**Embedded Systems Engineering**

Product Report

Trick question inc - Reverse Geocaching Box Development



**Embedded Systems Engineering Academy Engineering and Automotive HAN University of Applied Sciences**

**Authors**  
2158469 Federico Giovanni Accossato  
2153089 Henri Krinke  
2140725 Jabez Impano  
2158363 Mohammed Alkhomaish Hernandez  
2156231 Pedro von Sydow

**Tutor**

Remko Welling

**Customer**

Hugo Arends

**Date**

February 2025

**Version** 0.5

# Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| Version | When | Who | What |
| 0.1 | 21/02/2025 | Federico & Mohammed | Product Report File created, the Preface, Summary, and the Introduction were written |
| 0.1.1 | 21/02/2025 | Henri & Jabez | Wrote the functional design chapter |
| 0.1.2 | 28/02/2025 | Mohammed | Updated the introduction, added Technical Specifications, added “user interface section” |
| 0.1.3 | 8/03/2025 | Henri | Updated “user interface section” added puzzle 3 & 4 description |
| 0.1.4 | 9/03/2025 | Pedro | Updated “user interface section” added puzzle 2 description |
| 0.1.5 | 10/03/2025 | Mohammed | Updated “user interface section” added puzzle 1 description |
| 0.1.6 | 13/03/2025 | Federico | Updated “Functional Design” to be sent to the Tutor |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Preface

The Reverse Geocaching Box project explores the integration of embedded systems in secure storage solutions. This report documents the development process of a GPS-locked box that unlocks upon reaching predefined coordinates. Our team, under the guidance of our tutor and in collaboration with a client, worked on designing and implementing hardware and firmware solutions to meet the specified requirements.

# Summary

The goal of this project was to design and develop a reverse geocaching box—a locked container that only opens when it reaches a specific GPS location. This involved implementing GPS tracking, environmental data logging, an actuator-based locking mechanism, and interactive puzzles for unlocking. The box also communicates with a laptop for data retrieval and configuration.

The project was successfully developed within the constraints of a €50 budget (excluding the components provided) and met all functional and technical requirements.

**Contents**

[Revisions 1](#_Toc644646468)

[Preface 2](#_Toc90582159)

[Summary 3](#_Toc1573821124)

[1 Introduction 4](#_Toc1276092122)

[1.1 Background 5](#_Toc179899313)

[1.2 Objective 5](#_Toc1125474094)

[1.3 Report structure 5](#_Toc35249411)

[2 Functional Design 5](#_Toc772551411)

[2.1 Functional specification 6](#_Toc1303196694)

[2.2 Technical specifications 7](#_Toc1099067114)

[2.3 User interface 8](#_Toc115416977)

[Puzzle-Box: 9](#_Toc613381051)

[Companion laptop: 12](#_Toc582325324)

[3 Technical design 14](#_Toc175114693)

[1. Introduction 15](#_Toc515530813)

[1.1 Background 15](#_Toc771698600)

[1.2 Reason 15](#_Toc1863602987)

[1.3 Report Structure 15](#_Toc1402076059)

[2 Architecture 16](#_Toc997582235)

[2.1 Architecture diagram 16](#_Toc600708179)

[2.2 UML Component diagram 17](#_Toc1731640469)

[3 Interfaces 17](#_Toc843175334)

[3.1 Power supply 17](#_Toc532857503)

[3.2 Microcontroller – UART – GPS module 18](#_Toc19626921)

[Overview 18](#_Toc781126006)

[Working principle 18](#_Toc85183587)

[Specifications 18](#_Toc1488320609)

[Sequence diagram: 18](#_Toc1822148618)

[3.3 Microcontroller – Color sensor 19](#_Toc1696628438)

[Overview 19](#_Toc199441239)

[Working Principle 19](#_Toc1507200673)

[Specifications 19](#_Toc1825263025)

[Sequence Diagram 19](#_Toc121413120)

[3.4 Microcontroller – Tilt sensor 20](#_Toc459781116)

[Overview 20](#_Toc49291665)

[Working principle 20](#_Toc678200184)

[Specifications 20](#_Toc7373039)

[Sequence diagram 20](#_Toc596901295)

[3.5 Microcontroller – Temperature sensor 21](#_Toc665863572)

[Overview 21](#_Toc1678373234)

[Working Principle 21](#_Toc614687069)

[Specifications 22](#_Toc1212245762)

[Sequence Diagram: 22](#_Toc945369333)

[3.6 Microcontroller – Joystick 23](#_Toc372944033)

[Overview 23](#_Toc448027146)

[Working Principle 23](#_Toc723332813)

[Specifications 24](#_Toc1479200400)

[Sequence Diagram: 24](#_Toc432718151)

[3.7 Microcontroller – LCD-Display 24](#_Toc238952958)

[Overview 25](#_Toc1057823988)

[Working principle 25](#_Toc1379955045)

[Specifications 25](#_Toc221489495)

[Sequence Diagram: 25](#_Toc723387057)

[3.8 Microcontroller – Numpad 25](#_Toc1466780300)

[Overview 26](#_Toc1588872077)

[Working principle 26](#_Toc1249035485)

[Specifications 26](#_Toc384920743)

[Sequence Diagram: 26](#_Toc1931744450)

[3.9 Microcontroller – MicroSD Card 27](#_Toc1267009206)

[Overview 27](#_Toc1093925403)

[Working Principle 27](#_Toc688235835)

[Specifications 28](#_Toc935771464)

[Sequence diagram 28](#_Toc1915018163)

[3.10 Microcontroller – MOSFET – Solenoid 28](#_Toc872461852)

[Overview 29](#_Toc1575606801)

[Working principle: 29](#_Toc1894103311)

[Specifications 29](#_Toc56426698)

[Sequence Diagram: 29](#_Toc1400309595)

[4 Software 29](#_Toc210067613)

[4 Realization 30](#_Toc130562929)

[Hardware 31](#_Toc653979067)

[Software 31](#_Toc1992979027)

[Main 31](#_Toc1092101682)

[GPS/GPS Games 31](#_Toc956680604)

[SD-Card/Logging 34](#_Toc847769015)

[Solenoid/Unlocking 34](#_Toc997971147)

[Color Sensor 34](#_Toc2127580735)

[Temperature Sensor 34](#_Toc337165314)

[PCB 34](#_Toc872891746)

[LCD Display 34](#_Toc1091882725)

# Introduction

## Background

This report is part of the second semester project of the Embedded Systems Engineering bachelor’s program at the Hogeschool van Arnhem en Nijmegen (HAN). The surrounding second semester as well as the HAN facilities provide the students with all the tools and knowledge needed, like how to program a microcontroller using C, how to write professionally or access to the soldering area.

