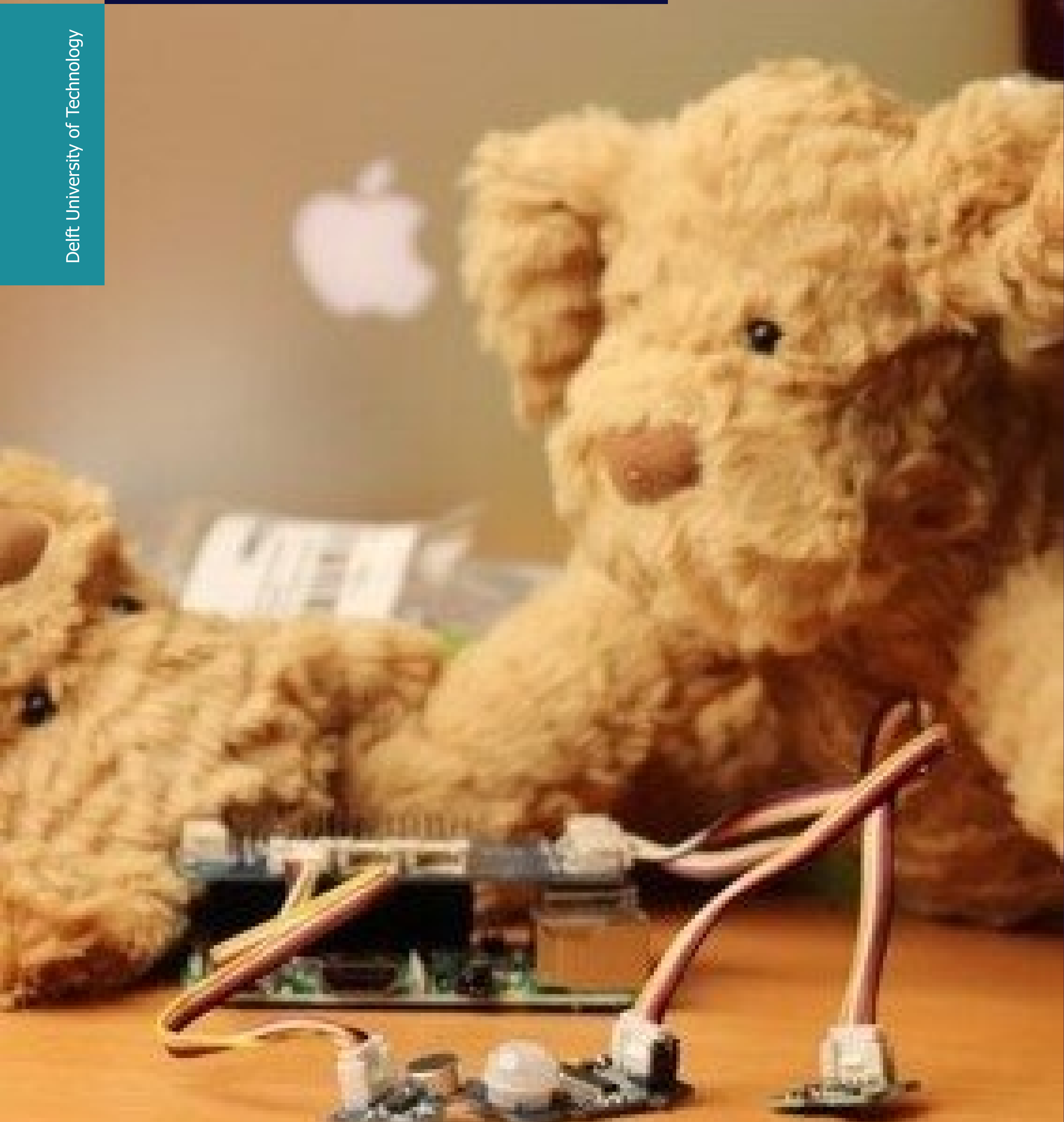
The Smart Teddy Project

Design of a data acquisition system to monitor seniors with dementia and detect dangerous situations

T. N. van der Spijk (4693817) & A. Hamo (4726960)

Delft University of Technology



The Smart Teddy Project

Design of a data acquisition system to monitor seniors with dementia and detect dangerous situations

by

T. N. van der Spijk (4693817) & A. Hamo (4726960)

In partial fulfillment of the requirements for the graduation project of

BSc Electrical Engineering

Date of Submission:

June 18, 2021

Students:	Tim N. van der Spijk	4693817
	Alan Hamo	4726960
Instructors:	Dr. Zaid Al-Ars	Associate Professor the Delft University of Technology
	Dr. Hani Al-Ers	Researcher at the Hague University of Applied Sciences
Thesis Committee:	Dr. Prof. Catholijn M. Jonker	Full professor at the Delft University of Technology
	Dr. Matthias Möller	Associate Professor at the Delft University of Technology
	Dr. Zaid Al-Ars	Associate Professor the Delft University of Technology
	Dr. Hani Al-Ers	Researcher at the Hague University of Applied Sciences
Institution:	Delft University of Technology	
Place:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft	
Date:	June 18, 2021	
Defence:	13:30-15:00, June 29, 2021.	

Delft University of Technology

Preface

The amount of people dealing with dementia is rising globally. The amount of caretakers is, however, not. Therefore, technological aids are needed to support people dealing with dementia and relieve the stress on their caretakers. Current solutions provide tracking of people with dementia. Also, different robots exist that provide people with companionship. However, no solution exists that combines tracking and companionship capabilities. Therefore, the Smart Teddy is introduced. The Smart Teddy can track different indicators that indicate the progress of dementia and simultaneously provide the user with companionship through interaction. The goal of this thesis is to design a data acquisition system that acquires meaningful data that can be used for the development of algorithms that will autonomously determine the progress of dementia. To achieve this, a system with a Teddy and a Base station has been designed. The Teddy has a sound-, a carbon monoxide-, a smoke- and a movement sensor. Also, a real-time module is implemented to be able to assign the current time to the measurement data. Lastly, a GPS and GSM module is implemented to be able to track seniors in case they wander. In the Base station, a mmWave sensor is implemented that tracks the position, velocity, and direction of the persons present in the room. Also, a processor is implemented that gathers and stores the data from the mmWave sensor and the data from the Teddy which is sent via a LoRa connection. In addition, the designed system can store the collected data for more than one week. The collected data can be used by an expert in dementia to extract meaningful information about dementia progress, after that, an expert in digital signal processing is needed to develop algorithms that estimate the quality of life of a senior suffering from dementia.

The Smart Teddy project is carried out by a total of six students of the Electrical Engineering Bachelor. The project was initiated by The Hague University of Applied Sciences in September 2018. This thesis is created by two students concluding their Bachelor in Electrical Engineering at the Delft University of Technology. This thesis is a product of the years the authors have enjoyed studying Electrical Engineering.

We would like to thank our client Hani Al-ers for his continued support and enthusiasm. We trust that our design is in good hands with you and look forward to the future development of the Smart Teddy. We would also like to thank our supervisor Dr. Zaid Al-ars for being critical and helping us learn from the process of creating this design and thesis. A big thank you to Jeroen Bastemeijer, Mathias Möller, Ezra, and Annemieke who spent time helping us during the course of this project. Your tips were of great value to us. Last but not least we would like to thank the rest of the team: Shea Haggerty, Laura Croes, Taha Küçükçelebi, and Lyana Usa for their amazing work, their collaboration, and most of all their strong work ethic.

T. N. van der Spijk (4693817) & A. Hamo (4726960)
Delft, June 2021

Contents

1 Introduction

This chapter starts with an assessment of the current situation and the main motivation of this project in Section 1.1. In Section 1.2, an overview of the Smart Teddy project is given. Section 1.3 provides a thorough definition of the problem at hand and concludes by giving the research question of this thesis. To be able to provide a knowledgeable design, first, the previous work in the Smart Teddy was analyzed in Subsection 1.4. Second, a literature study was conducted in the field of QoL and indicators of dementia in Subsection 1.4. Using the background, research was done into the state of the art considering sensing technologies in Section 1.5.

1.1. Situational assessment

Dementia is one of the biggest global health- and social care challenges people are facing today. The number of people with dementia was estimated to be 46.8 million in 2015. This number will almost double every 20 years [38]. Dementia is a collective name for syndromes in which the memory, thinking, behavior, and the ability to perform everyday activities deteriorate. Dementia has a physical, psychological, social, and economic impact, not only on people with dementia but also on their carers, families, and society [54]. In 2015, the global costs were estimated to be $818 billion US globally. Currently, the global costs are estimated to be above $1 trillion dollar US [3]. These costs are distributed between three major subcategories, namely, direct medical, social care, and informal care. Roughly 20% of the costs are spent on direct care, 40% on direct social sector costs, and the remaining 40% are spent on informal care [38]. A considerable portion of these informal costs is used in home-based care in the early stages of dementia. Informal caregivers who include spouses, adult children, daughters- and sons-in-law, and friends. Women are far more likely to be the caregivers in all countries. Providing care to people with dementia can result in significant strain for those who provide most of the care [55].

The field of research in dementia is an active- and growing field of research. In the Netherlands, for example, the senior population is growing where the amount of caregivers is not [46]. This creates an increasing need for technological aids in the health care system and the home care system in particular. An example of a technological aid is a social robot used to provide emotional, cognitive, and physical support for people with dementia. Such a social robot is intended to enhance the Quality of Life or QoL of seniors with dementia. They can be classified as pet, assistive, humanoid, and telepresence robots according to their predominantly intended use. Pet robots are intended to enhance social interaction with people affected by dementia, where assistive and humanoid robots are deployed to support people with dementia in daily life activities. Telepresence robots are used to provide social connectedness, remote support, care, and medical treatment [19].

1.2. The Smart Teddy

The Hague University of applied science aims to develop a product that can automate the measurement of QoL in senior citizens dealing with early-stage dementia. This product is disguised as a Smart Teddy, which aims to support patients to remain living independently in their own home and with that decrease stress on care homes. The Smart Teddy will offer seniors companionship and will measure indicators of QoL to track the progress of dementia, besides that, the Smart Teddy will be able to detect some dangerous situations to update or alert the caregiver and/or family members. The Smart Teddy can be seen as a combination of a pet robot and assistive robot and consists of a Teddy and a Base station.

Teddy: The Teddy is a soft and cuddly stuffed animal that will offer seniors companionship by exhibiting interaction like moving its tail and barking. Currently, the Teddy houses several sensors such as a touch sensor, a Global Positioning System or GPS, motion sensor, light sensor, Gyroscope, smoke sensor, and a microphone to collect data about daily activities that can indicate progress dementia of senior citizens. The collected data will be used to automate the estimation of the quality of life. The Teddy sends the collected data to a Base station.

Base station: Currently, the Base station houses a receiver, a processor board, a smoke sensor, a microphone, a camera, and a wireless charger for the Teddy and connectivity for the user. The processor processes the data received from the Teddy and the sensors.

1.3. Problem definition

As mentioned in Section 1.1, dementia has an impact on the economy and society. The Hague University of applied science aims to contribute to mitigating this impact by introducing the Smart Teddy. This aims to prolong the time for which seniors with dementia can live independently in their homes. This product is unique since it combines monitoring the progress of dementia, offering companionship, and alerting caregivers in dangerous situations. These functions combined will have a positive impact on seniors suffering from dementia, their caregivers, and people closely related to the particular seniors. To the best of our knowledge, no product exists yet that combines these three functions of the Smart Teddy.

The final goal of the Smart Teddy project is to estimate the QoL of seniors with dementia by applying algorithms to data collected by the system. However, the current data acquisition system does not function properly. In addition, the Teddy is not cuddly since the components inside are too large. Solving the technical issues the device faces requires the expected knowledge from an Electrical Engineer with a Bachelor's degree. However, the final goal of the project requires expertise in digital signal processing to process the collected data into useful information to be used by an expert in dementia who can extract the interpretation of the data in terms of the QoL.

In addition to the technical challenges, ethical, legal, and social implication are considered to prevent slow adoption, incorrect implementation and inappropriate use [23]. Important values that should be considered in the early stages of designing a monitoring system in dementia care are identified to be: Privacy, consent, respect, individuality, dignity, warmth, safety, and well-being [40]. Besides that, the acceptance and perception of seniors with dementia of the Teddy are also important because, for the Teddy to function properly, it should be in the vicinity of the senior. For example, the Teddy cannot measure wandering if the senior wanders somewhere else where the Teddy is placed.

The current situation of COVID-19 shows the urgency of such a product. A study conducted in the United States about the relationship between dementia and COVID-19 showed that patients with dementia were at increased risk for COVID-19 [53]. Those findings highlighted the need to protect seniors with dementia from COVID-19 but also pointed out the danger for caregivers and loved ones. This product will contribute to mitigating the risk for COVID-19 and help decrease the loneliness of seniors with dementia by offering companionship. This project will also contribute to the third goal of the SDG goals (sustainable development goals) which is to ensure healthy lives and promote well-being for all at all ages [49].

This project aims to redesign the current prototype such that data collection is automated and the Teddy is a stand-alone device with a battery and charging method. In addition, the project will take the responsibility to implement interaction to offer the seniors companionship. Therefore, the project is split into three sub-domains: Data acquisition, Human interaction & integration, and Power & charging. The last two sub-domains will be described in two other theses.

In this thesis, the focus will be on the data acquisition system. The primary goal is to design an automated system that collects useful data for algorithms that will be applied in the future to enable the estimation of QoL. In addition, this thesis will address the use of sensor data to detect dangerous situations such as falling senior or high carbon monoxide concentrations. In doing so, this thesis will answer the following question:

How to design a data acquisition system that firstly, collects data about seniors' dementia-related activities such as wandering, social contact, sleep rhythm, and emotions and secondly, detects dangerous situations such as falling and the presence of high concentrations of Carbon Monoxide gas?

1.4. Background

Previous work in the Smart Teddy project

Over the course of three years, Dr. Hani Al-Ers has been the supervisor of the development of three prototypes of the Smart Teddy. Throughout these years 18 students of The Hague University have worked on the project. In Appendix A a description of the latest work is given. In Table 1.1, the indicators they recommend in their work are given. Also, in Figure 1.1, a schematic view of the latest version of the Smart Teddy is given.

Table 1.1: Indicators as suggested by previous work in the Smart Teddy project.

Indicator	Sensor type
Wandering	Infrared sensor & Sound
Social contact	Infrared sensor & Sound
Life rhythm	GPS & Gyroscope
Day- & night rhythm	Infrared sensor & Sound
Senior motion & location	GPS & Gyroscope
Eating rhythm & Body weight	Camera
Emotion	Camera
Forgetting critical actions	Gas sensor
Falling	Sound & Camera

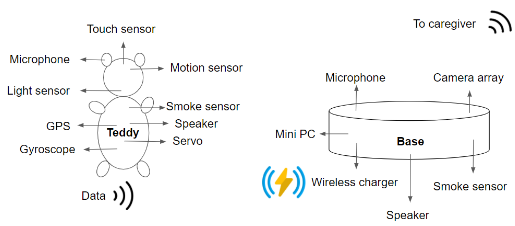


Figure 1.1: Third prototype of the Smart Teddy

Literature study

Since the goal of the Smart Teddy - project is to estimate the QoL of seniors with dementia, it is important for this project that the correct indicators are measured. The primary goal of this literature study is thus to verify the indicators provided by previous efforts with literature in the field of QoL indicators. In Table 1.2, the indicators that will be measured are given. The full analysis is given in Appendix B.

Table 1.2: Indicators to be measured with references

Indicator	References
Wandering	[11] [12]
Social Interaction	[48] [24]
Emotion	[57] [30]
Daily activities	[27] [16]
Sleep- and eating rhythm	[15] [20]
Falling	[14] [13]
Other dangerous situations	[26]

1.5. State of the art

In this section, the research previously conducted in the field of sensing of the mentioned indicators will be analyzed. The different technologies that can be used to measure an indicator will be discussed. A summary is given in Table 1.3. This analysis will form the basis for the program of requirements in Chapter 2 and ultimately for the detailed design in Chapter 3.

Wandering

Research in automated wandering detection focuses on two regions: Outdoor and indoor detection of wandering. Outdoor wandering can be detected using GPS sensors in a wearable to track the senior and

Table 1.3: Summary of the State of the Art

Indicators: Sensors	Wandering	Social interaction	Emotion	Daily activities	Sleep and eating rhythm	Fall detection	Other dangerous situations
mmWave	x	x	x	x	x	x	
PIR	x	x		x		x	
RFID	x						
Microphone			x	x		x	
GPS	x						
Video			x	x	x		
Wearables			x		x		
RGB-D					x		
Vibration						x	
Gas sens.							x

use machine learning to detect whether a person is wandering or not [51] [8].
Three methods of indoor detection of wandering will be discussed. In multi-room scenarios such as nursing homes, tracking can be done using Radio Frequency Identification presence detection [51]. The main downside of this method is the need for the senior to wear a tag that can be detected by the RFID sensor. In single-room scenarios Pyro-electric Infra-Red (PIR) sensors combined with a Deep Convolutional Neural Network Algorithm can be used to detect wandering [17]. The drawback of using PIR to track wandering is that the tracking has to be paused when multiple occupants are in the same room since this disturbs the detection algorithm.
For multi- or single-room scenarios, a mmWave sensor can be used [21]. The location of a user can be determined with 0.30m accuracy, thus path tracking is possible.

Social interaction

PIR sensors can be used for occupancy detection [56] [5]. Besides room occupancy, also occupancy count, location prediction and Human target differentiation is possible. For stationary occupants a vibrating PIR sensor has to be used [56]. Also, a mmWave sensor can be used for occupancy detection [18] [21].

In [18], a comparison between PIR and mmWave technology for occupancy detection was made. The conclusion was that for meeting rooms the mmWave sensor was more suitable to detect the number of occupants. The benefits of the mmWave sensor were: higher detection accuracy, higher reliability in different lighting scenarios, and overall higher performance due to the deterioration in measurement by the PIR sensor when other heat sources are present such as coffee cups or computers.

Emotion

Recording audio and video containing human activities and facial expressions can be used to estimate emotion [39]. Emotion can also be detected using wearable devices [25] [36]. Wandering can also be indicative of the emotional status of the senior: more wandering correlates to more negative emotions [29]. A mmWave sensor can be used to detect the heart rate of a person [52] and heart rate can, in turn, be used to track emotion [44].

Daily activities

Recording audio and video can be used to track precise daily activities. The number of daily activities can be determined by tracking a person's movements using a mmWave- or a PIR sensor [59].

Sleep- and eating rhythm

Precise, wireless sleep monitoring can be achieved using mmWave radar technology [58]. Breathing rhythm and heart rate are detected to determine sleeping [60]. According to [33], prediction of sleep using heart rate can be done using a convolutional neural network, even different stages of sleep can be classified.

Several wearable sensors can be used to track eating rhythm [10]. Examples are Accelerometer, Gyroscope, Microphone, EMG sensor, Piezoelectric sensor, or proximity sensors. For non-wearable sensors, camera-based monitoring can be used. Also, RGB-D sensors can be used to detect eating [34]. Lastly, a mmWave sensor can be used to track skeletal posture [42]. Skeletal posture can in turn be used to detect eating [22] [35].

Falling

Fall detection can be categorized into two kinds of devices: wearable and non-wearable [61]. For wearable devices, an accelerometer can be used to collect useful data for fall detection.

For non-wearable devices, recording audio and/or video could be used for fall detection. Using video for fall detection is more accurate than using sound. However, a camera is heavier and more expensive than a microphone. In addition, cameras cannot work well in dark environments [9]. The biggest issue with cameras is the invasiveness of a seniors' lawful and emotional privacy [61]. The danger is that when a senior feels they are being watched, according to the Hawthorne effect theory they might act differently and thus disturb the measurement results of the other sensors [41]. Also, floor vibration sensors can be used for fall detection [32]. Another option is the use of two pulse-Doppler range control radars [6]. The advantage of the radar over the floor vibration sensor is that it can distinguish a human falling from and other objects falling [32].

The Doppler effect can be used to detect falls using a non-stationary channel sender and receiver model for radio frequencies [6]. A combination of sound sensors can also be used for fall detection [31]. The time difference of arrival of 8 Circular microphones is measured to determine the location of a fall. However, using sound to detect falls requires quiet environments [43]. In [9], a combination of infrared-ultrasonic sensor fusion is used to collect data about location, size, and profile of the senior, of which the data can help in detecting a fall. The infrared sensor provides thermal information and the ultrasonic sensor measures the distance based on Time-of-Flight. In [43], a microwave Doppler sensor is used to collect data in fall detection where the frequency distribution trajectory was the focus, which corresponds to the velocity of the movements while falling. Recently, the advances in Millimeter Wave sensors enable a vast number of applications [4]. Using a support vector machine (SVM) model [4], the data from Millimeter wave sensors can be used to classify motion into five categories: resting, falling, sitting down, walking, and standing up. The Millimeter Wave sensor is based on reflecting electromagnetic waves to determine object range, angle, and velocity.

1.6. Thesis outline

This thesis commenced by conducting the literature study and the state-of-the-art analysis. Using that, the program of requirements is formulated in Chapter 2. The requirements are separated into functional and non-functional requirements. Using the requirements, the design of the data acquisition system is documented in Chapter 3. In Chapter 4, the design will be implemented and verified to the requirements. In Chapter 5, the conclusion and recommendations for future work are presented.

2

Program of Requirements

This chapter addresses the program of requirements for the Data acquisition sub-domain of the Smart Teddy project. In Section 2.1, assumptions made during the project will be discussed followed by verification methods in Section 2.2. The functional requirements and non-functional requirements are addressed in sections 2.3, and 2.4 respectively.

2.1. Assumptions and context

To be able to create a system able to collect data and detect dangerous situations in 11 weeks during the BAP project some assumptions that bound the scope of this project had to be made. These assumptions are made with the knowledge that they might be of influence to the data collected in the real world. In addition, these assumptions help to avoid unneeded complexity in the system. The assumptions are:

- The system operates in a bounded room where the senior does all activities including eating, sleeping & living.
- In the bounded room, the senior is always in range of the sensor in the base station.
- If the senior does leave, the senior always carries the Teddy with him.

All mentioned indicators in the state of art analysis in Section 1.5 are not physical quantities that can be measured directly using a sensor. However, for all indicators, presence detection is of key importance. Presence detection is the first step to track a person's activities. In addition, localizing the person in a room and determining his velocity, dimension, and direction was necessary for all indicators. Presence, position, velocity, dimension, and direction are physical quantities that can be measured using sensors. Besides that, sound and carbon monoxide can also be measured utilizing sensors. By translating the indicators into physical quantities, it was possible to set design requirements.

2.2. Verification methods

To be able to verify whether the requirements are met after the implementation of the design, verification methods are needed. According to Adams [2], verification of requirements can categorized in four methods of verification:

Inspection	The nondestructive examination of a product using one or more of the five senses.
Demonstration	The manipulation of a product as intended to be used to verify the results are as expected.
Test	The verification of a product using predefined inputs and comparing the output with the predefined outputs specified by the requirements.
Analysis	The verification of a product using models, calculations and testing equipment.

2.3. Functional requirements

In Table 2.1, the functional requirements specific to the Data Acquisition part of the Smart Teddy project are listed. The rationale behind the requirement and the verification strategy is included.

Table 2.1: Table with the functional Data Acquisition design requirements. The functional design requirements are labeled with DA.XX

ID	Requirement	Rationale	Verification Method
DA.01	The system should collect data that can be used to determine if a person is present.	Detection if a person can be used to track activity of a senior.	Demonstration
DA.02	The system should collect data that can be used to detect if more than one person is present.	Detecting amount of occupants can be used for Social interaction tracking.	Demonstration
DA.03	The system should collect data that can be used to differentiate between different movements of parts of the body.	Distinguishing between different movements can be used for tracking of physical activities.	Demonstration
DA.04	The system should collect data that can be used to localize people with an accuracy of 1 cm for a stationary person.	During sleep, high accuracy measurement is needed to be able to track breathing rhythm.	Test
DA.05	The system should collect data that can be used to localize people with an accuracy of 10 cm for a non-stationary person.	Detection of position can be used to track wandering and daily activities.	Test
DA.06	The system should collect data that can be used to measure movement velocity between >= 0 up-to free fall speed 9.6 m/s.	Measuring the velocity up to free fall speed is required to be able to measure falling.	Test
DA.07	The system should collect data that can be used to detect the height and width of a person.	Detecting the dimensions of a person help estimating social interaction.	Demonstration
DA.08	The system should keep track of real time.	Keeping track of real time allows the data recorded to be labeled with a time.	Inspection
DA.09	The Teddy should be able to store 1 week worth of data without transmission to the base station.	The storage should be designed, such that no data is lost when the connection with the base station is lost.	Demonstration
DA.10	The sound level should be measured in the range between 40dB (Living room, quiet classroom) 120dB (Human voice at its loudest, police siren).	The sound levels where chosen such that all sounds in a mainstream living room can be measured.	Test
DA.11	In case of detection of a dangerous situation, the caregiver should be alerted within 10 seconds.	Alerting a caregiver is main priority when a dangerous situation is detected and should thus happen as fast as possible.	Demonstration
DA.12	The power consumption of the teddy should be maximal 5 W.	The power used by the system determines the on-time and should thus be limited	Analysis
DA.13	The system should be able to measure the concentration of carbon monoxide gas and smoke.	Measuring the concentration of these specific gasses can be used to determine if the situation is dangerous.	Demonstration
DA.14	The system should be able to detect movement of the Teddy.	Detecting the movement of the Teddy is an indication of interaction between the senior and the Teddy.	Test
DA.15	The system should be able to locate the person in outdoor activity.	Locating the senior outside can be used for tracking of wandering, but also when the senior wanders of.	Test
DA.16	The system should collect data in a range of 8m from the base station.	8m Was chosen to be appropriate for mainstream living rooms.	Demonstration

2.4. Non-functional requirements

The first set of requirements in Table 2.2 are the non-functional general design requirements. Non-functional requirements are requirements that do not specify specific behaviours of the product, but are more general about the operation and appearance of the product. The rationale behind the requirement and the verification strategy is included.

Table 2.2: Table with non-functional general requirements. The non-functional requirements are labeled with G.XX.

ID	Requirement	Rationale	Verification Method
G.01	The dimensions of the data acquisition system should not exceed 10x10x5cm in the Teddy.	The dimensions of the system should be limited so the user does not feel the electronics inside the Teddy.	Inspection
G.02	No sensors should be placed outside the Teddy or Base station	Placing sensors outside the Teddy or Base station will disrupt the non-intrusive nature of the Smart Teddy	Inspection
G.03	The Smart Teddy project should not contain any wearable devices	A wearable will disrupt the non-intrusive nature of the Smart Teddy	Inspection

3
Detailed Design

In this chapter, the detailed design of the data acquisition system will be discussed. This chapter commences in Section 3.1, with the selection of the types of sensors needed. Then in Section 3.2, the selection of specific components and further considerations about components in the Teddy are discussed. The final section, Section 3.3, will discuss the selection of the specific components and further considerations about components in the Base station.

3.1. Selection of the types of sensors

From the state of the art analysis in Section 1.5, it was found that different types of sensors can be used to measure an indicator. In this section, the selection of what type of sensors are required for all indicators to be measured will be elaborated. Before all else, some considerations to take into account. The use of video to measure indicators is excluded from the design due to ethical and legal concerns such as privacy, informed consent, and autonomy. In the following, each indicator will be discussed with the possible sensors to be used.

Wandering

For tracking wandering, tracking the path of a senior is required. This can be done by determining the location of a senior and storing this location repeatedly in a chronological manner. This path can then be analyzed by a wandering detection algorithm as described in Section 1.5. In Table 3.1, a comparison between possible sensors for tracking wandering can be seen. Tracking wandering can be done using a mmWave sensor, a PIR sensor, RFID, and GPS.

Table 3.1: Comparison between possible sensors to track wandering

Feature	Presence	Speed	Distance	Direction	Through light walls	Indoor
mmWave sensor	Yes	Yes	Yes	Yes	Yes	Yes
PIR	Yes	No	No	No	No	Yes
RFID	Yes	Yes	Yes	No	Yes	Yes
GPS	No	Yes (wearable)	Yes (wearable)	No	Yes(wearable)	No (Not efficient)

Firstly, the mmWave sensor is chosen for indoor tracking, because of its ability to provide accurate presence, speed, distance, and direction estimations without disturbances from heat sources and reflective surfaces as the PIR sensor has. Also, a range of 10m in a field of view of 110°is possible. The RFID is not considered due to its intrusive nature, as the senior would always have to carry an RFID tag. Secondly, the GPS is chosen for outside location tracking. Selecting the mmWave sensor will lead to meet the following design requirements from Section 2.3: DA.01, DA.03, DA.04, DA.05, DA.06, DA.16

The GPS is chosen for outside tracking and will be mounted inside the Teddy. As stated in Section 2.1, it is assumed that the senior will always take the Teddy with him, so the path of the senior can be stored. Therefore, design requirement DA.15 can be met from Section 2.3. In addition, to be able to meet the design requirements DA.11 from Section 2.3, a GSM module will be used to communicate with the caregiver.

Social interaction

Determining the amount of social interaction a senior has, can be indicated by determining the amount and frequency of visitors over time. Determining the number of occupants in a room can be done using an analog, stationary PIR sensor or a mmWave sensor, the comparison between the two sensors can be seen in Table 3.2. The downside of the PIR sensor is the inability to detect a person if he is stationary. A solution to this would be to vibrate the sensor to still be able to detect stationary object [56]. Another option would be to use a mmWave sensor. As found in the State of Art analysis in Section 1.5, the mmWave sensor is suited best for occupancy detection, since the mmWave sensor has a higher detection accuracy and a higher detection reliability in different light conditions without disturbances caused by heat sources compared to the PIR sensor. Also, the mmWave sensor can be configured with a range between 8 and 10m, which satisfies design requirement DA.16. In addition, the mmWave sensor has a build-in algorithm to detect occupants. Therefore, the mmWave sensor will be used for determining the amount of social interaction a senior has. Therefore, design requirement DA.02 from Section 2.3 can be met.

