oberhaching, D. K. G. E. |. |. (2020). EnOcean for Europe: 868 MHz - ECO 200. EnOcean. https://www.enocean.com/en/products/enocean_modules/eco-200/
70. Sandisk SDSQDQ-016G-B35 flashgeheugen 16 GB MicroSDHC. (n.d.). Bol.Com. <https://www.bol.com/nl/p/sandisk-sdsdqm-016g-b35-flashgeheugen-16-gb-microsdhc/9000000012279555/?s2a=#productTitle>
71. Sandler, E. (2020, June 18). How to: Create Wireless Hosted Networks in Windows 7. Wi-FiPlanet.Com. <https://www.wi-fiplanet.com/how-to-create-wireless-hosted-networks-in-windows-7/>
72. Seifert, K., & Camacho, O. (2007). Implementing Positioning Algorithms Using Accelerometers. *Freemove Semiconductor*, 1. <https://www.nxp.com/docs/en/application-note/AN3397.pdf>
73. Send MKR1000 Data to Google Sheets. (2016). Arduino Project Hub. <https://create.arduino.cc/projecthub/detox/send-mkr1000-data-to-google-sheets-1175ca>
74. Sheets Tips - Google Sheets Tips and Tricks | G Suite Tips. (n.d.). [Gsuitetips](https://gsuitetips.com/tips/sheets/google-spreadsheet-limitations/). <https://gsuitetips.com/tips/sheets/google-spreadsheet-limitations/>
75. Smart Plug with Arduino MKR WiFi 1010. (2020). Arduino. https://create.arduino.cc/projecthub/Avilmaru/smart-plug-with-arduino-mkr-wifi-1010-63cb25?ref=tag&ref_id=ifttt&offset=4
76. Snap Fit Design. (n.d.). Gotstogo. http://www.gotstogo.com/misc/engineering_info/snap_design.htm
77. Spring 100st Battery Battery Shrapnel AA of AAA-batterij Spring 7 No. positieve en negatieve Contact Stukken 50pairs Drop Ship Easy to install: Amazon.nl. (n.d.). Amazon.Nl. https://www.amazon.nl/gp/product/B08DHRCL6/ref=ox_sc_act_title_2?smid=A2GWW94JYD74WR&pvc=1
78. Strain Reliefs Selection Guide | Engineering360. (n.d.). Globalspec. https://www.globalspec.com/learnmore/electrical_electronic_components/wires_cables_accessories/strain_reliefs#:~:text=Strain%20relief%20types%20include%20cable,piece%20of%20equipment%20or%20device.
79. SuperCalla | Charging / Data Cables Redesigned. (2020, August 3). Kickstarter. https://www.kickstarter.com/projects/1813437441/supercalla-charging-cables-redesigned?ref=project_link
80. Taylor, E. (2017, July 18). Turbocharge your HEMS start-up times. *AirMed&Rescue*. <https://www.airmedandrescue.com/latest/long-read/turbocharge-your-hems-start-times>
81. TPU filament - learn everything about the TPU material for 3D printing. (2020, July 13). Tractus3D. [https://tractus3d.com/materials/tpu#:~:text=Thermoplastic%20Polyurethane%20\(TPU%20material\)%20is,a%20flexible%20abrasion%20resistant%20thermoplastic.&text=3D%20printed%20parts%20with%20TPU,is%20resistant%20to%20many%20chemicals](https://tractus3d.com/materials/tpu#:~:text=Thermoplastic%20Polyurethane%20(TPU%20material)%20is,a%20flexible%20abrasion%20resistant%20thermoplastic.&text=3D%20printed%20parts%20with%20TPU,is%20resistant%20to%20many%20chemicals).
82. Ultimaker 2+: Robust single extrusion. (n.d.). Ultimaker.Com. <https://ultimaker.com/3d-printers/ultimaker-2-plus>
83. USE a BUZZER MODULE (PIEZO SPEAKER) USING ARDUINO UNO. (2018). Arduino Project Hub. <https://create.arduino.cc/projecthub/SURYATEJA/use-a-buzzer-module-piezo-speaker-using-arduino-uno-89df45>

84. VARTA CR2032 lithium knoopcellen 3V batterij in originele blisterverpakking, 10-pack: Amazon.nl. (n.d.). Amazon.NL. https://www.amazon.nl/gp/product/B018S4PTNW/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&pvc=1
85. VELLEMAN - K/MOWM Schakeldraad-assortiment - 10 kleuren - 60 m - 0,2 mm² volledig materiaal 276230: Amazon.nl. (n.d.). Amazon.NL. https://www.amazon.nl/VELLEMAN-Schakeldraad-assortiment-kleuren-voledig-materiaal/dp/B0011RVDV4/ref=sr_1_2?__mk_nl_NL=%C3%85M%C3%85C5%BD%C3%95%C3%91&dchild=1&keywords=arduino+wire&qid=1597746429&sr=8-2
86. Visaton 0.5W Miniature Speaker 16mm Dia. (n.d.). RS Components. [https://nl.rs-online.com/web/p/miniature-speakers/1218640?cm_mmc=Nl-PLA-DS3A-_-google-_-CSS_NL_NL_Passive_Components_Whoop-_\(NL:Whoop!\)+Miniature+Speakers-_-1218640&matchtype=&aud=772940708119:pla-319119479250&gclid=CjwKCAjwsan5BRAOEiwALzomX2mrePhLYdy2KP5R0x_ppf2b3V8Yd67z6p3hK0dFX05p6ygPrJrP1x0Cb-8QAvD_BwE&gclid=aw.ds](https://nl.rs-online.com/web/p/miniature-speakers/1218640?cm_mmc=Nl-PLA-DS3A-_-google-_-CSS_NL_NL_Passive_Components_Whoop-_(NL:Whoop!)+Miniature+Speakers-_-1218640&matchtype=&aud=772940708119:pla-319119479250&gclid=CjwKCAjwsan5BRAOEiwALzomX2mrePhLYdy2KP5R0x_ppf2b3V8Yd67z6p3hK0dFX05p6ygPrJrP1x0Cb-8QAvD_BwE&gclid=aw.ds)
87. What is MoSCoW Prioritization? | Overview of the MoSCoW Method. (2020, July 10). Productplan. <https://www.productplan.com/glossary/moscow-prioritization/>
88. What's the difference: WiFi Booster, Repeater or Extender? (n.d.). Waveform. <https://www.waveform.com/pages/wifi-booster-repeater-extender-differences>
89. Wikipedia contributors. (2001, October 29). Hypertext Transfer Protocol. Wikipedia. https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
90. Wikipedia contributors. (2020a, January 19). IC power-supply pin. Wikipedia. https://en.wikipedia.org/wiki/IC_power-supply_pin
91. Wikipedia contributors. (2020b, May 12). Voltage optimisation. Wikipedia. https://en.wikipedia.org/wiki/Voltage_optimisation#:~:text=The%20higher%20the%20voltage%20the,job%20due%20to%20atmospheric%20losses.
92. Wikipedia contributors. (2020c, May 31). Comma-separated values. Wikipedia. https://en.wikipedia.org/wiki/Comma-separated_values
93. Wikipedia contributors. (2020d, July 3). Microcontroller. Wikipedia. <https://en.wikipedia.org/wiki/Microcontroller>
94. Wikipedia contributors. (2020e, July 5). D-subminiature. Wikipedia. <https://en.wikipedia.org/wiki/D-subminiature>
95. Wikipedia contributors. (2020f, July 10). Thread (computing). Wikipedia. [https://en.wikipedia.org/wiki/Thread_\(computing\)](https://en.wikipedia.org/wiki/Thread_(computing))
96. Wikipedia contributors. (2020g, July 15). MagSafe. Wikipedia. <https://en.wikipedia.org/wiki/MagSafe>
97. Wikipedia contributors. (2020h, July 23). AA battery. Wikipedia. [https://en.wikipedia.org/wiki/AA_battery#:~:text=An%20AA%20cell%20measures%2049.2,%E2%80%930.57%20in\)%20in%20diameter.](https://en.wikipedia.org/wiki/AA_battery#:~:text=An%20AA%20cell%20measures%2049.2,%E2%80%930.57%20in)%20in%20diameter.)
98. Wikipedia contributors. (2020i, July 23). I2C. Wikipedia. <https://en.wikipedia.org/wiki/I%C2%B2C>
99. Wikipedia contributors. (2020j, August 2). Wi-Fi Protected Access. Wikipedia. https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access
100. Wikipedia contributors. (2020l, August 4). URL. Wikipedia. <https://en.wikipedia.org/wiki/URL>
101. Wikipedia contributors. (2020m, August 6). Velcro. Wikipedia. <https://en.wikipedia.org/wiki/Velcro>
102. Wikipedia contributors. (2020n, August 9). Silicone rubber. Wikipedia. https://en.wikipedia.org/wiki/Silicone_rubber
103. Wikipedia contributors. (2020o, August 12). Polylactic acid. Wikipedia. https://en.wikipedia.org/wiki/Polylactic_acid
104. Wikipedia contributors. (2020p, August 4). Serial Peripheral Interface. Wikipedia. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
105. X Series Monitor/Defibrillator for EMS - ZOLL Medical. (n.d.). [Www.Zoll.Com. https://www.zoll.com/products/defibrillators/x-series-for-ems](https://www.zoll.com/products/defibrillators/x-series-for-ems)
106. Zijlstra, J. (2020). Delft Design Guide (revised edition): Perspectives - Models - Approaches - Methods (Revised ed.). BIS Publishers.
107. Zoll. (2018). Zoll AutoPulse Resuscitation System Model 100 User Guide. Zoll.
108. ZOLL CPR-D-padz CPR Electrodes - Automated External Defibrillator Pads. (n.d.). [Www.Zoll.Com. https://www.zoll.com/medical-products/defibrillator-electrodes/cpr-d-padz-aed](https://www.zoll.com/medical-products/defibrillator-electrodes/cpr-d-padz-aed)

10

Appendix

List of appendices

A	Graduation project brief
B	ON-SCENE ECPR study RESEARCH PROTOCOL (March 2020) (Confidential)
C	Full activity timeline of HEMS crew in OHCA
D	Interview with Dr. Dinis Reis Miranda
E	CorPatch plug pinout (Confidential)
F	Arduino IoT cloud code
G	Arduino code of NeoPixel, buzzer and mini speaker
H	Final Google Script code
I	Final Arduino code: WPA2 personal version and WPA2 enterprise version
J	Solidworks assembly and parts drawings
K	Purchase links of the cost estimation
L	Final Solidworks model files
M	3D printed models and the working prototype
N	User test videos

IDE Master Graduation

Project team, Procedural checks and personal Project brief

This document contains the agreements made between student and supervisory team about the student's IDE Master Graduation Project. This document can also include the involvement of an external organisation, however, it does not cover any legal employment relationship that the student and the client (might) agree upon. Next to that, this document facilitates the required procedural checks. In this document:

- The student defines the team, what he/she is going to do/deliver and how that will come about.
- SSC E&SA (Shared Service Center, Education & Student Affairs) reports on the student's registration and study progress.
- IDE's Board of Examiners confirms if the student is allowed to start the Graduation Project.

! USE ADOBE ACROBAT READER TO OPEN, EDIT AND SAVE THIS DOCUMENT

Download again and reopen in case you tried other software, such as Preview (Mac) or a webbrowser.

STUDENT DATA & MASTER PROGRAMME

Save this form according the format "IDE Master Graduation Project Brief_familyname_firstname_studentnumber_dd-mm-yyyy". Complete all blue parts of the form and include the approved Project Brief in your Graduation Report as Appendix 1 !



family name Zhang
 initials Y. given name Yu
 student number 4363515
 street & no. _____
 zipcode & city _____
 country _____
 phone _____
 email _____

Your master programme (only select the options that apply to you):

IDE master(s): IPD Dfi SPD

2nd non-IDE master: _____

individual programme: _____ (give date of approval)

honours programme: Honours Programme Master

specialisation / annotation: Medisign

Tech. in Sustainable Design

Entrepreneurship

SUPERVISORY TEAM **

Fill in the required data for the supervisory team members. Please check the instructions on the right !

** chair Prof.dr.ir. R.H.M. Goossens dept. / section: Human Centered/ AED
 ** mentor Ir. R.J.H.G. van Heur dept. / section: Human Centered/ AED
 2nd mentor Dinis Reis Miranda MD, PhD
 organisation: Erasmus University Medical Center
 city: Rotterdam country: The Netherlands

Chair should request the IDE Board of Examiners for approval of a non-IDE mentor, including a motivation letter and c.v..



Second mentor only applies in case the assignment is hosted by an external organisation.



Ensure a heterogeneous team. In case you wish to include two team members from the same section, please explain why.

comments
(optional)

⋮

Procedural Checks - IDE Master Graduation

APPROVAL PROJECT BRIEF

To be filled in by the chair of the supervisory team.

rgoo
ssen
Digitally signed by rgoossens
Date: 2020.03.25 11:49:28 +01'00'

chair Prof.dr.ir. R.H.M. Goossens date 25 - 03 - 2020 signature S

CHECK STUDY PROGRESS

To be filled in by the SSC E&SA (Shared Service Center, Education & Student Affairs), after approval of the project brief by the Chair. The study progress will be checked for a 2nd time just before the green light meeting.

Master electives no. of EC accumulated in total: 15 EC

Of which, taking the conditional requirements into account, can be part of the exam programme 15 EC

List of electives obtained before the third semester without approval of the BoE

YES all 1st year master courses passed

NO missing 1st year master courses are:

name _____ date 27 3 - 2020 signature CB

FORMAL APPROVAL GRADUATION PROJECT

To be filled in by the Board of Examiners of IDE TU Delft. Please check the supervisory team and study the parts of the brief marked **. Next, please assess, (dis)approve and sign this Project Brief, by using the criteria below.

- Does the project fit within the (MSc)-programme of the student (taking into account, if described, the activities done next to the obligatory MSc specific courses)?
- Is the level of the project challenging enough for a MSc IDE graduating student?
- Is the project expected to be doable within 100 working days/20 weeks ?
- Does the composition of the supervisory team comply with the regulations and fit the assignment ?

Content: APPROVED NOT APPROVED

Procedure: APPROVED NOT APPROVED

- do not use abbreviations in title

comments

name Monique von Morgen date 14-04-2020 signature MvM

Design of a data collector for HEMS crew during OHCA

project title

Please state the title of your graduation project (above) and the start date and end date (below). Keep the title compact and simple. Do not use abbreviations. The remainder of this document allows you to define and clarify your graduation project.

start date 01 - 04 - 2020

18 - 08 - 2020 end date

INTRODUCTION **

Please describe, the context of your project, and address the main stakeholders (interests) within this context in a concise yet complete manner. Who are involved, what do they value and how do they currently operate within the given context? What are the main opportunities and limitations you are currently aware of (cultural- and social norms, resources (time, money,...), technology, ...).

The helicopter emergency medical services (HEMS) involves teams of highly specialized personnel, which are deployed very quickly. The HEMS are active over a wide area and are still able to deploy teams to the scene within 11 min of the initial emergency call.

Neurological recovery after out-of-hospital cardiac arrest (OHCA) is mainly dependent on the duration of the arrest. Some hospitals restore circulation with extracorporeal cardio-pulmonary resuscitation (E-CPR) administered on a patient's arrival at the hospital during refractory OHCA. In E-CPR, a miniaturized cardiopulmonary bypass system (similar to that used in open-heart surgery) replaces cardiac and pulmonary function and provides full adequate circulatory support to the body. To this end, large-bore cannulas are inserted into the inguinal artery and vein and are connected to the bypass system. Transporting patients in cardiac arrest to the hospital for applying ECPR can be very time consuming, decreasing the chance of neurological recovery.