The project itself is to develop a reverse geocaching box for the client, Hugo Arends, which is a box that unlocks itself once it has been brought to a specific location. On the way to that specific location the user must solve at least one puzzle that uses interfaces like a display.

This is to be developed within a semester and a budget of 50 euros excluding certain components with the goal of delivering a product that matches the requirements set by; and communicated with the client along with the necessary documentation. All of this serves the purpose of the students gaining technical skills as well as getting more comfortable with working in a professional setting.

## Objective

This report is meant to illustrate the development process of the reverse geocaching box. Specifically, to serve as a description of how functions have been implemented hard- and software-wise as well as sum up which functions have been achieved and successfully been displayed during testing.

These are done following the V-Model (add reference), starting with the requirement engineering phase, moving to the subsystem development and ending with full-system integration and appropriate testing. This step-by-step approach mitigates risk and is easy to document and follow.

## Report structure

**The structure of this report is as follows:**

* **Functional Design:** Includes functional and technical specifications as well as description of the user interface.
* **Technical Design:** Includes Architecture and a general wiring diagram as well as a description of how chosen components interface with each other and a general software diagram.
* **Realization:** Includes a wiring diagram of the final product and describes the code implemented to achieve the functional specifications.
* **Testing:** Includes test cases as well as their results.
* **Conclusion and recommendations:** Sums up the process, results and problems.

# Functional Design

The functional specifications define the behaviors and feature the Reverse Geocaching Box must exhibit. These specifications are written using the SMART criteria to ensure they are Specific, Measurable, Assignable, Realistic, and Time-related. They are also categorized into must, should and could have according to the MoSCoW method.

## Functional specification

|  |  |  |
| --- | --- | --- |
| Nr. | MoSCow | Description |
| F1 | M | Must consist of an openable box |
| F1.1 | M | Must be able to be locked and unlocked |
| F1.1.2 | M | Must use an actuator for the locking mechanism |
| F1.2 | S | Should be portable |
| F2 | M | The box must be able to track its own coordinates |
| F2.1 | M | Must have a tracking accuracy of a 10-meter radius |
| F2.2 | M | Must be tracked every second |
| F2.3 | M | Must be able to compare its own coordinates to target coordinates |
| F2.3.1 | S | Should be able to handle multiple target locations |
| F2.3.2 | M | Must be able to determine if it is at the target coordinates |
| F2.3.3 | M | Must be able to determine the direct distance to the coordinates |
| F2.3.4 | S | Should be able to determine in which compass direction the target coordinates are located |
| F3 | M | Must have multiple puzzle elements before the box can be opened |
| F3.1 | S | There should be puzzles at 4 different locations |
| F3.2 | S | Location one should contain a memory game |
| F3.2.1 | S | A sequence of arrows should be displayed on a display |
| F3.2.2 | S | The sequence should be replicated using a joystick |
| F3.3 | S | Location two should contain a colour dependant game |
| F3.3.1 | S | Depending on a presented colour different information should be displayed |
| F3.3.1.1 | S | When green is presented the compass direction of the next location should be shown for one minute |
| F3.3.1.2 | S | When brown is presented the distance to the next location should be shown for 30 seconds |
| F3.3.1.3 | S | When White/Grey is presented the option to receive a hint about the next location will be shown for 15 seconds |
| F3.3.1.4 | S | Each colour option should only be able to be chosen three times |
| F3.4 | S | Location three should contain a balancing game |
| F3.4.1 | S | The box should be kept straight within a margin of 20° until reaching the final location |
| F3.4.2 | S | The compass direction of the final location will be shown in degrees from 0° to 360° |
| F3.5 | M | At each of the location a light corresponding to a number should display |
| F3.5.1 | M | At the final location all the numbers should be entered to unlock the box |
| F4 | M | The box must be able to display relevant information on an interface |
| F4.1 | M | Must be able to show the puzzles |
| F4.1.1 | S | Should be able to display distance to target coordinates |
| F4.1.2 | S | Should be able to display compass location of target location |
| F4.1.3 | S | Should be able to display arrows in vertical, horizontal and diagonal directions |
| F4.1.2 | C | Could display puzzle descriptions |
| F4.2 | M | Must be able to display a menu that the user can use |
| F4.3 | M |  |
| F4.4 | S | Should be able to show the runtime of the batteries and |
| F4.5 | M | Must be able to show if a GPS connection has been established |
| F5 | M | The box must be able to log certain information |
| F5.1 | M | Must be able to log the GPS coordinates |
| F5.1.1 | S | Should be logged every 10 seconds |
| F5.2 | M | Must be able to log the temperature outside the box |
| F5.2.1 | S | Should be logged every 10 meters walked |
| F5.3 | M | Must be able to store puzzle progress |
| F5.3.1 | S | Should be logged after every solved puzzle |
| F5.4 | M | Logged information must be stored in persistent memory |
| F6 | M | The box must be able to communicate with a companion laptop |
| F6.1 | S | Should only be accessible once the box has been opened |
| F6.2 | M | Must be through cabled connection |
| F6.3 | M | Must be able to set target locations |
| F6.4 | M | Must be able to display logged information |
| F6.4.1 | S | Should be able to clear logged information from persistent memory |

## Technical specifications

The technical specifications contain additional specifications that the customer/client requires of the product. This can be, for example, the type of microcontroller or the programming language with which the product must be realized. Use a table to display the technical specifications in a clear and grouped way. Also give each specification an identification number so that it can be easily referred to in the rest of the report.

Technical Specifications

|  |  |
| --- | --- |
| **Nr.** | **Specifications** |
| T1 | Must be controlled using the FRDMMcxa153-development board |
| T2 | Must use the ATGM336H GPS module |
| T3 | Should use an OLED display |
| T3.1 | Should use I2C protocol |
| T4 | Should use a physical compass or compass module |
| T5 | Should use a ky-023 Joystick for the first game |
| T6 | Should use the TCS34725 colour sensor module for the second game |
| T7 | Should use the SW-520D tilt sensor module for the third game |
| T8 | Should use the JF-0530B solenoid as the locking mechanism actuator |
| T9 | Should use the ASAIR AHT21B temperature and humidity sensor for logging the temperature |
| T10 | Should use UART protocol to communicate with the companion laptop |
| T11 | Must be programmed using C programming language |
| T12 | Must use two separate power supplies |
| T12.1 | 5V supply for the development board. |
| T12.2 | 6V supply for the solenoid |
| T13 | Should be a maximum of 20cm\*20cm, 30cm\* 30cm |
| T14 | Should have a separate compartment for the power supply |