Table 3.2: Comparison between possible sensors for social interaction tracking

Feature	Presence of multiple persons	Speed	Distance	Direction	Need for algorithm	Indoor
mmWave sensor	Yes	Yes	Yes	Yes	No	Yes
PIR	Yes(vibrate the sensor)	No	No	No	Yes	Yes

Emotion

Tracking of the emotional state of the senior can be done using a mmWave sensor, a microphone, or using the camera to record video. The mmWave sensor is the same sensor used for tracking the indicators that require motion detection. Only an additional algorithm has to be added to the design. Using a microphone to track the emotional state of a senior would require continuous listening and speech recognition. This was implemented in the previous prototype of the Smart Teddy, therefore, a microphone will be added to the system to give the ability to use the previous algorithm and to meet design requirement DA.10 from Section 2.3. As mentioned in the introductory section of this chapter, Section 3.1, the video recording is excluded due to privacy and legal concerns.

Daily activities

Precise determination of the daily activities of a senior can be implemented using a camera, however, as in previous sections, that would violate the privacy and legal rights of a senior. Besides, computing power-intensive detection algorithms would have to be implemented. Instead, not the precise activities of a senior will be tracked, but the amount of activity of the senior will be tracked. This in itself is not an indicative measure, however, a change in the average activity of a senior is. Tracking the average activity of a senior can be done using the same motion and path tracking as discussed for Wandering and Social Interaction. Therefore, the same conclusion can be drawn that the use of the mmWave sensor is favorable over the other options.

In addition, a gyroscope-accelerometer sensor can be used to track the amount of interaction the senior has with the Teddy. Again, this in itself is not an indicative measure, but the change in the average activity of use is. Therefore the gyroscope-accelerometer will be implemented in the Teddy such that the design requirement DA.14 can be met from Section 2.3.

Sleep- and eating rhythm

As found in the literature study in Section 1.4, sleep- and eating rhythm can be indicative of the progress of dementia in a senior. As described in Section 1.5, sleep- and eating rhythm monitoring can be done wirelessly by using a mmWave sensor, a camera, or RGB-D. In Table 3.3, a comparison between the sensors can be seen.

As seen in Table 3.3, The mmWave sensor can classify the state of a person. So, when a person lies down in bed, the algorithm will define the state as lying down. When this is the case, precise measurement in the difference of the position of the chest will commence being able to track breathing rhythm and heart rate. Algorithms are already implemented in the mmWave sensor to measure heart rate and state detection. Therefore, this sensor is selected because it lowers the complexity of the system and will lead to meet the design requirement DA.04 from Section 2.3.

Table 3.3: Comparison between possible sensors to track sleeping

Feature	Presence	Accuracy	State detection
mmWave sensor	Yes	Accurate	Yes
Camera	Yes	Accurate	Algorithm needed
RGB-D	Yes	Accurate	Algorithm needed

Current solutions of detecting when a person is eating make use of cameras pointed to the dining table of the user. In this thesis, this is not an acceptable solution considering the violation of the ethical values presented in Section 3.1. A solution would be to use the mmWave sensor to detect objects such as spoon and dish since the sensor can detect the dimension of an object with mm accuracy. However, to achieve that, a new algorithm will have to be developed.

Falling

A falling senior can be detected with different non-wearable devices. The alternatives considered are a PIR sensor in combination with a microphone, a microwave sensor, an array of microphones, and a mmWave sensor, a comparison between these sensors can be seen in table 3.4. As stated in Section 1.5, fall detection using a PIR sensor is possible in combination with a microphone close to the PIR sensor. A microwave sensor and a mmWave sensor both operate similarly. The main difference between the two is the operating frequency and therefore their maximal accuracy. Since the mmWave sensor's operating frequency is higher than that of the microwave sensor, the accuracy will be notably higher. A concern of using a higher frequency would be the reduction in range, but since the measurement will be in the house of the senior only this will not be an issue. The mmWave sensor is also able to determine the dimension of a person to meet design requirement DA.07 from section 2.3, and it has a pre-implemented algorithm to detect falls using the change in the average height of a person. The use of a camera is excluded due to privacy concerns.

Table 3.4: Compression between possible sensors for fall detection

Feature	Presence	Speed	Need for algorithm	Indoor	Person dimensions
mmWave sensor	Yes	Yes	No	Yes	Yes
PIR	Yes	No	Yes	yes	No
Microwave sensor	Yes	Yes	Yes	Yes	No
microphone	Yes	Not applicable	Yes	Yes	Not applicable
Camera	Yes	Yes	Yes	Yes	Yes

Other dangerous situations

Besides falling, other dangerous situations involving gas concentrations are considered. Firstly, the concentration of the potentially lethal gas carbon monoxide will be measured. This gas is chosen since it can be produced by a malfunctioning boiler present in most homes. Also, the gas has no smell and can thus not be detected by humans without the aid of technology. Secondly, smoke is measured in the room to be able to notify the caregiver in case of fire. Using gas and smoke sensor will lead to meet the design requirement DA.13 from Section 2.3.

Final selection

The selected sensors can be seen in Figure 3.1. To meet the design requirements AD.08, and AD.09 from Section 2.3, a real-time module and SD card were added to the system. In the following sections, the detailed design of the sensors will be discussed.

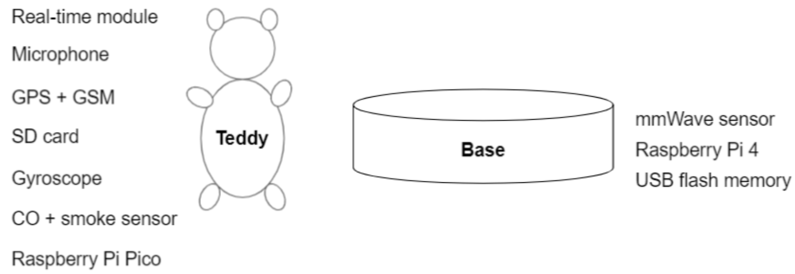


Figure 3.1: Data acquisition system components

3.2. Components in the Teddy

This section will discuss the different components that are housed inside the Teddy itself. First, in Subsection 3.2.1, the processor is discussed. Following, the sensors connected to the analog to digital converter will be discussed in Subsection 3.2.2. Then, the sensors connected via I2C will be discussed in Subsection 3.2.3. After that, the sensors connected via UART will be discussed in Subsection 3.2.4. Concluding, the storage and system flowchart will be discussed in Subsection 3.2.5 and 3.2.6 respectively.

3.2.1. The processor

The processor is the brain of the data acquisition system. Here, data processing and control of the mechanical part of the Teddy is done. What processor will be used is determined by the Human Interaction and Integration sub-group to be a Raspberry Pi Pico. However, it is important to highlight features needed for the data acquisition system here and compare it with other processors to give some background and overview of the design to verify that this processor is suitable. The Raspberry Pi Pico uses the RP2040 processor. In Table 3.5, a comparison between the RP2040 and three competitors is made.

Table 3.5: Features of RP2040 compared with three different processors

Feature	RP2040	ESP32	ESP32-s2	STM32F411
Chip size	QFN-56	QFN-48	QFN-56	QFN-48
CPU	2xCortex M0+, up to 125 MHz	2xLX6, 240 MHz	1xLX7, 240 MHz	Cortex-M4, 100 MHz
Co-processor	PIO	ULP	RISC-V ULP	none
RAM	264 KB	520 KB	320 KB	128 KB
Flash	External/ 2MB	External/ 4MB	External/ 4MB	512 KB
GPIO	30	26	30	32
ADC	12 bits	12 bits	12 bits	12 bits
UART	2	3	2	3
I2C	2	2	2	3
SPI	2	4	4	5
Current consumption	18 mA	53 mA	30 mA	26 mA

The size of the RP2040 chip is 7x7 mm. The central processing unit has a clock frequency up to 125 MHz and two M0+ cores which do not have a floating-point unit. The PIO is a co-processor that runs at full speed, however, it has only 9 instructions. This co-processor needs to be programmed in an assembler and is mainly used to output fast signals. The flash memory is where the program is stored, the RP2040 has 2 MB. The RAM is where the program is compiled and variables are temporarily stored, the RP2040 has 264 KB of RAM. The RP2040 has 30 General purpose in- output pins (GPIO), however, 3 of these pins are used for ADC and all communication peripherals use GPIO pins. The ADC is 12 bit. The RP2040 also supports communication peripherals such as UART, I2C, and SPI like the other processors. A remarkable feature of the RP2040 is the current consumption, which is 18 mA when running a simple blinking program.

3.2.2. Sensors connected to ADC

The sound sensor and the gas sensor in the data acquisition system have an analog output. Therefore, these sensors are connected to the ADC of the RP2040 processor. For the data acquisition system, it is enough to measure the sound intensity to meet the design requirements. However, the Human Interaction and Integration sub-group will implement a voice recognition, therefore, the MAX9814-microphone is selected. This sound sensor can measure sounds in the frequency range 20 Hz-20 kHz, has an automatic gain control, built-in amplifier, and has total harmonic distortion (THD) of 0.04%.

To measure carbon monoxide, the MQ-9 gas sensor is used. The sensor is also able to measure smoke and other gasses.

The ADC of the RP2040 processor of the Raspberry Pi Pico has 12 bits Successive Approximation Register Analogue to Digital converter (SAR). Capturing a sample takes 96 clock cycles and requires a 48 MHz clock cycle. Using that, the sample rate is calculated:

$$Sample\ rate = \frac{Clockfrequency}{No.Clockcycle/sample} = \frac{48MHz}{96} = 500kilosamples/s \tag{3.1}$$

The RP2040 has three analog inputs, for each of these inputs, the effective number of bits (ENOB) is 9 from the 12 bits. That is the number of bits of each measure that are on average not noise. The resolution of the ADC is calculated as followed:

$$Resolution = 2^{number\ of\ bits} = 2^{12} = 4096\ quantisation\ levels \tag{3.2}$$

The ADC has 4096 quantization levels. The voltage reference of the RP2040 is generated from the power supply, which is 3.3 Volt. The resolution of the ADC in terms of voltage is:

$$Voltage\ resolution = \frac{max.Voltage}{Quantisation\ levels} = \frac{3.3\ V}{4096} = 0.8\ mV \tag{3.3}$$

However, the useful resolution is limited by the signal-to-noise rate (SNR) and ENOB. The quantization error of this ADC is calculated as followed:

$$SQNR = 20\log(2^{Q}) = 72Q\ dB \tag{3.4}$$

Where Q is the number of quantization bits. The quantization error is 72 dB below the maximum level. The quantization error may occur from DC 0 Hz to the Nyquist frequency. Frequencies above the Nyquist frequencies will be mapped to lower frequency and therefore, incorrect detection. This is called aliasing, however, the selected sensors have integrated circuits IC's that perform an anti-aliasing filter.

Problems with the ADC:

- The reference voltage of the ADC is generated from a Switched Mode Power Supply (SMPS) at 3.3V by using an R-C filter. The output accuracy of the 3.3V SMPS is noisy. In addition, the ADC draws about 150 uA current which will lead to an offset of approximately 30 mV. It is suggested in the datasheet of the RP2040 that this offset may be reduced by tying an ADC channel to the ground and using this zero-measurement as an approximation to the offset. This solution reduces the number of available analog pins. This problem is solved by using a 74ch4051 multiplexer. The needed analog inputs for the Data acquisition system are 2 pins, therefore, a multiplexer with two channels would be sufficient. However, this 74ch4051 has eight channels, this was intentionally chosen to provide access to analog pins for the other two sub-groups and any other additional sensor in the future.
- The number of effective bits is 9 bits which means that the last lower bits are unreliable and noisy. To reduce the noise without impacting the signal, a simple averaging of samples will be applied to form a low pass Finite impulse response (FIR) digital filter. The simple moving average (SMA) will be used to smooth the data and reduce the noise. Where n samples are averaged, when calculating the next average, a new value comes into the sum and the oldest value drops out. The average of k data-points is given by the equation:

$$SMA_k = \frac{1}{k}\sum_{i=n-k+1}^{n} P_i \tag{3.5}$$

When new value comes in:

$$SMA_k, next = SMA_k, prev + \frac{1}{k}(P_{n+1} - P_{n-k+1}) \tag{3.6}$$

3.2.3. Sensors connected to I2C

The I2C is a synchronous serial communication bus and uses bidirectional lines, Serial Data Line (SDA), and Serial Clock Line (SCL), both lines are pulled up with internal resistors in the RP2040 chip. Also, the chip provides the possibility to connect two I2C buses, each has three operation modes:

- Standard mode with data rates from 0 to 100 kb/s.
- Fast mode with data rates less than or equal to 400 kb/s.
- Fast mode plus with data rates less than or equal to 1000 kb/s.

The data acquisition system has two sensors connected to an I2C bus. The first one is a DS3231 real-time module that keeps track of time and date which is important to know when an event occurs and to make the distinction between day and night. The module operates in either the 24-hour or 12-hour format with AM/PM indicators. The module has by default 0x68 address in hexadecimal, however, this address can be changed by soldering some pins with each other leading to 8 different possible addresses. To read the information from the buffer of the module, 7 bytes must be transmitted via the I2C. The datasheet stated that the I2C timing should be 400 Kb/s. For each byte to be transmitted, 9 clock cycles are needed. The data rate of the real-time module is:

$$Data\ Rate\ DS3231 = \frac{Data\ Rate\ I2C}{Number\ of\ Bytes * 9\ bits} = \frac{400kHz}{7 * 9} = 6.35KHz = 0.157ms \tag{3.7}$$

The second sensor connected to the I2C bus is the MPU6050 gyroscope and accelerometer. This sensor measures the movement of the Teddy in all directions and provides acceleration information of the movements. It is used to detect when the Teddy is moved. The movement of the Teddy is interpreted as an interaction between the Teddy and the senior. In the datasheet of the gyroscope+accelerometer, the I2C timing was given to be 400 Kb/s. Each byte will require a 9 clock cycle, 8 clock cycle for one word, and 1 clock cycle for acknowledging the signal. To read out all values from the sensor, 15 bytes over the I2C need to be transmitted. The data rate will be then:

$$Data\ Rate\ MPU6050 = \frac{Data\ Rate\ I2C}{Number\ of\ Bytes * 9\ bits} = \frac{400kHz}{15 * 9} = 2.96kHz = 0.3ms \tag{3.8}$$

Therefore, one reading of the MPU6050 will last 0.3 ms to transmit 15 bytes and one reading of DS3231 will last 0.157 ms to transmit 7 bytes.

For the data acquisition system, the second operation mode of the RP2040 will be used, since both sensors require 400 kb/s.

3.2.4. Sensors connected to UART

The data acquisition system in the Teddy has one sensor connected via UART which is the SIM808 board. The board contains a GPS engine, a GSM module to make calls or send & receive SMS, a Bluetooth module, the ability to access the internet via GSM, and a Li-Ion battery charger, all integrated into one chip. The choice was intentionally made such that other sub-group can also use the chip for communication or charging. Using the GSM, the Teddy can send an SMS to alert the caregiver about dangerous situations. The module is ultra-low power consumption where 1 mA is drawn in sleep mode and 24 mA in continuous tracking mode. The GPS receiver is sensitive with 22 tracking and 66 acquisition channels. The default baud rate of the module is 9600 Bd, since it is a digital output, the baud rate is the same as bits per second. The SIM808 is programmed via the "AT" commands which are a set of short text strings that can be combined to produce commands for operations.

3.2.5. Data logging & storage

According to design requirement DA.9, one week's worth of collected data must be able to be stored in the Teddy such that no data is lost when there is no transmission of data possible between the Teddy and the Base station. Therefore the needed storage space will be calculated. Since the three peripherals can not operate in parallel, the average data rate R_{avg} is calculated using Equation 3.9. Then, using the average data rate, the length of a week in seconds, and Equation 3.10 the total storage space required is calculated.

$$R_{avg} = \frac{(R_{ADC} + R_{I2C} + R_{UART})}{3} = \frac{500kb/s + 400kb/s + 9,6kb/s}{3} = 303,2kb/s \tag{3.9}$$

$$Storage\ space\ required = R_{avg} * t = 303.2kb/s * 604.800s = 183.375.360kb = 22.9GB \quad (3.10)$$

The first remark is that the RP2040 can not store the data required to meet design requirement DA.9, even when considering both flash and RAM. As calculated in Equation 3.10, the system will collect 22.9GB of data per week at its maximum.

The second remark is that when a failure occurs in the wireless transmission between the Teddy and the Base station, the data will be lost. In addition, wireless communication encounters some delays during transmission. The wireless communication is done by the Human Interaction & Integration sub-group. Lora is used as a wireless communication bus. Lora has a long-range, extremely low bandwidth of 7.8 to 500 kHz, with a link budget up to 154 dBm. The effective bit rate of the module is 0.018-37.5 kb/s which depends on the used spreading factor 6-12. LoRa-based network encounters two types of delay, the first one caused due to the transmission time of the data packet (time-on-air). The time-on-air depends on the spreading factor (SF), the channel bandwidth, and payload size. For example, 50 bytes and a channel bandwidth of 125 kHz will need 10 ms and 2.3s for SF 7 and SF 12, respectively. This time increase to 3.94s for a payload of 100 bytes for the same SF [62]. The second delay was caused due to the radio duty cycle regulations. A typical duty cycle in Europa is 1%, which means that the nodes need to wait for 99% of the total time [62]. Per 10 ms, the data acquisition system will collect 379 bytes, and the wireless communication can transmit 50 bytes. To solve these problems, multiple solutions can be used:

Reducing the sample rate

A solution to the problem of the shortage in storage space inside the Teddy would be to reduce the rate at which each sensor samples the data. Since human-related activities do not change rapidly for seniors with dementia, the sampling rate may be reduced. This would reduce R_{avg} and therefore the storage space required. This is however not acceptable for all sensors. According to the datasheet, the microphone can detect frequencies up to 20kHz. To be able to detect human voice, which has its peaks at 8kHz, according to the Nyquist theorem, a sample rate of 16kHz is required. This means the sampling rate of the microphone can be reduced to 16kHz. Also, when the Teddy moves, the accelerometer will output data with $R_{I2C} = 400kb/s$ data rate as mentioned in Section 3.2.3. This is a property of the module and can not be altered. The frequency at which the Teddy will be moved is determined by the user and should thus be expected to be all the time for abundance.

The idea of this solution is to sample at 5 Hz for gas and smoke detection. For acceleration and sound, sampling at the highest rate and storing only differential values. This solution may solve the problem but it has ambiguity since it is unknown for how long the Teddy will be moving or how long a sound will be generated in the room.

Using an external SD card memory

As mentioned in Subsection 3.2.5, the wireless communication may encounter difficulties in handling the amount of collected data and there is ambiguity with lowering the sample rate. An alternative solution could be to store the data on an external SD card and give wireless communication more freedom on transmission slots time. Using a 32 GB SD card will give the system the ability to collect data with no loss for more than one week. Therefore, no intervention for more than one week as mentioned in the design requirements DA.09 from section 2.3. This solution will be used since it meets this design requirement and reduces the ambiguity in the system.

The SD card is connected to the RP204 via the SPI protocol. The SPI protocol has a default baud rate of 9600 Bd similar to the UART. Each transmission requires 10 bits (1 bit for start, 1 bit for end, and 8 bits for a word or byte). The LoRa network is also connected to the SPI protocol, therefore, the SPI should use the slave selector pins where two bits are needed for this application.

3.2.6. System flowchart

In Figure 3.2, the flowchart of the system can be seen. The data acquisition system has the following events: measurements, storing, Acquire GPS, and sending location. Since the data acquisition system and Human Interaction & integration will operate on the same microprocessor, it is important to take their system into account when designing the flowchart. Human Interaction & Integration system has two events, namely, interaction (Breathing and moving tail), and data transmission.

The acknowledgment signal from the communication system could be used as an indication for when the location should be acquired. Losing the connection could mean that the teddy is outside. In the measurement event, one measurement will be performed (sound, gas, smoke, accelerometer, or time) and then stored during the store event such that the interaction event is checked fast and regularly. In the Transmit event, the data are transmitted from the Teddy to the Base station, the decision on whether the transmission is ready or not is done by the Human Interaction and Integration sub-group.

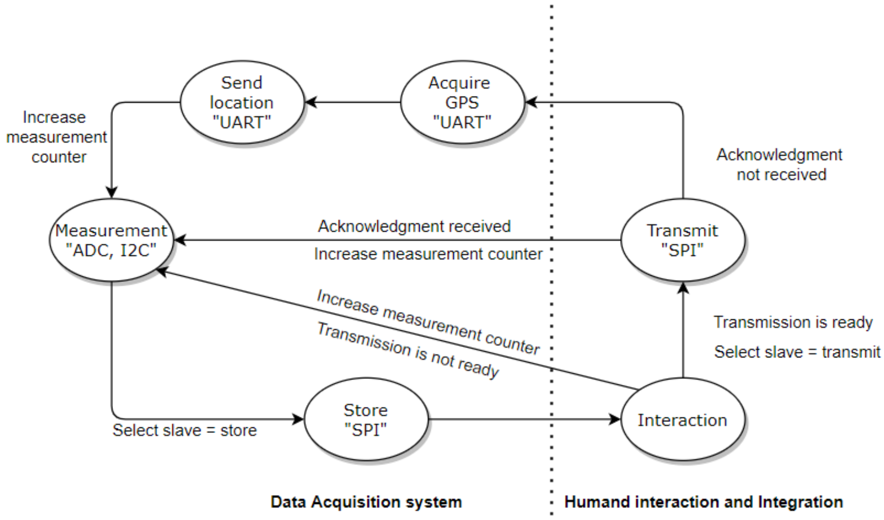


Figure 3.2: The Teddy flowchart system

3.3. Components in the Base station

This section will describe the components of the Base station of the Smart Teddy system. First, the sensor that acquires the data will be discussed in Subsection 3.3.1, following by the processor where the data is processed in Subsection 3.3.2. Lastly, the storage solution will be elaborated on in Subsection 3.3.3.

3.3.1. mmWave sensor

The mmWave sensor owes its name to the wavelength used, achieved by operating in a frequency range ranging from 60 up to 64 GHz. The main advantage of such a short wavelength, between 4.6 and 5.0mm, is the ability to do the distance, velocity, and angle measurement with millimeter accuracy. The basics of the principles used to implement measurements with the mmWave sensor are elaborated on in Appendix C.1.

The selected mmWave sensor is an IWR6843AOP antenna on the package (AOP) from Texas Instruments. This sensor is selected because it has advantages over the alternatives also provided by Texas Instruments. The comparison between possible sensors can be seen in table 3.6. Using the AOP simplifies the design and requires minimal RF expertise. The sensor has 4 receivers (RX), and 3 transmitters (TX) with 120 azimuth field of view (FoV) and 120 elevation FoV integrated into one chip. In addition to that, the sensor has a build-in DSP (C674c) for advanced signal processing, a hardware accelerator for FFT (Fast Fourier Transform), filtering, and CFAR (constant false alarm rate) processing.

Table 3.6: Features of IWR6843AOP compared with three different mmWave sensors

Feature	**IWR6843AOP**	**IWR6843**	**IWR1843**	**IWR1642**	**IWR1443**
Antenna on Package (AOP)	yes	-	-	-	-
Number of receiver	4	4	4	4	4
Number of transmitter	3	3	3	2	3
RF frequency range	60 to 64 GHz	60 to 64 GHz	76 to 81 GHz	76 to 81 GHz	76 to 81 GHz
On-chip memory	1.75 MB	1.74MB	2 MB	576 KB	
Max real sampling rate (Msps)	25	25	25	12.5	12.5
Micro-controller (R4F)	yes	yes	yes	yes	yes
Digital signal processing (C674x)	yes	yes	yes	yes	No
Access to point cloud via	USB	mmWaveBoost	mmWaveBoost	mmWaveBoost	mmWaveBoost

The sensor can process the data from front-end radar and provide positional data about targets (people), such as position, velocity, and angle. The processing chain starts with the analog output of front-end (FE) radar, these points are digitized utilizing an ADC. The ADC samples are used as an input for the detection process, in this phase, the range, azimuth angle, elevation angle, radial velocity, and SNR (Signal-to-noise ratio) values are detected. A collection of these measurement points is called point cloud data. The sensor uses the point cloud data to localize targets (people) in the localization phase which uses a 3D contact acceleration model. The algorithm of processing the data to track people consists of the following steps prediction, association, updating, and maintenance. A detailed explanation of the algorithm can be found in Appendix C.

The data acquired from the sensor has TLV (Type, Length, Value) protocol. Each TLV-packet has a fixed header (8 bytes) followed by a TLV-specific payload. The header has information about the type and the length. Each TLV can be one of three possible types:

- Point Cloud TLV where information about range, azimuth, Doppler, and SNR ratio are provided.
- Target List TLV where information about the number of detected targets is provided.
- Target structure TLV where 3D positional information about the target are provided (position, velocity, and acceleration)

Collecting the TLV data will be sufficient to meet design requirements DA.01, DA.02, DA.03, DA.04, DA.05, DA.06, and DA.07. The next paragraphs will describe the two possible options to collect the data.

Using pre-made solution

Texas Instruments, the manufacturer of the chosen mmWave sensor, provided a graphical user interface or GUI supported with Python and C codes with the sensor. However, this GUI is meant to operate on a Windows-based PC, while the operating system of the mini PC in the Base station is Linux-based. These two operating systems have different architectures since Windows has 86x architecture and Linux has ARM architecture. This solution will be used for verification of the capabilities of the sensor, however, storing the measured data is not possible.

Developing the code

Another way to collect the data is establishing a UART communication between the mmWave sensor and the mini PC. First, the sensor needs to be configured via the UART port, before the sensor starts measuring and sending the data. Then, a code should be written to receive the TLV data and store the data appropriately.

Since the first solution provides no way to log and store the data, a combination of the pre-made solution and the development of code is used. The basic algorithms and working principles of the delivered GUI will be implemented in a self-written code that supports the saving of the data. This will be discussed in Section 3.3.3.

3.3.2. The processor

The mmWave sensor has a powerful micro-controller (R4F) that has up to 6 ADC, 2 SPI ports, 2 UARTs, 1 CAN-FD interface, I2C, and GPIO. This micro-controller is used to process the data in the sensor and is programmed in C. However, the previous prototypes used a Raspberry Pi 3B+ processor and implemented algorithms in Python3 language. Comparing in Table 3.7, it is evident that newer and revised Raspberry Pi 4 is an improvement over the previous Pi 3B+ with a faster CPU, more RAM, and better connectivity. Therefore, the updated Raspberry Pi 4 will be used as the processor in the Base station. In doing so, the design is such, that the previously developed algorithms can still be implemented in the Base station.

Table 3.7: Features of Raspberry Pi 4 compared with the Raspberry Pi 3B+

Feature	Raspberry Pi 4	Raspberry Pi 3B+
CPU	Broadcom BCM2711 at 1.5 GHz	Broadcom BCM 2837Bo, , at 1.4GHz
	Quad-core Cortex-A72 (ARMv8)	Quad-core Cortex-A53 (ARMv8)
GPU	Broadcom VideoCore VI	Broadcom VideoCore VI
RAM	1, 2 or 4Gb	1GB
Flash	micro SD	micro SD
GPIO	40	40
USB	2x USB 3.0 2x USB 2.0	4x USB 2.0
Ethernet	Gigabit	Gigabit over USB 2.0
Bluetooth	5.0	4.2
Current consumption	3 A	2.5 A

3.3.3. Storage

The mmWave sensor is connected via UART to the processor with a minimal baud rate of 921600 Bd. This means that the sensor outputs data with a rate of 921600 bits/s since the signal is digital and thus one data unit is one bit. The amount of data storage required per day is calculated using Equation 3.12. When a decision is made for the required amount of time the system must be able to capture data, this value can be used to determine the required storage space. Also, in the following two paragraphs, two storage solutions are provided that can be implemented for storage.

$$R = \frac{921600Bd}{1} = 921.6\ kb/s \tag{3.11}$$

$$Storage\ space\ required/day = R * t = 921.6kb/s * 86400s = 79.626.240kb/day = 9,95GB/day \tag{3.12}$$

Cloud storage

The first storage solution considered is cloud storage. Cloud storage is a way of storing data on a server managed by a service company. A reoccurring fee is paid to make use of such storage and should be accessed via an internet connection. The amount of data that can be stored on such a server is practically endless, as much as the user is willing to pay. The downside of this solution is the burden it has on data security and thus the privacy of the senior. The cloud server has to be accessed via an internet connection, which creates the possibility of unauthorized access by third parties. In previous prototypes, the WiFi module of the Raspberry Pi was even disabled because of this risk.

Local storage

A solution that greatly reduced the risk of unauthorized access is local storage. Local storage can only be accessed via a physical connection to the device or removal of the storage device. For the Smart Teddy system, a USB Flash drive is the most suitable option considering the connectivity provided by the Raspberry Pi as seen in Table 3.7. A USB 3.0 connection will be used to connect a USB flash drive where the data acquired by the sensors in the Teddy and by the mmWave sensor in the Base station will be stored.

4

Implementation and Verification

In this chapter, the implementation of the design presented in Chapter 3 will be discussed. The sensors implemented in the Teddy and the Base station will be discussed in Section 4.1 and Section 4.2 respectively. The code used is found in Appendix D.