As the HEMS are at scene very rapidly and consist of a team of highly trained specialists, equipping the HEMS with E-CPR may potentially significantly reduce the time till restoration of adequate circulation in patients with persistent arrest of circulation. To study this, we plan a large nationwide study covering all OHCA patients younger than 50 years by all four HEMS stations, comparing deployment of HEMS without ECPR with deployment of HEMS with ECPR capabilities.

space available for images / figures on next page

introduction (continued): space for images



image / figure 1: HEMS team with a patient on a miniaturized heart-lung machine

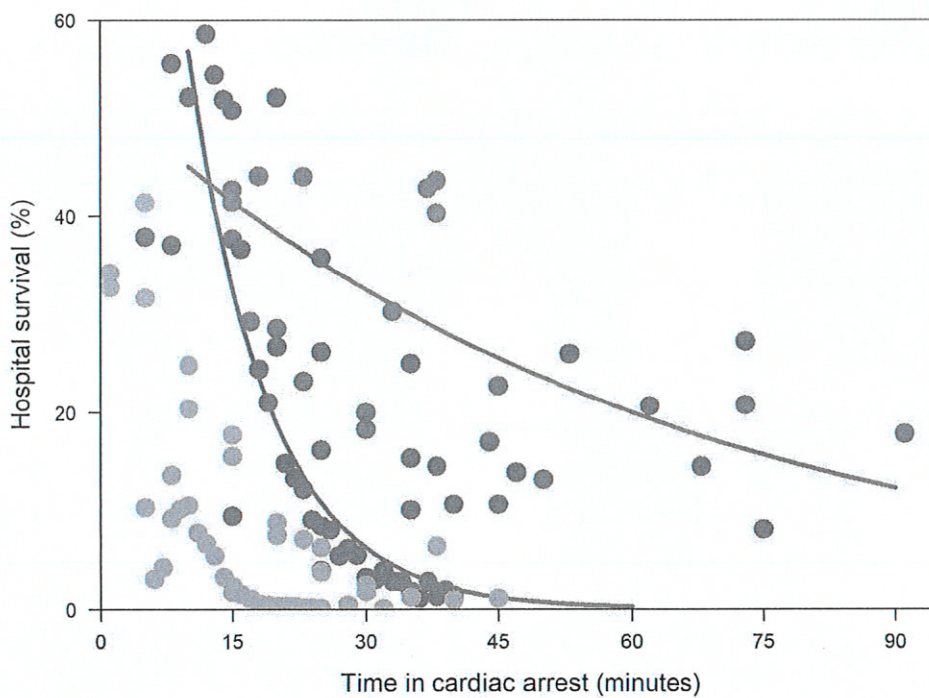


image / figure 2: Survival rate comparing to time in cardiac arrest (yellow: non-shockable, blue: shockable, red: ECMO)

PROBLEM DEFINITION **

Limit and define the scope and solution space of your project to one that is manageable within one Master Graduation Project of 30 EC (= 20 full time weeks or 100 working days) and clearly indicate what issue(s) should be addressed in this project.

As described above, neurological outcome is mainly dependent on time in cardiac arrest. As shown on figure 2, time record on phases of OHCA and patients' situation is vital for the comparison study (ECPR vs non-ECPR). However, due to the "hectic" setting during OHCA, with few staff, time in cardiac arrest is very difficult to measure adequately. In addition, at the end of the resuscitation, collection of data is also difficult as the HEMS often is dispatched to another assignment. However, for the study, adequate registration of time in cardiac arrest is of paramount importance. Thus, the major problem of this project would be: How to design a data collector, that requires as less action and attention as possible (or automatic) from the HEMS crew, to record current situation and register time precisely? The challenge is not only about implementing technologies on data collector but also about how to design the interaction between users (HEMS crew) and the device (data collector) in an emergency circumstance (OHCA). Therefore, research on following 2 topics should be carried out:

1. What kinds of data can be retracted from HEMS crew (e.g. physical inputs) and the patient (e.g. biomarkers) during the whole OHCA process for time registration on each OHCA phase (CPR -- shocks -- ECMO -- heart beats/ECMO runs/death)? For this research, time line and scenarios of each participant during OHCA should be drew out and analyzed. Plus, an observation study on actual OHCA case might be conducted if possible.
2. What kind of interaction should be designed for the HEMS crew while using the data collector in emergency situations? Should it be a passive automatic way, or a proactive way that requires inputs? To find out the solution, similar emergency cases from other fields (e.g. divers, skydivers, climbers) can be collected and analyzed. Based on this and results from the first research, simple prototypes can be built for user tests to seek the most suitable interaction between HEMS crew and the data collector.

ASSIGNMENT **

State in 2 or 3 sentences what you are going to research, design, create and / or generate, that will solve (part of) the issue(s) pointed out in "problem definition". Then illustrate this assignment by indicating what kind of solution you expect and / or aim to deliver, for instance: a product, a product-service combination, a strategy illustrated through product or product-service combination ideas, In case of a Specialisation and/or Annotation, make sure the assignment reflects this/these.

To design a data collector, carried by the HEMS personnel, which can register and store time in cardiac arrest in a very easy, practical and low-time-consuming way.

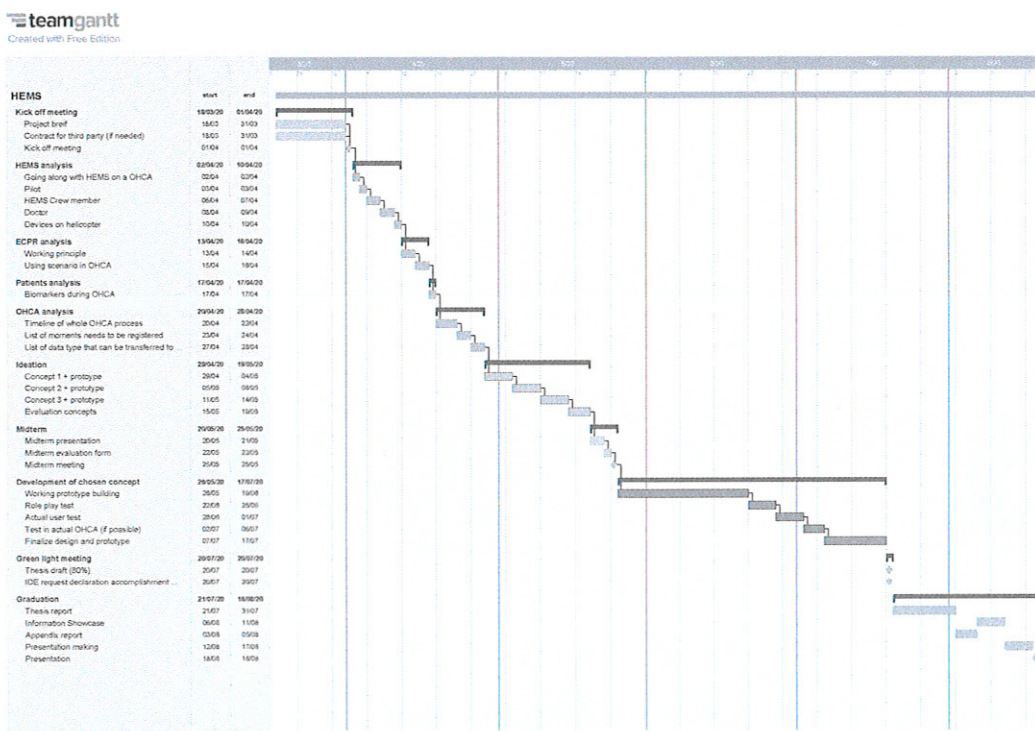
The expected outcome of the project would be an electronic product, which can be easily brought along with the HEMS crew, collect necessary medical data (mainly time of phases during OHCA) and send the data to a database (online/phone/PC). The collected data can be further retracted from the database for HEMS team's further research.

PLANNING AND APPROACH **

Include a Gantt Chart (replace the example below - more examples can be found in Manual 2) that shows the different phases of your project, deliverables you have in mind, meetings, and how you plan to spend your time. Please note that all activities should fit within the given net time of 30 EC = 20 full time weeks or 100 working days, and your planning should include a kick-off meeting, mid-term meeting, green light meeting and graduation ceremony. Illustrate your Gantt Chart by, for instance, explaining your approach, and please indicate periods of part-time activities and/or periods of not spending time on your graduation project, if any, for instance because of holidays or parallel activities.

start date 1 - 4 - 2020

18 - 8 - 2020 end date



The planning follows a common design approach. It starts with analysis on context, clients and target groups for the first 20 days, the result will be a list of design requirements. Then 3 concepts will be developed based on the list before midterm meeting by following weekly Scrum methods (approx. 20 days). After midterm meeting, one concept will be chosen for further development. Before green light meeting, a working prototype of the concept (20 days, Scrum) should be made, a user test with the prototype should be conducted (10 days) and thesis report is drafted (10 days). After green light meeting, a full thesis report, a showcase and final presentation will be prepared for the final graduation presentation.

MOTIVATION AND PERSONAL AMBITIONS

Explain why you set up this project, what competences you want to prove and learn. For example: acquired competences from your MSc programme, the elective semester, extra-curricular activities (etc.) and point out the competences you have yet developed. Optionally, describe which personal learning ambitions you explicitly want to address in this project, on top of the learning objectives of the Graduation Project, such as: in depth knowledge a on specific subject, broadening your competences or experimenting with a specific tool and/or methodology, Stick to no more than five ambitions.

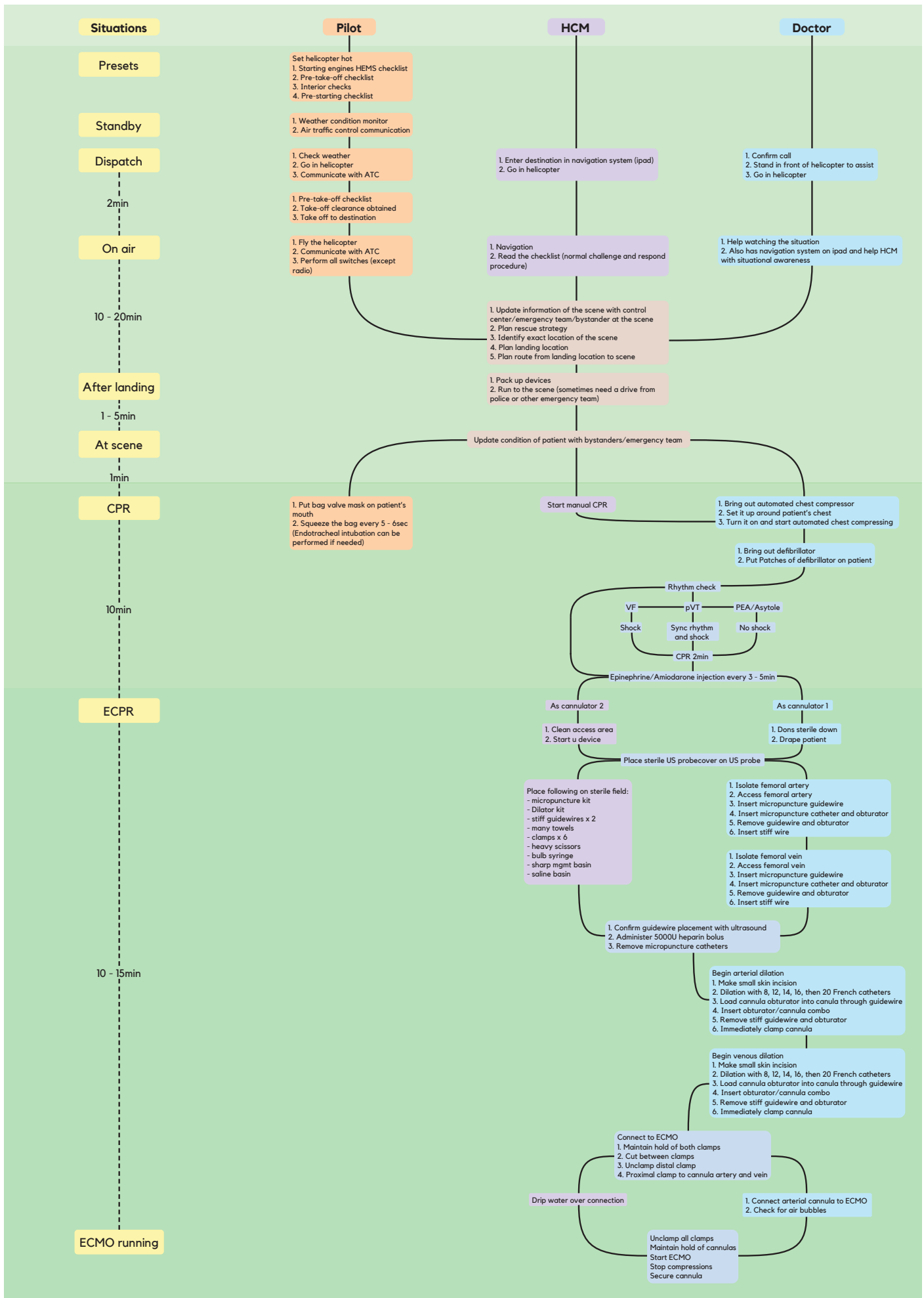
First, I want to graduate with a Medesign project. In AED, I experienced the whole process of electronic product development. During electives, I have finished Prototype Connected Product course, which illustrates the process of prototyping connected prototypes. In the Machine learning course I gained knowledge of using machine learning algorithm to analyze data. In JMP project, I even went to India to conduct researches and user tests. All these experiences is prepared for me to start a Medesign project includes connected product developing, data analysis and actual field tests. So this project suits me well. I have following ambitions for this project:

- In depth knowledge on Intensive care area
- Whole process of building working connected product, from electronics to online platform or app
- Whole process of data analysis, from data collection, data transform to data analysis

FINAL COMMENTS

In case your project brief needs final comments, please add any information you think is relevant.

Appendix B - Full activity timeline of HEMS crew in OHCA



Appendix D - Interview with Dr. Dinis Reis Miranda

- Does the team currently has a time counter or follow the defibrillator's counter?

Normally HEMS team doesn't bring defibrillator, unless they will arrive earlier than the ambulance, which is only 5% of the cases. The ambulance team always brings defibrillator but the brands are different for different regions/teams. For the time counter in the defibrillator, which counts down 2min after every reset. But the team hardly do reset it because it requires action and does not give warning about the chest compression time and rhythm check time.

- Does the HEMS team or the ambulance team use any type of automatic chest compressor?

HEMS teams never bring automatic chest compressor. For the ambulance teams, some regions use some not. Most of them use the LUCAS device, Rotterdam region uses Corpuls CPR (only with 10 ambulances), Amsterdam uses Autopulse.

- Does the ambulance team bring ECMO?

For the research, the HEMS team will bring one Cardiohelp ECMO.

- Is the ECMO Cardiohelp? And has you retracted data from Cardiohelp before?

Yes. You can only retract data after stopping the machine. If you start a new case, old case will be automatically deleted. And if you want to record the data, you need to go multiple steps through the menu to start recording before running the case, which is not ideal to require the team to do that in critical situations. But the data from ECMO would be nice to have but not necessary to have (wish).

- What is the protocol/requirement to apply ECMO on cardiac arrest patient? After how long CPR will the team switch to ECMO?

Yes it can be referred to the research protocol.

- Do you have an example of a case report that you normally need to formulate after each rescue?