## **2.3** User interface

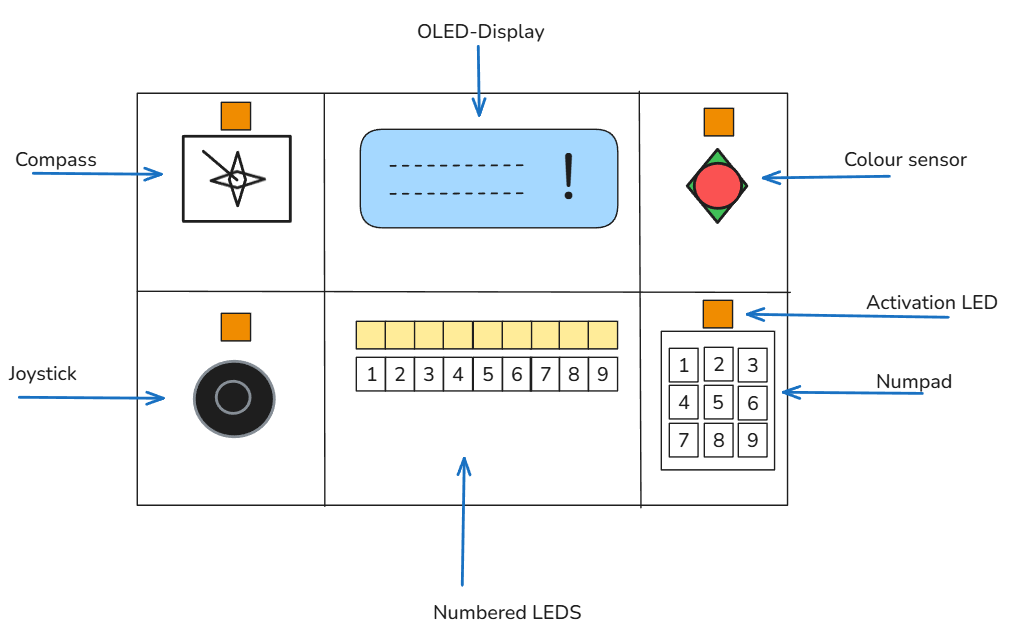
The user interface of the Reverse Geocaching Box plays a crucial role in how users interact with the system. It includes both input and output components that facilitate user engagement with the puzzle-solving and unlocking process. Additionally, the interface is split into two parts, the outside of the box with its sensors, modules and the display and the companion laptop with its keyboard and screen.

### Puzzle-Box:

#### Appearance and Layout

The Reverse Geocaching Box will have a compact and portable design, with a clear distinction between input and output elements. The enclosure will house the electronic components, including the GPS module, sensors, and microcontroller, while the interface elements will be positioned for intuitive user interaction.

#### Sketch of the interface:

p

#### Inputs

* **Joystick**: Used for the memory game, allowing users to replicate arrow sequences.
* **Color Sensor**: Detects different colors and triggers corresponding puzzle hints.
* **Tilt Sensor**: Monitors box stability for the balancing game. (Not visible)
* **Buttons:** Act as inputs for the unlocking sequence.

#### Outputs

* **OLED Display:** Shows puzzle-related information, including directional hints, distances, and game progress.
* **LED Indicators:** Provide feedback on completed puzzles and display numerical clues at each location.
* **Solenoid-Actuator:** Provides final feedback by opening the door.

#### Interaction and Changes

The level of influence each of the inputs has on the output modules depends on which puzzle the user is currently solving. This section will go through the respective interactions puzzle by puzzle. Information always visible on the OLED display is whether a GPS fix has been successfully acquired. This will be done through a “.” if there is no fix and an “!” if there is a fix.

**The first puzzle:**

The game utilizes a joystick, and an OLED display. The OLED display will continuously show arrows pointing in various directions, prompting the player to move the joystick accordingly. Each time an arrow appears; the player must quickly push the joystick in the corresponding direction. If the correct movement is made within a set time frame, the game progresses smoothly. However, if the joystick is moved in the wrong direction or if the player fails to respond in time, a countdown timer will appear on the display. If the player does not correct their input before the timer expires, they will have to restart from the previous stage. Successfully following a sequence of directional prompts will trigger a transition to the next puzzle. Throughout the game, an LED indicator will remain illuminated, signaling active gameplay.

**The second puzzle:**

This game utilizes a color sensor and the OLED display. Upon leaving the starting location the display will read “provide color”, upon showing a color the display will update with a corresponding hint. Each color will have two variables: display time and maximum number of scans. The colors and corresponding hints can be edited on the companion app, by default they will be set to:

Brown (e.g. tree bark) – This will display the distance to the target location (30 seconds)

Green (e.g. grass) – This will display a compass direction to the target location (1 minute)

Off-white/grey (e.g. concrete) – This will display a hint about the target location (e.g. near a small pond) (15 seconds)

If the target location cannot be found after all colors are used up, the player will be given the option to reset from the start location.

**The third puzzle:**

It uses the tilt sensor, the GPS module, the OLED display and a compass. The OLED-Display will constantly display the direction to the target location from the current location in degrees. This changes if the user walks to a significantly different GPS location with the box. At the same time the display will show a graphic representation of whether the box is being held leveled or whichever side it is shifted to according to the information received from the tilt sensor. If the box is tilted by more than 20 degrees a five second countdown will appear next to the representation. If the box is not level again after this a message asking the player to go to the previous target location to restart the puzzle will be shown on the display. If the box is successfully delivered to the target location while keeping it level a message informing of the transition to the final puzzle will be displayed. The entire time a numbered LED will be turned on.

**The final/overarching puzzle:**

It uses the OLED display, the buttons and the solenoid actuator. The display will show a message asking for a code after solving all the puzzles and reaching the final location. During each of the previous puzzles a numbered LED has been turned on either during the puzzle or upon solving it. Once the corresponding numbers have been entered using the buttons the display will display a congratulations message, and the solenoid will open the box.

## Companion laptop:

#### Appearance and Layout:

The laptop interface is contained in a window

#### Sketch of the interface:

Immagine che contiene elettronica, computer, Dispositivo di output, computer

Descrizione generata automaticamente

#### Inputs:

The used inputs are going to be the laptops keyboard as well as its cursor

#### Outputs:

The outputs are going to be displayed in a window on the laptop screen.

#### Interactions and changes:

For these interactions to work the laptop has to be plugged into the microcontroller. Upon opening the application, the user gets shown a menu in which they can decide whether to set something in the microcontroller, look at the logged information from the run or just exit the application. If the logging option is chosen the menu changes to display the names of the things that were logged, which are GPS location every 10 seconds and temperature every 10 meters. The temperature will be displayed in a graph of temperature over time. The GPS locations will be shown on a map as a highlighted path. If the setting option is chosen the menu will display the option to set the location, the hints for each color of the color sensor game and the option to reset the pers which would make it possible for the game to restart. If the color setting is chosen for each hint the first letter of the color now used for it has to be pressed with each color only being able to be assigned once. In the location setting option the user has the option to set them by manually typing longitude and latitude of the target locations or to move a pin on a map using WASD for vertical and horizontal movement. The menus can be traversed using W for up and S for down. Selecting an option or saving a setting is done using the enter key. In every state of the interface the user can press b to go back and new settings are only saved if they represent a valid input.