4.1. Teddy implementation

In the Teddy, all sensors are connected to the RP2040 processor. An overview of the system is given in Figure 4.1. This processor can be programmed in C/C++, Micro-python, Circuit-python, or Arduino IDE languages. The selection of the language is done by the Human Interaction and Integration subgroup. The chosen language is Micro-python. However, for the sensors used in the Data Acquisition system, no libraries were found to support the selected sensors. A library is a collection of code that can be used to control a sensor using functions. Libraries for other languages were however found and translated to work in Micro-python. In the following subsections, the implementation of each component will be discussed per communication protocol. First, the sensors connected to the analog to digital converter are discussed in Subsection 4.1.1. Following, the sensors connected to the I2C bus will be discussed in Subsection 4.1.2. Then, the GPS and GSM connected via UART will be discussed in 4.1.3. The storage, flowchart and filter implementation will be discussed in Subsection 4.1.4 and 4.1.5. Lastly, the power consumption will be discussed in Subsection 4.1.6.

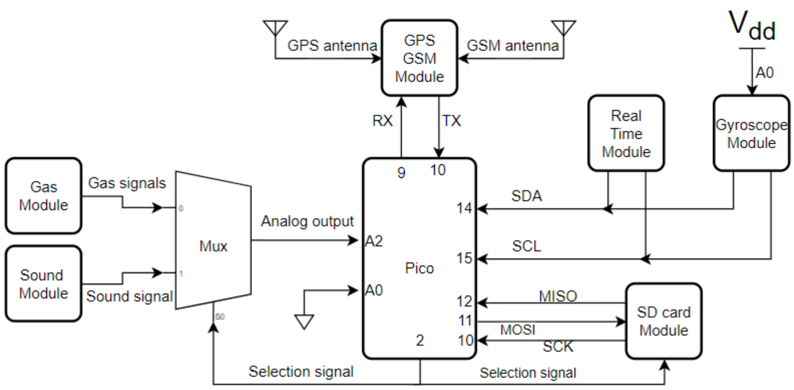


Figure 4.1: Overview of the data acquisition system inside the Teddy.

4.1.1. Implementation of sensors connected to ADC

As mentioned in the detailed design chapter, Chapter 3.2.2, two sensors are connected to the ADC, namely, sound, and gas sensors. During the implementation, it was observed that the amount of quantization levels of the ADC is 65536 instead of the expected 4096, even though the ADC has 12 bits and not 16. The 74ch4051 multiplexer uses a selection digital signal to select one of the two sensors. The implementation results can be seen in Figure 4.2. The green line is the sound measurement result when playing a song in the room. The sound module output is biased to 1.2 V. The smoke sensor output fluctuated between 0.038 - 0.039 ppm (Parts per Million) in a normal environment. When burning a paper next to the gas sensor, the sensor output increased to 0.062 ppm. The CO measurement was stable at around 0.01 ppm. Testing the change in CO measurements was not achieved due to safety reasons.

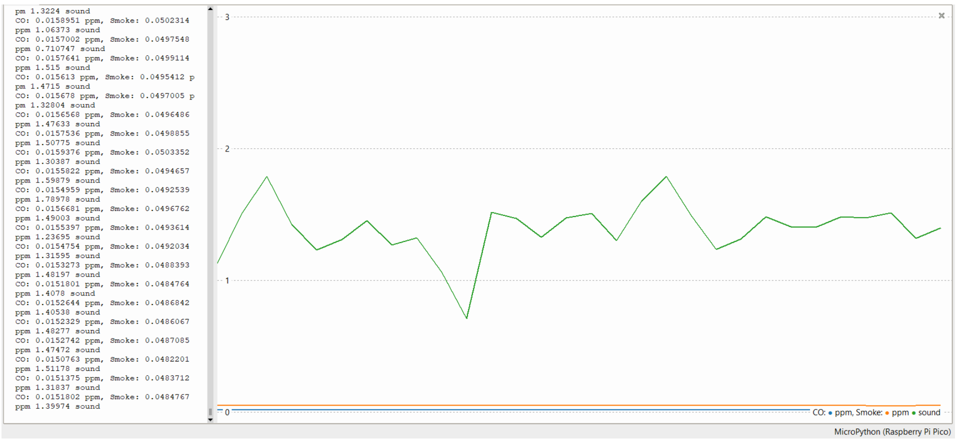


Figure 4.2: Test results of the sound- and gas sensor

Concluding, the sensors connected to the ADC work as expected and can provide the data required to meet design requirements DA.13 & DA.10. The multiplexer switches between the two ADC channels without difficulty. The CO sensor could not be verified with high concentrations of CO, but the measurement succeeded in normal conditions.

4.1.2. Implementation of sensors connected to I2C

The Data Acquisition system has two sensors connected to the I2C bus. The first one being the Gyroscope + Accelerometer, the second one being the Real-Time module. The two modules have the same address which is 0x68 in hexadecimal. Fortunately, the Gyroscope has an option to change the address to 0x69 by grounding one of the pins. As stated in 3.2.3, the used I2C frequency is 400 kHz. The test result can be seen in Figure 4.3 where the sensor is connected and randomly moved. The blue, red, and green lines represent x, y, and z directions respectively. It turns out that the sensor is also able to measure the temperature in the room. The result of temperature measurements can be also seen in Figure 4.3 the purple line.

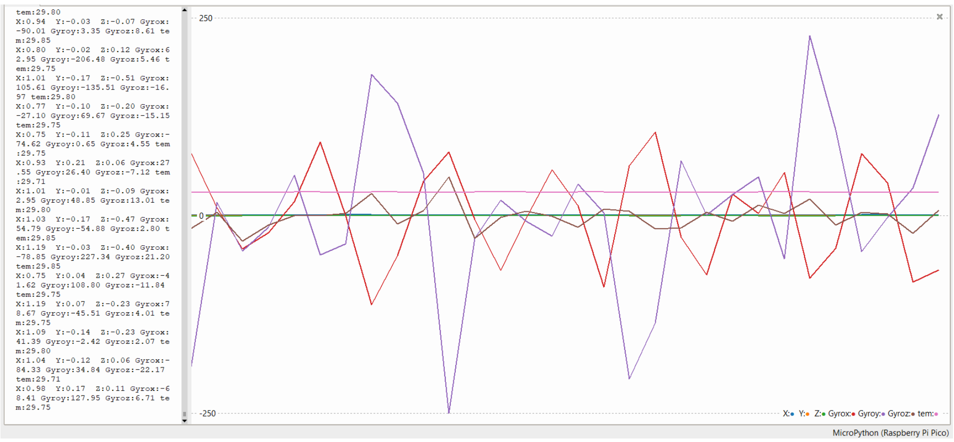


Figure 4.3: Results of testing the Gyroscope while moving the sensor in random directions. The blue, red, brown lines are the x,y, and z respectively. The purple line is the temperature in the room 29.2 degree

The real-time module is also connected to the I2C bus and works as expected. The module accepts an external battery such that it keeps track of the time when the main power supply is interrupted. In addition, the current time should be written to the start register 0x00 in the first time of using the module. In Figure 4.4, the result of implementing the module can be seen. The blue line represents the year. Orange, green, and red lines represent seconds, minutes, and hours respectively.

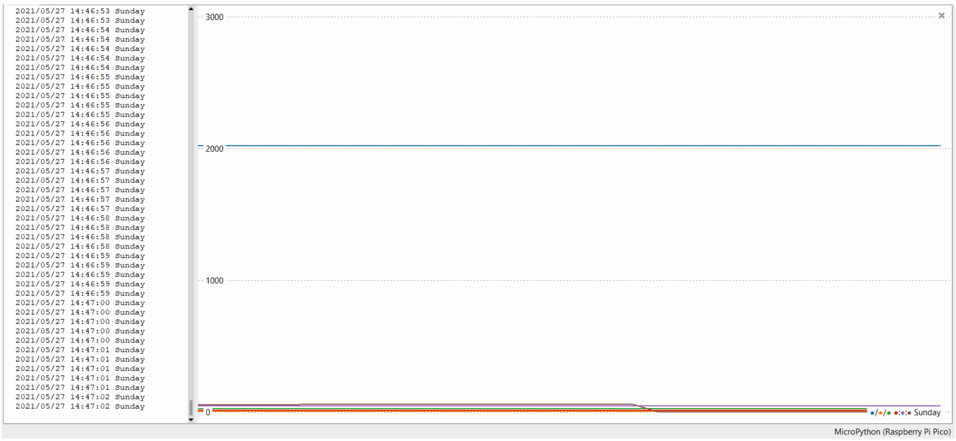


Figure 4.4: Results of testing real-time module while the Gyroscope is operating. The blue line is the year we are now 2021. Orange, green, and red colors are seconds, minutes, and hours respectively

To conclude, both the gyroscope + accelerometer and real-time module work properly with the real-time is tracked and Teddy's movements are detected. Therefore, design requirements DA.08 and DA.14 are met.

4.1.3. Implementation of sensors connected to UART

The SIM808 module of the Data Acquisition system is connected to the RP2040 via UART. The module is programmed via AT commands as mentioned in Section 3.2.4. For this module, no libraries were found to program the chip. Therefore, a code was written to acquire the location. In short, the code sends the required commands via the UART port and receives the results. To acquire the location, two steps should be done according to the datasheet. First, turning the engine on by sending the command "AT + CGPSPWR = 1", and second, acquire the information via the command "AT+CGNSINF". The last command will return longitude, latitude, altitude, UTC, time to first fix, satellites in view of fix, speed over ground, and course over ground. However, for Google map localization, only latitude and longitude are needed. These two are extracted from the received message by converting the received bytes into a string and then searching for the third, fourth, and fifth commas. Where longitude is between the third and fourth comma, and latitude is between the fourth and fifth comma. The result of implementing the GPS can be seen in Figure 4.5. Where the longitude and latitude are inserted in a readable link for the browser: "http://www,google.com/maps/place/longitude+ "/ " + latitude". The accuracy with respect to Google's estimation of the location is 26 m. In the datasheet of SIM808, it is given that the accuracy is between < 5 m. However, when comparing to the actual location in the building, the GPS shows a location more accurate than the Google estimation.

The GSM module of the SIM808 was tested by sending an SMS from the module to another phone number with the location as an content of the message. The test is done also by sending AT commands to the chip. The results of sending the location via SMS can be seen in Figure 4.5, where a screenshot is made from the Mobile-phone that received the message, the message was received in less than three seconds.

To conclude, both the GPS and the GSM module work properly with the ability to get the location of the Teddy and the ability to send the location via SMS. Therefore, design requirements DA.11 and DA.15 are met.

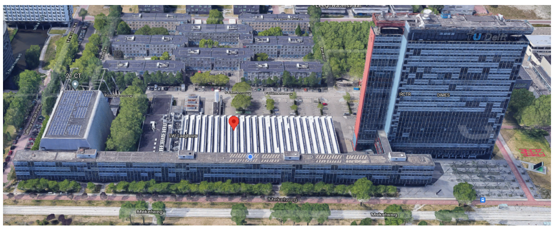


Figure 4.5: Results of testing the GPS. The red pointer is the location obtained from SIM808, and blue dot point is the google estimation of the location. The difference between the two is 26 m according to google maps.

4.1.4. Storage and data logging implementation

An SD card of 32 GB was used to store the collected data in the Teddy, where only differential data was stored. In Figure 4.6, the result of the implementation can be seen. The data was stored on the SD card, the SD card was plugged into a PC to plot the data in MATLAB. The collected data were the concentration of carbon monoxide, movement (position and acceleration), temperature, sound, and some percentage in the room. During the test, a song was played in the room in different time slots, the board was moved randomly, and a paper was burned in the room. The test was performed for 7 minutes. In the mentioned figure, it can be seen that the temperature, shown with the blue line, was constant at around 30° during the recording time, which was 7 minutes. The four segments in Figure 4.6 represent the random movement of the board. The sound is not clear to see because the song was played while moving the board.

The storage of the system works as expected and is able to store data from all sensors in the system. The used SD card gives the ability to store data for more than one week. Therefore design requirement DA.09 is met. During the integration with other subsystems, attention should be paid to the selection pin. Because the wireless communication LoRa also uses the same SPI protocol. In addition, there are no available pins left in the system to be used for two modules, therefore, the selection pin of the analog sensors will be used as a selector for the SPI protocol. When the pin is digitally low, the slave will be selected.

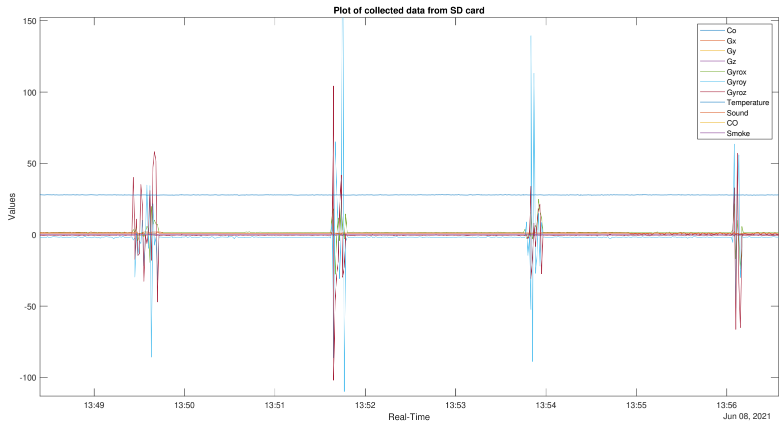


Figure 4.6: Results of testing the storage and data logging implementation where the collected data are plotted in MATLAB. The x-axis is the real-time axes where data were collected for 7 minutes. The y-axis is the value of each sensor. During the 7 minutes, music were played, the board was randomly moved, and a paper was burned in the room

4.1.5. Flowchart and filter implementation

The flowchart is implemented as explained in Figure 3.2. The interaction and transmission states are replaced by a delay. The working of the flowchart is verified by printing the state in the serial plotter, and as a result, the flowchart function properly.

The filter was not implemented because some data may be lost during filtering which may be important for the processing in the next prototype. This decision was also based on an interview with an engineer from the TU-Delft: Jeroen Bastemeijer. Where he stated that a data acquisition system should only collect data and filtering should be done in the processing stage.

4.1.6. Power consumption

In Table 4.1, the power consumption of components in the Teddy can be seen. The design requirement DA.12 from Section 2.3 states that the maximal power consumption should be 5 W. However, the designed system consumes approximately 2.3 W. Therefore, the design requirement DA.12 is met.

Table 4.1: Power consumption of the data acquisition system inside the Teddy

Component	average current(mA)	operating voltage(V)	average power (mW)
Raspberry Pi Pico	300	5.0	1500
gas MQ9	70	5.0	350
sd-card module	80	3.3	264
GPS GSM SIM808	24	4.0	96
Microphone	3	5.0	15
accelero + gyro MPU-600	3.9	3.3	12.9
real time module	0.3	5.0	1.5
total			**2239.4**

4.2. Base station implementation

The Base station consists of a mini-computer and the mmWave sensor. The mini-computer is a Raspberry Pi 4 which is programmed in Python3 language, the code can be found in Appendix D. The Raspberry Pi will be used for processing the data and applying algorithms in the next prototype. In the following sections, the implementation of the Base-station will be discussed. First the test of the mmWave sensor on windows operating system in section 4.2.1. Second, the communication between Raspberry Pi 4 and mmWave sensor in section 4.2.2. Finally, the implementation of storage and data logging in the Base-station in section 4.2.3.

4.2.1. Testing the mmWave sensor

Before starting with the implementation of the Base station, it was important to test the mmWave sensor to understand how the sensor works. The Texas Instruments provides a GUI (Graphical user interface) to visualize the collected data from the sensor with build-in algorithms to count people in the room and detect falls. The results of testing the sensor on a Windows operating system can be seen in Figure 4.7. The sensor can detect if a person is sitting, standing, or falling. On the up right side of the figure, it can be seen that the sensor detects one standing person. The sensor also provides information about instantaneous, average, and Delta heights. These heights are plotted in Figure 4.7 on the right-hand side. The position, velocity, and acceleration of the person are plotted in the GUI, these are the green points in the middle of Figure 4.7. The GUI also provides information about the number of frames, average plot time, and the number of points. The sensor functions properly and collects the needed data to meet design requirements DA.01, DA.02, DA.03, DA.04, DA.05, DA.06, DA.07. Also, the maximum detection range was 10m, which is satisfactory to meet design requirement DA.16. To implement the Base station, transmitting the data from the mmWave sensor to Raspberry Pi 4 is needed. The storage of this data is discussed in the following sections.

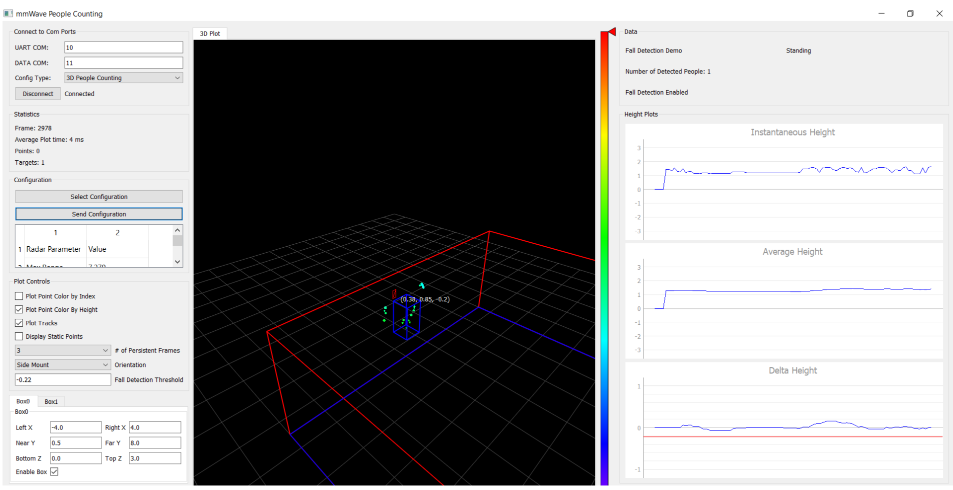


Figure 4.7: Results of testing the mmWave sensor using the provided GUI from Texas Instruments.

4.2.2. Communication between mmWave sensor and Raspberry Pi 4

The first challenge of establishing communication between the mmWave sensor and the Raspberry Pi was the operating systems used. The operating system the GUI from Texas Instruments was made for is Windows, where the Raspberry Pi 4 works on a Linux-based operating system called Raspberry Pi OS. The second challenge was using the QT-5 creator. This is a GUI software that can generate binary code that works on ARM architecture generated from 86x code.

The communication between the mmWave sensor and the Raspberry Pi 4 was achieved using the Universal Serial Bus or USB of the Raspberry Pi 4. Using the UART protocol, a configuration- and a data bus can be opened. The configuration bus is used to configure the board such that variables such as chirp time are set according to the desired measurements. The data bus is opened when measurements are being done. A code was written in Python3 that implements this protocol. The data was received correctly and was stored in a CSV file format. An example of the received data in hex TLV format can be seen in Appendix C.3

4.2.3. Storage implementation

Due to time- and budget constraints, a 4GB USB flash drive was used to store the collected data in the Base station. The TLV data was stored including the current time to every sample collected. The data will not be processed any further, as Jeroen Bastemeijer suggested, there is a risk that in processing some crucial data might get lost. The raw data can be used in the Texas Instrument software to generate the location, height, velocity, and angle of the detected object.

4.3. The prototype

During the implementation, the system was tested on a breadboard using jumper wires. For the prototype, a PCB seen in Figure 4.8 was designed, however, this PCB was not ordered due to time- and budget limitations. The PCB was electrically checked to have no connection errors and all units are designed according to the datasheets of the components. The schematic of the PCB can be found in Appendix E. However, to reduce the probability of connection errors during testing, a Breadboard PCB was used as seen in Figure 4.9. With this solution, design requirements G.01 and G.02 were met.

For integration with the other sub-groups, the Human Interaction & Integration designed and ordered a PCB where the modules can be mounted. This was however not finished at the time of writing this thesis.

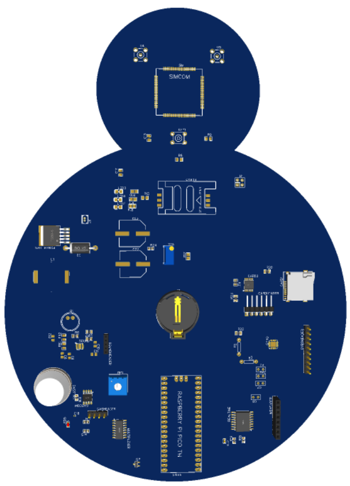


Figure 4.8: The PCB of the Data acquisition system in the Teddy

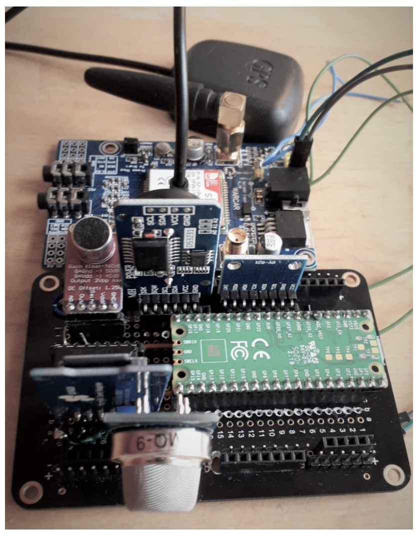


Figure 4.9: Implementation of the Data acquisition system in the Teddy

4.4. Program of design requirements check

After implementing the design from Chapter 3, the functional and non-functional design requirements that were met are listed in Table 4.2.

Table 4.2: Functional and non-functional design requirements that were met.

Requirement	DA.01	DA.02	DA.03	DA.04	DA.05	DA.06	DA.07	DA.08	DA.09	DA.10	DA.11	DA.12	DA.13	DA.14	DA.15	DA.16	G.01	G.02	G.03
Check	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

5

Conclusion and future work

This thesis aimed to design a data acquisition system that firstly, collects data about seniors' dementia-related activities such as wandering, social contact, sleep rhythm, and emotion, and secondly, detects dangerous situations such as falling, and the presence of a high concentration of Carbon monoxide gas.

In this thesis, we were able to design the data acquisition system by following the following procedure. Firstly, a literature study was conducted in the field of dementia to understand the indicators of dementia and to select what indicators should be measured. After that, the state of art was analyzed in Chapter 1 to know what has been done before regarding the measurement of indicators. Having done that, the design requirements were set in Chapter 2. In Chapter 3, the system was designed according to the requirements. Finally, the system was implemented and verified to meet all design requirements in Chapter 4.

Recommendations

- After integration with other sub-groups, the sound data did not achieve the desired sample rate due to the increase of complexity in the system. As a solution, the sound module could operate on the second core of the micro-processor in parallel to the system. This was not implemented due to time constraints. In addition, for efficient storage of the sound data, the data can be compressed in MP3 format.
- Acquiring the GPS location when the communication is lost between the Teddy and Base station raises an ethical question about the privacy and autonomy of the senior. When a senior with dementia leaves the house, this is not always wandering. Therefore, this also should be included in the ethical study of the next prototype.
- After integrating all sub-systems, a state machine was needed to automate the system reliably. In addition, an error detection algorithm should be included in the state machine such that the system can solve issues automatically and report them. Finally, a watchdog timer can be used to reset the system in case of a crashed program.

Future work

First, an expert in dementia and an expert in digital signal processing are needed for further development of the product. Where the dementia expert will be able to interpret the collected data concerning dementia progress, the expert in digital signal processing can design and implement algorithms from the collected data based on the result from the dementia expert.

Second, an ethical study is needed regarding human values and a deception concerns. Besides the technical and economical approaches, a human values approach regarding privacy, informed consent, autonomy, and psychological & physical well-being of seniors should be studied. This ensures a responsible innovation and avoids slow adoption, legal, and ethical issues. The deception concern originates from the fact that seniors could be led to believe that the Teddy is a real pet which is false.

References

[1] "A model for quality of life measures in patients with dementia: Lawron's next step". English. In: *Dementia and Geriatric Cognitive Disorders* 18.2 (2004), pp. 159–164. ISSN: 1420-8008. DOI: 10.1159/000079196.

[2] Chris Adams. URL: https://www.modernanalyst.com/Careers/InterviewQuestions/tabid/128/ID/1168/What-are-the-four-fundamental-methods-of-requirement-verification.aspx#:~:text=The%5C%20four%5C%20fundamental%5C%20methods%5C%20of%5C%20verification%5C%20are%5C%20Inspection%5C%2C%5C%20Demonstration%5C%2C%5C%20Test,or%5C%20system%5C%20with%5C%20increasing%5C%20rigor..

[3] *ADI - Dementia statistics*. URL: https://www.alzint.org/about/dementia-facts-figures/dementia-statistics.

[4] Mubarak A. Alanazi et al. "Machine Learning Models for Human Fall Detection using Millimeter Wave Sensor". In: *2021 55th Annual Conference on Information Sciences and Systems (CISS)*. 2021, pp. 1–5. DOI: 10.1109/CISS50987.2021.9400259.

[5] Jack Andrews et al. "A Motion Induced Passive Infrared (PIR) Sensor for Stationary Human Occupancy Detection". In: *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. IEEE, Apr. 2020. DOI: 10.1109/plans46316.2020.9109909. URL: https://doi.org/10.1109/plans46316.2020.9109909.

[6] Alireza Borhani and Matthias Pätzold. "A Non-Stationary Channel Model for the Development of Non-Wearable Radio Fall Detection Systems". In: *IEEE Transactions on Wireless Communications* 17.11 (2018), pp. 7718–7730. DOI: 10.1109/TWC.2018.2869782.

[7] M. Brod et al. "Conceptualization and Measurement of Quality of Life in Dementia: The Dementia Quality of Life Instrument (DQoL)". In: *The Gerontologist* 39.1 (1999), pp. 25–36. ISSN: 0016-9013. DOI: 10.1093/geront/39.1.25. URL: https://doi.org/10.1093/geront/39.1.25.

[8] Atul Chaudhary et al. "Sensor Signals-Based Early Dementia Detection System Using Travel Pattern Classification". In: *IEEE Sensors Journal* 20.23 (Dec. 2020), pp. 14474–14481. DOI: 10.1109/jsen.2020.3008063. URL: https://doi.org/10.1109/jsen.2020.3008063.

[9] Zhangjie Chen and Ya Wang. "Infrared–ultrasonic sensor fusion for support vector machine–based fall detection". In: *Journal of Intelligent Material Systems and Structures* 29.9 (2018), pp. 2027–2039. ISSN: 1045-389X. DOI: 10.1177/1045389x18758183. URL: https://doi.org/10.1177/1045389x18758183.

[10] Keum San Chun, Sarnab Bhattacharya, and Edison Thomaz. "Detecting Eating Episodes by Tracking Jawbone Movements with a Non-Contact Wearable Sensor". In: 2.1 (Mar. 2018). DOI: 10.1145/3191736. URL: https://doi.org/10.1145/3191736.

[11] Gabriele Cipriani et al. "Wandering and dementia". In: *Psychogeriatrics* 14.2 (2014), pp. 135–142. ISSN: 1346-3500. DOI: 10.1111/psyg.12044. URL: https://dx.doi.org/10.1111/psyg.12044.

[12] James K. Cooper and Dan Mungas. "Risk Factor and Behavioral Differences Between Vascular and Alzheimer's Dementias: The Pathway to End-Stage Disease". In: *Journal of Geriatric Psychiatry and Neurology* 6.1 (Jan. 1993), pp. 29–33. DOI: 10.1177/002383099300600105. URL: https://doi.org/10.1177/002383099300600105.

[13] P. T. M. van Dijk et al. "Falls in Dementia Patients". In: *The Gerontologist* 33.2 (Apr. 1993), pp. 200–204. DOI: 10.1093/geront/33.2.200. URL: https://doi.org/10.1093/geront/33.2.200.

[14] Carol Van Doorn et al. "Dementia as a Risk Factor for Falls and Fall Injuries Among Nursing Home Residents". In: *Journal of the American Geriatrics Society* 51.9 (Sept. 2003), pp. 1213–1218. DOI: 10.1046/j.1532-5415.2003.51404.x. URL: https://doi.org/10.1046/j.1532-5415.2003.51404.x.

[15] Phil Gehrman et al. "The relationship between dementia severity and rest/activity circadian rhythms". eng. In: *Neuropsychiatric disease and treatment* 1.2 (June 2005). PMC2413196[pmcid], pp. 155–163. ISSN: 1176-6328. DOI: 10.2147/nedt.1.2.155.61043. URL: https://doi.org/10.2147/nedt.1.2.155.61043.

[16] Clarissa M Giebel, Caroline Sutcliffe, and David Challis. "Activities of daily living and quality of life across different stages of dementia: a UK study". In: *Aging & Mental Health* 19.1 (2015), pp. 63–71.

[17] Munkhjargal Gochoo et al. "Device-Free Non-Privacy Invasive Classification of Elderly Travel Patterns in A Smart House Using PIR Sensors and DCNN". In: *IEEE Sensors Journal* (2017), pp. 1–1. DOI: 10.1109/jsen.2017.2771287. URL: https://doi.org/10.1109/jsen.2017.2771287.

[18] Christian Gross et al. "Towards an Occupancy Count Functionality for Smart Buildings - An Industrial Perspective". In: *2020 2nd IEEE International Conference on Industrial Electronics for Sustainable Energy Systems (IESES)*. IEEE, Sept. 2020. DOI: 10.1109/ieses45645.2020.9210641. URL: https://doi.org/10.1109/ieses45645.2020.9210641.

[19] Julian Hirt et al. "Social Robot Interventions for People with Dementia: A Systematic Review on Effects and Quality of Reporting". In: *Journal of Alzheimer's Disease* 79 (2021). 2, pp. 773–792. ISSN: 1875-8908. DOI: 10.3233/JAD-200347. URL: https://doi.org/10.3233/JAD-200347.

[20] Jan Homolak et al. "Circadian Rhythm and Alzheimer's Disease". eng. In: *Medical sciences (Basel, Switzerland)* 6.3 (June 2018). medsci6030052[PII], p. 52. ISSN: 2076-3271. DOI: 10.3390/medsci6030052. URL: https://doi.org/10.3390/medsci6030052.