Time of 112 call, arrival time of ambulance, arrival time in hospital, outcome of patients. These can be easily retracted from the hospital database later. The time when CCPR stops, when HEMS team arrives, when ECMO starts. Time of death/ROSC. When chest compression stops, only three results left for the patient: ROSC/ECMO/Death. For the research there are two groups: one with ECMO, one uses CCPR (also want the whole compression data).

- Does the pilot join the rescue or he will wait at the helicopter?

The pilot normally needs to guard helicopter. He can join the rescue only if no one is around the helicopter or the police is around the helicopter, because the helicopter always need official staff to guard.

Appendix F - Arduino IoT cloud code

```
//MAIN CODE//
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include <RTCZero.h>
#include "thingProperties.h"

const int MPU_addr = 0x68;
const int GMT = 2; //change this to adapt it to your time zone
String numString;
//String casename;
File sensorData;
RTCZero real_tc;

void setup() {
  //SD.begin(4);
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B); // PWR_MGMT_1 register
  Wire.write(0); // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  //pinMode(LED_BUILTIN,INPUT);
  // Initialize serial and wait for port to open:
  Serial.begin(115200);

  SD.begin(4);

  real_tc.begin(); // initialize RTC
  real_tc.setEpoch(WiFi.getTime() + 7200);
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void loop() {
  acc();
  currtime();
  dataString = timeString + " " + String(AcZ);
  ArduinoCloud.update();
  savedata();
  delay(1000);
}

void acc() {
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true); // request a total of 14 registers
  //AcX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  //AcY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
```

```

AcZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
}

String currtime() {
    return timeString = String(real_tc.getDay()) + "/" + String(real_tc.getMonth()) + "/" + "20" + String(real_tc.getYear()) + "," + print2digits(real_tc.getHours() + GMT) + ":" +
    print2digits(real_tc.getMinutes()) + ":" + print2digits(real_tc.getSeconds());
}

void savedata() {
    //casename = "case" + String(real_tc.getDay()) + "/" + String(real_tc.getMonth()) + "/" + String(real_tc.getYear()) + ".csv";
    sensorData = SD.open("sddata.csv", FILE_WRITE);
    if (sensorData) {
        sensorData.println(dataString);
        sensorData.close(); // close the file
        //digitalWrite(LED_BUILTIN, LOW);
    }
    else {
        //digitalWrite(LED_BUILTIN, HIGH);
        Serial.println("Error writing to file !");
    }
}

String print2digits(int num) {
    if (num < 10) {
        return numString = "0" + String(num);
    } else return numString = String(num);
}

//arduino_secrets.h//
#define SECRET_SSID ""
#define SECRET_PASS ""

//thingProperties.h//
#include <ArduinoloTCloud.h>
#include <Arduino_ConnectionHandler.h>
#include "arduino_secrets.h"

const char THING_ID[] = "";

const char SSID[] = SECRET_SSID; // Network SSID (name)
const char PASS[] = SECRET_PASS; // Network password (use for WPA, or use as key for WEP)

String timeString;
int AcZ;
String dataString;

void initProperties(){
    ArduinoCloud.setThingId(THING_ID);
    ArduinoCloud.addProperty(AcZ, READ, 1 * SECONDS, NULL);
    ArduinoCloud.addProperty(timeString, READ, 1 * SECONDS, NULL);
    ArduinoCloud.addProperty(dataString, READ, 1 * SECONDS, NULL);
}

WiFiConnectionHandler ArduinoloTPreferredConnection(SSID, PASS);

```


Appendix G - Arduino code of NeoPixel, buzzer and mini speaker

```
// NeoPixel Ring simple sketch (c) 2013 Shae Erisson
// Released under the GPLv3 license to match the rest of the
// Adafruit NeoPixel library

#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
#include <avr/power.h> // Required for 16 MHz Adafruit Trinket
#endif

// Which pin on the Arduino is connected to the NeoPixels?
#define PIN    6 // On Trinket or Gemma, suggest changing this to 1

// How many NeoPixels are attached to the Arduino?
#define NUMPIXELS 12 // Popular NeoPixel ring size

// When setting up the NeoPixel library, we tell it how many pixels,
// and which pin to use to send signals. Note that for older NeoPixel
// strips you might need to change the third parameter -- see the
// strandtest example for more information on possible values.
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

#define DELAYVAL 500 // Time (in milliseconds) to pause between pixels

void setup() {
  pixels.begin(); // INITIALIZE NeoPixel strip object (REQUIRED)
  pixels.setBrightness(6);
}

void loop() {
  //compress();
  //countdown();
  warn();
}

void compress() {
  for (int i = 0; i < 6; i++) { // For each pixel...
    pixels.setPixelColor(i, pixels.Color(150, 0, 0));
  }
  pixels.show(); // Send the updated pixel colors to the hardware.
  delay(600); // Pause before next pass through loop
  pixels.clear();

  for (int i = 6; i < 12; i++) {
    pixels.setPixelColor(i, pixels.Color(150, 0, 0));
  }
  pixels.show(); // Send the updated pixel colors to the hardware.
  delay(600); // Pause before next pass through loop
  pixels.clear();
}

void countdown() {
  for (int i = 1; i < 11; i++) { // For each pixel...
    pixels.setPixelColor(i, pixels.Color(150, 0, 0));
    pixels.show();
    delay(1000);
  }
}
```

```

}
pixels.clear();
}

void warn() {
  for (int i = 0; i < 12; i++) { // For each pixel...
    pixels.setPixelColor(i, pixels.Color(150, 0, 0));
  }
  pixels.show();
  delay(500);
  for (int i = 0; i < 12; i++) { // For each pixel...
    pixels.setPixelColor(i, pixels.Color(0, 0, 0));
  }
  pixels.show();
  delay(500);
}

```

// This code uses a neopixel ring, a buzzer to give audio and visual feedback to the rescuers. The rest part is same with the final code, like saving data in micro SD card, uploading data via WPA2 personal Wifi network.

```

#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include "arduino_secrets.h"
char server[] = "script.google.com";
char ssid[] = SECRET_SSID;
char pass[] = SECRET_PASS;
int status = WL_IDLE_STATUS;
WiFiSSLClient client;
String CPRdata = "";
//////////////////////////////////WIFI and IOT CLOUD//////////////////////////////////

#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
#include <avr/power.h> // Required for 16 MHz Adafruit Trinket
#endif

#define PIN    6 // On Trinket or Gemma, suggest changing this to 1
#define NUMPIXELS 12 // Popular NeoPixel ring size
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
#define DELAYVAL 500 // Time (in milliseconds) to pause between pixels
//////////////////////////////////NEOPIXEL SETUP//////////////////////////////////

int buzzerPin = 5;
int hertz1 = 262; //C4
int hertz2 = 523; //C5
int hertz3 = 1046; //C6
//#include <AudioZero.h>
long onTime = 300;
long offTime = 300;
long onTime2 = 500;
long offTime2 = 500;
long onTime3 = 250;
long offTime3 = 250;
long onTime4 = 2000;
long cycle = 20000; //130000s = 2min + 10s
int state = LOW;
int state2 = LOW;
unsigned long previous;

```

```

unsigned long before;
unsigned long before2 = 0;
int j = 11;
////////AUDIO////////

#include <Wire.h>
#define DS3231_I2C_ADDRESS 0x68
String monString;
String dayString;
String hourString;
String minString;
String secString;
String dateString;
String timeString;
String dataString;
////////RTC////////

byte decToBcd(byte val) {
    return ( val / 10 * 16) + (val % 10 );
}
byte bcdToDec(byte val) {
    return ( val / 16 * 10) + (val % 16 );
}
/////////DECIMAL TO BINARY////////

#include <SPI.h>
#include <SD.h>
String title = "Date,Timestamp,Acceleration,Rate";
String datetime;
String starttime;
String endtime;
String filename;
int SDPin = 4;
bool filenamed = false;
bool uploaded = true;
bool recorded = false;
//bool reset = false;
File sensorData;
//File compress;
//File countdown;
//File warn;
/////////SD////////

int accPin = A1;
int acc;
int c;
int compressionRate;
int maxThreshold = 910;
int minThreshold = 865;
int pinThreshold = 10;

unsigned long t;
unsigned long t1;
unsigned long t2;
unsigned long t3;
unsigned long t4;
const long interval = 300;
const long interval2 = 2000;

```

```
//////////ACCELERATION//////////
```

```
void setup() {  
  Wire.begin();  
  Serial.begin(115200);  
  SD.begin(SDPin);  
  // Serial.print("Initializing SD card...");  
  // if (!SD.begin(SDPin)) {  
  //   Serial.println("initialization failed!");  
  //   while (1);  
  // }  
  // Serial.println("initialization done.");  
  ////////////SD INITIALIZATION//////////  
  
  // byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;  
  // readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);  
  // //filename = "CPRdata.csv";//monString + dayString + String(hour, DEC) + ".csv";  
  // sensorData = SD.open(filename, FILE_WRITE);  
  // if (sensorData) {  
  //   //sensorData.println("Session start");  
  //   sensorData.println(title);  
  //   sensorData.close(); // close the file  
  //   //Serial.println(title);  
  // }  
  ////////////CREATE CASE FILE IN SD//////////  
  
  pixels.begin();  
  pixels.setBrightness(5);  
  ////////////INITIALIZE NEOPIXEL//////////  
  //AudioZero.begin(2*44100); //88200 SAMPLE RATE  
  ////////////INITIALIZE AUDIO//////////  
}
```

```
void loop() {  
  pixels.clear();  
  acceleration();  
  currtime();  
  if (analogRead(accPin) > pinThreshold) {  
    unsigned long present = millis();  
    if ((present - before2) < cycle) { //120s + 11s=131000  
      if ((compressionRate > 0 and compressionRate <= 90) or compressionRate >= 120) {  
        j = 11;  
        compress();  
      }  
      else if (compressionRate == 0) {  
        countdown();  
      }  
    }  
    else if ((present - before2) >= cycle) {  
      notify2min();  
      if (compressionRate == 0) {  
        before2 = present;  
      }  
    }  
  
    if (recorded == false) {  
      datetime = dateString + " ";  
      starttime = timeString + " ";  
    }  
  }  
}
```

```

Serial.println("Date:" + datetime);
Serial.println("Start:" + starttime);
recorded = true;
}

if ((present - before) > 999) {
  savedata();
  before = present;
}
uploaded = false;
}

else if ((analogRead(accPin) <= pinThreshold) && uploaded == false) {
  endtime = timeString + ",";
  Serial.println("End:" + endtime);
  while (status != WL_CONNECTED) {
    Serial.println("Connecting to wifi...");
    status = WiFi.begin(ssid, pass);
    if (analogRead(accPin) > pinThreshold) {
      break;
    }
    if (uploaded == true) {
      break;
    }
  }
  Serial.println("Connected to wifi");
  readdata();
  recorded = false;
  filenamed = false;
}

else {
  recorded = false;
  filenamed = false;
  Serial.println(analogRead(accPin));
}
}

void acceleration() {
  acc = analogRead(accPin);
  t1 = millis();
  t = t1 - t2;
  //t3 = t1 - t4;
  if (acc > maxThreshold) {
    if (t > 450) { // calculate compression rate every 300ms
      compressionRate = 60000 / t; //so maximal rate is 60s/0.3s=200pushes/min
      t2 = t1;
    }
  }
  else if (t > interval2 && acc < minThreshold) { // if no force sensed longer than 2s, rate is 0
    compressionRate = 0;
  }
}

////////////////////////////////////ACCELERATION////////////////////////////////////

void savedata() {
  if (filenamed == false) {
    filename = monString + dayString + hourString + ".csv";
    sensorData = SD.open(filename, FILE_WRITE);
  }
}

```

```

if (sensorData) {
    sensorData.println(title);
    sensorData.close();
}
else {
    Serial.println("fail to write");
}
filenamed = true;
}

dataString = dateString + "," + timeString + "," + String(acc) + "," + String(compressionRate); // convert to CSV
sensorData = SD.open(filename, FILE_WRITE);
if (sensorData) {
    sensorData.println(dataString);
    sensorData.close();
    Serial.println(dataString);
}
else {
    Serial.println("fail to write");
}
}

void readdata() {
    String avg = "112,";
    if (client.connectSSL(server, 443)) {
        Serial.println("connected to server");
        client.println("GET https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPy9038alQ0/exec?Date=" + datetime+","+
datetime+"&Timestamp="+starttime+","+starttime+"&Acceleration="+endtime+","+endtime+"&Rate="+avg+","+avg);
        //client.println("GET https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPy9038alQ0/exec?Date=" + datetime + datetime + datetime +
datetime + datetime + datetime + datetime + datetime + datetime + datetime + datetime + datetime + datetime + datetime + datetime + datetime + datetime +
datetime + "Start=" + starttime + starttime + starttime + starttime + starttime + starttime + starttime + starttime + starttime + starttime + starttime + starttime +
starttime + starttime + starttime + starttime + starttime + starttime + "End=" + endtime + endtime + endtime + endtime + endtime + endtime + endtime + endtime +
endtime + endtime + endtime + endtime + endtime + endtime + endtime + endtime + endtime + endtime + "Rate=" + avg + avg + avg + avg + avg + avg + avg + avg + avg +
avg + avg + avg + avg + avg + avg + avg + avg + avg + avg + avg);
        client.println("Host: script.google.com");
        client.println("Connection: close");
        client.println();
        //uploaded = true;
    } else Serial.println("fail");
}

String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = {0, -1};
    int maxIndex = data.length() - 1;
    for (int i = 0; i <= maxIndex && found <= index; i++) {
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i + 1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

////////////////////////////////SD////////////////////////////////

void compress() {

```

```

unsigned long current = millis();
if ((state == HIGH) && (current - previous) > onTime) {
  state = LOW;
  previous = current;
  if (state2 == LOW) {
    state2 = HIGH;
    for (int i = 0; i < 6; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(0, 150, 0));
    }
    for (int i = 6; i < 12; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(0, 0, 0));
    }
  }
  else {
    state2 = LOW;
    for (int i = 0; i < 6; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(0, 0, 0));
    }
    for (int i = 6; i < 12; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(0, 150, 0));
    }
  }
  pixels.show(); // Send the updated pixel colors to the hardware.
  tone(buzzerPin, hertz1, onTime);
}
else if ((state == LOW) && (current - previous) > offTime) {
  state = HIGH;
  previous = current;
  for (int i = 0; i < 12; i++) { // For each pixel...
    pixels.setPixelColor(i, pixels.Color(0, 0, 0));
  }
  pixels.show();
  noTone(buzzerPin);
}
}

```

```

void countdown() {
  unsigned long current = millis();
  if (j > 1) {
    if ((state == LOW) && (current - previous) > onTime2) {
      state = HIGH;
      previous = current;
      for (int i = 1; i < j; i++) {
        pixels.setPixelColor(i, pixels.Color(150, 0, 0));
      }
      pixels.show();
      tone(buzzerPin, hertz2, onTime2);
      j = j - 1;
    }
    else if ((state == HIGH) && (current - previous) > offTime2) {
      state = LOW;
      previous = current;
      pixels.clear();
      noTone(buzzerPin);
    }
  } else {
    warn();
  }
}