# Technical design

# Introduction

The Technical Design of the Reverse Geocaching Box defines how to implement the functional and technical specifications considering the teams preliminary research document on suitable components with reasonings for their selection.

## Background

This document is part of the development process for a Reverse Geocaching Box, which is an interactive system that integrates GPS tracking and puzzle-solving elements to create a fun experience for the users. Unlike traditional geocaching, where users locate a hidden object using GPS coordinates, this system requires players to solve a series of challenges before unlocking the box they brought with them.

## Reason

The purpose of this document is to outline the technical solutions for implementing the Reverse Geocaching Box. The technical design ensures that the system is not only functional but also reliable and meets the client's expectations. This section defines the integration of hardware components such as the GPS module, joystick, sensors, and solenoid lock, along with the software structure for handling data, user interactions, and overall system management.

By breaking down the system into its individual components and specifying their roles, this document offers a clear technical roadmap to guide development. It details the communication protocols, power supply considerations, and how the system will respond to user inputs, making it an essential part of the project's technical development.

## Report Structure

This section of the report focuses on the Technical Design of the Reverse Geocaching Box and is organized into the following key areas:

* **System Architecture**: Overview of the entire system, including hardware and software components, their interactions, and data flow.
* **Interfaces**: Describes the communication between different components, such as the microcontroller and peripheral devices (e.g., GPS module, joystick, sensors, solenoid).
* **Software**: Outlines the style of software being used and shows the structure of the main program

# Architecture

The architecture is depicted using an architecture and component diagram. The diagrams describe how the components will interface with each other and especially with the FRDM-MCXA153 development board. Each components respective communication types and protocols are noted on the arrows. The VDD going out of the microcontroller represents the voltages going to each component which is going to be supplied by the microcontrollers multiple output voltage options for example 3.3V.

The components can be grouped into three main categories:

**Actuators:**

* LCD-Display (I2C)
* MOSFET (GPIO)
* Solenoid (powered via voltage supply)

**Sensors:**

* Colour Sensor (TCS34725) – I2C
* Temperature Sensor (ASAIR AHT21B) – I2C
* Tilt Sensor (SW-520D) – GPIO
* Numpad (9 switches) – GPIO
* GPS-Module (ATGM336H) – UART
* Joystick – GPIO, Analog

**Communication:**

* Laptop Application via COM-Port – UART
* MicroSD Card – SPI (for reading and writing data)

### Architecture diagram

The architecture diagram focusses on the communication protocols and the communication directions.

A diagram of a computer system

AI-generated content may be incorrect.

Figure 1: Architecture Diagram

### UML Component diagram

The component diagram adds additional information about which layer every component belongs to.

A diagram of a computer

AI-generated content may be incorrect.

Figure 2: UML Component Diagram

# Interfaces

This section describes how the components are implemented and interface with each other along with some reasoning for components. More detailed research can be found in the exploratory research document (Inc, Exploratory Research, 2025).

## Power supply

The box uses a 6V 1A power supply made of batteries that can be turned off and on using a switch. The 6V 1A ensures it can power the component that draws the most power, the solenoid. For everything else the voltage will be regulated down to 5V, so it works with the microcontroller that has a 3.3V and 5V output to power the other components.

## Microcontroller – UART – GPS module

### Overview

According to F2 of the functional specifications the box needs to track its own position and as can be seen in T2 of the technical specifications (Inc, Functional Design, 2025) the client requires the module for that to be the ATGM336H.

### Working principle

The coordinates as well as additional information like altitude are acquired by the patch antenna and then communicated to the microcontroller using a UART interface. They are sent in form of a NMEA sentence (Gakstatter, 2015) made of ASCII characters. The important information in a NMEA sentence is separated by commas.

### Specifications

* **Frequency and accuracy**: The module acquires positional data from satellites and transmits it to the microcontroller every second with an accuracy of a 10m radius around the box.
* **UART setup:** The communication uses the lpuart2 peripheral of the FRDM-MXCA153 board and is set to a baud rate of 9600 bits per second with one stop bit and no parity to match the GPS modules specification that can be found in its datasheet
* **Pins:** RX and TX
* **Operating Voltage**: 3.3V (compatible with FRDM-MCXA153).

### Sequence diagram:

The software for receiving positional data is interrupt based. First an initialization of the lpuart2 peripheral, which was chosen because of it having pin headers on top of the board, takes place. This includes the set-up of its interrupt request handler for receiving data. As soon as that is triggered by the module sending processed and formatted data byte by byte the microcontroller puts it in a queue. After that it gets moved from this queue to a buffer which can then be parsed.

A diagram of a computer program

AI-generated content may be incorrect.

Figure 3: Sequence diagram of the GPS module interface

## Microcontroller – Color sensor

### Overview

According to functional specifications F3.3 (Inc, Functional Design, 2025), the Reverse Geocaching Box must include a color-dependent puzzle at one of its locations. Specifically, as detailed in F3.3.1 (Inc, Functional Design, 2025), the puzzle involves detecting different colors presented by the user, each triggering distinct hints or information related to navigating toward the next target location. The module chosen for that was the TCS34725 due to previous experience with the I2C interface.

### Working Principle

The color sensor used is the TCS34725, which operates by measuring the intensity of reflected red, green, blue, and clear (unfiltered) light from an illuminated surface using an integrated IR blocking filter and two built-in LEDs. The sensor communicates with the microcontroller via the I2C communication protocol.

### Specifications

* **Interface**: I2C
* **Operating Voltage:** 3.3V (compatible with FRDM-MXCA153 microcontroller)
* **Operating principle:** RGB digital output with using a sensor with integrated IR filter and LED illumination

### Sequence Diagram

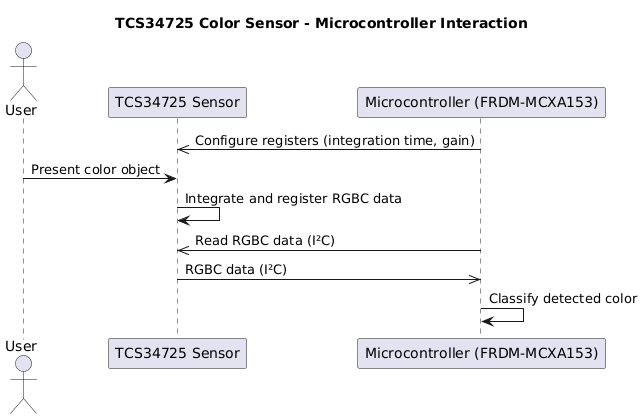


Figure 4: Sequence diagram of the color sensor

## Microcontroller – Tilt sensor

### Overview

The SW520D module has been chosen to fulfil F3.4 of the functional specifications (Inc, Functional Design, 2025) which is the balancing game. Since it only detects tilts on one axis two modules will be placed in a 90-degree angle to each other.