[21] Xu Huang et al. "Indoor Detection and Tracking of People Using mmWave Sensor". In: *Journal of Sensors* 2021 (Feb. 2021). Ed. by Bin Gao, pp. 1–14. DOI: 10.1155/2021/6657709. URL: https://doi.org/10.1155/2021/6657709.

[22] Thien Huynh-The et al. "Hierarchical topic modeling with pose-transition feature for action recognition using 3D skeleton data". In: *Information Sciences* 444 (May 2018), pp. 20–35. DOI: 10.1016/j.ins.2018.02.042. URL: https://doi.org/10.1016/j.ins.2018.02.042.

[23] Marcello Ienca et al. "Social and Assistive Robotics in Dementia Care: Ethical Recommendations for Research and Practice". In: *International Journal of Social Robotics* 8.4 (Aug. 2016), pp. 565–573. ISSN: 1875-4805. DOI: 10.1007/s12369-016-0366-7. URL: https://doi.org/10.1007/s12369-016-0366-7.

[24] Ying-Ling Jao et al. "Association between social interaction and affect in nursing home residents with dementia". In: *Aging & Mental Health* 22.6 (Mar. 2017), pp. 778–783. DOI: 10.1080/13607863.2017.1304526. URL: https://doi.org/10.1080/13607863.2017.1304526.

[25] Hyunbum Kim et al. "A Virtual Emotion Detection Architecture with Two-way Enabled Delay Bound toward Evolutional Emotion-based IoT Services". In: *IEEE Transactions on Mobile Computing* (2020), pp. 1–1. ISSN: 1536-1233. DOI: 10.1109/tmc.2020.3024059. URL: https://doi.org/10.1109/tmc.2020.3024059.

[26] Michael D. Kopelman. "Rates of forgetting in Alzheimer-type dementia and Korsakoff's syndrome". In: *Neuropsychologia* 23.5 (Jan. 1985), pp. 623–638. DOI: 10.1016/0028-3932(85)90064-8. URL: https://doi.org/10.1016/0028-3932(85)90064-8.

[27] M Powell Lawton and Elaine M Brody. "Assessment of older people: self-maintaining and instrumental activities of daily living". In: *The gerontologist* 9.3_Part_1 (1969), pp. 179–186.

[28] M. Powell Lawton. "Quality of Life in Alzheimer Disease". In: *Alzheimer Disease & amp; Associated Disorders* 8 (1994). DOI: 10.1097/00002093-199424004-00015.

[29] Kyung Hee Lee, Donna L. Algase, and Eleanor S. Mcconnell. "Relationship between observable emotional expression and wandering behavior of people with dementia". In: *International Journal of Geriatric Psychiatry* 29.1 (2014), pp. 85–92. ISSN: 0885-6230. DOI: 10.1002/gps.3977. URL: https://doi.org/10.1002/gps.3977.

[30] Kyung Hee Lee et al. "Pain and Psychological Well-Being Among People with Dementia in Long-Term Care". In: *Pain Medicine* 16.6 (June 2015), pp. 1083–1089. DOI: 10.1111/pme.12739. URL: https://doi.org/10.1111/pme.12739.

[31] Yun Li et al. "Acoustic fall detection using a circular microphone array". In: *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*. 2010, pp. 2242–2245. DOI: 10.1109/IEMBS.2010.5627368.

[32] Liang Liu et al. "Automatic fall detection based on Doppler radar motion signature". In: *2011 5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth) and Workshops*. 2011, pp. 222–225. DOI: 10.4108/icst.pervasivehealth.2011.245993.

[33] John Malik, Yu-Lun Lo, and Hau-tieng Wu. "Sleep-wake classification via quantifying heart rate variability by convolutional neural network". In: *Physiological Measurement* 39.8 (Aug. 2018), p. 085004. DOI: 10.1088/1361-6579/aad5a9. URL: https://doi.org/10.1088/1361-6579/aad5a9.

[34] Lars Meinel et al. "OPDEMIVA: An integrated assistance and information system for elderly with dementia". In: *2015 IEEE International Conference on Consumer Electronics (ICCE)*. 2015, pp. 76–77. DOI: 10.1109/ICCE.2015.7066325.

[35] Armando Nava, Leonardo Garrido, and Ramon F. Brena. "Recognizing Activities Using a Kinect Skeleton Tracking and Hidden Markov Models". In: *2014 13th Mexican International Conference on Artificial Intelligence*. IEEE, Nov. 2014. DOI: 10.1109/micai.2014.18. URL: https://doi.org/10.1109/micai.2014.18.

[36] Jingping Nie et al. "SPIDERS: Low-Cost Wireless Glasses for Continuous In-Situ Bio-Signal Acquisition and Emotion Recognition". In: *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 2020, pp. 27–39. DOI: 10.1109/IoTDI49375.2020.00011.

[37] J. Oostrom, R.J. Schlingmann, and H. Alers. "Using a Robot Companion to Detect and Visualize the Indicators of Dementia Progression and Quality of Life of People Aged 65 and Older". In: (2019), pp. 1–6.

[38] Martin Prince et al. *World Alzheimer Report 2015. The Global Impact of Dementia. An Analysis of Prevalence, Incidence, Cost and Trends*. Aug. 2015.

[39] Munaf Rashid, S. A. R. Abu-Bakar, and Musa Mokji. "Human emotion recognition from videos using spatio-temporal and audio features". In: *The Visual Computer* 29.12 (Dec. 2012), pp. 1269–1275. DOI: 10.1007/s00371-012-0768-y. URL: https://doi.org/10.1007/s00371-012-0768-y.

[40] Yvonne Schikhof, Ingrid Mulder, and Sunil Choenni. "Who will watch (over) me? Humane monitoring in dementia care". In: *International Journal of Human-Computer Studies* 68.6 (June 2010), pp. 410–422. ISSN: 1071-5819. URL: https://www.sciencedirect.com/science/article/pii/S1071581910000169.

[41] Philip Sedgwick and Nan Greenwood. "Understanding the Hawthorne effect". In: *BMJ* (Sept. 2015), h4672. DOI: 10.1136/bmj.h4672. URL: https://doi.org/10.1136/bmj.h4672.

[42] Arindam Sengupta et al. "mm-Pose: Real-Time Human Skeletal Posture Estimation Using mmWave Radars and CNNs". In: *IEEE Sensors Journal* 20.17 (Sept. 2020), pp. 10032–10044. DOI: 10.1109/jsen.2020.2991741. URL: https://doi.org/10.1109/jsen.2020.2991741.

[43] Kazuaki Shiba, Takashi Kaburagi, and Yosuke Kurihara. "Fall Detection Utilizing Frequency Distribution Trajectory by Microwave Doppler Sensor". In: *IEEE Sensors Journal* 17.22 (2017), pp. 7561–7568. DOI: 10.1109/JSEN.2017.2760911.

[44] Lin Shu et al. "Wearable Emotion Recognition Using Heart Rate Data from a Smart Bracelet". In: *Sensors* 20.3 (Jan. 2020), p. 718. DOI: 10.3390/s20030718. URL: https://doi.org/10.3390/s20030718.

[45] S Smith et al. "Measurement of health-related quality of life for people with dementia: development of a new instrument (DEMQOL) and an evaluation of current methodology". In: *Health Technology Assessment* 9.10 (2005). ISSN: 1366-5278. DOI: 10.3310/hta9100.

[46] Sociaal en Cultureel Planbureau. *Rijksoverheid.nl*. 2018. URL: https://digitaal.scp.nl/ouderenzorg/tekort-aan-mantelzorgers-en-professionals (visited on 04/27/2021).

[47] TI Developers. *TI DevTools*. 2021. URL: https://dev.ti.com/tirex/explore/node?node=AJoMGA2ID9pCPWEKPi16wg__VLyFKFf__LATEST (visited on 05/25/2021).

[48] *Understanding Social Interaction*. Feb. 2021. URL: https://socialsci.libretexts.org/@go/page/8023.

[49] United Nation. *Sustainable Development Goals: Goal 3: Ensure healthy lives and promote well-being for all at all ages*. 2021. URL: https://www.un.org/sustainabledevelopment/health/ (visited on 05/14/2021).

[50] Ladislav Volicer and Lisa Bloom-Charette. *Enhancing the quality of life in advanced dementia*. Psychology Press, 1999.

[51] N.K. Vuong, S. Chan, and C.T. Lau. "Automated detection of wandering patterns in people with dementia". In: *Gerontechnology* 12.3 (June 2014). DOI: 10.4017/gt.2014.12.3.001.00. URL: https://doi.org/10.4017/gt.2014.12.3.001.00.

[52] Fengyu Wang et al. "Radio Frequency Based Heart Rate Variability Monitoring". In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, June 2021. DOI: 10.1109/icassp39728.2021.9413465. URL: https://doi.org/10.1109/icassp39728.2021.9413465.

[53] Quanqiu Wang et al. "COVID 19 and dementia: Analyses of risk, disparity, and outcomes from electronic health records in the US". In: *Alzheimer's & Dementia* (2021). ISSN: 1552-5260. DOI: 10.1002/alz.12296. URL: https://doi.org/10.1002/alz.12296.

[54] WHO. *Dementia*. URL: https://www.who.int/news-room/fact-sheets/detail/dementia (visited on 05/30/2021).

[55] WHO. *SUPPORTING INFORMAL CAREGIVERS OF PEOPLE LIVING WITH DEMENTIA*. URL: https://www.who.int/mental_health/neurology/dementia/dementia_thematicbrief_informal_care.pdf (visited on 05/12/2021).

[56] Libo Wu and Ya Wang. "A Low-Power Electric-Mechanical Driving Approach for True Occupancy Detection Using a Shuttered Passive Infrared Sensor". In: *IEEE Sensors Journal* 19.1 (Jan. 2019), pp. 47–57. DOI: 10.1109/jsen.2018.2875659. URL: https://doi.org/10.1109/jsen.2018.2875659.

[57] Yu-Tzu Wu, Linda Clare, and Fiona E. Matthews. "Relationship between depressive symptoms and capability to live well in people with mild to moderate dementia and their carers: results from the Improving the experience of Dementia and Enhancing Active Life (IDEAL) programme". In: *Aging & Mental Health* 25.1 (Sept. 2019), pp. 38–45. DOI: 10.1080/13607863.2019.1671316. URL: https://doi.org/10.1080/13607863.2019.1671316.

[58] Zhicheng Yang et al. "Vital Sign and Sleep Monitoring Using Millimeter Wave". In: *ACM Transactions on Sensor Networks* 13.2 (June 2017), pp. 1–32. DOI: 10.1145/3051124. URL: https://doi.org/10.1145/3051124.

[59] Yunze Zeng et al. "Poster Abstract: Human Tracking and Activity Monitoring Using 60 GHz mmWave". In: *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 2016, pp. 1–2. DOI: 10.1109/IPSN.2016.7460704.

[60] Peijun Zhao et al. "Heart Rate Sensing with a Robot Mounted mmWave Radar". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2020. DOI: 10.1109/icra40945.2020.9197437. URL: https://doi.org/10.1109/icra40945.2020.9197437.

[61] Liyang Zhu et al. "A Survey of Fall Detection Algorithm for Elderly Health Monitoring". In: *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*. 2015, pp. 270–274. DOI: 10.1109/BDCloud.2015.35.

[62] Dimitrios Zorbas et al. "TS-LoRa: Time-slotted LoRaWAN for the Industrial Internet of Things". In: *Computer Communications* 153 (2020), pp. 1–10. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2020.01.056. URL: https://www.sciencedirect.com/science/article/pii/S0140366419314677.

Appendices

A

Previous Efforts

According to the work of Oostrom, Schlingmann & Alers [37] indicators can be found that can be used to calculate a Quality of Life score which in itself is an indication of the progress of dementia. The indicators they suggest can be seen in Table 1.1. Their focus was on trying to determine a QoL score using part of the DEMQOL standard [45]. According to their paper it is recommended to expand their work by being able to detect emotion of a patient using a form of speech recognition which recognizes words and classifies them to a corresponding emotion. Another recommendation is the tracking of physical health by tracking activities during the day, but also keeping track of sleep- and eating rhythm.

Following work was done by Caleb Quame. The goal of his work was to implement the suggestions done by Oostrom, Schlingmann & Alers. His work does not contain any literature research. His report does however describe practical experiences and issues he came across while building the system. These will be taken into account during implementation.

According to Kumar even more indicators that indicate progress of dementia exist. It should be noted that her work was based on interviews with Dutch healthcare workers. Which is why, in Section 1.4, the indicators given will be verified with literature. It should also be noted that her work was done with the focus on practical implementation of the Smart Teddy product, where she aimed to serve caretakers with data gathered by the Smart Teddy. The indicators she recommended are given in Table A.1.

Table A.1: Indicators as suggested by previous work in the Smart Teddy project.

Indicator	**Sensor type**
Wandering	Infrared sensor & Sound
Social contact	Infrared sensor & Sound
Life rhythm	GPS & Gyroscope
Day- & night rhythm	Infrared sensor & Sound
Senior motion & location	GPS & Gyroscope
Eating rhythm & Body weight	Camera
Emotion	Camera
Forgetting critical actions	Gas sensor
Falling	Sound & Camera

B

Literature support of indicators

Quality of Life

In [1], a model for QoL measurement is provided by discussing relevant literature on QoL research. The core dimension of QoL of patients with dementia is identified to be the psychological well-being. The paper presents a hierarchical model of QoL which is listed below.

- personal aspects not related to dementia such as religion,work, material possessions, coping style, living accommodation, and income [7].
- personal aspects related to dementia: cognitive functioning, physical health[15], eating, infections, and chronic physical disease, psychiatric symptoms [50].
- environmental factors: Daily activities, recreational activities, social behavior [50], social support, network [28],

Wandering

According to Dr. G. Cipriana [11], wandering is a physical and emotional problem of dementia for both the patient and the caregiver. Since the cause of wandering is yet to be discovered, it can only be said that a strong correlation between dementia and wandering exists. It should also be noted that not all forms of dementia cohere with wandering. For example, in people who suffer from vascular dementia only 18% of the patients in the sample of 502 people would be considered a wanderer [12]. Therefore, when wandering occurs this is quite a good indication that someone suffers from a form of dementia, while the reverse is not true.

Social Interaction

”A social interaction is a social exchange between two or more individuals. These interactions form the basis for social structure and therefore are a key object of basic social inquiry and analysis.” [48]. According to [24], social interaction can work two ways in the progress of dementia. More social interaction results in less progression of the disease while less social interaction indicates that the disease is actually progressing.

Emotion

According to [57], depression is a common condition in dementia. In [30] it is shown that negative emotion is related to the decline of cognitive functions and therefore the progress of dementia. Therefore, it can be concluded that a more negative emotions over a period of time can indicate dementia progress.

Daily Activities

Losing the ability to perform daily life activities is one of the most prevalent characteristics of dementia. An example of instrumental activities of daily life (IADLs) are: finances, household tasks, laundry, meal preparation, medication management, shopping, telephoning and transport [27], and basic activities of daily living (ADLs), including bathing, continence, dressing, feeding and toileting [16].

In [16], a study was conducted in the UK on 122 PWD (65 years or older) with their carers in different dementia stages, either living at home or recently admitted to long-term care. The study monitored the daily ADLs of the PWD and the results show that there is deterioration in early-stages dementia in ADL, including dressing, bathing, and continence.

Sleep- and eating Rhythm

Rhythm is a key factor in the health of a person. Key components of rhythm are the time at which a person goes to sleep and the time at which a user eats. It was found that not the day to day rhythm indicates progress of severity of dementia, but more the robustness of the rhythm is the indicator. Those with more robust rhythm had less severe dementia. [15] [20]

Falling

As suggested in [14] & [13], the risk of falling for seniors with dementia is far greater than that of healthy seniors. Both papers also suggest that in the beginning and moderate stages of the disease the risk of falling increases with the progress of dementia. When dementia is far progressed the falling rates drop, since seniors lose their mobility overall.

Other dangerous situations

Dangerous situations considered are the presence of flammable methane gas and toxic carbon monoxide gas. An increase in frequency of forgetting such critical action is considered to be part of the decrease in short term memory. As shown in [26], deterioration of the short term memory is however not only related only to dementia. This indicator can thus be used to track progress of dementia, but not prove it to begin with.

C

mmWave Sensing

C.1. Basics of mmWave sensing

Distance measurement in mmWave sensing technology is done by using frequency modulated continuous wave (FMCW). Distance is measured by sending out a chirp and determining the time it takes for the reflection to arrive. This is done by mixing the received signal with the transmitted signal. The mixed signal is called the beat signal. The frequency and phase of this signal are given by Equation C.1 and Equation C.2 subsequently where R is the distance to the object measured, B the bandwidth of the chirp, T the length of the chirp and c the speed of light in vacuum. From Equation C.1 the range can thus be calculated. Then Equation C.2 can be used for a more precise estimation of the range.

$$f_b = \frac{2BR}{cT} \tag{C.1}$$

$$\phi_b = \frac{4\pi f_{min} R}{c} \tag{C.2}$$

Velocity v is determined by sending out a second chirp after a specified time t and determining the difference in range after this time.

$$v = \frac{\Delta R}{t} \tag{C.3}$$

One might wonder what happens when the velocity is in the direction perpendicular to the measurement direction. This case is the reason angle estimation was introduced. Estimation of the angle is done by using 2 transceivers and determining the difference in distance measured from both transceivers. It should however be noted that this measurement can be ambiguous, since the difference between for example 130° and 310° can not be determined. This means a shield should be made on the sensor to ensure only measurement in the direction of measurement.

When distance and angle can be measured a continuous localization of points is possible. When a large number of points is measured, detection of certain objects, such as humans is possible. Since the measurement can be as accurate as micrometers, this measurement can also be used to determine for example heart rate and other health related properties. Also, the size of a point cloud can be used to recognize a person based on previous measurements.

C.2. Algorithm

The processing chain starts with the analog output of front-end (FE) radar, these points are digitized by means of an ADC. The ADC samples are used as an input for the detection process, in this phase, the range, azimuth angle, elevation angle, radial velocity, and Signal-to-noise ratio or SNR values are detected. A collection of these measurement points is called point cloud data. The sensor uses the point cloud data to localize targets (people) in the localization phase which uses a 3D contact acceleration model characterized by 9 elements state vector

$$s_{3DA}(n) = [x_n, y_n, z_n, \dot{x}_n, \dot{y}_n, \dot{z}_n, \ddot{x}_n, \ddot{y}_n, \ddot{z}_n]. \tag{C.4}$$

C.2.1. People Tracking Algorithm from Texas Instruments

The algorithm consists of the following steps prediction, association, updating, and maintenance. The steps will be elaborated on in this subsection. The tracking is implemented in Cartesian coordinates and can operate in either 2D or 3D Cartesian space. An constant acceleration model is used where for each discrete time step **T**, the position and velocity of an target can potentially change.

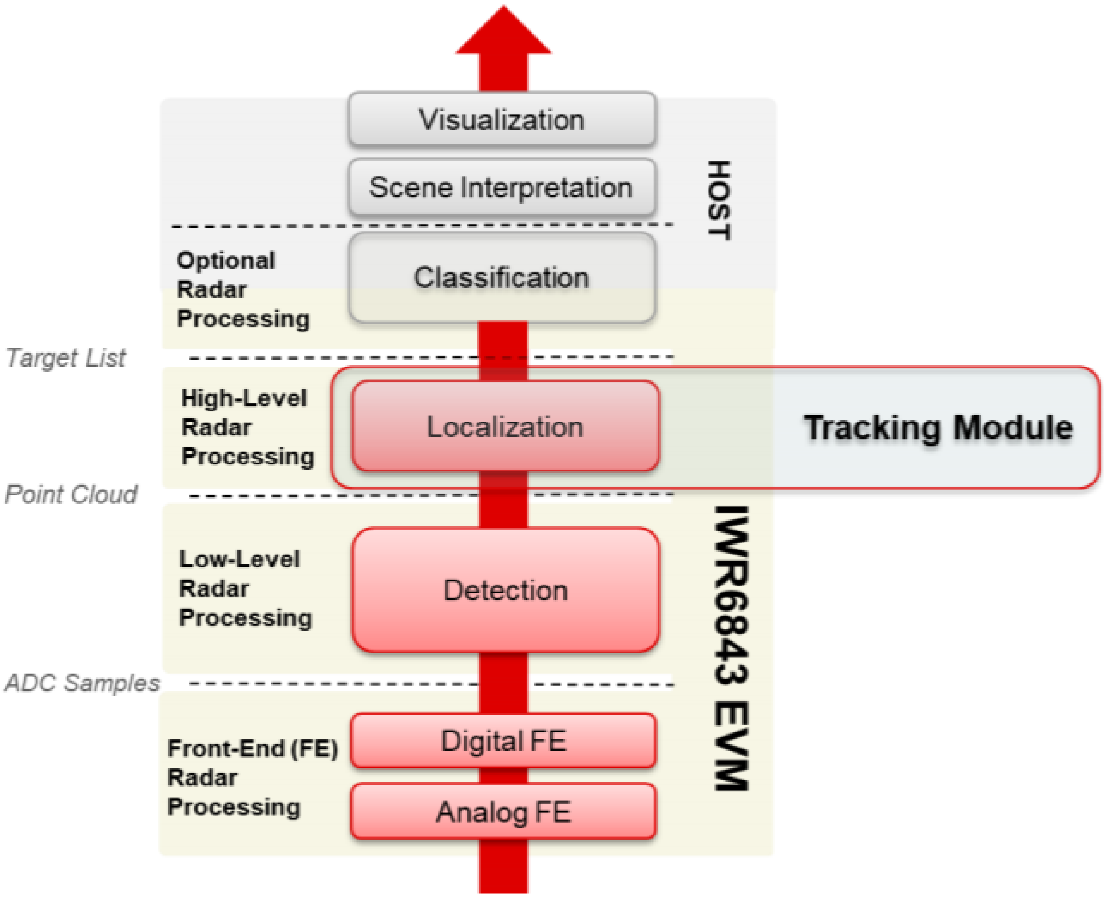


Figure C.1: Radar Processing Layers [47]

To specify the the dynamics of the motion model, TI has used the state transition matrix, which given by:

$$F_{3DA} = \begin{pmatrix} 1 & 0 & 0 & T & 0 & 0 & 0.5T^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 & 0 & 0.5T^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & T & 0 & 0 & 0.5T^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The state of the Kalman filter at time instant n is given as:

$$s(n) = Fs(n-1) + w(n) \tag{C.5}$$

Where $s(n)$ is the state vector given by C.4, F is the transition state matrix given by C.2.1, and $w(n)$ is the vector of process noise.

The input measurements vector $u(n)$ from the radar sensor is in spherical co-ordinates and includes range (r), azimuth (φ), elevation (θ), and radial velocity ($\dot{r}$)

$$u(n) = [r(n)\ \varphi(n)\ \theta(n)\ \dot{r}(n)]^T \tag{C.6}$$

The relation between the state $s(n)$ of the Kalman filter and measurement vector $u(n)$ is given by:

$$u(n) = H(s(n)) + v(n) \tag{C.7}$$

Where $v(n)$ is vector of measurement noise with co variance matrix $R(n)$ of size 4x4, and H is a measurement matrix given by:

$$H(s(n)) = \begin{pmatrix} \sqrt{x^2 + y^2 + z^2} \\ \tan^{-1}(x, y) \\ \tan^{-1}(z, \sqrt{x^2 + y^2}) \\ \frac{x\dot{x} + y\dot{y} + z\dot{z}}{\sqrt{x^2+y^2+z^2}} \end{pmatrix}$$

The function $\tan^{-1}(a, b)$ is defined as:

$$\tan^{-1}(a, b) = \begin{cases} \tan^{-1}(\frac{a}{b}) & b>0 \\ \frac{\pi}{2} & b = 0 \\ \tan^{-1}(\frac{a}{b}) + \pi & b<0 \end{cases}$$

Prediction Step:

In equation C.7, the measurement vector $u(n)$ is related tot he state vector $s(n)$ via a non-linear relation, therefore, in this step, extended kalman filter (EKF) is used that linearizes the non-linear function using derivative of the non-linear function around current state. After linearization, the relation between the state $s(n)$ and measurement vector $u(n)$ is:

$$u(n) = H(s_{apr}(n) + J_H(s_{apr}(n))[s(n) - s_{apr}(n)] + v(n) \tag{C.8}$$

Where $s_{apr}(n)$ is a-priori (predicted) estimates of tracking state at time n, and J_H is the Jacobian matrix given by:

$$J_H(s_{3DA}) = \begin{pmatrix} \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{y}{x^2+y^2} & \frac{-x}{x^2+y^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{-x}{r^2}\frac{z}{\sqrt{x^2+y^2}} & \frac{-y}{r^2}\frac{z}{\sqrt{x^2+y^2}} & \frac{\sqrt{x^2+y^2}}{r^2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{y(\dot{x}y-\dot{y}x)+z(\dot{x}z-\dot{z}x)}{r^3} & \frac{x(\dot{y}x-\dot{x}y)+z(\dot{y}z-\dot{z}y)}{r^3} & \frac{x(\dot{z}x-\dot{x}z)+y(\dot{z}y-\dot{y}z)}{r^3} & \frac{x}{r} & \frac{y}{r} & \frac{z}{r} & 0 & 0 & 0 \end{pmatrix}$$

Association Step

The association step consists of associating radar measurements to a unique existing track. This is accomplished by 3 main steps namely Gating, Scoring and Allocate. Gating is the selection of points based on their proximity to each other. If a point is close enough to the neighbouring, the point is included. When a point is included, the scoring step determines what point has the highest power. The measurement point with the highest power is assigned to that point. Points that are not assigned yet, go through the allocate function. There, points are first grouped based on their distance. If the groups of points satisfy several tests, the group is converted to a new track.

Updating and maintenance

fEach track goes through a life cycle of events. At the maintenance step the state is changed or the track is deleted that is not active any more. Tracks are updated using the set of associated points from the previous step.