```

```

}

void warn() {
  unsigned long current = millis();
  if ((state == LOW) && (current - previous) > onTime3) {
    state = HIGH;
    previous = current;
    for (int i = 0; i < 12; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(150, 0, 0));
    }
    pixels.show();
    tone(buzzerPin, hertz3, onTime3);
  }
  else if ((state == HIGH) && (current - previous) > offTime3) {
    state = LOW;
    previous = current;
    for (int i = 0; i < 12; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(0, 0, 0));
    }
    pixels.show();
    noTone(buzzerPin);
  }
}

void notify2min() {
  unsigned long current = millis();
  if ((state == LOW) && (current - previous) > onTime3) {
    state = HIGH;
    previous = current;
    for (int i = 0; i < 12; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(0, 150, 0));
    }
    pixels.show();
    tone(buzzerPin, hertz3, onTime3);
  }
  else if ((state == HIGH) && (current - previous) > offTime3) {
    state = LOW;
    previous = current;
    for (int i = 0; i < 12; i++) { // For each pixel...
      pixels.setPixelColor(i, pixels.Color(0, 0, 0));
    }
    pixels.show();
    noTone(buzzerPin);
  }
}

//////////////////////////////////NEOPIXEL AND BUZZER//////////////////////////////////

void readDS3231time(byte * second, byte * minute, byte * hour, byte * dayOfWeek, byte * dayOfMonth, byte * month, byte * year)
{
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set DS3231 register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
  // request seven bytes of data from DS3231 starting from register 00h
  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
}

```



```

*dayOfMonth = bcdToDec(Wire.read());
*month = bcdToDec(Wire.read());
*year = bcdToDec(Wire.read());

if (*month < 10) {
    monString = "0" + String(*month, DEC);
} else monString = String(*month, DEC);

if (*dayOfMonth < 10) {
    dayString = "0" + String(*dayOfMonth, DEC);
} else dayString = String(*dayOfMonth, DEC);

if (*minute < 10) {
    minString = "0" + String(*minute, DEC);
} else minString = String(*minute, DEC);

if (*second < 10) {
    secString = "0" + String(*second, DEC);
} else secString = String(*second, DEC);
}

void currtime()
{
    byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
    readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
    dateString = "20" + String(year, DEC) + "-" + monString + "-" + dayString;
    hourString = String(hour, DEC);
    timeString = hourString + ":" + minString + ":" + secString;
}

void setDS3231time(byte second, byte minute, byte hour, byte dayOfWeek, byte dayOfMonth, byte month, byte year)
{
    // sets time and date data to DS3231
    Wire.beginTransmission(DS3231_I2C_ADDRESS);
    Wire.write(0); // set next input to start at the seconds register
    Wire.write(decToBcd(second)); // set seconds
    Wire.write(decToBcd(minute)); // set minutes
    Wire.write(decToBcd(hour)); // set hours
    Wire.write(decToBcd(dayOfWeek)); // set day of week (1=Sunday, 7=Saturday)
    Wire.write(decToBcd(dayOfMonth)); // set date (1 to 31)
    Wire.write(decToBcd(month)); // set month
    Wire.write(decToBcd(year)); // set year (0 to 99)
    Wire.endTransmission();
}

////////////////////////////////////////RTC////////////////////////////////////////
//arduino_secrets.h//
#define SECRET_SSID ""
#define SECRET_PASS ""

```

// This code uses a neopixel ring, a speaker to give audio and visual feedback to the rescuers. It needs wav. files to be saved inside the micro sd card for audio playing. The rest part is same with the final code, like saving data in micro SD card, uploading data via WPA2 personal Wifi network.

```
#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include "arduino_secrets.h"
char server[] = "script.google.com";
char ssid[] = SECRET_SSID;
char pass[] = SECRET_PASS;
int status = WL_IDLE_STATUS;
WiFiSSLClient client;
String CPRdata = "";
/////////////////////////////////WIFI and IOT CLOUD////////////////////////////////

#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
#include <avr/power.h> // Required for 16 MHz Adafruit Trinket
#endif
#define PIN 6 // On Trinket or Gemma, suggest changing this to 1
#define NUMPIXELS 12 // Popular NeoPixel ring size
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
#define DELAYVAL 500 // Time (in milliseconds) to pause between pixels
/////////////////////////////////NEOPIXEL SETUP////////////////////////////////

int buzzerPin = 5;
int hertz1 = 262; //C4
int hertz2 = 523; //C5
int hertz3 = 1046; //C5
#include <AudioZero.h>
long onTime = 300;
long offTime = 300;
long onTime2 = 500;
long offTime2 = 500;
long onTime3 = 250;
long offTime3 = 250;
int state = LOW;
int state2 = LOW;
unsigned long previous;
unsigned long before;
int j = 11;
File audioFile;
//////////AUDIO//////////

#include <Wire.h>
#define DS3231_I2C_ADDRESS 0x68
String monString;
String dayString;
String hourString;
String minString;
String secString;
String dateString;
String timeString;
String dataString;
//////////RTC//////////

byte decToBcd(byte val) {
  return ( (val / 10 * 16) + (val % 10) );
}
```

```

byte bcdToDec(byte val) {
    return ( val / 16 * 10) + (val % 16 );
}

//////////DECIMAL TO BINARY//////////

#include <SPI.h>
#include <SD.h>
String title = "Date,Timestamp,Acceleration,Rate";
String datetime;
String starttime;
String endtime;
String filename;
int SDPin = 4;
bool filenamed = false;
bool uploaded = true;
bool recorded = false;
File sensorData;
//File compress;
//File countdown;
//File warn;
//////////SD//////////

int accPin = A1;
int acc;
int compressionRate;
int maxThreshold = 900;
int minThreshold = 865;
int pinThreshold = 10;
unsigned long t;
unsigned long t1;
unsigned long t2;
const long interval = 300;
const long interval2 = 2000;
//////////ACCELERATION//////////

void setup() {
    Wire.begin();
    Serial.begin(115200);
    SD.begin(SDPin);
    // Serial.print("Initializing SD card...");
    // if (!SD.begin(SDPin)) {
    //     Serial.println("initialization failed!");
    //     while (1);
    // }
    // Serial.println("initialization done.");
    //////////SD INITIALIZATION//////////

    // byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
    // readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
    // //filename = "CPRdata.csv";//monString + dayString + String(hour, DEC) + ".csv";
    // sensorData = SD.open(filename, FILE_WRITE);
    // if (sensorData) {
    //     //sensorData.println("Session start");
    //     sensorData.println(title);
    //     sensorData.close(); // close the file
    //     //Serial.println(title);
    // }
    //////////CREATE CASE FILE IN SD//////////

```

```

pixels.begin();
pixels.setBrightness(5);
////////////////////////////////////INITIALIZE NEOPIXEL////////////////////////////////////
//AudioZero.begin(16000); //16000 SAMPLE RATE
////////////////////////////////////INITIALIZE AUDIO////////////////////////////////////
}

void loop() {
if (analogRead(accPin) > pinThreshold) {
  acceleration();
  currtime();
  if ((compressionRate > 0 and compressionRate < 95) or compressionRate > 110) {
    j = 11;
    compress();
    pixels.clear();
  } else if (compressionRate == 0) {
    countdown();
    pixels.clear();
  }
}

if (recorded == false) {
  datetime = dateString;
  starttime = timeString;
  Serial.println("Date:" + datetime);
  Serial.println("Start:" + starttime);
  recorded = true;
}

unsigned long present = millis();
if ((present - before) > 999) {
  savedata();
  before = present;
}

uploaded = false;
}

else if ((analogRead(accPin) < pinThreshold) && uploaded == false) {
  endtime = timeString;
  Serial.println("End:" + endtime);
  while (status != WL_CONNECTED) {
    Serial.println("Connecting to wifi...");
    status = WiFi.begin(ssid, pass);
    if (analogRead(accPin) > pinThreshold) {
      break;
    }
    if (uploaded == true) {
      break;
    }
  }
  Serial.println("Connected to wifi");
  readdata();
  recorded = false;
  filenameed = false;
}

else {
  recorded = false;
  filenameed = false;
}

```

```

Serial.println(analogRead(accPin));
}
}

void acceleration() {
acc = analogRead(accPin);
t1 = millis();
t = t1 - t2;
if (acc > maxThreshold) {
if (t > interval) { // calculate compression rate every 300ms
compressionRate = 60000 / t; //so maximal rate is 60s/0.3s=200pushes/min
t2 = t1;
}
} else if (t > interval2 && acc < minThreshold) { // if no force sensed longer than 2s, rate is 0
compressionRate = 0;
}
}
////////////////////////////////////ACCELERATION////////////////////////////////////

void savedata() {
if (filenamed == false) {
filename = monString + dayString + hourString + ".csv";
sensorData = SD.open(filename, FILE_WRITE);
if (sensorData) {
sensorData.println(title);
sensorData.close();
}
else {
Serial.println("fail to write");
}
filenamed = true;
}

dataString = dateString + "," + timeString + "," + String(acc) + "," + String(compressionRate); // convert to CSV
sensorData = SD.open(filename, FILE_WRITE);
if (sensorData) {
sensorData.println(dataString);
sensorData.close();
Serial.println(dataString);
}
else {
Serial.println("fail to write");
}
}

void readdata() {
if (client.connectSSL(server, 443)) {
Serial.println("connected to server");
client.println("GET https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNf5q0CgYL3DwjHHyvu9rY4PIPy9038alQ0/exec?Date=" +
datetime + "&Start=" + starttime + "&End=" +
endtime);
client.println("Host: script.google.com");
client.println("Connection: close");
client.println();
uploaded = true;
} else Serial.println("fail");
}

String getValue(String data, char separator, int index)

```

```

{
int found = 0;
int strIndex[] = {0, -1};
int maxIndex = data.length() - 1;
for (int i = 0; i <= maxIndex && found <= index; i++) {
if (data.charAt(i) == separator || i == maxIndex) {
found++;
strIndex[0] = strIndex[1] + 1;
strIndex[1] = (i == maxIndex) ? i + 1 : i;
}
}
return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
////////////////////////////////SD////////////////////////////////

```

```

void compress() {
unsigned long current = millis();
if ((state == HIGH) && (current - previous) > onTime) {
state = LOW;
previous = current;
if (state2 == LOW) {
state2 = HIGH;
for (int i = 0; i < 6; i++) { // For each pixel...
pixels.setPixelColor(i, pixels.Color(0, 150, 0));
}
for (int i = 6; i < 12; i++) { // For each pixel...
pixels.setPixelColor(i, pixels.Color(0, 0, 0));
}
}
else {
state2 = LOW;
for (int i = 0; i < 6; i++) { // For each pixel...
pixels.setPixelColor(i, pixels.Color(0, 0, 0));
}
for (int i = 6; i < 12; i++) { // For each pixel...
pixels.setPixelColor(i, pixels.Color(0, 150, 0));
}
}
AudioZero.begin(16000);
audioFile = SD.open("compress.wav");
pixels.show(); // Send the updated pixel colors to the hardware.
AudioZero.play(audioFile);
}
else if ((state == LOW) && (current - previous) > offTime) {
state = HIGH;
previous = current;
for (int i = 0; i < 12; i++) { // For each pixel...
pixels.setPixelColor(i, pixels.Color(0, 0, 0));
}
pixels.show();
//AudioZero.end();
}
}

```

```

void countdown() {
unsigned long current = millis();
if (j > 1) {

```

```

if ((state == LOW) && (current - previous) > onTime2) {
    state = HIGH;
    previous = current;
    for (int i = 1; i < j; i++) {
        pixels.setPixelColor(i, pixels.Color(150, 0, 0));
    }
    if (j == 10 or j == 9) {
        AudioZero.begin(16000);
        audioFile = SD.open("resu.wav");
        AudioZero.play(audioFile);
    }
    else if (j == 8 or j == 7) {
        AudioZero.begin(16000);
        audioFile = SD.open("87.wav");
        AudioZero.play(audioFile);
    }
    else if (j == 6 or j == 5) {
        AudioZero.begin(16000);
        audioFile = SD.open("65.wav");
        AudioZero.play(audioFile);
    }
    else if (j == 4 or j == 3) {
        AudioZero.begin(16000);
        audioFile = SD.open("43.wav");
        AudioZero.play(audioFile);
    }
    else {
        AudioZero.begin(16000);
        audioFile = SD.open("21.wav");
        AudioZero.play(audioFile);
    }
    pixels.show();
    j = j - 1;
}
else if ((state == HIGH) && (current - previous) > offTime2) {
    state = LOW;
    previous = current;
    pixels.clear();
    //AudioZero.end();
}
} else {
    warn();
}
}

```

```

void warn() {
    unsigned long current = millis();
    if ((state == LOW) && (current - previous) > onTime3) {
        state = HIGH;
        previous = current;
        for (int i = 0; i < 12; i++) { // For each pixel...
            pixels.setPixelColor(i, pixels.Color(150, 0, 0));
        }
        AudioZero.begin(16000);
        audioFile = SD.open("warn.wav");
        pixels.show();
        AudioZero.play(audioFile);
    }
}

```

```

}
else if ((state == HIGH) && (current - previous) > offTime3) {
    state = LOW;
    previous = current;
    for (int i = 0; i < 12; i++) { // For each pixel...
        pixels.setPixelColor(i, pixels.Color(0, 0, 0));
    }
    pixels.show();
    //AudioZero.end();
}
}
}
//////////////////////////////////NEOPIXEL AND BUZZER//////////////////////////////////

void readDS3231time(byte *second, byte *minute, byte *hour, byte *dayOfWeek, byte *dayOfMonth, byte *month, byte *year)
{
    Wire.beginTransmission(DS3231_I2C_ADDRESS);
    Wire.write(0); // set DS3231 register pointer to 00h
    Wire.endTransmission();
    Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
    // request seven bytes of data from DS3231 starting from register 00h
    *second = bcdToDec(Wire.read() & 0x7f);
    *minute = bcdToDec(Wire.read());
    *hour = bcdToDec(Wire.read() & 0x3f);
    *dayOfWeek = bcdToDec(Wire.read());
    *dayOfMonth = bcdToDec(Wire.read());
    *month = bcdToDec(Wire.read());
    *year = bcdToDec(Wire.read());

    if (*month < 10) {
        monString = "0" + String(*month, DEC);
    } else monString = String(*month, DEC);

    if (*dayOfMonth < 10) {
        dayString = "0" + String(*dayOfMonth, DEC);
    } else dayString = String(*dayOfMonth, DEC);

    if (*minute < 10) {
        minString = "0" + String(*minute, DEC);
    } else minString = String(*minute, DEC);

    if (*second < 10) {
        secString = "0" + String(*second, DEC);
    } else secString = String(*second, DEC);
}

void currtime()
{
    byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
    readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
    dateString = "20" + String(year, DEC) + "-" + monString + "-" + dayString;
    hourString = String(hour, DEC);
    timeString = hourString + ":" + minString + ":" + secString;
}

void setDS3231time(byte second, byte minute, byte hour, byte dayOfWeek, byte dayOfMonth, byte month, byte year)
{
    // sets time and date data to DS3231
    Wire.beginTransmission(DS3231_I2C_ADDRESS);

```



```
Wire.write(0); // set next input to start at the seconds register
Wire.write(decToBcd(second)); // set seconds
Wire.write(decToBcd(minute)); // set minutes
Wire.write(decToBcd(hour)); // set hours
Wire.write(decToBcd(dayOfWeek)); // set day of week (1=Sunday, 7=Saturday)
Wire.write(decToBcd(dayOfMonth)); // set date (1 to 31)
Wire.write(decToBcd(month)); // set month
Wire.write(decToBcd(year)); // set year (0 to 99)
Wire.endTransmission();
}
//////////////////////////////////////////////////////////////////RTC//////////////////////////////////////////////////////////////////

//arduino_secrets.h//
#define SECRET_SSID ""
#define SECRET_PASS ""
```