### Working principle

The module is based on 2 balls in a tube. Once the module is tilted far enough the balls connect which leads to them conducting current. While the current is flowing the module sends a signal to the microcontroller.

### Specifications

* **Axes**: Detects tilt on one axis, determined by the modules orientation.
* **Pins:** 1 GPIO
* **Operating Voltage**: 3.3V to 5V (compatible with FRDM-MCXA153).

### Sequence diagram

After initializing the pin needed for the module the sensor sends a HIGH voltage signal to the microcontroller when it detects a tilt and a LOW voltage signal when it does not.

A diagram of a computer program

AI-generated content may be incorrect.

Figure 5: Tilte sensor sequence diagram

## Microcontroller – Temperature sensor

### Overview

According to F5.2 of the Functional Design and T5.2 of the Technical Specifications (Inc, Functional Design, 2025)

, the Reverse Geocaching Box must log the external temperature every 10 seconds and store it in persistent memory.

The chosen temperature sensor is the ASAIRAHT21B, which communicates with the FRDM-MCXA153 microcontroller via the I2C protocol. This sensor ensures precise environmental temperature measurements, critical for logging and user feedback during gameplay.

### Working Principle

The ASAIR AHT21B is a digital humidity and temperature sensor with a calibrated output. For this application, only the temperature functionality is utilized.

The sensor uses an I2C interface to transmit temperature data to the microcontroller. After initialization, the microcontroller sends a measurement command, triggering the sensor to perform a temperature reading.

The sensor then returns a 16-bit digital value representing the ambient temperature, which is converted to degrees Celsius (°C) using the formula specified in the sensor’s datasheet.

### Specifications

* **Interface**: I2C Communication Protocol (Address: **0x38**).
* **Operating Voltage**: 3.3V (compatible with FRDM-MCXA153).
* **Measurement Range**: -40°C to +85°C.
* **Accuracy**: ±0.3°C (typical).ss
* **Update Interval**: Temperature is sampled and logged every 10 seconds (as mentioned in **T5.2**).

### Sequence Diagram:

The I2C peripheral of the FRDM-MCXA153 is configured with a clock speed of **100 kHz** (standard mode) and the sensor is initialized by sending a soft-reset command (0xBA – Datasheet Retrieved) to ensure it enters a known state.

The microcontroller sends a measurement command (0xAC) to the sensor and the sensor performs a temperature conversion (typical delay: **80 ms**).

The microcontroller reads 6 bytes of raw data from the sensor’s I2C register and, by taking the raw data from bytes [3], [4], and [5], then converted to °C using the following formula:

T°C=(raw\_value220×200−50)

Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente

Figure 6: Sequence diagram of the temperature sensor module interface

## Microcontroller – Joystick

### Overview

According to F3.2 of the functional specifications (Inc, Functional Design, 2025), the geocache box must include a memory game at one of the locations. As specified in F3.2.1 and F3.2.2, (Inc, Functional Design, 2025)

The game involves displaying a sequence of arrows on a screen, which the player must replicate using a joystick.

The joystick serves as the primary input device for user interaction. It is u

sed to navigate through menus, replicate puzzle sequences, and confirm selections. The FRDM-MXCA153microcontroller is responsible for reading joystick inputs and processing them accordingly.

### Working Principle

The joystick is an analog input device that measures movement along two axes (X and Y) and includes a button for selections. The microcontroller reads these inputs and determines the user’s intended actions.

### Specifications

* **Axes:** The joystick operates on two analog axes (X and Y)
* **Pins Axes:** The joystick position is read via 2 GPIO pins set to ADC (Analog-to-Digital Converter)
* **Pins button**: The joystick button (SW) is connected to a GPIO digital input
* **Operating voltage:** 5V

### Sequence Diagram:

The joystick input is handled using interrupts, allowing for quick response times and minimal processing delay. Movements are detected through ADC readings, while button presses trigger GPIO interrupts.

A diagram of a computer program

AI-generated content may be incorrect.

Figure 7: Sequence diagram of the Joystick module interface

## Microcontroller – LCD-Display

### Overview

The LCD-Display works to display the information that we as the builders have specified whether it be the puzzles or relevant information that is used for the game as specified in F4 of the functional design (Inc, Functional Design, 2025).

### Working principle

The LCD display works using the I2C interface. The PCB attached to the LCD sends the data to the actual LCD pins byte by byte. Thes bytes either represent an ASCII character to be printed or a command like clearing the screen.

### Specifications

* **Size**: 4 rows , 20 columns (4x20)
* **Interface**: I2C
* **Operating voltage:** 5V

### Sequence Diagram:

After an initialization the screen displays ASCII symbols using a print function or executes LCD commands like clear if prompted by the microcontroller.

A diagram of a computer screen

AI-generated content may be incorrect.

Figure 8: Sequence diagram of the LCD-Display module interface

## Microcontroller – Numpad

### Overview

As stated in F3.5.1 of the functional specifications (Inc, Functional Design, 2025), the geocache box needs to let the user enter a number code to unlock it at the final location. While the spec doesn’t say how that input should be done, we chose a 3x3 matrix numpad because it’s simple, reliable, and easy to use.

The numpad is connected to the FRDM-MCXA153 development board, which reads the inputs, checks if the entered code is correct, and then controls the unlocking mechanism.

### Working principle

The numpad has 9 buttons arranged in 3 rows and 3 columns. When a button is pressed, it connects a row to a column. The microcontroller scans the numpad by interrupts getting triggered if the buttons are pressed. Depending on the pressed button the corresponding number will be stored in a buffer. The software also uses a millis based debounde to prevent accidental double pressing.

### Specifications

* **Type:** 3x3 matrix keypad (keys 0–9)
* **Connections:** 9 GPIO inputs with pull-up resistors
* **Scanning Method:** Software based polling
* **Debouncing:** Done in software to prevent pressing multiple times by accident

### Sequence Diagram:

The buttons work based on interrupts so once a button is pressed its corresponding number is put in a buffer and the button is debounced.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 9: Sequence diagram of the Numpad module interface

## Microcontroller – MicroSD Card

### Overview

According to F5.1, F5.2 and F5.3 (Inc, Functional Design, 2025)

the Reverse Geocaching Box must log certain information such as GPS coordinates, temperature, and puzzle progress in persistent memory that does not clear once the power is turned off. To fulfill this requirement, the MicroSDHC-I card is used. The communication interface used between the microcontroller and the microSD module is the Serial Peripheral Interface (SPI) bus, storing all log data and allowing retrieval when the box is connected to a companion laptop application.