C.3. mmWave TLV data

Figure C.2: Collected data from mmWave sensor in TLV protocol


```
10:36:35 b''
10:36:35 b'\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x03d\x00\x00\x00Ch\n'
10:36:35
b'\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xdd\x1d\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x01\x00Rf\x06\x00\x00\x004\x00\x00\x00\n'
10:36:35 b'\xd7#<\n'
10:36:35 b'\xd7#<\xf7\xcc\x929o\x12\x839\n'
10:36:35
b'\xd7#=\xff1\x07\xff\xd8\x0c\x03\x01\x021\x07\xff\xd5\r\xe7\x00\xfaZ\x07\xff\xe1\t\x8d\x00\x02\x01\x04\
x03\x06\x05\x08\x07\x04\x00\x05\x03\\\x00\x00\x00Ch\n'
10:36:35
b'\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00w\x1d\x00\x00\x0c\x00\x00\x00:\x05\x00\x00\x0
1\x00ya\x06\x00\x00\x00,\x00\x00\x00\n'
10:36:36 b'\xd7#<\n'
10:36:36 b'\xd7#<\xf7\xcc\x929o\x12\x839\n'
10:36:36 b'\xd7#=\xfd[\x07\xff\xe1\t\x9f\x00\xff_\x07\xff\xde\n'
10:36:36 b'\x8d\x00\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x03T\x00\x00\x00Ch\n'
10:36:36
b'\x00\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xf9\x1c\x00\x00\x07\x00\x00\x00\xa5\x05\x00\x
00\x01\x00\x98a\x06\x00\x00\x00$\x00\x00\x00\n'
10:36:36 b'\xd7#<\n'
10:36:36 b'\xd7#<\xf7\xcc\x929o\x12\x839\n'
10:36:36
10:36:36
b'\x00\x04\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x1c\x00\x00\x05\x00\x00\x00(\x06\x00\x00\x
00\x00\x95a\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x030\x00\x00\x00Ch\n'
10:36:36
"b'\x00\x05\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x1c\x00\x00\x03\x00\x00\x00\x9e\x02\x00\
10:36:36
b'\x00\x06\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x1c\x00\x00\x03\x00\x00\x00\x9e\x02\x00\x
00\x00\x00\x1fe\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x030\x00\x00\x00Ch\n'
10:36:36
b'\x00\x07\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x1c\x00\x00\x03\x00\x00\x00\x9e\x02\x00\x
00\x00\x00\x1ee\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x030\x00\x00\x00Ch\n'
10:36:36
b'\x00\x08\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x1c\x00\x00\x03\x00\x00\x00\x9e\x02\x00\x
00\x00\x00\x1de\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x030\x00\x00\x00Ch\n'
10:36:36
b'\x00\t\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x1c\x00\x00\x03\x00\x00\x00\x9e\x02\x00\x00\
x00\x00\x1ce\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x030\x00\x00\x00Ch\n'
10:36:36 b'\x00\n'
10:36:36
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x1c\x00\x00\x03\x00\x00\x00\x9e\x02\x00\x00\x00\x
00\x1be\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x030\x00\x00\x00Ch\n'
10:36:36
b'\x00\x0b\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x1c\x00\x00\x03\x00\x00\x00\x9e\x02\x00\x
00\x00\x00\x1ae\x02\x01\x04\x03\x06\x05\x08\x07\x04\x00\x05\x030\x00\x00\x00Ch\n'
```

D
Python Codes

D.1. main.py

```
##############################################################################
# Data Acquisition system                                                    #
# Authors: Alan, Tim                                                         #
# Discription: In this code, sound sensor, gas sensor, gyroscope,            #
# and real-time module are connected. Gas sensor and sound are analog        #
# and are connected to pin A2 (GP28). A multiplexer is used to alternate     #
# between the two sensors using a selector (Sel). Gyroscope and real-time    #
# module are connected via an I2C bus (1), the gyroscope's address is changed #
# from 0x68 to 0x69 by connecting the AD0 pin to 3.3 v. The real-time module  #
# has an address of 0x68.                                                    #
##############################################################################
# NOTE: The real-time needs to be calibrated each time you disconnect the power
# Unless you connect an external battery. However, to recalibrate the module
# use the following code in the sell:
# bus = I2C(1,scl=Pin(15),sda=Pin(14), freq=400000)
#                 sec min hour week day month year
#    NowTime = b'\x00\x44\x18\x02\x27\x05\x21'
# bus.writeto_mem(int(0x68),int(0x00),NowTime)
from machine import Pin, ADC    # Import Pins and ADCs
import os, sdcard
import mpu6050                  # Import library for gyroscope and accelerometer
from mq9 import MQ              # Import library for gas sensor
import time                    # Import time library
from ds3231 import ds3231       # Import real time module library

# Declare objects
mpu = mpu6050.MPU6050() # Define gyroscope object. Connected to i2c1 with address 0x69
mq = MQ()               # Define gas sensor object.
real_time = ds3231(1,15,14)  # Define real time object connected to i2c1 with address 0x68
# Define pins
sel = machine.Pin (2, machine.Pin.OUT) # use pin 2 as slector to select between gas and sound
sound = ADC(28)
# SPI SDcard
spi = machine.SPI(1)
spi.init()  # Ensure right baudrate
sd_spi = machine.SPI(1, sck = machine.Pin(10, machine.Pin.OUT), mosi = machine.Pin(11,
    machine.Pin.OUT), miso = machine.Pin(12, machine.Pin.OUT))
sd = sdcard.SDCard(sd_spi, machine.Pin(9))
vfs = os.VfsFat(sd)
os.mount(vfs, "/fc")
print("Filesystem check")
print(os.listdir("/fc"))
fn = "/sd/teddy.txt"

# Define Varaibles
conversion_factor = 3.3/(65536)        # To read the correct value from the ADC
gyroData=[]
gasData=[]
soundData=[]
timeData=[]
```

```
iteration = 10
cnt=0
# define a function to store data
def store_with_space(value):
    with open(fn, "a") as f:
        n = f.write(value+",")
        #print(n, "bytes written")
def store_with_enter(value):
    with open(fn, "a") as f:
        n = f.write(value+"\n")
def store_time(value):
    with open(fn, "a") as f:
        n = f.write(value+":")
def store(value):
    with open(fn, "a") as f:
        n = f.write(value)

# Loop
while (cnt<5):
      tt = real_time.read_time()
      store_time(str("%02x" %(tt[2])))
      time.sleep(0.001)
      store_time(str("%02x" %(tt[1])))
      time.sleep(0.001)
      store_with_space(str("%02x" %(tt[0])))
      time.sleep(0.001)
#        store(str("%02x" %(tt[3])))
#        time.sleep(0.01)
#        store(str("%02x" %(tt[4])))
#        time.sleep(0.01)
#        store(str("%02x" %(tt[5])))
#        time.sleep(0.01)
#        store(str("20%x" %(tt[6])))
#        time.sleep(0.01)
      g=mpu.readData()
      store_with_space(str(g.Gx))
      time.sleep(0.001)
      store_with_space(str(g.Gy))
      time.sleep(0.001)
      store_with_space(str(g.Gz))
      time.sleep(0.001)
      store_with_space(str(g.Gyrox))
      time.sleep(0.001)
      store_with_space(str(g.Gyroy))
      time.sleep(0.001)
      store_with_space(str(g.Gyroz))
      time.sleep(0.001)
      store_with_space(str(g.Temperature))
      #print("X:{:.2f}  Y:{:.2f}  Z:{:.2f} Gyrox:{:.2f} Gyroy:{:.2f} Gyroz:{:.2f} tem:{:.2f}".format(g.Gx,g.Gy,g.Gz,g.Gyrox,g.Gyroy,g.Gyroz,g.Temperature))
      time.sleep(0.001)
      sel.value(0)
      perc = mq.MQPercentage()
      time.sleep(0.001)
      store_with_space(str(perc["CO"]))
      time.sleep(0.001)
      store_with_space(str(perc["SMOKE"]))
      time.sleep(0.001)
      sel.value(1)
      #print("CO: %g ppm, Smoke: %g ppm %g sound" % (perc["CO"], perc["SMOKE"], sound.read_u16() *conversion_factor))

      #print(sound.read_u16()*conversion_factor)
      #utime.sleep(0.1)

      s = sound.read_u16()*conversion_factor
      store_with_enter(str(s))
      time.sleep(0.001)
      cnt=cnt+1
      #store(str(tt))
      #print(t)
#        print(g.Gx,g.Gy,g.Gz,g.Gyrox,g.Gyroy,g.Gyroz,perc["CO"], perc["SMOKE"], sound.read_u16()*conversion_factor)
```

```
    #print(g.Gx,g.Gy,g.Gz,g.Gyrox,g.Gyroy,g.Gyroz, s)
```

D.2. ds3231.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from machine import Pin, I2C
import time
import binascii

#    the first version use i2c1
#I2C_PORT = 1
#I2C_SDA = 6
#I2C_SCL = 7

#    the new version use i2c0,if it dont work,try to uncomment the line 14 and comment line 17
#    it should solder the R3 with 0R resistor if want to use alarm function,please refer to the Sch file on waveshare Pico-RTC-DS3231 wiki
#    https://www.waveshare.net/w/upload/0/08/Pico-RTC-DS3231_Sch.pdf
I2C_PORT = 0
I2C_SDA = 20
I2C_SCL = 21

ALARM_PIN = 3


class ds3231(object):
#            13:45:00 Mon 24 May 2021
#  the register value is the binary-coded decimal (BCD) format
#               sec min hour week day month year
    NowTime = b'\x00\x44\x18\x02\x27\x05\x21'
    w  = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"];
    address = 0x68
    start_reg = 0x00
    alarm1_reg = 0x07
    control_reg = 0x0e
    status_reg = 0x0f

    def __init__(self,i2c_port,i2c_scl,i2c_sda):
        self.bus = I2C(1,scl=Pin(15),sda=Pin(14), freq=400000)

    def set_time(self,new_time):
        hour = new_time[0] + new_time[1]
        minute = new_time[3] + new_time[4]
        second = new_time[6] + new_time[7]
        week = "0" + str(self.w.index(new_time.split(",",2)[1])+1)
        year = new_time.split(",",2)[2][2] + new_time.split(",",2)[2][3]
        month = new_time.split(",",2)[2][5] + new_time.split(",",2)[2][6]
        day = new_time.split(",",2)[2][8] + new_time.split(",",2)[2][9]
        now_time = binascii.unhexlify((second + " " + minute + " " + hour + " " + week + " " + day + " " + month + " " + year).replace(' ',''))
        #print(binascii.unhexlify((second + " " + minute + " " + hour + " " + week + " " + day + " " + month + " " + year).replace(' ','')))
        #print(self.NowTime)
        self.bus.writeto_mem(int(self.address),int(self.start_reg),now_time)

    def read_time(self):
        t = self.bus.readfrom_mem(int(self.address),int(self.start_reg),7)
        a = t[0]&0x7F  #second
        b = t[1]&0x7F  #minute
        c = t[2]&0x3F  #hour
        d = t[3]&0x07  #week
        e = t[4]&0x3F  #day
        f = t[5]&0x1F  #month
        print("20%x/%02x/%02x %02x:%02x:%02x %s" %(t[6],t[5],t[4],t[2],t[1],t[0],self.w[t[3]-1]))

    def set_alarm_time(self,alarm_time):
        #    init the alarm pin
        self.alarm_pin = Pin(ALARM_PIN,Pin.IN,Pin.PULL_UP)
        #    set alarm irq
        self.alarm_pin.irq(lambda pin: print("alarm1 time is up"), Pin.IRQ_FALLING)
```

```
        #    enable the alarm1 reg
        self.bus.writeto_mem(int(self.address),int(self.control_reg),b’\x05’)
        #    convert to the BCD format
        hour = alarm_time[0] + alarm_time[1]
        minute = alarm_time[3] + alarm_time[4]
        second = alarm_time[6] + alarm_time[7]
        date = alarm_time.split(”,”,2)[2][8] + alarm_time.split(”,”,2)[2][9]
        now_time = binascii.unhexlify((second + ” ” + minute + ” ” + hour +  ” ” + date).replace(’ ’,’’))
        #    write alarm time to alarm1 reg
        self.bus.writeto_mem(int(self.address),int(self.alarm1_reg),now_time)

if __name__ == ’__main__’:
    rtc = ds3231(I2C_PORT,I2C_SCL,I2C_SDA)
    rtc.set_time(’13:45:50,Monday,2021-05-24’)
    rtc.read_time()
    rtc.set_alarm_time(’13:45:55,Monday,2021-05-24’)
```

D.3. mpu6050.py

```
import struct
import math
import utime
from machine import Pin, I2C

#https://www.raspberrypi.org/forums/viewtopic.php?t=302363

class MPU6050Data:

    def __init__(self):
        self.Gx=0
        self.Gy=0
        self.Gz=0
        self.Temperature=0
        self.Gyrox=0
        self.Gyroy=0
        self.Gyroz=0

class MPU6050:

    AccelerationFactor= 2.0/32768.0;   #assuming +/- 16G
    GyroFactor=500.0 / 32768.0;         #assuming 500 degree / sec

    # Temperature in degrees C = (TEMP_OUT Register Value as a signed quantity)/340 + 36.53
    TemperatureGain = 1.0 / 340.0
    TemperatureOffset = 36.53

    #converted from Jeff Rowberg code https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/MPU6050.h


    #register definition

    MPU6050_RA_XG_OFFS_TC = 0x00 # [7] PWR_MODE, [6:1] XG_OFFS_TC, [0] OTP_BNK_VLD
    MPU6050_RA_YG_OFFS_TC = 0x01 # [7] PWR_MODE, [6:1] YG_OFFS_TC, [0] OTP_BNK_VLD
    MPU6050_RA_ZG_OFFS_TC = 0x02 # [7] PWR_MODE, [6:1] ZG_OFFS_TC, [0] OTP_BNK_VLD
    MPU6050_RA_X_FINE_GAIN = 0x03 # [7:0] X_FINE_GAIN
    MPU6050_RA_Y_FINE_GAIN = 0x04 # [7:0] Y_FINE_GAIN
    MPU6050_RA_Z_FINE_GAIN = 0x05  # [7:0] Z_FINE_GAIN
    MPU6050_RA_XA_OFFS_H = 0x06  # [15:0] XA_OFFS
    MPU6050_RA_XA_OFFS_L_TC = 0x07
    MPU6050_RA_YA_OFFS_H = 0x08  #[15:0] YA_OFFS
    MPU6050_RA_YA_OFFS_L_TC = 0x09
    MPU6050_RA_ZA_OFFS_H = 0x0A  #[15:0] ZA_OFFS
    MPU6050_RA_ZA_OFFS_L_TC = 0x0B
    MPU6050_RA_XG_OFFS_USRH = 0x13  #[15:0] XG_OFFS_USR
    MPU6050_RA_XG_OFFS_USRL = 0x14
    MPU6050_RA_YG_OFFS_USRH = 0x15  #[15:0] YG_OFFS_USR
    MPU6050_RA_YG_OFFS_USRL = 0x16
    MPU6050_RA_ZG_OFFS_USRH = 0x17  #[15:0] ZG_OFFS_USR
    MPU6050_RA_ZG_OFFS_USRL = 0x18
    MPU6050_RA_SMPLRT_DIV = 0x19
    MPU6050_RA_CONFIG = 0x1A
```

```
    MPU6050_RA_GYRO_CONFIG = 0x1B
    MPU6050_RA_ACCEL_CONFIG = 0x1C
    MPU6050_RA_FF_THR = 0x1D
    MPU6050_RA_FF_DUR = 0x1E
    MPU6050_RA_MOT_THR = 0x1F
    MPU6050_RA_MOT_DUR = 0x20
    MPU6050_RA_ZRMOT_THR = 0x21
    MPU6050_RA_ZRMOT_DUR = 0x22
    MPU6050_RA_FIFO_EN = 0x23
    MPU6050_RA_I2C_MST_CTRL = 0x24
    MPU6050_RA_I2C_SLV0_ADDR = 0x25
    MPU6050_RA_I2C_SLV0_REG = 0x26
    MPU6050_RA_I2C_SLV0_CTRL = 0x27
    MPU6050_RA_I2C_SLV1_ADDR = 0x28
    MPU6050_RA_I2C_SLV1_REG = 0x29
    MPU6050_RA_I2C_SLV1_CTRL = 0x2A
    MPU6050_RA_I2C_SLV2_ADDR = 0x2B
    MPU6050_RA_I2C_SLV2_REG = 0x2C
    MPU6050_RA_I2C_SLV2_CTRL = 0x2D
    MPU6050_RA_I2C_SLV3_ADDR = 0x2E
    MPU6050_RA_I2C_SLV3_REG = 0x2F
    MPU6050_RA_I2C_SLV3_CTRL = 0x30
    MPU6050_RA_I2C_SLV4_ADDR = 0x31
    MPU6050_RA_I2C_SLV4_REG = 0x32
    MPU6050_RA_I2C_SLV4_DO = 0x33
    MPU6050_RA_I2C_SLV4_CTRL = 0x34
    MPU6050_RA_I2C_SLV4_DI = 0x35
    MPU6050_RA_I2C_MST_STATUS = 0x36
    MPU6050_RA_INT_PIN_CFG = 0x37
    MPU6050_RA_INT_ENABLE = 0x38
    MPU6050_RA_DMP_INT_STATUS = 0x39
    MPU6050_RA_INT_STATUS = 0x3A
    MPU6050_RA_ACCEL_XOUT_H = 0x3B
    MPU6050_RA_ACCEL_XOUT_L = 0x3C
    MPU6050_RA_ACCEL_YOUT_H = 0x3D
    MPU6050_RA_ACCEL_YOUT_L = 0x3E
    MPU6050_RA_ACCEL_ZOUT_H = 0x3F
    MPU6050_RA_ACCEL_ZOUT_L = 0x40
    MPU6050_RA_TEMP_OUT_H = 0x41
    MPU6050_RA_TEMP_OUT_L = 0x42
    MPU6050_RA_GYRO_XOUT_H = 0x43
    MPU6050_RA_GYRO_XOUT_L = 0x44
    MPU6050_RA_GYRO_YOUT_H = 0x45
    MPU6050_RA_GYRO_YOUT_L = 0x46
    MPU6050_RA_GYRO_ZOUT_H = 0x47
    MPU6050_RA_GYRO_ZOUT_L = 0x48
    MPU6050_RA_EXT_SENS_DATA_00 = 0x49
    MPU6050_RA_EXT_SENS_DATA_01 = 0x4A
    MPU6050_RA_EXT_SENS_DATA_02 = 0x4B
    MPU6050_RA_EXT_SENS_DATA_03 = 0x4C
    MPU6050_RA_EXT_SENS_DATA_04 = 0x4D
    MPU6050_RA_EXT_SENS_DATA_05 = 0x4E
    MPU6050_RA_EXT_SENS_DATA_06 = 0x4F
    MPU6050_RA_EXT_SENS_DATA_07 = 0x50
    MPU6050_RA_EXT_SENS_DATA_08 = 0x51
    MPU6050_RA_EXT_SENS_DATA_09 = 0x52
    MPU6050_RA_EXT_SENS_DATA_10 = 0x53
    MPU6050_RA_EXT_SENS_DATA_11 = 0x54
    MPU6050_RA_EXT_SENS_DATA_12 = 0x55
    MPU6050_RA_EXT_SENS_DATA_13 = 0x56
    MPU6050_RA_EXT_SENS_DATA_14 = 0x57
    MPU6050_RA_EXT_SENS_DATA_15 = 0x58
    MPU6050_RA_EXT_SENS_DATA_16 = 0x59
    MPU6050_RA_EXT_SENS_DATA_17 = 0x5A
    MPU6050_RA_EXT_SENS_DATA_18 = 0x5B
    MPU6050_RA_EXT_SENS_DATA_19 = 0x5C
    MPU6050_RA_EXT_SENS_DATA_20 = 0x5D
    MPU6050_RA_EXT_SENS_DATA_21 = 0x5E
    MPU6050_RA_EXT_SENS_DATA_22 = 0x5F
    MPU6050_RA_EXT_SENS_DATA_23 = 0x60
    MPU6050_RA_MOT_DETECT_STATUS = 0x61
    MPU6050_RA_I2C_SLV0_DO = 0x63
    MPU6050_RA_I2C_SLV1_DO = 0x64
```

```
    MPU6050_RA_I2C_SLV2_DO = 0x65
    MPU6050_RA_I2C_SLV3_DO = 0x66
    MPU6050_RA_I2C_MST_DELAY_CTRL = 0x67
    MPU6050_RA_SIGNAL_PATH_RESET = 0x68
    MPU6050_RA_MOT_DETECT_CTRL = 0x69
    MPU6050_RA_USER_CTRL = 0x6A
    MPU6050_RA_PWR_MGMT_1 = 0x6B
    MPU6050_RA_PWR_MGMT_2 = 0x6C
    MPU6050_RA_BANK_SEL = 0x6D
    MPU6050_RA_MEM_START_ADDR = 0x6E
    MPU6050_RA_MEM_R_W = 0x6F
    MPU6050_RA_DMP_CFG_1 = 0x70
    MPU6050_RA_DMP_CFG_2 = 0x71
    MPU6050_RA_FIFO_COUNTH = 0x72
    MPU6050_RA_FIFO_COUNTL = 0x73
    MPU6050_RA_FIFO_R_W = 0x74
    MPU6050_RA_WHO_AM_I = 0x75

    ZeroRegister = [
        MPU6050_RA_FF_THR, #Freefall threshold of |0mg|  LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FF_THR, 0x00);
        MPU6050_RA_FF_DUR, #Freefall duration limit of 0   LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FF_DUR, 0x00);
        MPU6050_RA_MOT_THR, #Motion threshold of 0mg     LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_THR, 0x00);
        MPU6050_RA_MOT_DUR, #Motion duration of 0s    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_DUR, 0x00);
        MPU6050_RA_ZRMOT_THR, #Zero motion threshold    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_ZRMOT_THR, 0x00);
        MPU6050_RA_ZRMOT_DUR, #Zero motion duration threshold    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_ZRMOT_DUR, 0x00);
        MPU6050_RA_FIFO_EN, #Disable sensor output to FIFO buffer    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FIFO_EN, 0x00);
        MPU6050_RA_I2C_MST_CTRL, #AUX I2C setup    //Sets AUX I2C to single master control, plus other config    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_MST_CTRL, 0x00);
        MPU6050_RA_I2C_SLV0_ADDR, #Setup AUX I2C slaves    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_ADDR, 0x00);
        MPU6050_RA_I2C_SLV0_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_REG, 0x00);
        MPU6050_RA_I2C_SLV0_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_CTRL, 0x00);
        MPU6050_RA_I2C_SLV1_ADDR, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_ADDR, 0x00);
        MPU6050_RA_I2C_SLV1_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_REG, 0x00);
        MPU6050_RA_I2C_SLV1_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_CTRL, 0x00);
        MPU6050_RA_I2C_SLV2_ADDR, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_ADDR, 0x00);
        MPU6050_RA_I2C_SLV2_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_REG, 0x00);
        MPU6050_RA_I2C_SLV2_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_CTRL, 0x00);
        MPU6050_RA_I2C_SLV3_ADDR, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_ADDR, 0x00);
        MPU6050_RA_I2C_SLV3_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_REG, 0x00);
        MPU6050_RA_I2C_SLV3_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_CTRL, 0x00);
        MPU6050_RA_I2C_SLV4_ADDR, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_ADDR, 0x00);
        MPU6050_RA_I2C_SLV4_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_REG, 0x00);
        MPU6050_RA_I2C_SLV4_DO, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_DO, 0x00);
        MPU6050_RA_I2C_SLV4_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_CTRL, 0x00);
        MPU6050_RA_I2C_SLV4_DI, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_DI, 0x00);
        MPU6050_RA_INT_PIN_CFG, #MPU6050_RA_I2C_MST_STATUS //Read-only    //Setup INT pin and AUX I2C pass through    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_INT_PIN_CFG, 0x00);
        MPU6050_RA_INT_ENABLE, #Enable data ready interrupt      LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_INT_ENABLE, 0x00);
```

```
    MPU6050_RA_I2C_SLV0_DO, #Slave out, dont care    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_DO, 0x00);
    MPU6050_RA_I2C_SLV1_DO, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_DO, 0x00);
    MPU6050_RA_I2C_SLV2_DO, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_DO, 0x00);
    MPU6050_RA_I2C_SLV3_DO, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_DO, 0x00);
    MPU6050_RA_I2C_MST_DELAY_CTRL, #More slave config      LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_MST_DELAY_CTRL, 0x00);
    MPU6050_RA_SIGNAL_PATH_RESET, #Reset sensor signal paths    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_SIGNAL_PATH_RESET, 0x00);
    MPU6050_RA_MOT_DETECT_CTRL, #Motion detection control    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_DETECT_CTRL, 0x00);
    MPU6050_RA_USER_CTRL, #Disables FIFO, AUX I2C, FIFO and I2C reset bits to 0    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_USER_CTRL, 0x00);
    MPU6050_RA_CONFIG, #Disable FSync, 256Hz DLPF    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_CONFIG, 0x00);
    MPU6050_RA_FF_THR, #Freefall threshold of |0mg|    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FF_THR, 0x00);
    MPU6050_RA_FF_DUR, #Freefall duration limit of 0    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FF_DUR, 0x00);
    MPU6050_RA_MOT_THR, #Motion threshold of 0mg    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_THR, 0x00);
    MPU6050_RA_MOT_DUR, #Motion duration of 0s    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_DUR, 0x00);
    MPU6050_RA_ZRMOT_THR, #Zero motion threshold    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_ZRMOT_THR, 0x00);
    MPU6050_RA_ZRMOT_DUR, #Zero motion duration threshold    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_ZRMOT_DUR, 0x00);
    MPU6050_RA_FIFO_EN, #Disable sensor output to FIFO buffer    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FIFO_EN, 0x00);
    MPU6050_RA_I2C_MST_CTRL, #AUX I2C setup    //Sets AUX I2C to single master control, plus other config    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_MST_CTRL, 0x00);
    MPU6050_RA_I2C_SLV0_ADDR, #Setup AUX I2C slaves    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_ADDR, 0x00);
    MPU6050_RA_I2C_SLV0_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_REG, 0x00);
    MPU6050_RA_I2C_SLV0_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_CTRL, 0x00);
    MPU6050_RA_I2C_SLV1_ADDR, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_ADDR, 0x00);
    MPU6050_RA_I2C_SLV1_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_REG, 0x00);
    MPU6050_RA_I2C_SLV1_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_CTRL, 0x00);
    MPU6050_RA_I2C_SLV2_ADDR, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_ADDR, 0x00);
    MPU6050_RA_I2C_SLV2_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_REG, 0x00);
    MPU6050_RA_I2C_SLV2_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_CTRL, 0x00);
    MPU6050_RA_I2C_SLV3_ADDR, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_ADDR, 0x00);
    MPU6050_RA_I2C_SLV3_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_REG, 0x00);
    MPU6050_RA_I2C_SLV3_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_CTRL, 0x00);
    MPU6050_RA_I2C_SLV4_ADDR, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_ADDR, 0x00);
    MPU6050_RA_I2C_SLV4_REG, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_REG, 0x00);
    MPU6050_RA_I2C_SLV4_DO, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_DO, 0x00);
    MPU6050_RA_I2C_SLV4_CTRL, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_CTRL, 0x00);
    MPU6050_RA_I2C_SLV4_DI, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_DI, 0x00);
    MPU6050_RA_I2C_SLV0_DO, #Slave out, dont care    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_DO, 0x00);
    MPU6050_RA_I2C_SLV1_DO, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_DO, 0x00);
    MPU6050_RA_I2C_SLV2_DO, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_DO, 0x00
```

```
    );
        MPU6050_RA_I2C_SLV3_DO, #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_DO, 0x00);
        MPU6050_RA_I2C_MST_DELAY_CTRL, #More slave config    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_MST_DELAY_CTRL, 0x00);
        MPU6050_RA_SIGNAL_PATH_RESET, #Reset sensor signal paths    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_SIGNAL_PATH_RESET, 0x00);
        MPU6050_RA_MOT_DETECT_CTRL, #Motion detection control    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_DETECT_CTRL, 0x00);
        MPU6050_RA_USER_CTRL, #Disables FIFO, AUX I2C, FIFO and I2C reset bits to 0 LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_USER_CTRL, 0x00);
        MPU6050_RA_INT_PIN_CFG, #MPU6050_RA_I2C_MST_STATUS //Read-only    //Setup INT pin and AUX I2C pass through    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_INT_PIN_CFG, 0x00);
        MPU6050_RA_INT_ENABLE, #Enable data ready interrupt    LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_INT_ENABLE, 0x00);
        MPU6050_RA_FIFO_R_W ] #LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FIFO_R_W, 0x00);


    def reg_write(self,reg_addr,value):
        self.i2c.writeto_mem(self.MPU6050_ADDRESS,reg_addr,value)

    def reg_writeByte(self,reg_addr,value):
        self.reg_write(reg_addr,bytearray(value))

    def reg_read(self,reg_addr, count):
        return self.i2c.readfrom_mem(self.MPU6050_ADDRESS,reg_addr,count)

    def __init__(self, bus=1, address=0x69, scl=Pin(15), sda=Pin(14), freq=400000):
        self.i2c = I2C(bus,scl=scl,sda=sda,freq=freq)
        self.MPU6050_ADDRESS = address
        self.setSampleRate(100)
        self.setGResolution(2)
        self.setGyroResolution(250)
        # Disable gyro self tests, scale of 500 degrees/s
        self.reg_writeByte(self.MPU6050_RA_GYRO_CONFIG, 0b00001000)

        for loop in self.ZeroRegister:
            self.reg_writeByte(loop,0)

        # Sets clock source to gyro reference w/ PLL
        self.reg_writeByte(self.MPU6050_RA_PWR_MGMT_1, 0b00000010)

        #Controls frequency of wakeups in accel low power mode plus the sensor standby modes
        self.reg_writeByte(self.MPU6050_RA_PWR_MGMT_2, 0x00)

        self.reg_writeByte(self.MPU6050_RA_INT_ENABLE, 0x01)
        self.readStatus()
        self.fifoCount =0

    def readDataFromFifo(self):
        # first check how many bytes in temporary fifo counter
        if self.fifoCount == 0 :
            self.fifoCount = self.readFifoCount()

        #max block transfer in i2c is 32 bytes including the address
        # accelerometer, gyro and temperature  data=> 7 short  = 14 bytes  => 31 bytes / 14 = 2
        # then it will be 28
        if (self.fifoCount > 28) :
            nCount = 28
        else:
            nCount = self.fifoCount
        GData = self.reg_read(self.MPU6050_RA_FIFO_R_W, nCount)
        self.fifoCount = self.fifoCount - nCount
        return GData

    def readData(self):
        #read accelerometers , temperature and gyro
        GData = self.reg_read(self.MPU6050_RA_ACCEL_XOUT_H,14)
        #convert list of 14 values bytes into MPU6050Data struct in engineering units
        return self.convertData(GData)

    def convertData(self,ListData):
```

```python
        ShortData = struct.unpack(">hhhhhhh", bytearray(ListData))
        #lets create the Data Class
        AccData = MPU6050Data()

        # first 3 short value are Accelerometer

        AccData.Gx = ShortData[0] * self.AccelerationFactor
        AccData.Gy = ShortData[1] * self.AccelerationFactor
        AccData.Gz = ShortData[2] * self.AccelerationFactor

        #temperature
        AccData.Temperature = ShortData[3] * self.TemperatureGain + self.TemperatureOffset

        #and the 3 last ar'e the gyro data

        AccData.Gyrox = ShortData[4] * self.GyroFactor
        AccData.Gyroy = ShortData[5] * self.GyroFactor
        AccData.Gyroz = ShortData[6] * self.GyroFactor

        return AccData

    def setGyroResolution(self, value):
        #use dictionary to get correct G resolution 2,4,8 or 16G
        self.reg_writeByte(self.MPU6050_RA_GYRO_CONFIG,{250 : 0 , 500 : 8 , 1000 : 16 , 2000
    : 24}[value])
        self.GyroFactor= value/32768.0;


    def setGResolution(self, value):
        #use dictionary to get correct G resolution 2,4,8 or 16G
        self.reg_writeByte(self.MPU6050_RA_ACCEL_CONFIG,{2 : 0 , 4 : 8 , 8 : 16 , 16 : 24}[
    value])
        self.AccelerationFactor= value/32768.0;


    def setSampleRate(self, Rate):
        SampleReg =  int(( 8000 / Rate) -1)
        self.SampleRate = 8000.0 / (SampleReg + 1.0)
        self.reg_writeByte(self.MPU6050_RA_SMPLRT_DIV,SampleReg)


    def readStatus(self):
        return  self.reg_read(self.MPU6050_RA_INT_STATUS,1
                          )

    def readFifoCount(self):
        GData=self.reg_read(self.MPU6050_RA_FIFO_COUNTH,2)
        self.fifoCount = (GData[0] * 256 + GData[1])
        return self.fifoCount

    def readFifo(self, ByteCount):
        GData = self.reg_read(self.MPU6050_RA_FIFO_R_W ,ByteCount)
        return GData

    def resetFifo(self):
        self.reg_writeByte(self.MPU6050_RA_USER_CTRL,0b00000000)
        pass
        self.reg_writeByte(self.MPU6050_RA_USER_CTRL,0b00000100)
        pass
        self.reg_writeByte(self.MPU6050_RA_USER_CTRL,0b01000000)

    def enableFifo(self,flag):
        self.reg_writeByte(self.MPU6050_RA_FIFO_EN,0)
        if flag:
            self.resetFifo()
            self.reg_writeByte(self.MPU6050_RA_FIFO_EN,0b11111000)


if __name__ == "__main__":
    mpu = MPU6050()
    while True:
            g=mpu.readData()
            print("X:{:.2f}  Y:{:.2f}  Z:{:.2f}".format(g.Gx,g.Gy,g.Gz))
```