Appendix H - Final Google Script code

```
// This code shows how the Google Script takes the uploaded data from the Arduino and store it in its corresponding Google spreadsheets and make analysis and charts over the data.
The corresponding sheet link is: https://docs.google.com/spreadsheets/d/1WH5DgohmEY1PSwjyZ5xM3mm7EYOXnFjsHGcCtIg_Q/edit#gid=340263445
var id = '1WH5DgohmEY1PSwjyZ5xM3mm7EYOXnFjsHGcCtIg_Q'; // This is the spreadsheet ID that corresponds to this script
var spreadsheet = SpreadsheetApp.openById(id);
var sheet = SpreadsheetApp.openById(id).getActiveSheet();

function doGet(e) { // Get the CPRdata as e from Arduino from the Https request: https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPYy9038a
IQ0/exec?CPRdata=" + CPRdata, 'AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPYy9038aIQ0' is the ID of this script
  Logger.log( JSON.stringify(e) ); // Make e as a string

  var result = 'Ok';

  if (e.parameter == undefined) {
    result = 'No Parameters';
  }
  else {
    var newRow;
    for (var param in e.parameter) {
      Logger.log('In for loop, param='+param);
      var value = stripQuotes(e.parameter[param]); // stripQuotes() function eliminate potential garbled characters from e
      switch (param) {
        case 'CPRdata': //Parameter
          var CPRdata = value.split(","); // Split value (e) by comma and assign it into CPRdata array
          break;
        default:
          result = "Wrong parameter";
      }
    }
    Logger.log(JSON.stringify(CPRdata));

    newRow = sheet.getLastRow() + 1;
    var newRange = sheet.getRange(newRow, 1, 100, 4); // Assign 400 cells (100 lines of data: date, time, ACC, rate) all in once
    newRange.setValues([[CPRdata[0],CPRdata[1],CPRdata[2],CPRdata[3]], [CPRdata[4],CPRdata[5],CPRdata[6],CPRdata[7]], [CPRdata[8],CPRdata[9],CPRdata[10],CPRdata[11]],
      [CPRdata[12],CPRdata[13],CPRdata[14],CPRdata[15]], [CPRdata[16],CPRdata[17],CPRdata[18],CPRdata[19]], [CPRdata[20],CPRdata[21],CPRdata[22],CPRdata[23]],
      [CPRdata[24],CPRdata[25],CPRdata[26],CPRdata[27]], [CPRdata[28],CPRdata[29],CPRdata[30],CPRdata[31]], [CPRdata[32],CPRdata[33],CPRdata[34],CPRdata[35]],
      [CPRdata[36],CPRdata[37],CPRdata[38],CPRdata[39]], [CPRdata[40],CPRdata[41],CPRdata[42],CPRdata[43]], [CPRdata[44],CPRdata[45],CPRdata[46],CPRdata[47]],
      [CPRdata[48],CPRdata[49],CPRdata[50],CPRdata[51]], [CPRdata[52],CPRdata[53],CPRdata[54],CPRdata[55]], [CPRdata[56],CPRdata[57],CPRdata[58],CPRdata[59]],
      [CPRdata[60],CPRdata[61],CPRdata[62],CPRdata[63]], [CPRdata[64],CPRdata[65],CPRdata[66],CPRdata[67]], [CPRdata[68],CPRdata[69],CPRdata[70],CPRdata[71]],
      [CPRdata[72],CPRdata[73],CPRdata[74],CPRdata[75]], [CPRdata[76],CPRdata[77],CPRdata[78],CPRdata[79]], [CPRdata[80],CPRdata[81],CPRdata[82],CPRdata[83]],
      [CPRdata[84],CPRdata[85],CPRdata[86],CPRdata[87]], [CPRdata[88],CPRdata[89],CPRdata[90],CPRdata[91]], [CPRdata[92],CPRdata[93],CPRdata[94],CPRdata[95]],
      [CPRdata[96],CPRdata[97],CPRdata[98],CPRdata[99]], [CPRdata[100],CPRdata[101],CPRdata[102],CPRdata[103]], [CPRdata[104],CPRdata[105],CPRdata[106],CPRdata[107]],
      [CPRdata[108],CPRdata[109],CPRdata[110],CPRdata[111]], [CPRdata[112],CPRdata[113],CPRdata[114],CPRdata[115]], [CPRdata[116],CPRdata[117],CPRdata[11
      8],CPRdata[119]],
      [CPRdata[120],CPRdata[121],CPRdata[122],CPRdata[123]], [CPRdata[124],CPRdata[125],CPRdata[126],CPRdata[127]], [CPRdata[128],CPRdata[129],CPRdata[13
      0],CPRdata[131]],
      [CPRdata[132],CPRdata[133],CPRdata[134],CPRdata[135]], [CPRdata[136],CPRdata[137],CPRdata[138],CPRdata[139]], [CPRdata[140],CPRdata[141],CPRdata[14
      2],CPRdata[143]],
      [CPRdata[144],CPRdata[145],CPRdata[146],CPRdata[147]], [CPRdata[148],CPRdata[149],CPRdata[150],CPRdata[151]], [CPRdata[152],CPRdata[153],CPRdata[15
      4],CPRdata[155]],
      [CPRdata[156],CPRdata[157],CPRdata[158],CPRdata[159]], [CPRdata[160],CPRdata[161],CPRdata[162],CPRdata[163]], [CPRdata[164],CPRdata[165],CPRdata[16
      6],CPRdata[167]],
      [CPRdata[168],CPRdata[169],CPRdata[170],CPRdata[171]], [CPRdata[172],CPRdata[173],CPRdata[174],CPRdata[175]], [CPRdata[176],CPRdata[177],CPRdata[17
      8],CPRdata[179]],
      [CPRdata[180],CPRdata[181],CPRdata[182],CPRdata[183]], [CPRdata[184],CPRdata[185],CPRdata[186],CPRdata[187]], [CPRdata[188],CPRdata[189],CPRdata[19
      0],CPRdata[191]],
```

```

    [CPRdata[192],CPRdata[193],CPRdata[194],CPRdata[195]],[CPRdata[196],CPRdata[197],CPRdata[198],CPRdata[199]],[CPRdata[200],CPRdata[201],CPRdata[202],CPRdata[203]],
    [CPRdata[204],CPRdata[205],CPRdata[206],CPRdata[207]],[CPRdata[208],CPRdata[209],CPRdata[210],CPRdata[211]],[CPRdata[212],CPRdata[213],CPRdata[214],CPRdata[215]],
    [CPRdata[216],CPRdata[217],CPRdata[218],CPRdata[219]],[CPRdata[220],CPRdata[221],CPRdata[222],CPRdata[223]],[CPRdata[224],CPRdata[225],CPRdata[226],CPRdata[227]],
    [CPRdata[228],CPRdata[229],CPRdata[230],CPRdata[231]],[CPRdata[232],CPRdata[233],CPRdata[234],CPRdata[235]],[CPRdata[236],CPRdata[237],CPRdata[238],CPRdata[239]],
    [CPRdata[240],CPRdata[241],CPRdata[242],CPRdata[243]],[CPRdata[244],CPRdata[245],CPRdata[246],CPRdata[247]],[CPRdata[248],CPRdata[249],CPRdata[250],CPRdata[251]],
    [CPRdata[252],CPRdata[253],CPRdata[254],CPRdata[255]],[CPRdata[256],CPRdata[257],CPRdata[258],CPRdata[259]],[CPRdata[260],CPRdata[261],CPRdata[262],CPRdata[263]],
    [CPRdata[264],CPRdata[265],CPRdata[266],CPRdata[267]],[CPRdata[268],CPRdata[269],CPRdata[270],CPRdata[271]],[CPRdata[272],CPRdata[273],CPRdata[274],CPRdata[275]],
    [CPRdata[276],CPRdata[277],CPRdata[278],CPRdata[279]],[CPRdata[280],CPRdata[281],CPRdata[282],CPRdata[283]],[CPRdata[284],CPRdata[285],CPRdata[286],CPRdata[287]],
    [CPRdata[288],CPRdata[289],CPRdata[290],CPRdata[291]],[CPRdata[292],CPRdata[293],CPRdata[294],CPRdata[295]],[CPRdata[296],CPRdata[297],CPRdata[298],CPRdata[299]],
    [CPRdata[300],CPRdata[301],CPRdata[302],CPRdata[303]],[CPRdata[304],CPRdata[305],CPRdata[306],CPRdata[307]],[CPRdata[308],CPRdata[309],CPRdata[310],CPRdata[311]],
    [CPRdata[312],CPRdata[313],CPRdata[314],CPRdata[315]],[CPRdata[316],CPRdata[317],CPRdata[318],CPRdata[319]],[CPRdata[320],CPRdata[321],CPRdata[322],CPRdata[323]],
    [CPRdata[324],CPRdata[325],CPRdata[326],CPRdata[327]],[CPRdata[328],CPRdata[329],CPRdata[330],CPRdata[331]],[CPRdata[332],CPRdata[333],CPRdata[334],CPRdata[335]],
    [CPRdata[336],CPRdata[337],CPRdata[338],CPRdata[339]],[CPRdata[340],CPRdata[341],CPRdata[342],CPRdata[343]],[CPRdata[344],CPRdata[345],CPRdata[346],CPRdata[347]],
    [CPRdata[348],CPRdata[349],CPRdata[350],CPRdata[351]],[CPRdata[352],CPRdata[353],CPRdata[354],CPRdata[355]],[CPRdata[356],CPRdata[357],CPRdata[358],CPRdata[359]],
    [CPRdata[360],CPRdata[361],CPRdata[362],CPRdata[363]],[CPRdata[364],CPRdata[365],CPRdata[366],CPRdata[367]],[CPRdata[368],CPRdata[369],CPRdata[370],CPRdata[371]],
    [CPRdata[372],CPRdata[373],CPRdata[374],CPRdata[375]],[CPRdata[376],CPRdata[377],CPRdata[378],CPRdata[379]],[CPRdata[380],CPRdata[381],CPRdata[382],CPRdata[383]],
    [CPRdata[384],CPRdata[385],CPRdata[386],CPRdata[387]],[CPRdata[388],CPRdata[389],CPRdata[390],CPRdata[391]],[CPRdata[392],CPRdata[393],CPRdata[394],CPRdata[395]],
    [CPRdata[396],CPRdata[397],CPRdata[398],CPRdata[399]]];
}

var end = sheet.getRange(sheet.getLastRow(),1).getValue(); // Get the value of the last line of the sheet
if (end == "END"){ // If the last value is "END", it means the upload is done
    setName(); // Set the sheet name
    setValue(); // Caculate compression start time, duration, end time, pause interval and average rate
    newChart(); // Generate a chart summary
    newSheet(); // Create a new sheet for next session
    sendEmail(); // Send email to the HEMS team if this spreadsheet is full
}
// Return result of operation
return ContentService.createTextOutput(result);
}

function stripQuotes( value ) {
    return value.replace(/^[\"']|['\"]$/g, "");
}

function setName(){
    var name = Utilities.formatDate(new Date(), "CET", "yy-MM-dd/HH:mm"); // Set the sheet name based on current time
    sheet.setName(name);
}

```

function newChart() { // Generate a chart summary of this session, blue line represents the ACC value and red represents compression rate

```
var lastRow = sheet.getLastRow()-2;
var chart = sheet.newChart()
    .setChartType(Charts.ChartType.LINE)
    .addRange(sheet.getRange(2,3,lastRow,2))
    .setPosition(4, 10, 2, 2)
    .setOption('title', 'CPR Session summary')
    .setOption('width', 1000)
    .setOption('height', 400);
sheet.insertChart(chart.build());
}
```

```
function setValue(){ //
var rate;
var avg;
var start;
var end;
var row;
var change = false;
var finish = false;
var j = 1;
var data = sheet.getRange(3,2,sheet.getLastRow(),1).getValues(); // Take the whole values of ACC into array data
for (i = 2; i< sheet.getLastRow(); i++){ // Go through every line of the ACC value in the loop
if (data[i] >= 90 && change == false){ // If ACC value is bigger than 90, which counts as valid compression rate
    row = i;
    start = sheet.getRange(i,2).getValue(); // Get corresponding time
    sheet.getRange(j,5).setValue(start); //Set this time as compression start time
    rate = sheet.getRange(i,4).getValue(); //Set corresponding compression rate as start rate
    sheet.getRange(j,8).setFormula("=MINUS(R[1]C[-3],R[0]C[-1])"); // Set pause interval
    change = true; // Change state
} else if (data[i] >= 90 && change == true){ // If ACC value is bigger than 90 and state changes
    rate = rate + val; // Accumulate rate values
} else if (data[i] <90 && change == true){ // If ACC valus is smaller than 90, which counts as invalid compression rate
    end = sheet.getRange(i,2).getValue(); // Get corresponding time
    sheet.getRange(j,7).setValue(end); //Set this time as compression end time
    sheet.getRange(j,6).setFormula("=MINUS(R[0]C[1],R[0]C[-1])"); // Calculate CPR duration
    avg = float2int(rate/(i - row)); // Calculate average compression rate
    sheet.getRange(j,9).setValue(avg); // Set average rate
    j++; // Move to next row
    change = false; // Change state back
}
}
}
```

// Current loop takes more than 6min to execute if the whole data is too large (> 3600 lines), which would cause a corruption because Google restrict the function execution time to 6min.

// Following code tries to solve this problem by going through the loop and calculating values with arrays. But this code is not working now. It needs further development to make sure that at least 7200 lines of data can be taken care of by the function within 6min.

```
// var ratedata = sheet.getRange(3,2,sheet.getLastRow(),1).getValues();
// var timedata = sheet.getRange(1,2,sheet.getLastRow(),1).getValues();
// var result = new Array(5);
// var values = [];
// for (i = 0; i< sheet.getLastRow(); i++){
// if (ratedata[i] >= 90 && change == false){
//     row = i;
//     result[0] = timedata[i]; //start time
//     rate = ratedata [i]; //start rate
//     result[3] = result[0] - result[2]; //pause interval
//     change = true;
// } else if (ratedata[i] >= 90 && change == true){
```

```

// rate = rate + ratedata[i];
// } else if (ratedata[i] <90 && change == true){
//   result[2] = timedata[i]; //end time
//   result[1] = result[2] - result[0]; //duration
//   result[4] = float2int(rate/(i - row)); //avg rate
//   change = false;
//   finish = true;
// }
// if (finish == true){
//   values.push(result);
//   j++;
//   finish = false;
// }
// }
// sheet.getRange(2,5,j,5).setValues(values);

// for (i = 2; i< sheet.getLastRow(); i++){
// }

function float2int (value) { // Transfer float to integer
  return value | 0;
}

function newSheet(){ // Create a new sheet for the next session based on the template sheet
  var template = spreadsheet.getSheetByName('template');
  var options = {template: template};
  spreadsheet.insertSheet(0,options);
}

function sendEmail () { // Send email to HEMS team if this spreadsheet is full with 200 sheets
  var numSheets = spreadsheet.getSheets().length; // Get the number of sheets inside this spreadsheet
  if (numSheets == 200){ // If it contains 200 sheets, which means it is full
    DriveApp.getFileById(spreadsheet.getId()).makeCopy(); // Make a full copy of this spreadsheet in the same Google Drive to backup the spreadsheet
    var address = 'xxx@mail.com'; // Send email to this address with following text
    var subject = 'Data of 199 CPR sessions are collected';
    var body = 'Hi xxx, the data of 199 CPR sessions are collected and saved in the HEMS folder';
    MailApp.sendEmail(address,subject,body);
    for (i = 0;i<199; i++){ // Delete all the sheets in this spreadsheet except the template sheet, so this spreadsheet is ready for more sessions
      spreadsheet.deleteActiveSheet();
    }
  }
}