### Working Principle

The MicroSD-Card will be written to using the Generic FAT Filesystem Module(FATFS) and SPI communication. Using the filesystem computer readable files like csv can be created and written to. After they are opened the SPI interface can transmit data blocks of 512 bytes in which the data to be logged is stored to the file every 10 seconds. For reading it data blocks of the size of a multiple of 4 can be read from the card one by one.

### Specifications

* **Storage**: 16GB
* **Transmission speed**: Up to 98MB/s
* **Communication protocol**: SPI
* **File system module**: FATFS
* **Operating voltage:** 3.3V

### Sequence diagram

First the SD card is initialized which includes pins as well as putting it in SPI mode. After the FATFS is used to open/create a file using the proper parameters. Then either F\_Write or F\_Read can be called depending on the wanted action. Under the hood the SD-Driver uses SPI functions like SPI\_transmitBlock or SPI\_receiveBlock that are defined in another file. After the action is done the file gets closed again.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 10: SD-Card sequence diagram

## Microcontroller – MOSFET – Solenoid

### Overview

In F1.1 and F1.1.2 of the functional design (Inc, Functional Design, 2025)

it is specified that an actuator must be used to lock and unlock the box, the chosen component is the JF-0530B Solenoid. It was chosen as its pin being pushed out by a spring in its default state leads to it not needing to get powered constantly. Additionally, it is simple to control with a MOSFET since it is voltage controlled. For that the IRLZ34n MOSFET has been chosen since its datasheet matches the needs in terms of gate threshold voltage and source-drain breakdown levels.

### Working principle:

The solenoid has an internal resistance that makes it pull enough current to operate when it receives the specified 6V of voltage.

The MOSFET will connect drain and source when it receives logic level voltage to the gate.

### Specifications

* **MOSFET specifications:** The MOSFET is able to be controlled with logic level voltage between 1.7V to 3.6V and is able to handle 6V of source-drain voltage and 1A source-drain current while leaking less than 300mA of current.
* **MOSFET pins:** 1 GPIO pin set to digital output
* **Operating voltage:** 6V

### Sequence Diagram:

The control is done using one GPIO pin which is connected to the gate. When it is set to HIGH, its output is between 1.7V to 3.3V which is enough to activate the gate and let the 6V and 1A through to the solenoid. Once it is cleared and thereby set to LOW again the gate is in the cutoff region not letting any current through or Voltage over since the connection will no longer be grounded.

A diagram of a computer component

AI-generated content may be incorrect.

Figure 11: Solenid sequence diagram

# Software

The software is designed in a cyclic executive way with interrupts handling events. First the needed peripherals needed for the components as well as for the smooth operating of the software are initialized including interrupt handlers. Then the main loop starts in which the changeable settings like target locations as well as buffers that need to be cleared get initialized for this run. Then it goes through the games taking their nature into account of whether they start once a location is reached or whether their point is to reach a certain location. Once the games are solved the code to unlock the box is executed and it will start the laptop communication. After that has been successfully finished the main loop repeats and initializes the now potentially changed settings. If the final game is not solved correctly, it will go back to the settings initialization The interrupts mainly play a role for the game and display functions but for example the reset button uses a pin interrupt routine.

A diagram of a function

AI-generated content may be incorrect.

Figure 12: Software flowchart

# Realization

## Hardware

##### LCD Display

The hardware used in this part of the project are specifically curated for the NXP FRDM-MCXA153 development board and uses a 16x2 LCD display that has an I2C connection with the address 0x27.

### Joystick

The joystick module contains two analog axes (VRx and VRy) connected to ADC channels, and a digital push-button (SW) connected to a GPIO input. The axes give voltage signals based on position, while the button is active-low and triggered by physical press.

The lowest level of all this is the joystick\_init() function in joystick.c, which sets up both ADC0 and GPIO1 peripherals for use.

### PCB

## Software

In this section the implementation of the codes will be explained using parts from the source code that highlight the most important parts of why it works.

### Main

### GPS/GPS Games

The GPS module has been implemented using UART communication and is responsible for gathering the box’s location which is the used to calculated distances etc.. The files related to this are gps.c, gps.h, gpsGame.c, gpsGame.h, fifo.c, fifo.h, lpuart2\_interupt.c and lpuar2t\_interrupt.h.

The lowest level of all this is lpuart2 which is set up to generate hardware interrupts upon receiving code. This is then handled by an interrupt handler which sends the received char to a receiving first-in-first-out(fifo) datastructure.

// Data received?

**if**((LPUART0->STAT & LPUART\_STAT\_RDRF\_MASK) != 0)

{

// Read data

c = (uint8\_t)(LPUART0->DATA);

// Put in receive FIFO

**if**(!f\_push(&rx, c))

{

// Error: receive FIFO full!!

// Should not happen, so freeze the system. Update FIFO size to

// match your application.

**while** (1)

{}

}

This fifo can be used using the lpuart2\_getchar() which simply pops the oldest thing in there and returns it, so everything gets taken out in the order it was received.

**int** **lpuart0\_getchar**(**void**)

{

uint8\_t c=0;

// Wait for data.

// If waiting is not desired, call the function lpuart0\_rxcnt() first to

// make sure data is available.

**while**(!f\_pop(&rx, &c))

{}

**return** (**int**)c;

}

This is being used by updatePosition in gps.c, however in order not to get stuck in a waiting loop for the next char it only gets called as long as the fifo is not empty.

Then it filters each of the GPS sentences sent via UART for the GNGGA one which contains the information we need in. This is done by saving each of the sentences in a buffer and using strcmp to check if the first 6 chars are the correct ones.

**void** **updatePosition**()

{

**while**(lpuart2\_rxcnt() > 0)

{

**char** c = (**char**)lpuart2\_getchar();

strncat(buffer, &c, 1);

**if**( c == '\n')

{

**if**(strncmp(buffer, "$GNGGA",6) == 0)

{

parseNMEA(buffer, &boxCoordinates);

**float** d = distance(&testCoordinates,&targetCoordinates);

**float** b = calculateBearing(&testCoordinates,&targetCoordinates);

buffer[0] = '\0'; //clear buffer

}

**else**

{

buffer[0] = '\0'; //clear buffer

}

}

}

}

Once that character has been found a parsing function gets called. This seperates the elements of the string on commas and assigns them to a struct containing the boxes longitude and latitude as well as variables stored in static memory. This function uses strsep() which is part of string.h which does not properly include so the function has been copied from its documentation.