```
            utime.sleep_ms(100)
```

D.4. mq9.py

```
# adapted from https://github.com/tutRPi/Raspberry-Pi-Gas-Sensor-MQ
##https://github.com/leech001/MQ9
import time
import math
from machine import ADC


class MQ:
    # Hardware Related Macros
    RL_VALUE = 10  # define the load resistance on the board, in kilo ohms
    RO_CLEAN_AIR_FACTOR = 9.83  # RO_CLEAR_AIR_FACTOR=(Sensor resistance in clean air)/RO,
    # which is derived from the chart in datasheet

    # Software Related Macros
    CALIBARAION_SAMPLE_TIMES = 50  # define how many samples you are going to take in the calibration phase
    CALIBRATION_SAMPLE_INTERVAL = 500  # define the time interal(in milisecond) between each samples in the
    # cablibration phase
    READ_SAMPLE_INTERVAL = 50  # define how many samples you are going to take in normal operation
    READ_SAMPLE_TIMES = 5  # define the time interal(in milisecond) between each samples in
    # normal operation

    # Application Related Macros
    GAS_LPG = 0
    GAS_CO = 1
    GAS_SMOKE = 2

    def __init__(self, ro=10):
        self.ro = ro
        self.adc = ADC(2)

        self.LPGCurve = [2.3, 0.21, -0.47]  # two points are taken from the curve.
        # with these two points, a line is formed which is "approximately equivalent"
        # to the original curve.
        # data format:{ x, y, slope}; point1: (lg200, 0.21), point2: (lg10000, -0.59)
        self.COCurve = [2.3, 0.72, -0.34]  # two points are taken from the curve.
        # with these two points, a line is formed which is "approximately equivalent"
        # to the original curve.
        # data format:[ x, y, slope]; point1: (lg200, 0.72), point2: (lg10000,  0.15)
        self.SmokeCurve = [2.3, 0.53, -0.44]  # two points are taken from the curve.
        # with these two points, a line is formed which is "approximately equivalent"
        # to the original curve.
        # data format:[ x, y, slope]; point1: (lg200, 0.53), point2: (lg10000,  -0.22)

        print("Calibrating...")
        self.ro = self.MQCalibration()
        print("Calibration is done...\n")
        print("Ro=%f kohm" % self.ro)

    def MQPercentage(self):
        val = {}
        read = self.MQRead()
        val["GAS_LPG"] = self.MQGetGasPercentage(read / self.ro, self.GAS_LPG)
        val["CO"] = self.MQGetGasPercentage(read / self.ro, self.GAS_CO)
        val["SMOKE"] = self.MQGetGasPercentage(read / self.ro, self.GAS_SMOKE)
        return val

    # MQResistanceCalculation
    # Input:   raw_adc - raw value read from adc, which represents the voltage
    # Output:  the calculated sensor resistance
    # Remarks: The sensor and the load resistor forms a voltage divider. Given the voltage
    #          across the load resistor and its resistance, the resistance of the sensor
    #          could be derived.
    def MQResistanceCalculation(self, raw_adc):
        return float(self.RL_VALUE * (1023.0 - raw_adc) / float(raw_adc))

    # MQCalibration
```

```
    # Output:  Ro of the sensor
    # Remarks: This function assumes that the sensor is in clean air. It use
    #          MQResistanceCalculation to calculates the sensor resistance in clean air
    #          and then divides it with RO_CLEAN_AIR_FACTOR. RO_CLEAN_AIR_FACTOR is about
    #          10, which differs slightly between different sensors.
    def MQCalibration(self):
        val = 0.0
        for i in range(self.CALIBARAION_SAMPLE_TIMES):  # take multiple samples
            val += self.MQResistanceCalculation(self.adc.read_u16()*3.3/(65536))
            time.sleep(self.CALIBRATION_SAMPLE_INTERVAL / 1000.0)

        val = val / self.CALIBARAION_SAMPLE_TIMES  # calculate the average value

        val = val / self.RO_CLEAN_AIR_FACTOR  # divided by RO_CLEAN_AIR_FACTOR yields the Ro
        # according to the chart in the datasheet
        return val

    # MQRead
    # Output:  Rs of the sensor
    # Remarks: This function use MQResistanceCalculation to caculate the sensor resistenc (Rs
    ).
    #          The Rs changes as the sensor is in the different consentration of the target
    #          gas. The sample times and the time interval between samples could be
    configured
    #          by changing the definition of the macros.
    def MQRead(self):
        rs = 0.0

        for i in range(self.READ_SAMPLE_TIMES):
            rs += self.MQResistanceCalculation(self.adc.read_u16()*3.3/(65536))
            time.sleep(self.READ_SAMPLE_INTERVAL / 1000.0)

        rs = rs / self.READ_SAMPLE_TIMES

        return rs

    # MQGetGasPercentage
    # Input:   rs_ro_ratio - Rs divided by Ro
    #          gas_id      - target gas type
    # Output:  ppm of the target gas
    # Remarks: This function passes different curves to the MQGetPercentage function which
    #          calculates the ppm (parts per million) of the target gas.
    def MQGetGasPercentage(self, rs_ro_ratio, gas_id):
        if gas_id == self.GAS_LPG:
            return self.MQGetPercentage(rs_ro_ratio, self.LPGCurve)
        elif gas_id == self.GAS_CO:
            return self.MQGetPercentage(rs_ro_ratio, self.COCurve)
        elif gas_id == self.GAS_SMOKE:
            return self.MQGetPercentage(rs_ro_ratio, self.SmokeCurve)
        return 0

    # MQGetPercentage
    # Input:   rs_ro_ratio - Rs divided by Ro
    #          pcurve      - pointer to the curve of the target gas
    # Output:  ppm of the target gas
    # Remarks: By using the slope and a point of the line. The x(logarithmic value of ppm)
    #          of the line could be derived if y(rs_ro_ratio) is provided. As it is a
    #          logarithmic coordinate, power of 10 is used to convert the result to non-
    logarithmic
    #          value.
    def MQGetPercentage(self, rs_ro_ratio, pcurve):
        return math.pow(10, (((math.log(rs_ro_ratio) - pcurve[1]) / pcurve[2]) + pcurve[0]))
```

D.5. sdcard.py

```
"""
MicroPython driver for SD cards using SPI bus.
Requires an SPI bus and a CS pin.  Provides readblocks and writeblocks
methods so the device can be mounted as a filesystem.
Example usage on pyboard:
    import pyb, sdcard, os
    sd = sdcard.SDCard(pyb.SPI(1), pyb.Pin.board.X5)
    pyb.mount(sd, '/sd2')
```

```
    os.listdir('/')
Example usage on ESP8266:
    import machine, sdcard, os
    sd = sdcard.SDCard(machine.SPI(1), machine.Pin(15))
    os.mount(sd, '/sd')
    os.listdir('/')
"""

from micropython import const
import time


_CMD_TIMEOUT = const(100)

_R1_IDLE_STATE = const(1 << 0)
# R1_ERASE_RESET = const(1 << 1)
_R1_ILLEGAL_COMMAND = const(1 << 2)
# R1_COM_CRC_ERROR = const(1 << 3)
# R1_ERASE_SEQUENCE_ERROR = const(1 << 4)
# R1_ADDRESS_ERROR = const(1 << 5)
# R1_PARAMETER_ERROR = const(1 << 6)
_TOKEN_CMD25 = const(0xFC)
_TOKEN_STOP_TRAN = const(0xFD)
_TOKEN_DATA = const(0xFE)


class SDCard:
    def __init__(self, spi, cs):
        self.spi = spi
        self.cs = cs

        self.cmdbuf = bytearray(6)
        self.dummybuf = bytearray(512)
        self.tokenbuf = bytearray(1)
        for i in range(512):
            self.dummybuf[i] = 0xFF
        self.dummybuf_memoryview = memoryview(self.dummybuf)

        # initialise the card
        self.init_card()

    def init_spi(self, baudrate):
        try:
            master = self.spi.MASTER
        except AttributeError:
            # on ESP8266
            self.spi.init(baudrate=baudrate, phase=0, polarity=0)
        else:
            # on pyboard
            self.spi.init(master, baudrate=baudrate, phase=0, polarity=0)

    def init_card(self):
        # init CS pin
        self.cs.init(self.cs.OUT, value=1)

        # init SPI bus; use low data rate for initialisation
        self.init_spi(100000)

        # clock card at least 100 cycles with cs high
        for i in range(16):
            self.spi.write(b"\xff")

        # CMD0: init card; should return _R1_IDLE_STATE (allow 5 attempts)
        for _ in range(5):
            if self.cmd(0, 0, 0x95) == _R1_IDLE_STATE:
                break
        else:
            raise OSError("no SD card")

        # CMD8: determine card version
        r = self.cmd(8, 0x01AA, 0x87, 4)
        if r == _R1_IDLE_STATE:
            self.init_card_v2()
```

```
        elif r == (_R1_IDLE_STATE | _R1_ILLEGAL_COMMAND):
            self.init_card_v1()
        else:
            raise OSError(”couldn’t determine SD card version”)

        # get the number of sectors
        # CMD9: response R2 (R1 byte + 16-byte block read)
        if self.cmd(9, 0, 0, 0, False) != 0:
            raise OSError(”no response from SD card”)
        csd = bytearray(16)
        self.readinto(csd)
        if csd[0] & 0xC0 == 0x40:  # CSD version 2.0
            self.sectors = ((csd[8] << 8 | csd[9]) + 1) * 1024
        elif csd[0] & 0xC0 == 0x00:  # CSD version 1.0 (old, <=2GB)
            c_size = csd[6] & 0b11 | csd[7] << 2 | (csd[8] & 0b11000000) << 4
            c_size_mult = ((csd[9] & 0b11) << 1) | csd[10] >> 7
            self.sectors = (c_size + 1) * (2 ** (c_size_mult + 2))
        else:
            raise OSError(”SD card CSD format not supported”)
        # print(’sectors’, self.sectors)

        # CMD16: set block length to 512 bytes
        if self.cmd(16, 512, 0) != 0:
            raise OSError(”can’t set 512 block size”)

        # set to high data rate now that it’s initialised
        self.init_spi(1320000)

    def init_card_v1(self):
        for i in range(_CMD_TIMEOUT):
            self.cmd(55, 0, 0)
            if self.cmd(41, 0, 0) == 0:
                self.cdv = 512
                # print(”[SDCard] v1 card”)
                return
        raise OSError(”timeout waiting for v1 card”)

    def init_card_v2(self):
        for i in range(_CMD_TIMEOUT):
            time.sleep_ms(50)
            self.cmd(58, 0, 0, 4)
            self.cmd(55, 0, 0)
            if self.cmd(41, 0x40000000, 0) == 0:
                self.cmd(58, 0, 0, 4)
                self.cdv = 1
                # print(”[SDCard] v2 card”)
                return
        raise OSError(”timeout waiting for v2 card”)

    def cmd(self, cmd, arg, crc, final=0, release=True, skip1=False):
        self.cs(0)

        # create and send the command
        buf = self.cmdbuf
        buf[0] = 0x40 | cmd
        buf[1] = arg >> 24
        buf[2] = arg >> 16
        buf[3] = arg >> 8
        buf[4] = arg
        buf[5] = crc
        self.spi.write(buf)

        if skip1:
            self.spi.readinto(self.tokenbuf, 0xFF)

        # wait for the response (response[7] == 0)
        for i in range(_CMD_TIMEOUT):
            self.spi.readinto(self.tokenbuf, 0xFF)
            response = self.tokenbuf[0]
            if not (response & 0x80):
                # this could be a big-endian integer that we are getting here
                for j in range(final):
                    self.spi.write(b”\xff”)
```

```
                if release:
                    self.cs(1)
                    self.spi.write(b”\xff”)
                return response

        # timeout
        self.cs(1)
        self.spi.write(b”\xff”)
        return -1

    def readinto(self, buf):
        self.cs(0)

        # read until start byte (0xff)
        for i in range(_CMD_TIMEOUT):
            self.spi.readinto(self.tokenbuf, 0xFF)
            if self.tokenbuf[0] == _TOKEN_DATA:
                break
            time.sleep_ms(1)
        else:
            self.cs(1)
            raise OSError(”timeout waiting for response”)

        # read data
        mv = self.dummybuf_memoryview
        if len(buf) != len(mv):
            mv = mv[: len(buf)]
        self.spi.write_readinto(mv, buf)

        # read checksum
        self.spi.write(b”\xff”)
        self.spi.write(b”\xff”)

        self.cs(1)
        self.spi.write(b”\xff”)

    def write(self, token, buf):
        self.cs(0)

        # send: start of block, data, checksum
        self.spi.read(1, token)
        self.spi.write(buf)
        self.spi.write(b”\xff”)
        self.spi.write(b”\xff”)

        # check the response
        if (self.spi.read(1, 0xFF)[0] & 0x1F) != 0x05:
            self.cs(1)
            self.spi.write(b”\xff”)
            return

        # wait for write to finish
        while self.spi.read(1, 0xFF)[0] == 0:
            pass

        self.cs(1)
        self.spi.write(b”\xff”)

    def write_token(self, token):
        self.cs(0)
        self.spi.read(1, token)
        self.spi.write(b”\xff”)
        # wait for write to finish
        while self.spi.read(1, 0xFF)[0] == 0x00:
            pass

        self.cs(1)
        self.spi.write(b”\xff”)

    def readblocks(self, block_num, buf):
        nblocks = len(buf) // 512
        assert nblocks and not len(buf) % 512, ”Buffer length is invalid”
        if nblocks == 1:
```

```
            # CMD17: set read address for single block
            if self.cmd(17, block_num * self.cdv, 0, release=False) != 0:
                # release the card
                self.cs(1)
                raise OSError(5)  # EIO
            # receive the data and release card
            self.readinto(buf)
        else:
            # CMD18: set read address for multiple blocks
            if self.cmd(18, block_num * self.cdv, 0, release=False) != 0:
                # release the card
                self.cs(1)
                raise OSError(5)  # EIO
            offset = 0
            mv = memoryview(buf)
            while nblocks:
                # receive the data and release card
                self.readinto(mv[offset : offset + 512])
                offset += 512
                nblocks -= 1
            if self.cmd(12, 0, 0xFF, skip1=True):
                raise OSError(5)  # EIO

    def writeblocks(self, block_num, buf):
        nblocks, err = divmod(len(buf), 512)
        assert nblocks and not err, "Buffer length is invalid"
        if nblocks == 1:
            # CMD24: set write address for single block
            if self.cmd(24, block_num * self.cdv, 0) != 0:
                raise OSError(5)  # EIO

            # send the data
            self.write(_TOKEN_DATA, buf)
        else:
            # CMD25: set write address for first block
            if self.cmd(25, block_num * self.cdv, 0) != 0:
                raise OSError(5)  # EIO
            # send the data
            offset = 0
            mv = memoryview(buf)
            while nblocks:
                self.write(_TOKEN_CMD25, mv[offset : offset + 512])
                offset += 512
                nblocks -= 1
            self.write_token(_TOKEN_STOP_TRAN)

    def ioctl(self, op, arg):
        if op == 4:  # get number of blocks
            return self.sectors
```

D.6. testGPS.py

```
from machine import UART, Pin
import utime
Link = "http://www.google.com/maps/place/"
loc = ""

def update_serial():
    while gps.any()>0:
        response = gps.read(1)
        print(response)



gps = UART(1, baudrate = 9600, tx=Pin(8), rx=Pin(9))
location = bytes()
gps.write(b'AT+CGPSPWR=1\r')
utime.sleep(0.1)
update_serial()
# gps.write(b'AT+CGPSSTATUS?\r')
# utime.sleep(0.1)
# update_serial()
# gps.write(b'AT+CGPSINF=0\r')
```

```
# utime.sleep(0.1)
# update_serial()
# gps.write(b'AT+CGPSINF=32\r')
# utime.sleep(0.1)
# update_serial()




def prepare_message(location):
    first_comma = location.index(',')
    second_comma = location.index(',', first_comma + 1)
    third_comma = location.index(',', second_comma+1)
    fourth_comma = location.index(',', third_comma+1)
    fifth_comma = location.index(',', fourth_comma+1)
    Longitude = ''
    iterator = third_comma;
    while iterator < fourth_comma:
        iterator = iterator + 1
        Longitude = Longitude + location[iterator];
    iterator = fourth_comma;
    Latitude = ''
    while iterator < fifth_comma-1:
        iterator = iterator + 1
        Latitude = Latitude + location[iterator]
    print('Latitude = ', Latitude)
    print('Longitude = ',Longitude)
    Link = "http://www.google.com/maps/place/" + Longitude+ Latitude
    print(Link)




gps.write(b'AT+CGNSINF\r')
utime.sleep(0.1)
while gps.any()>0:
    location += gps.read(1)

loc = location.decode('utf-8')
print('location = ', loc)
# loc = loc + str(location)
prepare_message(loc)

print(loc)
```

D.7. testSDcard.py

```
# import sdcard
# import machine
# import uos
# sd_spi = machine.SPI(1, sck = machine.Pin(10, machine.Pin.OUT), mosi = machine.Pin(11, machine.Pin.OUT), miso = machine.Pin(12, machine.Pin.OUT))
# sd = sdcard.SDCard(sd_spi, machine.Pin(9))
# uos.mount(sd, "/sd")
#
# print("Size: {} MB" .format(sd.sectors/2048))
# print(uos.listdir("/sd"))

# Test for sdcard block protocol
# Peter hinch 30th Jan 2016
import os, sdcard, machine
import time

def sdtest():
    spi = machine.SPI(1)
    spi.init()  # Ensure right baudrate
    sd_spi = machine.SPI(1, sck = machine.Pin(10, machine.Pin.OUT), mosi = machine.Pin(11, machine.Pin.OUT), miso = machine.Pin(12, machine.Pin.OUT))
    #sd = sdcard.SDCard(spi, machine.Pin.board.X21)  # Compatible with PCB
    sd = sdcard.SDCard(sd_spi, machine.Pin(9))
    vfs = os.VfsFat(sd)
    os.mount(vfs, "/fc")
```

```
    print("Filesystem check")
    print(os.listdir("/fc"))

    line = "abcdefghijklmnopqrstuvwxyz\n"
    lines = line * 200  # 5400 chars
    short = "1234567890\n"

    fn = "/sd/rats.txt"
    print()
    print("Multiple block read/write")
    with open(fn, "w") as f:
        n = f.write(lines)
        print(n, "bytes written")
#         n = f.write(short)
#         print(n, "bytes written")
#         n = f.write(lines)
#         print(n, "bytes written")

    with open(fn, "r") as f:
        result1 = f.read()
        print(len(result1), "bytes read")
#
#     fn = "/fc/rats1.txt"
#     print()
#     print("Single block read/write")
#     with open(fn, "w") as f:
#         n = f.write(short)  # one block
#         print(n, "bytes written")
#
#     with open(fn, "r") as f:
#         result2 = f.read()
#         print(len(result2), "bytes read")
#
#     os.umount("/fc")
#
#     print()
#     print("Verifying data read back")
#     success = True
#     if result1 == "".join((lines, short, lines)):
#         print("Large file Pass")
#     else:
#         print("Large file Fail")
#         success = False
#     if result2 == short:
#         print("Small file Pass")
#     else:
#         print("Small file Fail")
#         success = False
#     print()
#     print("Tests", "passed" if success else "failed")

sdtest()
```

D.8. testTimer.py

```
# from machine import Pin, Timer
#
# led = Pin(25, Pin.OUT)
# tim = Timer()
# def tick(timer):
#     global led
#     led.toggle()
#     print("lala")
#     print("baba")
# tim.init(freq=1, mode=Timer.PERIODIC, callback=tick)
#
# while True:
#     count = 100000
from rp2 import PIO, StateMachine, asm_pio
from machine import Pin, Timer
from machine import Pin
#import time
from machine import Pin, ADC    # Import Pins and ADCs
```

```
import mpu6050                  # Import library for gyroscope and accelerometer
from mq9 import MQ              # Import library for gas sensor
import utime                    # Import time library
from ds3231 import ds3231       # Import real time module library
#mq = MQ()               # Define gas sensor object.
time = ds3231(1,15,14)  # Define real time object connected to i2c1 with address 0x68
tim = Timer()
mpu = mpu6050.MPU6050() # Define gyroscope object. Connected to i2c1 with address 0x69

# Define pins
sel = machine.Pin (2, machine.Pin.OUT) # use pin 2 as slector to select between gas and sound
sound = ADC(28)
# Define Varaibles
conversion_factor = 3.3/(65536)       # To read the correct value from the ADC

gyroData=[]
gasData=[]
soundData=[]
timeData=[]
Data = []
Data_to_send = []
iteration = 10
start_measurement = False
send = False
gx_previous = 0
gy_previous = 0
gz_previous = 0
gyrox_previous = 0
gyroy_previous = 0
gyroz_previous = 0
previous_time = 0
counter =0
sound = ADC(28)
# Define Varaibles
conversion_factor = 3.3/(65536)       # To read the correct value from the ADC
def tick(timer):
    global start_measurement
    global counter
    global send
    counter = counter +1
    if counter == 10:
        send = True
        counter = 0
    else:
        send = False

    start_measurement = True
tim.init(freq=1, mode=Timer.PERIODIC, callback=tick)

# @asm_pio(set_init=PIO.OUT_LOW)
# def measure():
#     global start_measurement
#     start_measurement = True
#     print(start_measurement)
#
# sm1 = StateMachine(1, measure, freq=10000, set_base=Pin(2))
# sm1.active(1)

while True:
    if start_measurement == True:
        t = time.read_time()
        current_time = str("%02x" %t[0])
        print(current_time)
        start_measurement = False
        g=mpu.readData()
        gx_current  = g.Gx
        gy_current  = g.Gy
        gz_current  = g.Gz
        gyrox_current = g.Gyrox
        gyroy_current = g.Gyroy
        gyroz_current = g.Gyroz
        if ((-0.5 <= gx_current - gx_previous >= 0.5) or (-0.5 <= gy_current - gy_previous >=
     0.5) or (-0.5 <= gz_current - gz_previous >= 0.5)):
```

```
            Data.append(str("%02x" %t[2])+"/"+str("%02x" %t[1]) + "/" + str("%02x" %t[0]) + "M")
        if ((-0.5 <= gyrox_current - gyrox_previous >= 0.5) or (-0.5 <= gyroy_current - gyroy_previous >= 0.5) or (-0.5 <= gyroz_current - gyroz_previous >= 0.5)):
            Data.append(str("%02x" %t[2])+"/"+str("%02x" %t[1]) + "/" + str("%02x" %t[0]) + "M")
        #print(Data)
        if ((send == True) and ((current_time) != (previous_time)) ):
            Data_to_send.append(str("%02x" %t[0]) +"/" + str("%x" %len(Data)) + "M")

        gx_previous = gx_current
        gy_previous = gy_current
        gz_previous = gz_current
        gyrox_previous = gyrox_current
        gyroy_previous = gyroy_current
        gyroz_previous = gyroz_current
        previous_time = current_time
        print(previous_time)
        print(send)
        print(counter)
        print(Data_to_send)
       # g.Gx,g.Gy,g.Gz,g.Gyrox,g.Gyroy,g.Gyroz,g.Temperature

#         utime.sleep_ms(100)
#         sel.value(0)
#         #perc = mq.MQPercentage()
#         utime.sleep_ms(100)
#         sel.value(1)
#       #print("CO: %g ppm, Smoke: %g ppm %g sound" % (perc["CO"], perc["SMOKE"], sound.read_u16() *conversion_factor))
#
#       #print(sound.read_u16()*conversion_factor)
#       #utime.sleep(0.1)
#         t = time.read_time()
#         utime.sleep(0.1)
```

D.9. Basestation.py

```
import sys
import serial
from oob_parser import uartParserSDK
from graphUtilities import *
from gl_classes import GLTextItem


import random
import numpy as np
import time
import math
import struct
import os
import csv
configFileNaam = 'AOP_6m_default.cfg'
ompileGui = 0

class BaseStation():
    def __init__(self,s_height,az_tilt,elev_tilt, persistentFramesInput):

        if (1): #set to 1 to save terminal output to logFile, set 0 to show terminal output
            ts = time.localtime()
            terminalFileName = str('logData/logfile_'+ str(ts[2]) + str(ts[1]) + str(ts[0]) + '_' + str(ts[3]) + str(ts[4]) +'.txt')
            #sys.stdout = open(terminalFileName, 'w')

        print('Python is ', struct.calcsize("P")*8, ' bit')
        print('Python version: ', sys.version_info)
        self.frameTime = 50
        self.graphFin = 1
        self.hGraphFin = 1
        self.threeD = 1
        self.lastFramePoints = np.zeros((5,1))
        self.plotTargets = 1
        self.frameNum = 0
```

```
        self.lastTID = []
        self.profile = {'startFreq': 60.25, 'numLoops': 64, 'numTx': 3, 'sensorHeight':3, 'maxRange':10, 'az_tilt':0, 'elev_tilt':0}
        self.lastFrameHadTargets = False
        self.sensorHeight = 1.5
        self.numFrameAvg = 20
        self.configSent = 0
        self.previousFirstZ = -1
        self.yzFlip = 0
        self.configFileName = 'AOP_6m_default.cfg'
        #self.fallDetData()
        self.configType = '3D People Counting'
        self.uart = '/dev/ttyUSB1'
        self.data = '/dev/ttyUSB0'
        self.data_recorded = []
    def updateSensorPosition(self):
        try:
            float(self.s_height)
            float(self.az_tilt)
            float(self.elev_tilt)
        except:
            print("fail to update")
            return
        command = "sensorPosition " + self.s_height + " " + self.az_tilt + " " + self.elev_tilt + " \n"
        self.cThread = sendCommandThread(self.parser,command)
        self.cThread.start(priority=QThread.HighestPriority-2)
        self.gz.translate(dx=0,dy=0,dz=self.profile['sensorHeight'])
        self.profile['sensorHeight'] = float(self.s_height)
        self.gz.translate(dx=0,dy=0,dz=-self.profile['sensorHeight'])

    def updateGraph(self, parsedData):
        updateStart = int(round(time.time()*1000))
        self.useFilter = 0
        classifierOutput = []
        pointCloud = parsedData[0]
        targets = parsedData[1]
        indexes = parsedData[2]
        numPoints = parsedData[3]
        numTargets = parsedData[4]
        self.frameNum = parsedData[5]
        fail = parsedData[6]
        classifierOutput = parsedData[7]
        #print("indexes: ", indexes)
        fallDetEn = 0
        indicesIn = []
        #pass target XYZ vals and rotate due to elevation tilt angle (rotX uses Euler rotation around X axis)
        print('elev_tilt = ',self.profile['elev_tilt'])
        print('targets = ',targets)
        rotTargetDataX,rotTargetDataY,rotTargetDataZ = rotX (targets[1],targets[2],targets[3],-1*self.profile['elev_tilt'])
        print('Rotated Data TID,X,Y = ' +str(rotTargetDataX)+', '+str(rotTargetDataY)+', '+str(rotTargetDataZ))
        targets[1] = rotTargetDataX
        targets[2] = rotTargetDataY
        targets[3] = rotTargetDataZ

        #pass pointCloud XYZ vals and rotate due to elevation tilt angle (rotX uses Euler rotation around X axis)
        for i in range(numPoints):
            print('graph point cloud pt = ',pointCloud[:,i])
            print('graph point cloud Y = ',pointCloud[1][i])
            print('graph point cloud Z = ',pointCloud[2][i])
            rotPointDataX,rotPointDataY,rotPointDataZ = rotX ([pointCloud[0,i]],[pointCloud[1,i]],[pointCloud[2,i]],-1*self.profile['elev_tilt'])
            print('graph point cloud rotated pt = ',rotPointDataX,rotPointDataY,rotPointDataZ)
            print('graph point cloud Y = ',pointCloud[1][i])
            print('graph point cloud Z = ',pointCloud[2][i])
            pointCloud[0,i] = rotPointDataX
            pointCloud[1,i] = rotPointDataY
            pointCloud[2,i] = rotPointDataZ
```

```
        if (fail != 1):
            #left side
            #pointstr = 'Points: '+str(numPoints)
            #targetstr = 'Targets: '+str(numTargets)
            #self.numPointsDisplay.setText(pointstr)
            #self.numTargetsDisplay.setText(targetstr)
            #right side fall detection
            peopleStr = 'Number of Detected People: '+str(numTargets)
            if (numTargets == 0):
                print('Fall Detection Disabled - No People Detected')
            elif (numTargets == 1):
                print('Fall Detection Enabled')
                fallDetEn = 1
            elif (numTargets > 1):
                print('Fall Detected Disabled - Too Many People')
            #self.numDetPeople.setText(peopleStr)
            #self.fallDetEnabled.setText(fdestr)
        if (len(targets) < 13):
            targets = []
            classifierOutput = []
        if (fail):
            return
        #check for mounting position
        if (self.yzFlip == 1):
            pointCloud[[1, 2]] = pointCloud[[2, 1]]
            pointCloud[2,:] = -1*pointCloud[2,:]
            targets[[2,3]] = targets[[3,2]]
            targets[3,:] = -1*targets[3,:]