```

Appendix I - Final Arduino code: WPA2 personal and enterprise version

// This is the final code for the Arduino MKR 1010 Wifi. It collects data and sends data by connecting to a WPA2 personal Wifi network.

```
//////////////////////////////////WIFI and IOT CLOUD//////////////////////////////////
//This part sets up the libraries and variables for Wifi connection and Https requests
#include <WiFiNINA.h> // Necessary library
#include <WiFiUdp.h> // Necessary library
#include "arduino_secrets.h" // Contains password
char server[] = "script.google.com";
char ssid[] = SECRET_SSID;
char pass[] = SECRET_PASS;
int status = WL_IDLE_STATUS;
WiFiSSLClient client;
String CPRdata = "";
//////////////////////////////////WIFI and IOT CLOUD//////////////////////////////////
```

```
//////////////////////////////////RTC//////////////////////////////////
//This part sets up the libraries and variables for the DS3231 RTC module
#include <Wire.h>
#define DS3231_I2C_ADDRESS 0x68
String monString;
String dayString;
String hourString;
String minString;
String secString;
String dateString;
String timeString;
String dataString;
byte decToBcd(byte val) {
  return ( val / 10 * 16) + (val % 10);
}
byte bcdToDec(byte val) {
  return ( val / 16 * 10) + (val % 16);
}
//////////////////////////////////RTC//////////////////////////////////
```

```
//////////////////////////////////SD//////////////////////////////////
//This part sets up the libraries and variables for the micro SD card module
#include <SPI.h>
#include <SD.h>
String filename = "CPR_Data.csv";
String title = "Date,Timestamp,Acceleration,Rate";
String sheetname;
int SDPin = 7; // CS (chip selected) Pin to digital pin 7
int p;
int vol;
bool uploaded = true;
bool recorded = false;
bool filenameed = false;
bool marked = false;
File sensorData;
File sdfile;
//////////////////////////////////SD//////////////////////////////////
```

```

////////////////////////////////////////ACC////////////////////////////////////////
//This part sets up the libraries and variables for calculate the chest compression rate based on the ACC readings from CorPatch
int accPin = A1; // ACC pin to Analog pin 1

int acc;

int maxval;

int compressionRate;

int maxThreshold = 910; // ACC readings that are higher than this threshold count as a valid compression
int minThreshold = 875; // ACC readings that are lower than this threshold count as no compression
int pinThreshold = 100; // If ACC reading is higher than this threshold, it means CorPatch is plugged in
unsigned long t;
unsigned long t1;
unsigned long t2;
unsigned long before;

const long interval = 400; // Chest compression rate is calculated every 400ms
const long interval2 = 2000; // If longer than 2000ms,
////////////////////////////////////////ACC////////////////////////////////////////

////////////////////////////////////////SETUP FUNCTION////////////////////////////////////////
// This setup function only run once at the beginning when the device is turned on
void setup() {
  Wire.begin();
  Serial.begin(115200);
  SD.begin(SDPin);
  //setDS3231time(40, 44, 13, 2, 11, 8, 20); // This function is used to sync current time
}
////////////////////////////////////////SETUP FUNCTION////////////////////////////////////////

////////////////////////////////////////LOOP FUNCTION////////////////////////////////////////
// This loop function is looped as long as the device is turned on after executing the setup function
void loop() {
  if (analogRead(accPin) > pinThreshold) { // If the ACC value is higher than pinThreshold (100), it means that the CorPatch is plugged in
    acceleration(); // Calculate compression rate
    currtime(); // Get current time
    unsigned long present = millis();
    if ((present - before) > 999) { //called every 1000ms
      savedata(); // Create a new csv file, save the current time (date, time), ACC value, compression rate value to this file in the micro SD card
      before = present; // refresh timer
    }
    uploaded = false; // This file is not uploaded to Google sheet yet
    marked = false; // This saved csv file is not marked as 'END' yet
  }
  else if ((analogRead(accPin) <= pinThreshold) and uploaded == false) { //If ACC value is lower than pinThreshold (100), it means CorPatch is unplugged, and if the data is not uploaded
yet, upload the data here
    markend(); // Mark an 'END' string at the end of the just saved csv file
    while (status != WL_CONNECTED) { // This is a while loop function, which means it only stops until the condition is not true or a 'break' is called during the loop. Here if Wifi is not
connected, the function loops until Wifi is connected or 'break' is called
      Serial.println("Connecting to wifi...");
      status = WiFi.begin(ssid, pass); // Connect to Wifi with SSID and password
      if (analogRead(accPin) > pinThreshold) { // If the CorPatch is plugged in again, break this loop
        break;
      }
    }
    if (uploaded == true) { // If data is uploaded, break this loop
      break;
    }
  }
  Serial.println("Connected to wifi");
  readdata(); // send data to Google sheets
  recorded = false; //

```

```

filenamed = false; // The new file for next case is not created yet
}

else { //After uploading the data of this case, the device goes to standby mode, which does nothing until the CorPatch is plugged in again
recorded = false;
filenamed = false; // The new file for the next case is not created yet
marked = false; // The new file for the next case is not marked as 'END' yet
Serial.println(analogRead(accPin));
}
}
//////////////////////////////////LOOP FUNCTION//////////////////////////////////

//////////////////////////////////ACCELERATION//////////////////////////////////
//This function is called to get current ACC value and calculate compression rate every 400ms
void acceleration() {
acc = analogRead(accPin); // Get ACC raw value
if (millis() - t > interval) { // Do this every 400ms
if (acc >= maxval) { // If the ACC value is increasing
maxval = acc; // Max ACC value is the new reading
} else { // If the ACC value starts to decrease, which means the last maxval is the peak during this period
if (maxval > maxThreshold) { // If the peak ACC value is bigger than maxThreshold (910)
compressionRate = 60000 / (millis() - t); // Calculate the compression rate by dividing 60000ms by the interval between current and last time the peak value occurs
maxval = 0; // maxval set back to 0
t = millis(); // current time is recorded to t
}
}
}
if (millis() - t > interval2 && acc < minThreshold) { // If ACC value is smaller than minThreshold (875) for more than 2 seconds
compressionRate = 0; // Compression rate is set as 0
}
}
//////////////////////////////////ACCELERATION//////////////////////////////////

//////////////////////////////////SAVE DATA TO SD CARD//////////////////////////////////
// This function is called to save the current data to a newly created csv file in the micro SD card
void savedata() {
if (filenamed == false) { // If a new file is not created yet for the new rescue case
sheetname = monString + dayString + hourString + minString + ".csv"; // Create a new file by naming it with current time, this file is used to upload data to Google sheets later
filenamed = true; // File is created for the rescue case
sdfile = SD.open(filename, FILE_WRITE); // Open the newly created file
if (sdfile) {
sdfile.println(title); // Write the file title "Date,TimeStamp,Acceleration,Rate" in the first row of the csv file, so here 4 columns with these 4 titles are created inside the csv file
sdfile.close();
}
else {
Serial.println("fail to write");
}
}
dataString = dateString + "," + timeString + "," + String(acc) + "," + String(compressionRate) + ","; // Combine the whole data into one string with commas in between
sensorData = SD.open(sheetname, FILE_WRITE); // Save the data to the file that is used to upload to Google sheets
if (sensorData) {
if (p < 99) { // Write the 100 lines of the dataString in one row inside the csv file, so they can be uploaded in batch later
sensorData.print(dataString);
p++;
}
else {
sensorData.println(dataString);
p = 0;
}
}
}
}

```



```

}
sensorData.close();
}
else {
  Serial.println("fail to write");
}

sdfile = SD.open(filename, FILE_WRITE); // Open the backup CPRData.csv file,, which contains all the historic data
if (sdfile) {
  sdfile.println(vol);
  sdfile.println(dataString); // Write the new data into the file
  sdfile.close();
}
else {
  Serial.println("fail to write");
}
Serial.println(dataString);
}
//////////////////////////////////////SAVE DATA TO SD CARD//////////////////////////////////////

//////////////////////////////////////MARK FILE AS END//////////////////////////////////////
//This function is used to mark an "END" string at the end of an unuploaded file, so the Arduino knows when to stop upload data
void markend() {
  if (marked == false) { // If the file is not marked
    sensorData = SD.open(sheetname, FILE_WRITE); // Open the upload file
    if (sensorData) {
      sensorData.println("\nEND\n"); // Add "END" at then end of the file
      sensorData.close();
    }
    else {
      Serial.println("fail to write");
    }
    sdfile = SD.open(filename, FILE_WRITE); // Open the backup CPRData.csv file
    if (sdfile) {
      sdfile.println("\nEND\n"); // Add "END" at then end of the file
      sdfile.close();
    }
    else {
      Serial.println("fail to write");
    }
    marked = true; // The file is marked
  }
}
//////////////////////////////////////MARK FILE AS END//////////////////////////////////////

//////////////////////////////////////UPLOAD DATA TO GOOGLE SHEETS//////////////////////////////////////
//This function uploads the saved csv file to the google sheets
void readdata() {
  sensorData = SD.open(sheetname); // Open the upload file
  if (sensorData) {
    while (sensorData.available()) { // During upload, this while loop only stops is the SD card module is not communicating or a break is called
      CPRdata = sensorData.readStringUntil('\n'); // Read the first line and assign it to CPRdata string, which contains 100 lines of the dataString (100s of data)
      Serial.println(CPRdata);
      if (client.connectSSL(server, 443)) {
        Serial.println("connected to server");
        client.println("GET https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHyvu9rY4PIPy9038aIQ0/exec?CPRdata=" + CPRdata); // Attach the CPRdata string
        at the end of this Https address to upload data to Google sheets
      }
    }
  }
}

```

```

client.println("Host: script.google.com");
client.println("Connection: close");
client.println();
}
if (CPRdata == "END") { // If the CPRdata is assigned as "END", which means this is the end of the file
  uploaded = true; // Mark the file as uploaded
  break; // Get out of the while loop
}
}
} else Serial.println("fail");
}
}
//////////UPLOAD DATA TO GOOGLE SHEETS//////////

//////////GET CURRENT TIME//////////
// This function gets the current time from the DS3231 module
void readDS3231time(byte * second, byte * minute, byte * hour, byte * dayOfWeek, byte * dayOfMonth, byte * month, byte * year)
{
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set DS3231 register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
  // request seven bytes of data from DS3231 starting from register 00h
  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month = bcdToDec(Wire.read());
  *year = bcdToDec(Wire.read());

  if (*month < 10) { // Following codes format the corresponding time unit into 2 digits
    monString = "0" + String(*month, DEC);
  } else monString = String(*month, DEC);

  if (*dayOfMonth < 10) {
    dayString = "0" + String(*dayOfMonth, DEC);
  } else dayString = String(*dayOfMonth, DEC);

  if (*minute < 10) {
    minString = "0" + String(*minute, DEC);
  } else minString = String(*minute, DEC);

  if (*second < 10) {
    secString = "0" + String(*second, DEC);
  } else secString = String(*second, DEC);
}
//////////GET CURRENT TIME//////////

//////////FORMAT CURRENT TIME INTO A STRING//////////
// This function combines the time data into one formatted string as "2020-02-09" and "23:04:12", for example.
void currtime()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
  dateString = "20" + String(year, DEC) + "-" + monString + "-" + dayString;
  hourString = String(hour, DEC);
  timeString = hourString + ":" + minString + ":" + secString;
}
}

```

```

//////////FORMAT CURRENT TIME INTO A STRING//////////

//////////RESET TIME//////////

// This function reset the current time in the DS3231 module
void setDS3231time(byte second, byte minute, byte hour, byte dayOfWeek, byte dayOfMonth, byte month, byte year)
{
  // sets time and date data to DS3231
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set next input to start at the seconds register
  Wire.write(decToBcd(second)); // set seconds
  Wire.write(decToBcd(minute)); // set minutes
  Wire.write(decToBcd(hour)); // set hours
  Wire.write(decToBcd(dayOfWeek)); // set day of week (1=Sunday, 7=Saturday)
  Wire.write(decToBcd(dayOfMonth)); // set date (1 to 31)
  Wire.write(decToBcd(month)); // set month
  Wire.write(decToBcd(year)); // set year (0 to 99)
  Wire.endTransmission();
}

//////////RESET TIME//////////

//arduino_secrets.h//
// This file stores the SSID nad password of the Wifi network

#define SECRET_SSID "" // Put the SSID between the double quotes
#define SECRET_PASS "" //Put the password between the double quotes

```

// This is the final code for the Arduino MKR 1010 Wifi. It collects data and sends data by connecting to a WPA2 enterprise Wifi network.

//////////////////////////////////WIFI and IOT CLOUD//////////////////////////////////

//This part sets up the libraries and variables for Wifi connection and Https requests

#include <WiFiNINA.h> // Necessary library

#include <WiFiUdp.h> // Necessary library

#include "arduino_secrets.h" // Contains password

char server[] = "script.google.com";

char ssid[] = SECRET_SSID;

char pass[] = SECRET_PASS;

int status = WL_IDLE_STATUS;

WiFiSSLClient client;

String CPRdata = "";

//////////////////////////////////WIFI and IOT CLOUD//////////////////////////////////

//////////////////////////////////RTC//////////////////////////////////

//This part sets up the libraries and variables for the DS3231 RTC module

#include <Wire.h>

#define DS3231_I2C_ADDRESS 0x68

String monString;

String dayString;

String hourString;

String minString;

String secString;

String dateString;

String timeString;

String dataString;

byte decToBcd(byte val) {

return (val / 10 * 16) + (val % 10);

}

byte bcdToDec(byte val) {

return (val / 16 * 10) + (val % 16);

}

//////////////////////////////////RTC//////////////////////////////////

//////////////////////////////////SD//////////////////////////////////

//This part sets up the libraries and variables for the micro SD card module

#include <SPI.h>

#include <SD.h>

String filename = "CPR_Data.csv";

String title = "Date,Timestamp,Acceleration,Rate";

String sheetname;

int SDPin = 7; // CS (chip selected) Pin to digital pin 7

int p;

int vol;

bool uploaded = true;

bool recorded = false;

bool filenameed = false;

bool marked = false;

File sensorData;

File sdfile;