**while** ((token = strsep(&ptr, ",")) != NULL) {

**switch**(field)

{

**case** 2:

lat = atof(token);

**break**;

**case** 3:

latDir = token[0];

**break**;

**case** 4:

lon = atof(token);

**break**;

**case** 5:

lonDir = token[0];

**break**;

**case** 6:

fix = atoi(token);

**break**;

**case** 7:

nrSat = atoi(token);

**break**;

}

field++;

token = strsep(&buffer, ",");

}

**if** (lat && lon && latDir && lonDir)

{

boxCoordinates->lon = convertToDecimal(lon,lonDir);

boxCoordinates->lat = convertToDecimal(lat,latDir);

boxCoordinates->lonDir = lonDir;

boxCoordinates->latDir = latDir;

}

Using the current GPS data that has now been saved in a struct through a getter, gpsGame.c can compare it to a predefined target location.

It can either calculate the distance to the target using the haversine formula which accounts for the earths curvature.

**float** **distance**(coordinates\_t\* loc1, coordinates\_t\* loc2) { //Haversine distance formula

**float** differenceLat = degToRad(loc2->lat - loc1->lat);

**float** differenceLon = degToRad(loc2->lon - loc1->lon);

**float** a = sin(differenceLat / 2) \* sin(differenceLat / 2) +

cos(degToRad(loc1->lat)) \* cos(degToRad(loc2->lat)) \*

sin(differenceLon / 2) \* sin(differenceLon / 2);

**float** c = 2 \* atan2(sqrt(a), sqrt(1 - a));

**float** d = EARTH\_RADIUS \* c; // Distance in meters

**return** d;

}

Or the bearing towards that target.

**float** **calculateBearing**(coordinates\_t \*loc1, coordinates\_t\* loc2) { //formula to calculate bearing/compass direction of the second coordinate from the first in degrees and sort it into actual directions

**float** lat1 = degToRad(loc1->lat);

**float** lon1 = degToRad(loc1->lon);

**float** lat2 = degToRad(loc2->lat);

**float** lon2 = degToRad(loc2->lon);

**float** differenceLon = lon2 - lon1;

**float** x = sin(differenceLon) \* cos(lat2);

**float** y = cos(lat1) \* sin(lat2) - sin(lat1) \* cos(lat2) \* cos(differenceLon);

**float** bearing = atan2(x, y);

bearing = radToDeg(bearing); // Convert to degrees

**if** (bearing < 0) {

bearing += 360; // Normalize to 0-360 degrees

}

**return** bearing;

}

The layered design of everything concerning the GPS module ultimately allows for easy use through top level functions like distance() that can then be used for victory conditions.

### SD-Card/Logging

### Solenoid/Unlocking

### Color Sensor

### Temperature Sensor

### Joystick

**void** **joystick\_init**(**void**)

{

// Enable clocks for PORT1, GPIO1, ADC0

MRCC0->MRCC\_GLB\_CC0\_SET = MRCC\_MRCC\_GLB\_CC0\_PORT1(1) | MRCC\_MRCC\_GLB\_CC0\_ADC0(1);

MRCC0->MRCC\_GLB\_CC1\_SET = MRCC\_MRCC\_GLB\_CC1\_GPIO1(1);

MRCC0->MRCC\_GLB\_RST0\_SET = MRCC\_MRCC\_GLB\_RST0\_PORT1(1) | MRCC\_MRCC\_GLB\_RST0\_ADC0(1);

MRCC0->MRCC\_GLB\_RST1\_SET = MRCC\_MRCC\_GLB\_RST1\_GPIO1(1);

// Configure Joystick SW pin as GPIO input with pull-up

PORT1->PCR[7] = PORT\_PCR\_LK(1) | PORT\_PCR\_IBE(1) | PORT\_PCR\_PE(1) | PORT\_PCR\_PS(1) | PORT\_PCR\_MUX(0);

// Falling edge interrupt on P1\_07

GPIO1->ICR[7] = GPIO\_ICR\_ISF(1) | GPIO\_ICR\_IRQC(0b1010);

**This function enables clocks and resets for the peripherals needed to interface with the joystick.**

* The analog axes (VRx and VRy) use **ADC0**, so it is powered and initialized.
* The joystick button (SW) is connected to **PORT1 pin 7**, set up as an input with **pull-up resistor** and **interrupt on falling edge**.

**Reading Analog Movement (VRx and VRy)**

The joystick's analog movement is handled using two separate ADC channels — one for each axis. The VRx axis is connected to ADC **channel 10**, and VRy is connected to **channel 11**.

// Joystick VRx

uint16\_t **joystick\_vrx**(**void**)

{

ADC0->STAT |= ADC\_STAT\_TCOMP\_INT\_MASK;

ADC0->TSTAT |= ADC\_TSTAT\_TCOMP\_FLAG(0b0001);

ADC0->CMD[0].CMDL = ADC\_CMDL\_MODE(1) | ADC\_CMDL\_ADCH(10);

ADC0->TCTRL[0] = ADC\_TCTRL\_TCMD(0b001);

ADC0->SWTRIG = ADC\_SWTRIG\_SWT0(1);

uint32\_t timeout = 10000;

**while** (((ADC0->STAT & ADC\_STAT\_TCOMP\_INT\_MASK) == 0) && --timeout) {}

ADC0->STAT |= ADC\_STAT\_TCOMP\_INT\_MASK;

timeout = 10000;

**while** (((ADC0->TSTAT & ADC\_TSTAT\_TCOMP\_FLAG\_MASK) == 0) && --timeout) {}

ADC0->TSTAT |= ADC\_TSTAT\_TCOMP\_FLAG(0b0001);

**return** (uint16\_t)(ADC0->RESFIFO);

}

// Joystick VRy

uint16\_t **joystick\_vry**(**void**)

{

ADC0->STAT |= ADC\_STAT\_TCOMP\_INT\_MASK;

ADC0->TSTAT |= ADC\_TSTAT\_TCOMP\_FLAG(0b0001);

ADC0->CMD[0].CMDL = ADC\_CMDL\_MODE(1) | ADC\_CMDL\_ADCH(11);

ADC0->TCTRL[0] = ADC\_TCTRL\_TCMD(0b001);

ADC0->SWTRIG = ADC\_SWTRIG\_SWT0(1);

uint32\_t timeout = 10000;

**while** (((ADC0->STAT & ADC\_STAT\_TCOMP\_INT\_MASK) == 0) && --timeout) {}

ADC0->STAT |= ADC\_STAT\_TCOMP\_INT\_MASK;

timeout = 10000;

**while** (((ADC0->TSTAT & ADC\_TSTAT\_TCOMP\_FLAG\_MASK) == 0) && --timeout) {}

ADC0->TSTAT |= ADC\_TSTAT\_TCOMP\_FLAG(0b0001);

**return** (uint16\_t)(ADC0->RESFIFO);

}

* Each function **triggers a single ADC conversion** for a specific joystick axis.
* ADC\_CMDL\_ADCH(x) selects the ADC input channel (10 for VRx and 11 for VRy).
* SWTRIG starts the conversion manually.
* A **timeout loop** waits for the conversion to complete by checking interrupt flags.
* The result is then returned from the ADC’s **result FIFO** as a 16-bit integer.