        #remove static points
        if (self.configType == '3D People Counting' or self.configType == 'Capon3DAOP' or self.configType == 'Sense and Detect HVAC Control'):
            if (not self.staticclutter.isChecked()):
                statics = np.where(pointCloud[3,:] == 0)
                try:
                    firstZ = statics[0][0]
                    numPoints = firstZ
                    pointCloud = pointCloud[:,:firstZ]
                    indexes = indexes[:,:self.previousFirstZ]
                    self.previousFirstZ = firstZ
                except:
                    firstZ = -1
        #point cloud persistence
        fNum = self.frameNum%10
        if (numPoints):
            self.previousCloud[:5,:numPoints,fNum] = pointCloud[:5,:numPoints]
            self.previousCloud[5,:len(indexes),fNum] = indexes
        self.previousPointCount[fNum]=numPoints
        #plotting 3D - get correct point cloud (persistent points and synchronize the frame)
        if (self.configType == 'SDK3xPeopleCount'):
            pointIn = pointCloud
        else:
            totalPoints = 0
            persistentFrames = int(self.persistentFramesInput)
            #allocate new array for all the points
            for i in range(1,persistentFrames+1):
                totalPoints += self.previousPointCount[fNum-i]
            pointIn = np.zeros((5,int(totalPoints)))
            indicesIn = np.ones((1, int(totalPoints)))*255
            totalPoints = 0
            #fill array for indices and points
            for i in range(1,persistentFrames+1):
                prevCount = int(self.previousPointCount[fNum-i])
                pointIn[:,totalPoints:totalPoints+prevCount] = self.previousCloud[:5,:prevCount,fNum-i]
                if (numTargets > 0):
                    indicesIn[0,totalPoints:totalPoints+prevCount] = self.previousCloud[5,:prevCount,fNum-i]
                totalPoints+=prevCount

        #height plotting - only if 3D plot is good to go
        #first loop is instantaneous absolute height, relative height, length, and width
        if (self.configType == '3D People Counting'):
```

```
            pointIn = self.previousCloud[:,:int(self.previousPointCount[fNum-1]),fNum-1]
        elif (self.configType == ’Long Range People Detection’):
            pointIn = self.previousCloud[:,:int(self.previousPointCount[fNum]),fNum]
        fNum = self.frameNum%100
        for t in range(numTargets):
            tid = int(targets[t,0])
            print(”TID: ”, tid)
            tIndices = np.where(np.array(indexes) == tid)
            print(”Indexex: ”, np.size(indexes),” , pointIn: ”,  np.size(pointIn,1))
            if (np.size(tIndices) and np.size(pointIn,1) == np.size(indexes)):
                #print(”indices statement”)
                tPoints = np.take(pointIn, tIndices, 1)
                self.targetSize[0,tid,fNum] = np.amax(tPoints[2,0,:]) + self.sensorHeight #absolute height
                self.targetSize[1,tid,fNum] = np.amax(tPoints[2,0,:]) - np.amin(tPoints[2,:]) #relative height
                self.targetSize[2,tid,fNum] = np.amax(tPoints[1,0,:]) - np.amin(tPoints[1,:]) #length
                self.targetSize[3,tid,fNum] = np.amax(tPoints[0,0,:]) - np.amin(tPoints[0,:]) #width
                if tid in self.lastTID:
                    #print(”lastTID”)
                    self.targetSize[4,tid,0] = self.targetSize[4,tid,0]+1
                    age = self.targetSize[4,tid,0] #age
                    a = 1/self.numFrameAvg*self.targetSize[0,tid,fNum]
                    b = ((self.numFrameAvg-1)/self.numFrameAvg)*self.targetSize[5,tid,(fNum-1)%100]
                    c = a + b
                    print(’a: ’,a,’ b: ’, b,’ c: ’,c)
                    self.targetSize[5,tid,fNum]= (1/self.numFrameAvg*self.targetSize[0,tid,fNum])+((self.numFrameAvg-1)/self.numFrameAvg)*self.targetSize[5,tid,(fNum-1)%100] #avg height over 10 frames
                    #need 2 seconds to get accurate height
                    if(age>40):
                        self.targetSize[6,tid,fNum]= self.targetSize[5,tid,fNum]-self.targetSize[5,tid,(fNum-10)%100] #delta height after 10 frames
                        if (self.targetSize[6,tid,fNum] < self.fallThresh and fallDetEn):
                            print(’Fallen!’)
                            #self.fallPic.setPixmap(self.fallingPicture)
                            if (self.fallResetTimerOn == 0):
                                self.fallResetTimerOn = 1
                                self.fallTimer.start(5000) #5 second timer
                    else:
                        self.targetSize[6,tid,fNum] = 0
                else:
                    self.targetSize[4,tid,0] = 1
                    self.targetSize[5,tid,fNum]= self.targetSize[0,tid,fNum]
                    self.targetSize[6,tid,fNum]=0
            #nothing detected use values from last frame
            else:
                self.targetSize[0,tid,fNum]=self.targetSize[0,tid,fNum-1]
                self.targetSize[1,tid,fNum]=self.targetSize[1,tid,fNum-1]
                self.targetSize[2,tid,fNum]=self.targetSize[2,tid,fNum-1]
                self.targetSize[3,tid,fNum]=self.targetSize[3,tid,fNum-1]
                self.targetSize[4,tid,fNum]=self.targetSize[4,tid,fNum-1]
                self.targetSize[5,tid,fNum]=self.targetSize[5,tid,fNum-1]
                self.targetSize[6,tid,fNum]=self.targetSize[6,tid,fNum-1]

        #state tracking
        if (numTargets > 0):
            self.lastFrameHadTargets = True
        else:
            self.lastFrameHadTargets = False
        if (numTargets):
            self.lastTID = targets[0,:]
        else:
            self.lastTID = []

    def graphDone(self):
        plotend = int(round(time.time()*1000))
        plotime = plotend - self.plotstart
        try:
            if (self.frameNum > 1):
```

```python
                self.averagePlot = (plotime*1/self.frameNum) + (self.averagePlot*(self.frameNum-1)/(self.frameNum))
            else:
                self.averagePlot = plotime
        except:
            self.averagePlot = plotime
        self.graphFin = 1
        pltstr = 'Average Plot time: '+str(plotime)[:5] + ' ms'
        fnstr = 'Frame: '+str(self.frameNum)
        print(fnstr)
        print(pltstr)

    def connectCom(self):
        #get parser
        self.parser = uartParserSDK(type=self.configType)
        self.parser.frameTime = self.frameTime
        print('Parser type: ',self.configType)
        #init threads and timers
        #self.uart_thread = parseUartThread(self.parser)
        #if (self.configType != 'Replay'):
            #self.uart_thread.fin.connect(self.parseData)
        #self.uart_thread.fin.connect(self.updateGraph)
        #self.parseTimer = QTimer()
        #self.parseTimer.setSingleShot(False)
        #self.parseTimer.timeout.connect(self.parseData)
        try:
            #uart = "COM"+ self.uartCom.text()       #deb_gp
            #data = "COM"+ self.dataCom.text()       #deb_gp
#TODO: find the serial ports automatically.
            self.parser.connectComPorts(self.uart, self.data)
            print('Connected from main')     #deb_gp
            #print('Disconnect')    #deb_gp
#TODO: create the disconnect button action
        except Exception as e:
            print (e)
            print('Unable to Connect')
        if (self.configType == "Replay"):
            self.connectStatus = ('Replay')
        if (self.configType == "Long Range People Detection"):
            self.frameTime = 400
#
# Select and parse the configuration file
# TODO select the cfgfile automatically based on the profile.


    def sendCfg(self):
        try:
            if (self.configType!= "Replay"):
                self.parser.sendCfg(self.cfg)
                self.configSent = 1
            self.parseTimer.start(self.frameTime)
        except Exception as e:
            print(e)
            print ('No cfg file selected!')

    def serialConfig(self, configFileName):

        global CLIport
        global Dataport
    # Open the serial ports for the configuration and the data ports

    # Raspberry pi
        CLIport = serial.Serial('/dev/ttyUSB0', 115200)
        Dataport = serial.Serial('/dev/ttyUSB1', 921600)

    # Windows
    #CLIport = serial.Serial('COM3', 115200)
    #Dataport = serial.Serial('COM4', 921600)

    # Read the configuration file and send it to the board
        config = [line.rstrip('\r\n') for line in open(configFileName)]
        for i in config:
            CLIport.write((i+'\n').encode())
```

```
            print(i)
            time.sleep(0.01)

        return CLIport, Dataport

    def parseCfg(self):
        cfg_file = open(self.configFileName)
        self.cfg = cfg_file.readlines()
        counter = 0
        chirpCount = 0
        for line in self.cfg:
            args = line.split()
            if (len(args) > 0):
                if (args[0] == 'cfarCfg'):
                    zy = 4
                    #self.cfarConfig = {args[10], args[11], '1'}
                elif (args[0] == 'AllocationParam'):
                    zy=3
                    #self.allocConfig = tuple(args[1:6])
                elif (args[0] == 'GatingParam'):
                    zy=2
                    #self.gatingConfig = tuple(args[1:4])
                elif (args[0] == 'SceneryParam' or args[0] == 'boundaryBox'):
                    self.boundaryLine = counter
                    self.profile['leftX'] = float(args[1])
                    self.profile['rightX'] = float(args[2])
                    self.profile['nearY'] = float(args[3])
                    self.profile['farY'] = float(args[4])
                    if (self.configType== '3D People Counting'):
                        self.profile['bottomZ'] = float(args[5])
                        self.profile['topZ'] = float(args[6])
                    else:
                        self.profile['bottomZ'] = float(-3)
                        self.profile['topZ'] = float(3)
                    #self.setBoundaryTextVals(self.profile)
                    #self.boundaryBoxes[0]['checkEnable'].setChecked(True)
                elif (args[0] == 'staticBoundaryBox'):
                    self.staticLine = counter
                elif (args[0] == 'profileCfg'):
                    self.profile['startFreq'] = float(args[2])
                    self.profile['idle'] = float(args[3])
                    self.profile['adcStart'] = float(args[4])
                    self.profile['rampEnd'] = float(args[5])
                    self.profile['slope'] = float(args[8])
                    self.profile['samples'] = float(args[10])
                    self.profile['sampleRate'] = float(args[11])
                    print(self.profile)
                elif (args[0] == 'frameCfg'):
                    self.profile['numLoops'] = float(args[3])
                    self.profile['numTx'] = float(args[2])+1
                elif (args[0] == 'chirpCfg'):
                    chirpCount += 1
                elif (args[0] == 'sensorPosition'):
                    self.profile['sensorHeight'] = float(args[1])
                    self.profile['az_tilt'] = float(args[2])
                    self.profile['elev_tilt'] = float(args[3])
            counter += 1
        self.profile['maxRange'] = self.profile['sampleRate']*1e3*0.9*3e8/(2*self.profile['slope']*1e12)
        #update boundary box
        #self.drawBoundaryGrid(self.profile['maxRange']) #2D legacy version
        #self.gz.translate(0, 0, 3-self.profile['sensorHeight']) #reposition the ground level to be at sensor height
        #self.changeBoundaryBox() #redraw bbox from cfg file values
        #update chirp table values
        bw = self.profile['samples']/(self.profile['sampleRate']*1e3)*self.profile['slope']*1e12
        rangeRes = 3e8/(2*bw)
        Tc = (self.profile['idle']*1e-6 + self.profile['rampEnd']*1e-6)*chirpCount
        lda = 3e8/(self.profile['startFreq']*1e9)
        maxVelocity = lda/(4*Tc)
        velocityRes = lda/(2*Tc*self.profile['numLoops']*self.profile['numTx'])
        #self.configTable.setItem(1,1,QTableWidgetItem(str(self.profile['maxRange'])[:5]))
```

```
        #self.configTable.setItem(2,1,QTableWidgetItem(str(rangeRes)[:5]))
        #self.configTable.setItem(3,1,QTableWidgetItem(str(maxVelocity)[:5]))
        #self.configTable.setItem(4,1,QTableWidgetItem(str(velocityRes)[:5]))
        #update sensor position
        #print(str(self.profile['az_tilt']))
        #print(str(self.profile['elev_tilt']))
        #print(str(self.profile['sensorHeight']))

    def connectBase(self):
        self.parser = uartParserSDK(type=self.configType)
        self.parser.frameTime = 50
        #self.parseTimer.timeout.connect(self.parseData)
        self.parser.connectComPorts('/dev/ttyUSB1','/dev/ttyUSB0')

    def readData(self, data):
        #data_rec = self.parser.tlvHeader(data)
        data_read = self.parser.readAndParseUart(data)
        return data_read [3]





test = BaseStation(1,1,1,4)
test.connectBase()
#test.connectCom()
test.serialConfig(configFileNaam)
test.parseCfg()

#test.sendCfg()
#test.updateGraph()

#with open('output.csv', 'w') as csvfile:
#    while True:
#        readBuffer = Dataport.readline(Dataport.in_waiting)
#        print(readBuffer)
#        writer = csv.DictWriter(csvfile, readBuffer)

f = open('output.csv', 'w')
n = 100
while True: #n >0:
    readBuffer = Dataport.readline(Dataport.in_waiting)
    t = time.localtime()
    current_time = time.strftime("%H:%M:%S", t)
    spamWriter = csv.writer(f, delimiter=' ')
    print(readBuffer)
    print(current_time)
    spamWriter.writerow([current_time] + [readBuffer])
    n -= 1
f.close()
```

D.10. oob parser.py

```
import struct
import sys
import serial
import binascii
import time
import numpy as np
import math

from graphUtilities import rotX

#Initialize this Class to create a UART Parser. Initialization takes one argument:
# 1: String Lab_Type - These can be:
#    a. 3D People Counting
#    b. SDK Out of Box Demo
#    c. Long Range People Detection
#    d. Indoor False Detection Mitigation
#    e. (Legacy): Overhead People Counting
#    f. (Legacy) 2D People Counting
# Default is (f). Once initialize, call connectComPorts(self, UartComPort, DataComPort) to
    connect to device com ports.
```

```
# Then call readAndParseUart() to read one frame of data from the device. The gui this is packaged with calls this every frame period.
# readAndParseUart() will return all radar detection and tracking information.
class uartParserSDK():
    def __init__(self,type=’(Legacy) 2D People Counting’):
        self.headerLength = 52
        self.magicWord = 0x708050603040102
        self.threeD = 0
        self.ifdm = 0
        self.replay = 0
        self.SDK3xPointCloud = 0
        self.SDK3xPC = 0
        self.capon3D = 0
        self.aop = 0
        self.maxPoints = 1150
        if (type==’(Legacy): Overhead People Counting’):
            self.threeD = 1
        elif (type==’Sense and Detect HVAC Control’):
            self.ifdm = 1
        elif (type==’Replay’): # unused
            self.replay = 1
        elif (type==”SDK Out of Box Demo”):
            self.SDK3xPointCloud = 1
        elif (type==”Long Range People Detection”):
            self.SDK3xPC = 1
        elif (type==’3D People Counting’):
            self.capon3D = 1
        elif (type == ’Capon3DAOP’): #unused
            self.capon3D = 1
            self.aop = 1
        #data storage
        self.pcPolar = np.zeros((5,self.maxPoints))
        self.pcBufPing = np.zeros((5,self.maxPoints))
        self.numDetectedObj = 0
        self.targetBufPing = np.ones((10,20))*-1
        self.indexBufPing = np.zeros((1,self.maxPoints))
        self.classifierOutput = []
        self.frameNum = 0
        self.missedFrames = 0
        self.byteData = bytes(1)
        self.oldData = []
        self.indexes = []
        self.numDetectedTarget = 0
        self.fail = 0
        self.unique = []
        self.savedData = []
        self.saveNum = 0
        self.saveNumTxt = 0
        self.replayData = []
        self.startTimeLast = 0
        self.saveReplay = 0
        self.savefHist = 0
        self.saveBinary = 0
        self.saveTextFile = 0
        self.fHistRT = np.empty((100,1), dtype=np.object)
        self.plotDimension = 0
        self.getUnique = 0
        self.CaponEC = 0

        self.printVerbosity = 0 #set 0 for limited logFile printing, 1 for more logging

        if (self.capon3D):
            #3D people counting format
            #[frame #][header,pt cloud data,target info]
            #[][header][magic, version, packetLength, platform, frameNum, subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum]
            #[][pt cloud][pt index][#elev, azim, doppler, range, snr]
            #[][target][Target #][TID,x,y,z,vx,vy,vz,ax,ay,az]
            self.textStructCapon3D = np.zeros(1000*3*self.maxPoints*10).reshape((1000,3,self.maxPoints,10))#[frame #][header,pt cloud data,target info]

        if (self.ifdm):
            #Sense and direct format
```

```
            #[frame #][header,pt cloud data,target info]
            #[][header][magic, version, platform, timestamp, packetLength, frameNum, subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum]
            #[][pt cloud][pt index][#range, azim, doppler, snr]
            #[][target][Target #][TID,x,y,vx,vy,ax,ay]
            self.textStruct2D = np.zeros(1000*3*self.maxPoints*7).reshape((1000,3,self.maxPoints,7))#[frame #][header,pt cloud data,target info]

#below funtions are used for converting output of labs that do not match SDK 3.x DPIF output
    #convert 2D polar People Counting to 3D Cartesian
    def polar2Cart(self):
        self.pcBufPing = np.empty((5,self.numDetectedObj))
        for n in range(0, self.numDetectedObj):
            self.pcBufPing[1,n] = self.pcPolar[0,n]*math.cos(self.pcPolar[1,n])  #y
            self.pcBufPing[0,n] = self.pcPolar[0,n]*math.sin(self.pcPolar[1,n])  #x
        self.pcBufPing[3,:] = self.pcPolar[2,0:self.numDetectedObj] #doppler
        self.pcBufPing[4,:] = self.pcPolar[3,0:self.numDetectedObj] #snr
        self.pcBufPing[2,:self.numDetectedObj] = 0                                 #Z is zero in 2D case

    #convert 3D people counting polar to 3D cartesian
    def polar2Cart3D(self):
        self.pcBufPing = np.empty((5,self.numDetectedObj))
        for n in range(0, self.numDetectedObj):
            self.pcBufPing[2,n] = self.pcPolar[0,n]*math.sin(self.pcPolar[2,n]) #z
            self.pcBufPing[0,n] = self.pcPolar[0,n]*math.cos(self.pcPolar[2,n])*math.sin(self.pcPolar[1,n]) #x
            self.pcBufPing[1,n] = self.pcPolar[0,n]*math.cos(self.pcPolar[2,n])*math.cos(self.pcPolar[1,n]) #y
        self.pcBufPing[3,:] = self.pcPolar[3,0:self.numDetectedObj] #doppler
        self.pcBufPing[4,:] = self.pcPolar[4,0:self.numDetectedObj] #snr
        #print(self.pcBufPing[:,:10])

    #decode People Counting TLV Header
    def tlvHeaderDecode(self, data):
        #print(len(data))
        tlvType, tlvLength = struct.unpack('2I', data)
        return tlvType, tlvLength

    #decode People Counting Point Cloud TLV
    def parseDetectedObjects(self, data, tlvLength):
        objStruct = '4f'
        objSize = struct.calcsize(objStruct)
        self.numDetectedObj = int((tlvLength)/16)
        for i in range(self.numDetectedObj):
            try:
                self.pcPolar[0,i], self.pcPolar[1,i], self.pcPolar[2,i], self.pcPolar[3,i] = struct.unpack(objStruct,data[:objSize])
                data = data[16:]
            except:
                self.numDectedObj = i
                break
        self.polar2Cart()

    #decode IFDM point Cloud TLV
    def parseDetectedObjectsIFDM(self, data, tlvLength):
        pUnitStruct = '4f'
        pUnitSize = struct.calcsize(pUnitStruct)
        pUnit = struct.unpack(pUnitStruct, data[:pUnitSize])
        data = data[pUnitSize:]
        objStruct = '2B2h'
        objSize = struct.calcsize(objStruct)
        self.numDetectedObj = int((tlvLength-16)/objSize)
        #print('Parsed Points: ', self.numDetectedObj)
        for i in range(self.numDetectedObj):
            try:
                az, doppler, ran, snr = struct.unpack(objStruct, data[:objSize])
                data = data[objSize:]
                #get range, azimuth, doppler, snr
                self.pcPolar[0,i] = ran*pUnit[2]           #range
                if (az >= 128):
                    az -= 256
                self.pcPolar[1,i] = math.radians(az*pUnit[0])   #azimuth
```

```
                self.pcPolar[2,i] = doppler*pUnit[1]       #doppler
                self.pcPolar[3,i] = snr*pUnit[3]           #snr

                #Sense and direct format
                #[frame #][header,pt cloud data,target info]
                #[][header][magic, version, platform, timestamp, packetLength, frameNum,
    subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum]
                #[][pt cloud][pt index][#range, azim, doppler, snr]
                #[][target][Target #][TID,x,y,vx,vy,ax,ay]
                self.textStruct2D[self.frameNum%1000,1,i,0] = self.pcPolar[0,i] #range
                self.textStruct2D[self.frameNum%1000,1,i,1] = self.pcPolar[1,i] #az
                self.textStruct2D[self.frameNum%1000,1,i,2] = self.pcPolar[2,i] #doppler
                self.textStruct2D[self.frameNum%1000,1,i,3] = self.pcPolar[3,i] #snr

            except:
                self.numDetectedObj = i
                break
        self.polar2Cart()

    #decode 3D People Counting Point Cloud TLV
    def parseDetectedObjects3D(self, data, tlvLength):
        objStruct = '5f'
        objSize = struct.calcsize(objStruct)
        self.numDetectedObj = int(tlvLength/20)
        for i in range(self.numDetectedObj):
            try:
                self.pcPolar[0,i], self.pcPolar[1,i], self.pcPolar[2,i], self.pcPolar[3,i],
    self.pcPolar[4,i] = struct.unpack(objStruct,data[:objSize])
                data = data[20:]
            except:
                self.numDectedObj = i
                print('failed to get point cloud')
                break
        self.polar2Cart3D()

    #support for Capoin 3D point cloud
    #decode Capon 3D point Cloud TLV
    def parseCapon3DPolar(self, data, tlvLength):
        pUnitStruct = '5f'  #elev, azim, doppler, range, snr
        pUnitSize = struct.calcsize(pUnitStruct)
        pUnit = struct.unpack(pUnitStruct, data[:pUnitSize])
        data = data[pUnitSize:]
        objStruct = '2bh2H' #2 int8, 1 int16, 2 uint16
        objSize = struct.calcsize(objStruct)
        self.numDetectedObj = int((tlvLength-pUnitSize)/objSize)
        #if (self.printVerbosity == 1):
        #print('Parsed Points: ', self.numDetectedObj)
        for i in range(self.numDetectedObj):
            try:
                elev, az, doppler, ran, snr = struct.unpack(objStruct, data[:objSize])
                #print(elev, az, doppler, ran, snr)
                data = data[objSize:]
                #get range, azimuth, doppler, snr
                self.pcPolar[0,i] = ran*pUnit[3]           #range
                if (az >= 128):
                    print ('Az greater than 127')
                    az -= 256
                if (elev >= 128):
                    print ('Elev greater than 127')
                    elev -= 256
                if (doppler >= 32768):
                    print ('Doppler greater than 32768')
                    doppler -= 65536
                self.pcPolar[1,i] = az*pUnit[1]  #azimuth
                self.pcPolar[2,i] = elev*pUnit[0] #elevation
                self.pcPolar[3,i] = doppler*pUnit[2]       #doppler
                self.pcPolar[4,i] = snr*pUnit[4]           #snr

                #add pt cloud data to textStructCapon3DCapon3D for text file printing
                #self.textStructCapon3DCapon3D[,,,] = [frame #][header,pt cloud data,target
    info]
                #[][pt cloud = 0][pt index][#elev, azim, doppler, range, snr]
                self.textStructCapon3D[self.frameNum%1000,1,i,0] = self.pcPolar[2,i] #elev
```

```
                self.textStructCapon3D[self.frameNum%1000,1,i,1] = self.pcPolar[1,i] #az
                self.textStructCapon3D[self.frameNum%1000,1,i,2] = self.pcPolar[3,i] #doppler
                self.textStructCapon3D[self.frameNum%1000,1,i,3] = self.pcPolar[0,i] #range
                self.textStructCapon3D[self.frameNum%1000,1,i,4] = self.pcPolar[4,i] #snr
            except:
                self.numDetectedObj = i
                print('Point Cloud TLV Parser Failed')
                break
        self.polar2Cart3D()

    #decode 2D People Counting Target List TLV
    def parseDetectedTracks(self, data, tlvLength):
        if (self.plotDimension):
            targetStruct = 'I8f9ff'
        else:
            targetStruct = 'I6f9ff'
        targetSize = struct.calcsize(targetStruct)
        self.numDetectedTarget = int(tlvLength/targetSize)
        targets = np.empty((13,self.numDetectedTarget))
        for i in range(self.numDetectedTarget):
            targetData = struct.unpack(targetStruct,data[:targetSize])
            targets[0,i]=int(targetData[0]) #TID
            targets[1:3,i]=targetData[1:3] #X,Y
            targets[3,i]=0 #Z=0
            targets[4:6,i]=targetData[3:5] #vX,Vy
            targets[6,i]=0#vZ=0
            targets[7:9,i]=targetData[5:7] #aX,aY
            targets[9,i]=0 #az=0
            if (self.plotDimension):
                targets[10:12,i]=targetData[7:9]
                targets[12,i]=1
            else:
                targets[10:12,i]=[0.75,0.75]
                targets[12,i]=1
            data = data[targetSize:]

            if (self.saveTextFile):
                self.textStruct2D[self.frameNum%1000,2,i,0] = targets[0,i] #TID
                self.textStruct2D[self.frameNum%1000,2,i,1] = targets[1,i] #x
                self.textStruct2D[self.frameNum%1000,2,i,2] = targets[2,i] #y

                self.textStruct2D[self.frameNum%1000,2,i,3] = targets[4,i] #vx
                self.textStruct2D[self.frameNum%1000,2,i,4] = targets[5,i] #vy

                self.textStruct2D[self.frameNum%1000,2,i,5] = targets[7,i] #ax
                self.textStruct2D[self.frameNum%1000,2,i,6] = targets[8,i] #ay

                if (self.printVerbosity == 1):
                    print('target added to textStructCapon3D')
        self.targetBufPing = targets

    #decode 3D People Counting Target List TLV
    def parseDetectedTracks3D(self, data, tlvLength):
        targetStruct = 'I9f'
        targetSize = struct.calcsize(targetStruct)
        self.numDetectedTarget = int(tlvLength/targetSize)
        targets = np.empty((13,self.numDetectedTarget))
        for i in range(self.numDetectedTarget):
            targetData = struct.unpack(targetStruct,data[:targetSize])
            targets[0:7,i]=targetData[0:7]
            targets[7:10,i]=[0,0,0]
            targets[10:13,i] = targetData[7:10]
            data = data[targetSize:]
        self.targetBufPing = targets

    #decode Target Index TLV
    def parseTargetAssociations(self, data):
        targetStruct = 'B'
        targetSize = struct.calcsize(targetStruct)
        numIndexes = int(len(data)/targetSize)
        self.indexes = []
        self.unique = []
        try:
```

```
            for i in range(numIndexes):
                ind = struct.unpack(targetStruct, data[:targetSize])
                self.indexes.append(ind[0])
                data = data[targetSize:]
            if (self.getUnique):
                uTemp = self.indexes[math.ceil(numIndexes/2):]
                self.indexes = self.indexes[:math.ceil(numIndexes/2)]
                for i in range(math.ceil(numIndexes/8)):
                    for j in range(8):
                        self.unique.append(getBit(uTemp[i], j))
        except:
            print('TLV Index Parse Fail')

    #decode Classifier output
    def parseClassifierOutput(self, data):
        classifierDataStruct = 'Ii'
        clOutSize = struct.calcsize(classifierDataStruct)
        self.classifierOutput = np.zeros((2,self.numDetectedTarget))
        for i in range(self.numDetectedTarget):
            self.classifierOutput[0,i], self.classifierOutput[1,i] = struct.unpack(classifierDataStruct, data[:clOutSize])
            data = data[clOutSize:]

#below is for labs that are compliant with SDK 3.x  This code can parse the point cloud TLV and point cloud side info TLV from the OOB demo.
#It can parse the SDK3.x Compliant People Counting demo "tracker_dpc"
    #get SDK3.x Cartesian Point Cloud
    def parseSDK3xPoints(self, dataIn, numObj):
        pointStruct = '4f'
        pointLength = struct.calcsize(pointStruct)
        try:
            for i in range(numObj):
                self.pcBufPing[0,i], self.pcBufPing[1,i], self.pcBufPing[2,i], self.pcBufPing[3,i] = struct.unpack(pointStruct, dataIn[:pointLength])
                dataIn = dataIn[pointLength:]
            self.pcBufPing = self.pcBufPing[:,:numObj]
        except Exception as e:
            print(e)
            self.fail = 1

    #get Side Info SDK 3.x
    def parseSDK3xSideInfo(self, dataIn, numObj):
        sideInfoStruct = '2h'
        sideInfoLength = struct.calcsize(sideInfoStruct)
        try:
            for i in range(numObj):
                self.pcBufPing[4,i], unused = struct.unpack(sideInfoStruct, dataIn[:sideInfoLength])
                dataIn = dataIn[sideInfoLength:]
        except Exception as e:
            print(e)
            self.fail = 1

    #convert SDK compliant Polar Point Cloud to Cartesian
    def polar2CartSDK3(self):
        self.pcBufPing = np.empty((5,self.numDetectedObj))
        for n in range(0, self.numDetectedObj):
            self.pcBufPing[2,n] = self.pcPolar[0,n]*math.sin(self.pcPolar[2,n]) #z
            self.pcBufPing[0,n] = self.pcPolar[0,n]*math.cos(self.pcPolar[2,n])*math.sin(self.pcPolar[1,n]) #x
            self.pcBufPing[1,n] = self.pcPolar[0,n]*math.cos(self.pcPolar[2,n])*math.cos(self.pcPolar[1,n]) #y
        self.pcBufPing[3,:] = self.pcPolar[3,0:self.numDetectedObj] #doppler

    #decode SDK3.x Format Point Cloud in Polar Coordinates
    def parseSDK3xPolar(self, dataIn, tlvLength):
        pointStruct = '4f'
        pointLength = struct.calcsize(pointStruct)
        self.numDetectedObj = int(tlvLength/pointLength)
        try:
            for i in range(self.numDetectedObj):
                self.pcPolar[0,i], self.pcPolar[1,i], self.pcPolar[2,i], self.pcPolar[3,i] = struct.unpack(pointStruct, dataIn[:pointLength])
```

```
                dataIn = dataIn[pointLength:]
        except:
            self.fail = 1
            return
        self.polar2CartSDK3()

    #decode 3D People Counting Target List TLV

    #3D Struct format

    #uint32_t     tid;     /*! @brief   tracking ID */
    #float        posX;    /*! @brief   Detected target X coordinate, in m */
    #float        posY;    /*! @brief   Detected target Y coordinate, in m */
    #float        posZ;    /*! @brief   Detected target Z coordinate, in m */
    #float        velX;    /*! @brief   Detected target X velocity, in m/s */
    ##float       velY;    /*! @brief   Detected target Y velocity, in m/s */
    #float        velZ;    /*! @brief   Detected target Z velocity, in m/s */
    #float        accX;    /*! @brief   Detected target X acceleration, in m/s2 */
    #float        accY;    /*! @brief   Detected target Y acceleration, in m/s2 */
    #float        accZ;    /*! @brief   Detected target Z acceleration, in m/s2 */
    #float        ec[16];  /*! @brief   Target Error covariance matrix, [4x4 float], in row major order, range, azimuth, elev, doppler */
    #float        g;
    #float        confidenceLevel;    /*! @brief   Tracker confidence metric*/

    def parseDetectedTracksSDK3x(self, data, tlvLength):
        if (self.printVerbosity == 1):
            print(tlvLength)
        if (self.CaponEC):
            targetStruct = 'I27f'
        else:
            #targetStruct = 'I15f'
            targetStruct = 'I27f'
        targetSize = struct.calcsize(targetStruct)
        if (self.printVerbosity == 1):
            print('TargetSize=',targetSize)
        self.numDetectedTarget = int(tlvLength/targetSize)
        if (self.printVerbosity == 1):
            print('Num Detected Targets = ',self.numDetectedTarget)
        targets = np.empty((16,self.numDetectedTarget))
        rotTarget = [0,0,0]
        #theta = self.profile['elev_tilt']
        #print('theta = ',theta)
        #Rx = np.matrix([[ 1, 0           , 0           ],
        #           [ 0, math.cos(theta),-math.sin(theta)],
        #           [ 0, math.sin(theta), math.cos(theta)]])
        try:
            for i in range(self.numDetectedTarget):
                targetData = struct.unpack(targetStruct,data[:targetSize])
                if (self.printVerbosity == 1):
                    print(targetData)
                #tid, x, y
                if (self.CaponEC):
                    targets[0:13,i]=targetData[0:13]
                else:
                    #tid, pos x, pos y
                    targets[0:3,i]=targetData[0:3]
                    if (self.printVerbosity == 1):
                        print('Target Data TID,X,Y = ',targets[0:3,i])
                        print('i = ',i)
                    # pos z
                    targets[3,i] = targetData[3]

                    #rotTargetDataX,rotTargetDataY,rotTargetDataZ = rotX (targetData[1], targetData[2],targetData[3],self.profile['elev_tilt'])

                    #print('Target Data TID,X,Y = ',rotTargetDataX,', ',rotTargetDataY,', ', rotTargetDataZ)
                    #vel x, vel y
                    targets[4:6,i] = targetData[4:6]
                    #vel z
                    targets[6,i] = targetData[6]
                    # acc x, acc y
```

```
                    targets[7:9,i] = targetData[7:9]
                    # acc z
                    targets[9,i] = targetData[9]
                    #ec[16]
                    #targets[10:14,i]=targetData[10:14]
                    targets[10:13,i]=targetData[10:13]#Chris 2020-12-18
                    if (self.printVerbosity == 1):
                        print('ec = ',targets[10:13,i])
                    #g
                    #targets[14,i]=targetData[14]
                    targets[14,i]=targetData[26]
                    if (self.printVerbosity == 1):
                        print('g= ',targets[14,i])
                    #confidenceLevel
                    #targets[15,i]=targetData[15]
                    targets[15,i]=targetData[27]
                    if (self.printVerbosity == 1):
                        print('Confidence Level = ',targets[15,i])


                    #self.textStructCapon3D[[frame #],[header,pt cloud data,target info],index,data]
                    #[][header][magic, version, packetLength, platform, frameNum, subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum]
                    #[][pt cloud][pt index][#elev, azim, doppler, range, snr]
                    #[][target][Target #][TID,x,y,z,vx,vy,vz,ax,ay,az]
                    if (self.saveTextFile):
                        self.textStructCapon3D[self.frameNum%1000,2,i,0] = targets[0,i] #TID
                        self.textStructCapon3D[self.frameNum%1000,2,i,1] = targets[1,i] #x
                        self.textStructCapon3D[self.frameNum%1000,2,i,2] = targets[2,i] #y
                        self.textStructCapon3D[self.frameNum%1000,2,i,3] = targets[3,i] #z
                        self.textStructCapon3D[self.frameNum%1000,2,i,4] = targets[4,i] #vx
                        self.textStructCapon3D[self.frameNum%1000,2,i,5] = targets[5,i] #vy
                        self.textStructCapon3D[self.frameNum%1000,2,i,6] = targets[6,i] #vz
                        self.textStructCapon3D[self.frameNum%1000,2,i,7] = targets[7,i] #ax
                        self.textStructCapon3D[self.frameNum%1000,2,i,8] = targets[8,i] #ay
                        self.textStructCapon3D[self.frameNum%1000,2,i,9] = targets[9,i] #az
                        if (self.printVerbosity == 1):
                            print('target added to textStructCapon3D')
                data = data[targetSize:]
        except:
            print('Target TLV parse failed')
        self.targetBufPing = targets
        if (self.printVerbosity == 1):
            print(targets)



    #all TLV header decoding functions are below. Each lab with a Unique header or unique TLV set has its own header parsing function
    #decode Header and rest of TLVs for Legacy Labs and Indoor False detection mitigation
    def tlvHeader(self, data):
        #search for magic word
        self.targetBufPing = np.zeros((12,1))
        self.pcBufPing = np.zeros((5,self.maxPoints))
        self.indexes = []
        frameNum = -1
        self.numDetectedTarget = 0
        self.numDetectedObj = 0
        #search until we find magic word
        while (1):
            try:
                magic, version, platform, timestamp, packetLength, frameNum, subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum =  struct.unpack('Q10I2H', data[:self.headerLength])
            except:
                #bad data, return
                self.fail = 1
                return data
            if (magic != self.magicWord):
                #wrong magic word, increment pointer by 1 and try again
                data = data[1:]
            else:
```

```
                #we have correct magic word, proceed to parse rest of data
                break

        #Sense and direct format
        #[][header][magic, version, platform, timestamp, packetLength, frameNum, subFrameNum,
     chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum]
        if (self.saveTextFile):
            self.textStruct2D[self.frameNum%1000,0,0,0] = magic
            self.textStruct2D[self.frameNum%1000,0,1,0] = version
            self.textStruct2D[self.frameNum%1000,0,2,0] = platform
            self.textStruct2D[self.frameNum%1000,0,3,0] = timestamp
            self.textStruct2D[self.frameNum%1000,0,4,0] = packetLength
            self.textStruct2D[self.frameNum%1000,0,5,0] = frameNum
            self.textStruct2D[self.frameNum%1000,0,6,0] = subFrameNum
            self.textStruct2D[self.frameNum%1000,0,7,0] = chirpMargin
            self.textStruct2D[self.frameNum%1000,0,8,0] = frameMargin
            self.textStruct2D[self.frameNum%1000,0,9,0] = uartSentTime
            self.textStruct2D[self.frameNum%1000,0,10,0] = trackProcessTime
            self.textStruct2D[self.frameNum%1000,0,11,0] = numTLVs
            self.textStruct2D[self.frameNum%1000,0,12,0] = checksum
            if (self.printVerbosity == 1):
                print('FrameNumber = ',self.textStruct2D[self.frameNum%1000,0,5,0])

        if (self.frameNum != frameNum):
            self.missedFrames += 1
            self.frameNum = frameNum
        self.frameNum += 1
        if (len(data) < packetLength):
            ndata = self.dataCom.read(packetLength-len(data))
            if (self.saveBinary):
                self.oldData += ndata
            data += ndata
        data = data[self.headerLength:]
        for i in range(numTLVs):
            try:
                tlvType, tlvLength = self.tlvHeaderDecode(data[:8])
            except:
                print('read fail: not enough data')
                self.missedFrames += 1
                self.fail=1
                break
            try:
                data = data[8:]
                if (tlvType == 6):
                    if(self.threeD):
                        self.parseDetectedObjects3D(data[:tlvLength], tlvLength-8)
                    elif(self.ifdm):
                        self.parseDetectedObjectsIFDM(data[:tlvLength], tlvLength-8)
                    else:
                        self.parseDetectedObjects(data[:tlvLength], tlvLength-8)
                elif (tlvType == 7):
                    if(self.threeD):
                        self.parseDetectedTracks3D(data[:tlvLength], tlvLength-8)
                    else:
                        self.parseDetectedTracks(data[:tlvLength], tlvLength-8)
                elif (tlvType == 8):
                    self.parseTargetAssociations(data[:tlvLength-8])
                elif (tlvType == 9):
                    self.parseClassifierOutput(data[:tlvLength-8])
                data = data[tlvLength-8:]
            except:
                print('Not enough data')
                print('Data length: ', len(data))
                print('Reported Packet Length: ', packetLength)
                self.fail=1
                return data
        return data

    #parsing for SDK 3.x Point Cloud
    def sdk3xTLVHeader(self, dataIn):
        #reset point buffers
        self.pcBufPing = np.zeros((5,self.maxPoints))
        headerStruct = 'Q8I'
```

```
        headerLength = struct.calcsize(headerStruct)
        tlvHeaderLength = 8
        #search until we find magic word
        while(1):
            try:
                magic, version, totalPacketLen, platform, self.frameNum, timeCPUCycles, self.numDetectedObj, numTLVs, subFrameNum = struct.unpack(headerStruct, dataIn[:headerLength])
            except:
                #bad data, return
                self.fail = 1
                return dataIn
            if (magic != self.magicWord):
                #wrong magic word, increment pointer by 1 and try again
                dataIn = dataIn[1:]
            else:
                #we have correct magic word, proceed to parse rest of data
                break
        dataIn = dataIn[headerLength:]
        remainingData = totalPacketLen - len(dataIn)
        count = 0
        #check to ensure we have all of the data
        while (remainingData > 0 and count < 3):
            newData = self.dataCom.read(remainingData)
            remainingData = totalPacketLen - len(dataIn) - len(newData)
            dataIn += newData
            count += 1
            if (self.saveBinary):
                self.oldData += newData
        #now check TLVs
        #print ('got tlvs sdk3x')
        for i in range(numTLVs):
            try:
                tlvType, tlvLength = self.tlvHeaderDecode(dataIn[:tlvHeaderLength])
            except Exception as e:
                print(e)
                print ('failed to read OOB SDK3.x TLV')
            dataIn = dataIn[tlvHeaderLength:]
            if (tlvType == 1):
                self.parseSDK3xPoints(dataIn[:tlvLength], self.numDetectedObj)
                dataIn = dataIn[tlvLength:]
            elif (tlvType == 7):
                self.parseSDK3xSideInfo(dataIn[:tlvLength], self.numDetectedObj)
                dataIn = dataIn[tlvLength:]
        return dataIn


    #parsing for SDK 3.x DPIF compliant People Counting
    def sdk3xPCHeader(self, dataIn):
        #reset point buffers
        self.pcBufPing = np.zeros((5,self.maxPoints))
        self.targetBufPing = np.zeros((13,20))
        self.indexes = []
        tlvHeaderLength = 8
        #search until we find magic word
        while (1):
            try:
                magic, version, platform, timestamp, packetLength, self.frameNum, subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum =  struct.unpack('Q10I2H', dataIn[:self.headerLength])
            except:
                #bad data, return
                self.fail = 1
                return dataIn
            if (magic != self.magicWord):
                #wrong magic word, increment pointer by 1 and try again
                dataIn = dataIn[1:]
            else:
                #we have correct magic word, proceed to parse rest of data
                break
        dataIn = dataIn[self.headerLength:]
        remainingData = packetLength - len(dataIn)
        if (self.printVerbosity == 1):
            print('pl: ', packetLength)
```

```
            print('remainingData ', remainingData)
        #check to ensure we have all of the data
        #check to ensure we have all of the data
        count = 0
        while (remainingData > 0 and count < 3):
            if (self.printVerbosity == 1):
                print('RD Loop')
            newData = self.dataCom.read(remainingData)
            remainingData = packetLength - len(dataIn) - len(newData)
            dataIn += newData
            count += 1
            if (remainingData == 0):
                if (self.saveBinary):
                    self.oldData += newData
        #now check TLVs
        if (self.printVerbosity == 1):
            print('Frame: ', self.frameNum)
            print(len(dataIn))
            print(numTLVs)
        for i in range(numTLVs):
            try:
                #print("DataIn Type", type(dataIn))
                tlvType, tlvLength = self.tlvHeaderDecode(dataIn[:tlvHeaderLength])
                if (self.printVerbosity == 1):
                    print('TLV length = ',tlvLength)
            except Exception as e:
                if (self.printVerbosity == 1):
                    print(e)
                    print ('failed to read OOB SDK3.x TLV')
                    print('TLV num: ',i)
            dataIn = dataIn[tlvHeaderLength:]
            dataLength = tlvLength
            if (tlvType == 6):
                #DPIF Polar Coordinates
                #print('pointcloud lrpd')
                self.parseSDK3xPolar(dataIn[:dataLength], dataLength)
            elif (tlvType == 7):
                #target 3D
                self.parseDetectedTracksSDK3x(dataIn[:dataLength], dataLength)
            elif (tlvType == 8):
                #target index
                self.parseTargetAssociations(dataIn[:dataLength])
            elif (tlvType == 9):
                #side info
                self.parseSDK3xSideInfo(dataIn[:dataLength], self.numDetectedObj)
            dataIn = dataIn[dataLength:]
        return dataIn

    #parsing for 3D People Counting lab
    def Capon3DHeader(self, dataIn):
        #reset point buffers
        self.pcBufPing = np.zeros((5,self.maxPoints))
        self.pcPolar = np.zeros((5,self.maxPoints))
        self.targetBufPing = np.zeros((13,20))
        self.numDetectedTarget = 0
        self.numDetectedObj = 0
        self.indexes = []
        tlvHeaderLength = 8
        headerLength = 48
        #stay in this loop until we find the magic word or run out of data to parse
        while (1):
            try:
                magic, version, packetLength, platform, frameNum, subFrameNum, chirpMargin,
    frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum =  struct.unpack('Q9I2H',
    dataIn[:headerLength])
            except Exception as e:
                #bad data, return
                #print("Cannot Read Frame Header")
                #print(e)
                self.fail = 1
                return dataIn
            if (magic != self.magicWord):
                #wrong magic word, increment pointer by 1 and try again
```

```
                dataIn = dataIn[1:]
            else:
                #got magic word, proceed to parse
                break


        dataIn = dataIn[headerLength:]
        remainingData = packetLength - len(dataIn) - headerLength
        #check to ensure we have all of the data
        #print('remaining data = ',remainingData)
        if (remainingData > 0):
            newData = self.dataCom.read(remainingData)
            remainingData = packetLength - len(dataIn) - headerLength - len(newData)
            dataIn += newData
            if (self.saveBinary):
                self.oldData += newData
        if (self.saveTextFile):
            self.textStructCapon3D[self.frameNum%1000,0,0,0] = magic
            self.textStructCapon3D[self.frameNum%1000,0,1,0] = version
            self.textStructCapon3D[self.frameNum%1000,0,2,0] = packetLength
            self.textStructCapon3D[self.frameNum%1000,0,3,0] = platform
            self.textStructCapon3D[self.frameNum%1000,0,4,0] = frameNum
            self.textStructCapon3D[self.frameNum%1000,0,5,0] = subFrameNum
            self.textStructCapon3D[self.frameNum%1000,0,6,0] = chirpMargin
            self.textStructCapon3D[self.frameNum%1000,0,7,0] = frameMargin
            self.textStructCapon3D[self.frameNum%1000,0,8,0] = uartSentTime
            self.textStructCapon3D[self.frameNum%1000,0,9,0] = trackProcessTime
            self.textStructCapon3D[self.frameNum%1000,0,10,0] = numTLVs
            self.textStructCapon3D[self.frameNum%1000,0,11,0] = checksum
            if (self.printVerbosity == 1):
                print('FrameNumber = ',self.textStructCapon3D[self.frameNum%1000,0,4,0])

        #now check TLVs
        for i in range(numTLVs):
            #try:
            #print("DataIn Type", type(dataIn))
            try:
                tlvType, tlvLength = self.tlvHeaderDecode(dataIn[:tlvHeaderLength])
                dataIn = dataIn[tlvHeaderLength:]
                dataLength = tlvLength-tlvHeaderLength
            except:
                print('TLV Header Parsing Failure')
                self.fail = 1
                return dataIn
            if (tlvType == 6):
                #DPIF Polar Coordinates
                self.parseCapon3DPolar(dataIn[:dataLength], dataLength)
            elif (tlvType == 7):
                #target 3D
                self.parseDetectedTracksSDK3x(dataIn[:dataLength], dataLength)
            elif (tlvType == 8):
                #target index
                self.parseTargetAssociations(dataIn[:dataLength])
            elif (tlvType == 9):
                if (self.printVerbosity == 1):
                    print('type9')
                #side info
                #self.parseSDK3xSideInfo(dataIn[:dataLength], self.numDetectedObj)
            dataIn = dataIn[dataLength:]
            #except Exception as e:
            #    print(e)
            #    print ('failed to read OOB SDK3.x TLV')
        if (self.frameNum + 1 != frameNum):
            self.missedFrames += frameNum - (self.frameNum + 1)
        self.frameNum = frameNum
        return dataIn


    # This function is always called - first read the UART, then call a function to parse the
     specific demo output
    # This will return 1 frame of data. This must be called for each frame of data that is
    expected. It will return a dict containing:
    #   1. Point Cloud
```

```
    #   2. Target List
    #   3. Target Indexes
    #   4. number of detected points in point cloud
    #   5. number of detected targets
    #   6. frame number
    #   7. Fail - if one, data is bad
    #   8. classifier output
    # Point Cloud and Target structure are liable to change based on the lab. Output is
    always cartesian.
    def readAndParseUart(self, data):
        self.fail = 0
        if (self.replay):
            print('ik ben by 799')
            return self.replayHist()
        numBytes = 4666
        #data = self.dataCom.read(numBytes)
        if (self.byteData is None):
            print('804')
            self.byteData = data
        else:
            self.byteData += data
        if (self.saveBinary):
            self.oldData += data
        #try:
        if (self.SDK3xPointCloud == 1):
            self.byteData = self.sdk3xTLVHeader(self.byteData)
        elif (self.SDK3xPC == 1):
            self.byteData = self.sdk3xPCHeader(self.byteData)
        elif (self.capon3D == 1):
            self.byteData = self.Capon3DHeader(self.byteData)
        else:
            self.byteData = self.tlvHeader(self.byteData)
        #except Exception as e:
        #    print(e)
        #    self.fail = 1
        #return data after parsing and save to replay file
        if (self.fail):
            return self.pcBufPing, self.targetBufPing, self.indexes, self.numDetectedObj,
    self.numDetectedTarget, self.frameNum, self.fail, self.classifierOutput
        if (self.saveBinary):
            if (self.frameNum%1000 == 0):
                toSave = bytes(self.oldData)
                fileName = 'binData/pHistBytes_'+str(self.saveNum)+'.bin'
                self.saveNum += 1
                bfile = open(fileName, 'wb')
                bfile.write(toSave)
                self.oldData = []
                print ('Missed Frames ' + str(self.missedFrames)+'/1000')
                self.missedFrames = 0
                bfile.close
        if (self.saveTextFile):
            if (self.frameNum%1000 == 0):
                if (self.capon3D):
                    toSave = self.textStructCapon3D
                elif (self.ifdm):
                    toSave = self.textStruct2D
                print('Saved data file ', self.saveNumTxt)
                fileName = 'binData/pHistText_'+str(self.saveNumTxt)+'.csv'
                if (self.saveNumTxt < 75):
                    self.saveNumTxt += 1
                else:
                    self.saveNumTxt = 0
                tfile = open(fileName, 'w')
                tfile.write('This file contains parsed UART data in sensor centric
    coordinates\n')
                tfile.write('file format version 1.0\n')
                #tfile.write(str(toSave))


                if (self.capon3D):
                    #[frame #][header,pt cloud data,target info]
                    #[][header][magic, version, packetLength, platform, frameNum, subFrameNum
    , chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum]
```

```
                #[][pt cloud][pt index][#elev, azim, doppler, range, snr]
                #[][target][Target #][TID,x,y,z,vx,vy,vz,ax,ay,az]

                for i in range (1000):
                    tfile.write('magic, version, packetLength, platform, frameNum, subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum\n')
                    for j in range (0,12):
                        tfile.write(str(self.textStructCapon3D[i,0,j,0]))
                        tfile.write(',')
                        #print(str(self.textStructCapon3D[i,0,j,0]))
                    tfile.write('\n')
                    tfile.write('elev, azim, doppler, range, snr\n')
                    for j in range (np.count_nonzero(self.textStructCapon3D[i,1,:,0])): #self.numDetectedObj):#len(self.textStructCapon3D[i,1,:,0]!=0)):
                        for k in range(5):
                            tfile.write(str(self.textStructCapon3D[i,1,j,k]))
                            tfile.write(',')
                        tfile.write('\n')

                    tfile.write('TID,x,y,z,vx,vy,vz,ax,ay,az\n')
                    for j in range (np.count_nonzero(self.textStructCapon3D[i,2,:,1])):
                        for k in range(10):
                            tfile.write(str(self.textStructCapon3D[i,2,j,k]))
                            tfile.write(',')
                        tfile.write('\n')
                self.textStructCapon3D = np.zeros(1000*3*12*self.maxPoints).reshape((1000,3,12,self.maxPoints))#[frame #][header,pt cloud data,target info]
                tfile.close

            if (self.ifdm):
                #Sense and direct format
                #[frame #][header,pt cloud data,target info]
                #[][header][magic, version, platform, timestamp, packetLength, frameNum, subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum]
                #[][pt cloud][pt index][#range, azim, doppler, snr]
                #[][target][Target #][TID,x,y,vx,vy,ax,ay]
                for i in range (1000):
                    tfile.write('magic, version, platform, timestamp, packetLength, frameNum, subFrameNum, chirpMargin, frameMargin, uartSentTime, trackProcessTime, numTLVs, checksum\n')
                    for j in range (13):
                         tfile.write(str(self.textStruct2D[i,0,j,0]))
                         tfile.write(',')
                    tfile.write('\n')
                    tfile.write('range, azim, doppler, snr\n')
                    for j in range (np.count_nonzero(self.textStruct2D[i,1,:,0])):
                        for k in range(4):
                            tfile.write(str(self.textStruct2D[i,1,j,k]))
                            tfile.write(',')
                        tfile.write('\n')
                    tfile.write('TID,x,y,vx,vy,ax,ay\n')
                    for j in range (np.count_nonzero(self.textStruct2D[i,2,:,1])):
                        for k in range(7):
                            tfile.write(str(self.textStruct2D[i,2,j,k]))
                            tfile.write(',')
                        tfile.write('\n')
                self.textStruct2D = np.zeros(1000*3*self.maxPoints*7).reshape((1000,3,self.maxPoints,7))#[frame #][header,pt cloud data,target info]
                tfile.close



    parseEnd = int(round(time.time()*1000))
    print (self.pcBufPing)
    print (self.targetBufPing)
    print (self.indexes)
    print (self.numDetectedObj)
    print (self.numDetectedTarget)
    return self.pcBufPing, self.targetBufPing, self.indexes, self.numDetectedObj, self.numDetectedTarget, self.frameNum, self.fail, self.classifierOutput

#find various utility functions here for connecting to COM Ports, send data, etc...
```

```
    #connect to com ports
    # Call this function to connect to the comport. This takes arguments self (intrinsic), uartCom, and dataCom. No return, but sets internal variables in the parser object.
    def connectComPorts(self, uartCom, dataCom):

        self.uartCom = serial.Serial(uartCom, 115200,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE,timeout=0.3)
        if (self.capon3D == 1 and self.aop == 0):
            self.dataCom = serial.Serial(dataCom, 921600*1,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE,timeout=0.025)
        else:
            self.dataCom = serial.Serial(dataCom, 921600,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE,timeout=0.025)
        self.dataCom.reset_output_buffer()
        return self.dataCom
        print('Connected for parser')

    #send cfg over uart
    def sendCfg(self, cfg):
        for line in cfg:
            time.sleep(.1)
            self.uartCom.write(line.encode())
            ack = self.uartCom.readline()
            print(ack)
            ack = self.uartCom.readline()
            print(ack)
        time.sleep(3)
        self.uartCom.reset_input_buffer()
        self.uartCom.close()

    #send single command to device over UART Com.
    def sendLine(self, line):
        self.uartCom.write(line.encode())
        ack = self.uartCom.readline()
        print(ack)
        ack = self.uartCom.readline()
        print(ack)

    def replayHist(self):
        if (self.replayData):
            #print('reading data')
            #print('fail: ',self.fail)
            #print(len(self.replayData))
            #print(self.replayData[0:8])
            self.replayData = self.Capon3DHeader(self.replayData)
            #print('fail: ',self.fail)
            return self.pcBufPing, self.targetBufPing, self.indexes, self.numDetectedObj, self.numDetectedTarget, self.frameNum, self.fail, self.classifierOutput
            #frameData = self.replayData[0]
            #self.replayData = self.replayData[1:]
            #return frameData['PointCloud'], frameData['Targets'], frameData['Indexes'], frameData['Number Points'], frameData['NumberTracks'],frameData['frame'],0, frameData['ClassifierOutput'], frameData['Uniqueness']
        else:
            filename = 'overheadDebug/binData/pHistBytes_'+str(self.saveNum)+'.bin'
            #filename = 'Replay1Person10mShort/pHistRT'+str(self.saveNum)+'.pkl'
            self.saveNum+=1
            try:
                dfile = open(filename, 'rb', 0)
            except:
                print('cant open ', filename)
                return -1
            self.replayData = bytes(list(dfile.read()))
            if (self.replayData):
                print('entering replay')
                return self.replayHist()
            else:
                return -1

def getBit(byte, bitNum):
    mask = 1 << bitNum
    if (byte&mask):
        return 1
```

```
    else:
        return 0
```

E

Schematic

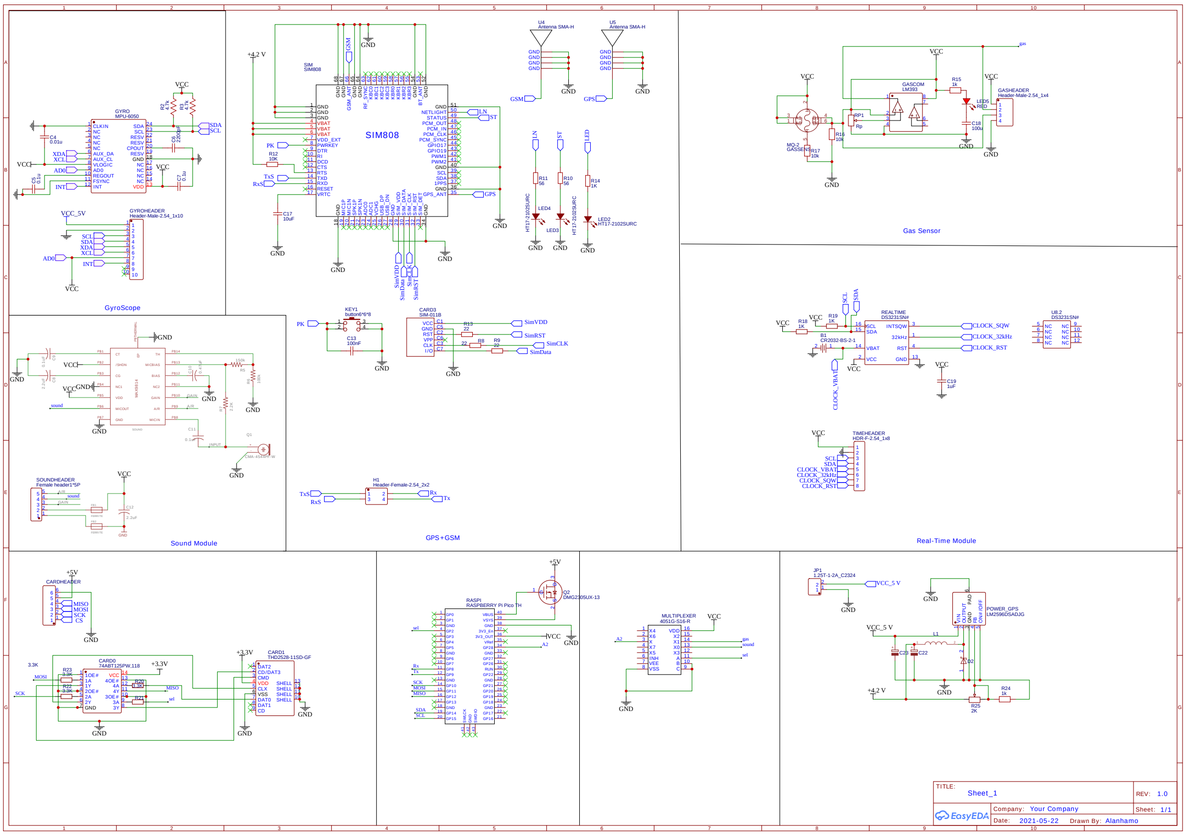


Figure E.1: Schematic for Teddy components PCB