//////////////////////////////////SD//////////////////////////////////

//////////////////////////////////ACC//////////////////////////////////

//This part sets up the libraries and variables for calculate the chest compression rate based on the ACC readings from CorPatch

int accPin = A1; // ACC pin to Analog pin 1

int acc;

int maxval;

```

int compressionRate;
int maxThreshold = 910; // ACC readings that are higher than this threshold count as a valid compression
int minThreshold = 875; // ACC readings that are lower than this threshold count as no compression
int pinThreshold = 100; // If ACC reading is higher than this threshold, it means CorPatch is plugged in
unsigned long t;
unsigned long t1;
unsigned long t2;
unsigned long before;
const long interval = 400; // Chest compression rate is calculated every 400ms
const long interval2 = 2000; // If longer than 2000ms,
////////////////////////////////////////ACC////////////////////////////////////////

////////////////////////////////////////SETUP FUNCTION////////////////////////////////////////
// This setup function only run once at the beginning when the device is turned on
void setup() {
  Wire.begin();
  Serial.begin(115200);
  SD.begin(SDPin);
  //setDS3231time(40, 44, 13, 2, 11, 8, 20); // This function is used to sync current time
}
////////////////////////////////////////SETUP FUNCTION////////////////////////////////////////

////////////////////////////////////////LOOP FUNCTION////////////////////////////////////////
// This loop function is looped as long as the device is turned on after executing the setup function
void loop() {
  if (analogRead(accPin) > pinThreshold) { // If the ACC value is higher than pinThreshold (100), it means that the CorPatch is plugged in
    acceleration(); // Calculate compression rate
    currtime(); // Get current time
    unsigned long present = millis();
    if ((present - before) > 999) { //called every 1000ms
      savedata(); // Create a new csv file, save the current time (date, time), ACC value, compression rate value to this file in the micro SD card
      before = present; // refresh timer
    }
    uploaded = false; // This file is not uploaded to Google sheet yet
    marked = false; // This saved csv file is not marked as 'END' yet
  }
  else if ((analogRead(accPin) <= pinThreshold) and uploaded == false) { //If ACC value is lower than pinThreshold (100), it means CorPatch is unplugged, and if the data is not uploaded
yet, upload the data here
    markend(); // Mark an 'END' string at the end of the just saved csv file
    while (status != WL_CONNECTED) { // This is a while loop function, which means it only stops until the condition is not true or a 'break' is called during the loop. Here if Wifi is not
connected, the function loops until Wifi is connected or 'break' is called
      Serial.println("Connecting to wifi...");
      status = WiFi.beginEnterprise(ssid, user, pass); // Connect to Wifi with SSID, username and password
      delay(10000); //Delay for 10s is maybe needed because it takes time to connect to WPA2 enterprise
      if (analogRead(accPin) > pinThreshold) { // If the CorPatch is plugged in again, break this loop
        break;
      }
      if (uploaded == true) { // If data is uploaded, break this loop
        break;
      }
    }
    Serial.println("Connected to wifi");
    readdata(); // send data to Google sheets
    recorded = false; //
    filenameed = false; // The new file for next case is not created yet
  }
  else { //After uploading the data of this case, the device goes to standby mode, which does nothing until the CorPatch is plugged in again

```

```

recorded = false;
filenamed = false; // The new file for the next case is not created yet
marked = false; // The new file for the next case is not marked as 'END' yet
Serial.println(analogRead(accPin));
}
}
/////////////////////////////////LOOP FUNCTION/////////////////////////////////

/////////////////////////////////ACCELERATION/////////////////////////////////
//This function is called to get current ACC value and calculate compression rate every 400ms
void acceleration() {
acc = analogRead(accPin); // Get ACC raw value
if (millis() - t > interval) { // Do this every 400ms
if (acc >= maxval) { // If the ACC value is increasing
maxval = acc; // Max ACC value is the new reading
} else { // If the ACC value starts to decrease, which means the last maxval is the peak during this period
if (maxval > maxThreshold) { // If the peak ACC value is bigger than maxThreshold (910)
compressionRate = 60000 / (millis() - t); // Calculate the compression rate by dividing 60000ms by the interval between current and last time the peak value occurs
maxval = 0; // maxval set back to 0
t = millis(); // current time is recorded to t
}
}
}
if (millis() - t > interval2 && acc < minThreshold) { // If ACC value is smaller than minThreshold (875) for more than 2 seconds
compressionRate = 0; // Compression rate is set as 0
}
}
/////////////////////////////////ACCELERATION/////////////////////////////////

/////////////////////////////////SAVE DATA TO SD CARD/////////////////////////////////
// This function is called to save the current data to a newly created csv file in the micro SD card
void savedata() {
if (filenamed == false) { // If a new file is not created yet for the new rescue case
sheetname = monString + dayString + hourString + minString + ".csv"; // Create a new file by naming it with current time, this file is used to upload data to Google sheets later
filenamed = true; // File is created for the rescue case
sdfile = SD.open(filename, FILE_WRITE); // Open the newly created file
if (sdfile) {
sdfile.println(title); // Write the file title "Date,TimeStamp,Acceleration,Rate" in the first row of the csv file, so here 4 columns with these 4 titles are created inside the csv file
sdfile.close();
}
else {
Serial.println("fail to write");
}
}
dataString = dateString + "," + timeString + "," + String(acc) + "," + String(compressionRate) + ","; // Combine the whole data into one string with commas in between
sensorData = SD.open(sheetname, FILE_WRITE); // Save the data to the file that is used to upload to Google sheets
if (sensorData) {
if (p < 99) { // Write the 100 lines of the dataString in one row inside the csv file, so they can be uploaded in batch later
sensorData.print(dataString);
p++;
}
else {
sensorData.println(dataString);
p = 0;
}
sensorData.close();
}
else {

```

```

Serial.println("fail to write");
}

sdfile = SD.open(filename, FILE_WRITE); // Open the backup CPRData.csv file,, which contains all the historic data
if (sdfile) {
  sdfile.println(vol);
  sdfile.println(dataString); // Write the new data into the file
  sdfile.close();
}
else {
  Serial.println("fail to write");
}
Serial.println(dataString);
}
//////////////////////////////////////SAVE DATA TO SD CARD//////////////////////////////////////

//////////////////////////////////////MARK FILE AS END//////////////////////////////////////
//This function is used to mark an "END" string at the end of an unuploaded file, so the Arduino knows when to stop upload data
void markend() {
  if (marked == false) { // If the file is not marked
    sensorData = SD.open(sheetname, FILE_WRITE); // Open the upload file
    if (sensorData) {
      sensorData.println("\nEND\n"); // Add "END" at then end of the file
      sensorData.close();
    }
    else {
      Serial.println("fail to write");
    }
    sdfile = SD.open(filename, FILE_WRITE); // Open the backup CPRData.csv file
    if (sdfile) {
      sdfile.println("\nEND\n"); // Add "END" at then end of the file
      sdfile.close();
    }
    else {
      Serial.println("fail to write");
    }
    marked = true; // The file is marked
  }
}
//////////////////////////////////////MARK FILE AS END//////////////////////////////////////

//////////////////////////////////////UPLOAD DATA TO GOOGLE SHEETS//////////////////////////////////////
//This function uploads the saved csv file to the google sheets
void readdata() {
  sensorData = SD.open(sheetname); // Open the upload file
  if (sensorData) {
    while (sensorData.available()) { // During upload, this while loop only stops is the SD card module is not communicating or a break is called
      CPRdata = sensorData.readStringUntil('\n'); // Read the first line and assign it to CPRdata string, which contains 100 lines of the dataString (100s of data)
      Serial.println(CPRdata);
      if (client.connectSSL(server, 443)) {
        Serial.println("connected to server");
        client.println("GET https://script.google.com/macros/s/AKfycbwZVNu1a-d2qICRNFSq0CgYL3DwjHHYvu9rY4PIPYy9038alQ0/exec?CPRdata=" + CPRdata); // Attach the CPRdata string
        at the end of this Https address to upload data to Google sheets
        client.println("Host: script.google.com");
        client.println("Connection: close");
        client.println();
      }
    }
  }
}

```

```

if (CPRdata == "END") { // If the CPRdata is assigned as "END", which means this is the end of the file
    uploaded = true; // Mark the file as uploaded
    break; // Get out of the while loop
}
}
} else Serial.println("fail");
}
////////////////////////////////////

////////////////////////////////////GET CURRENT TIME////////////////////////////////////
// This function gets the current time from the DS3231 module
void readDS3231time(byte * second, byte * minute, byte * hour, byte * dayOfWeek, byte * dayOfMonth, byte * month, byte * year)
{
    Wire.beginTransmission(DS3231_I2C_ADDRESS);
    Wire.write(0); // set DS3231 register pointer to 00h
    Wire.endTransmission();
    Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
    // request seven bytes of data from DS3231 starting from register 00h
    *second = bcdToDec(Wire.read() & 0x7f);
    *minute = bcdToDec(Wire.read());
    *hour = bcdToDec(Wire.read() & 0x3f);
    *dayOfWeek = bcdToDec(Wire.read());
    *dayOfMonth = bcdToDec(Wire.read());
    *month = bcdToDec(Wire.read());
    *year = bcdToDec(Wire.read());

    if (*month < 10) { // Following codes format the corresponding time unit into 2 digits
        monString = "0" + String(*month, DEC);
    } else monString = String(*month, DEC);

    if (*dayOfMonth < 10) {
        dayString = "0" + String(*dayOfMonth, DEC);
    } else dayString = String(*dayOfMonth, DEC);

    if (*minute < 10) {
        minString = "0" + String(*minute, DEC);
    } else minString = String(*minute, DEC);

    if (*second < 10) {
        secString = "0" + String(*second, DEC);
    } else secString = String(*second, DEC);
}
////////////////////////////////////GET CURRENT TIME////////////////////////////////////

////////////////////////////////////FORMAT CURRENT TIME INTO A STRING////////////////////////////////////
// This function combines the time data into one formatted string as "2020-02-09" and "23:04:12", for example.
void currtime()
{
    byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
    readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
    dateString = "20" + String(year, DEC) + "-" + monString + "-" + dayString;
    hourString = String(hour, DEC);
    timeString = hourString + ":" + minString + ":" + secString;
}
////////////////////////////////////FORMAT CURRENT TIME INTO A STRING////////////////////////////////////

////////////////////////////////////RESET TIME////////////////////////////////////
// This function reset the current time in the DS3231 module

```



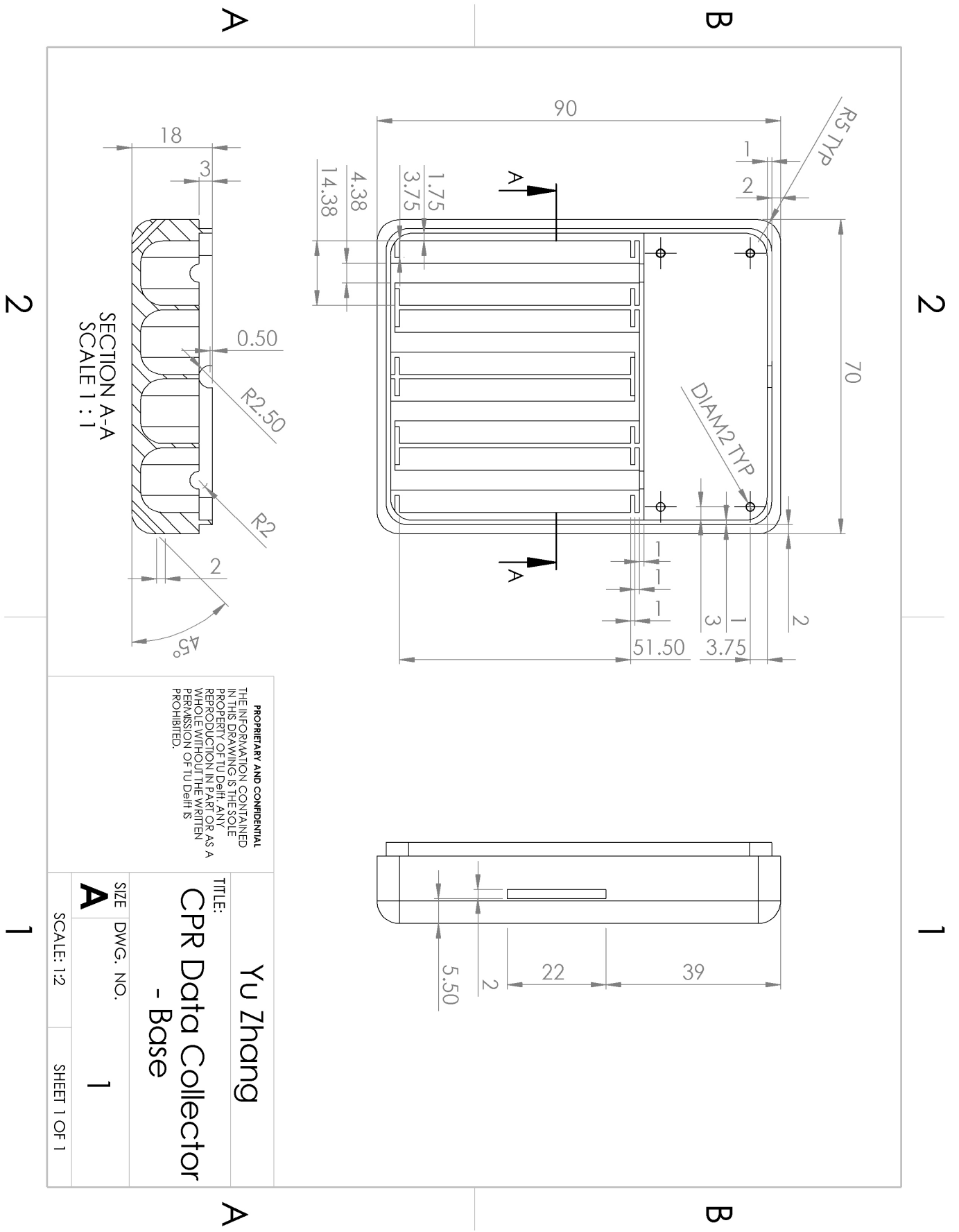
```
void setDS3231time(byte second, byte minute, byte hour, byte dayOfWeek, byte dayOfMonth, byte month, byte year)
```

```
{  
  // sets time and date data to DS3231  
  Wire.beginTransmission(DS3231_I2C_ADDRESS);  
  Wire.write(0); // set next input to start at the seconds register  
  Wire.write(decToBcd(second)); // set seconds  
  Wire.write(decToBcd(minute)); // set minutes  
  Wire.write(decToBcd(hour)); // set hours  
  Wire.write(decToBcd(dayOfWeek)); // set day of week (1=Sunday, 7=Saturday)  
  Wire.write(decToBcd(dayOfMonth)); // set date (1 to 31)  
  Wire.write(decToBcd(month)); // set month  
  Wire.write(decToBcd(year)); // set year (0 to 99)  
  Wire.endTransmission();  
}
```

```
////////////////////////////////////RESET TIME////////////////////////////////////
```

```
//arduino_secrets.h//  
// This file stores the SSID and password of the Wifi network  
  
#define SECRET_SSID "" // Put the SSID between the double quotes  
#define SECRET_PASS "" //Put the password between the double quotes
```

Appendix J - Solidworks model assembly and parts drawings



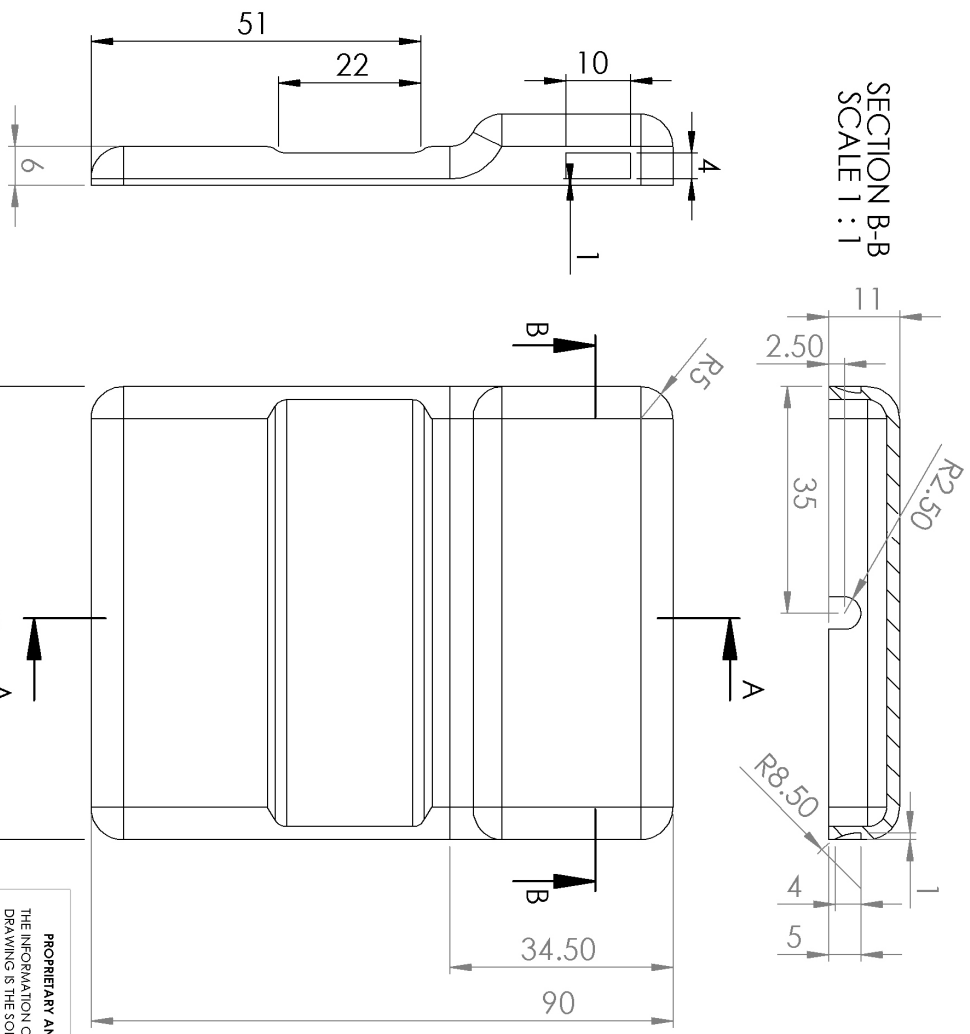
PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED
 IN THIS DRAWING IS THE SOLE
 PROPERTY OF TU DEFT. ANY
 REPRODUCTION IN PART OR AS A
 WHOLE WITHOUT THE WRITTEN
 PERMISSION OF TU DEFT IS
 PROHIBITED.

YU Zhang		TITLE:	
CPR Data Collector		- Base	
SIZE	DWG. NO.	1	
A	1	SCALE: 1:2	
1		SHEET 1 OF 1	

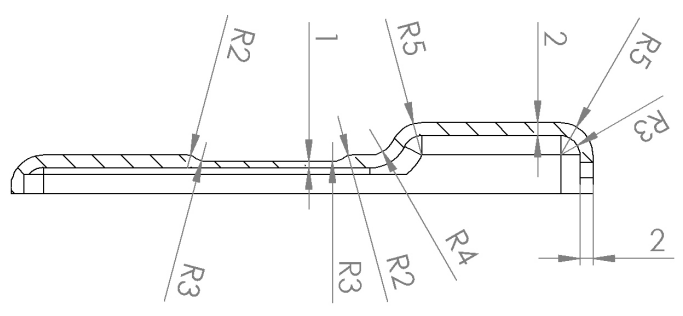
2

1

SECTION B-B
SCALE 1 : 1



SECTION A-A
SCALE 1 : 1



A

B

A

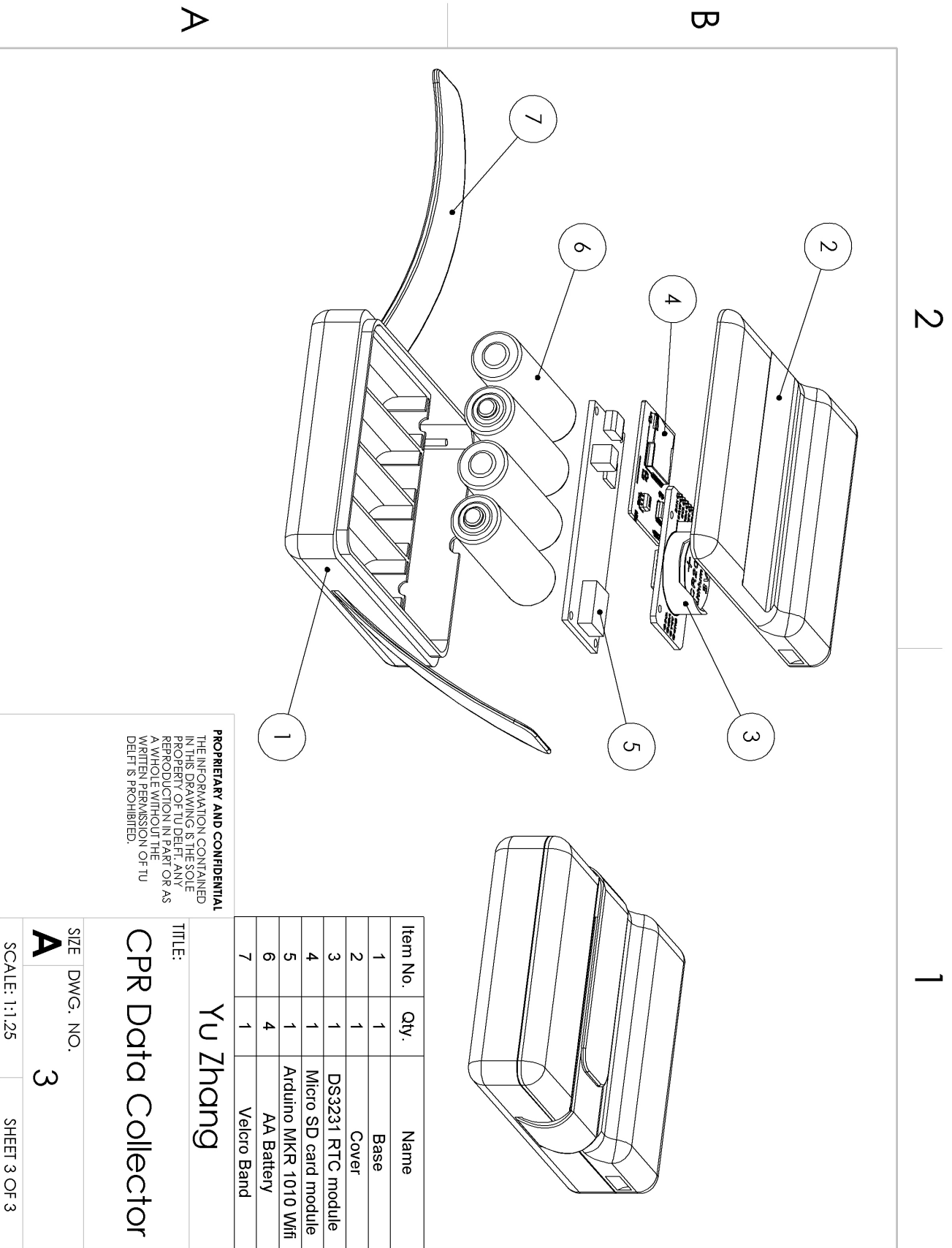
B

2

1

PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED IN THIS
 DRAWING IS THE SOLE PROPERTY OF
 <INSERT COMPANY NAME HERE>. ANY
 REPRODUCTION IN PART OR AS A WHOLE
 WITHOUT THE WRITTEN PERMISSION OF
 <INSERT COMPANY NAME HERE> IS
 PROHIBITED.

TITLE:		Yu Zhang	
CPR Data Collector			
- Cover			
SIZE	DWG. NO.		
A	2		
MATERIAL	WEIGHT		
PLA	17g		
SCALE: 1:1	SHEET 2 OF 3		



Item No.	Qty.	Name
1	1	Base
2	1	Cover
3	1	DS3231 RTC module
4	1	Micro SD card module
5	1	Arduino MKR 1010 Wifi
6	4	AA Battery
7	1	Velcro Band

PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED
 IN THIS DRAWING IS THE SOLE
 PROPERTY OF TU DELFT. ANY
 REPRODUCTION IN PART OR AS
 A WHOLE WITHOUT THE
 WRITTEN PERMISSION OF TU
 DELFT IS PROHIBITED.

TITLE: Yu Zhang
 CPR Data Collector

SIZE DWG. NO. A 3
 SCALE: 1:1.25 SHEET 3 OF 3

2

1

2

1

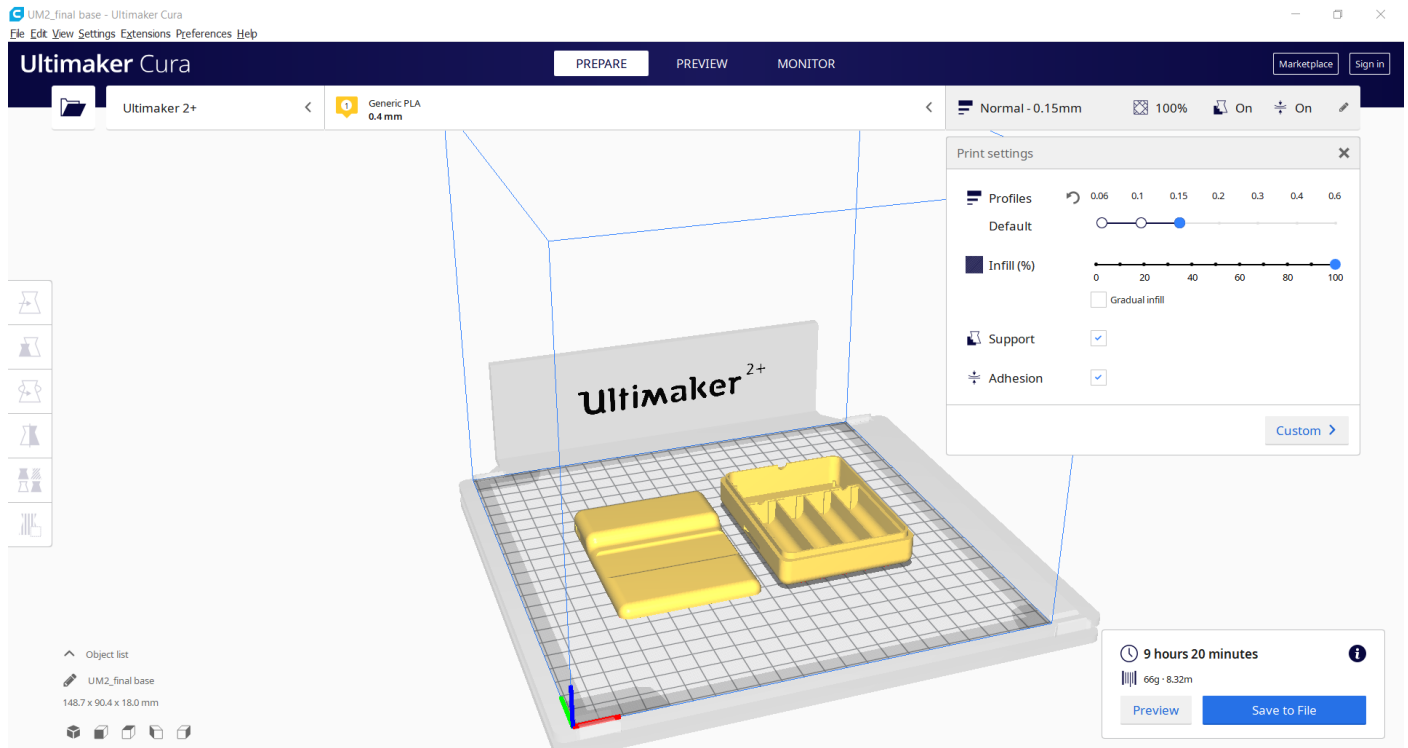
A

B

A

B

Appendix K - Purchase links of the cost estimation



As shown in the figure, to print one set of housing in Ultimaker 2+, it needs 9 hours 20 minutes and 66 grams of filament. If the manufacture size is 15 data collectors, the total printing time will be 150 hours (6.25 days) and the total filament needed will be 990 grams. According to <http://goedkoop3dprinterhuren.nl/prijzen/>, renting an Ultimaker 2+ for a week costs €130, and 1000 grams of filament costs €20. In total, €150 is estimated to print the housings of 15 data collectors. Following are purchase links that are used to estimate the cost:

3D printing parts

Prijzen – Goedkoop 3D printer huren. (n.d.). Goedkoop3dprinterhuren. <http://goedkoop3dprinterhuren.nl/prijzen/>

DS3231 RTC module

AZDelivery Real Time Clock RTC DS3231 I2C Real time klok voor Arduino met eBook: Amazon.nl. (n.d.). Amazon.Nl. https://www.amazon.nl/gp/product/B01M2B7HQB/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1

Micro SD card module

Kiwi Electronics. (n.d.-b). MicroSD card breakout board+. <https://www.kiwi-electronics.nl/microsd-card-breakout-board-plus?search=micro%20sd&description=true>

SanDisk micro SD card 16GB

Sandisk SDSAQM-016G-B35 flashgeheugen 16 GB MicroSDHC. (n.d.). Bol.Com. <https://www.bol.com/nl/p/sandisk-sdsdqm-016g-b35-flashgeheugen-16-gb-microsdhc/9000000012279555/?s2a=#productTitle>

Arduino MKR 1010 Wifi

Arduino MKR WiFi 1010 | Arduino Official Store. (2020). Arduino. <https://store.arduino.cc/arduino-mkr-wifi-1010>

10kΩ resistor

Kiwi Electronics. (n.d.-c). Weerstand 10K Ohm - 1/4 watt - 5% - 10 stuks. <https://www.kiwi-electronics.nl/Weerstand-10K-ohm-1-4-watt-5-procent-10-stuks?search=resistor%2010k&description=true>

AA battery

Duracell batterij 72 pack AA. (n.d.). Bol.Com. https://www.bol.com/nl/p/duracell-batterij-72-pack-aa/9200000074443115/?Referrer=ADVNLGO0002018-G-58263947517-S-493987207082-9200000074443115&gclid=Cj0KCQjw-O35BRDVARIsAJU5mQXBfA8yNOgvndlxoUct4dugEVn16sWyt8vuOUcybdNWKqzj8Mq-bFMaAsUXEALw_wcB

CR2032 cell battery

VARTA CR2032 lithium knoopcellen 3V batterij in originele blisterverpakking, 10-pack: Amazon.nl. (n.d.). Amazon.NL. https://www.amazon.nl/gp/product/B018S4PTNW/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&psc=1

Velcro bands double sided

KabelDirekt - klittenband kabelbinders hersluitbaar - 20 mm x 5 m - (rol voor kabels, vrij op maat te snijden & herbruikbaar, wit): Amazon.nl. (n.d.). Amazon.NL. https://www.amazon.nl/gp/product/B07BYLY15R/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1

9 pins D-Sub cable 2M

Premium seriële RS232 kabel 9-pins SUB-D (m) - 9-pins SUB-D (m) / gegoten connectoren - 2 meter - 9-pin SUB-D (RS232/RS485) - SUB-D (9p/15p/25p) - Computer | Onlinekabelshop.nl. (n.d.). Onlinekabelshop. <https://www.onlinekabelshop.nl/seriele-kabel-9pins-sub-d-mannelijk-9pins-sub-d-mannelijk-2-meter>

Wires

VELLEMAN - K/MOWM Schakeldraad-assortiment - 10 kleuren - 60 m - 0,2 mm2 volledig materiaal 276230: Amazon.nl. (n.d.). Amazon.NL. https://www.amazon.nl/VELLEMAN-Schakeldraad-assortiment-kleuren-volledig-materiaal/dp/B001IRVDV4/ref=sr_1_2?__mk_nl_NL=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=arduino+wire&qid=1597746429&sr=8-2

Contact plates

Spring 100st Battery Battery Shrapnel AA of AAA-batterij Spring 7 No. positieve en negatieve Contact Stukken 50pairs Drop Ship Easy to install: Amazon.nl. (n.d.). Amazon.NL. https://www.amazon.nl/gp/product/B08DHRCL6/ref=ox_sc_act_title_2?smid=A2GWW94JYD74WR&psc=1