These functions are non-blocking and can be polled as needed in the main game loop to get the joystick’s live position.

**Button Press Detection (SW)**

The joystick’s built-in button is connected to a GPIO pin and detected using an **interrupt-based counter**, which increments on each press and is later checked using joystick\_sw().

**void** **GPIO1\_IRQHandler**(**void**)

{

NVIC\_ClearPendingIRQ(*GPIO1\_IRQn*);

**if** ((GPIO1->ISFR[0] & GPIO\_ISFR\_ISF7(1)) != 0) {

GPIO1->ISFR[0] = GPIO\_ISFR\_ISF7(1);

joystick\_sw\_press\_count++;

}

}

// Joystick SW press

bool **joystick\_sw**(**void**)

{

**if** (joystick\_sw\_press\_count > 0) {

joystick\_sw\_press\_count--;

**return** true;

}

**return** false;

}

* An **interrupt handler** is triggered on button press.
* A **counter** (joystick\_sw\_press\_count) is incremented and later checked using joystick\_sw().
* This makes the button read **non-blocking** and usable inside the main loop without delay.

**Using the Joystick in the Game (main.c)**

Direction **getJoystickDirection**(**void**) {

**int** x = joystick\_vrx();

**int** y = joystick\_vry();

**if** (x > 60000 && y < 55000) **return** *RIGHT*;

**if** (x < 5000 && y < 55000) **return** *LEFT*;

**if** (y < 1000 && x < 55000) **return** *UP*;

**if** (y > 60000 && x < 55000) **return** *DOWN*;

**return** *NONE*;

}

This function checks joystick movement direction using thresholds.

* It translates **raw ADC values** from the joystick into logical directions: UP, DOWN, LEFT, or RIGHT.
* A **neutral zone** is enforced, so small hand movements or electrical noise don't trigger false inputs.
* The output of this function is used to **compare against the direction shown on the LCD**, allowing the game to validate whether the player's reaction was correct.

### LCD Display

##### Software

The software that was used in this program

# Testing Phase

# 5.1 Component Testing

In this section the tests done to confirm the components function as expected are explained and results are documented.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Testnr | Component | Test description | Expected Result | Pass/Fail |
| CT1 | GPS-Module | Power GPS module; Connect logic analyser to RX and Ground | Receives data in UART format | Pass |
| CT1.1 | GPS-Module | Power GPS module; Print the received data byte by byte | NMEA sentence printed character by character | Pass |
| CT2 | Solenoid | Slowly increase voltage till 6V then decrease it again | Pulls pin back at 6V; Releases pin around 1.8V | Pass |
| CT3 | Joystick | Power Joystick module; Press Joystick button | Receive an input signal when the button is pressed | Pass |
| CT3.1 | Joystick | Power Joystick module; Move Joystick around 45° at a time; Print analog values | Print analog values corresponding to the directions | Pass |
| CT4 | Color sensor | Power color sensor module; Send measurement signal; Print RGB values received via I2C | Prints an RGB value. | Pass |
| CT4.1 | Color sensor | Repeat steps from T4 three times; Present the sensor a red, a green and a blue object in this order. | Prints an RGB value corresponding to the color shown. | Pass |
| CT5 | Tilt sensor | Power tilt sensor module; Tilt module above threshold and then shift it back; continuously print output; | HIGH output while above threshold; LOW output while below threshold; |  |
| CT6 | Temperature sensor | Power temperature sensor module;  Receive raw temperature data via I2C;  Print received data; | Print raw temperature data | Pass |
| CT6.1 | Temperature sensor | Power temperature sensor module;  Receive raw temperature data via I2C;  Convert raw data to Celsius  Print temperature in Celsius; | Print a realistic temperature for the test environment(around room temperature (20°C)) | Pass |
| CT7 | SD Card | write data |  | Pass |
| CT7.1 | SD Card | Reads data and prints to terminal |  | Pass |
| CT7.2 | SD Card |  |  |  |
| CT7.3 | SD Card |  |  |  |
| CT8 | LCD-Screen | Power LCD-Screen; Print a character on all 32 available spaces; Clear the screen after a 5 second delay; | LCD-Screen prints the chosen character on every space and then clears the screen after 5 seconds. | Pass |

In this section the subsystems made multiple components are tested for their functionalities working like expected. Examples are the games as well as the logging function.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Testnr | Functionality being tested | Components in use | Test description | Expected/Result | Pass/Fail |
| ST1 | F1.1, F1.1.1  (Locking and unlocking) | MOSFET,  Solenoid | Set pin connected to MOSFET gate to HIGH; Delay one second; Set pin connected to MOSFET gate to low | Solenoid retracts for one second |  |
| ST2 | F2, F2.2, F2.2.2  (Compares its location to target location) | GPS-Module, LCD-Screen | Walk towards target location;  Wait for 30 seconds;  Walk 10 more meters | LCD-Screen prints message informing of arrival once the user is at the target location;  After walking the 10 meters the message stops displaying; |  |
| ST2.1 | F2, F2.2, F2.2.1, F2.2.2  (Compares its location to target location handling multiple target locations) | GPS-Module, LCD-Screen | Walk to the target locations in reverse order; Walk to the target locations in correct order; Wait for 30 seconds at every location | The LCD-Screen will display nothing until the location meant be reached first is reached; Then the screen will show information about arrival at the correct location for the duration of the stay at every location; |  |
| ST2.2 | F2, F2.1, F2.2, F.2.2.1, F2.2.2, F2.2.3 | GPS-Module, LCD-Screen | Walk to the target locations in order; | LCD shows distances to target location with a refresh rate of 1 second; At target location information of arrival will be displayed for five seconds; Distance to next target location will be shown; If there are no more locations the screen prints nothing |  |
| ST2.3 | F2, F2.1, F2.2, F2.2.1, F2.2.2 F2.2.4 | GPS-Module, LCD-Screen |  | LCD shows compass direction of target location with a refresh rate of 1 second; At target location information of arrival will be displayed for five seconds; Compass direction of next target location will be shown; If there are no more locations the screen prints nothing |  |
| ST3 |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |