System Design Document

<simplified version>

Afbeelding met persoon, binnen

Automatisch gegenereerde beschrijving

**Project name: WarePro**

**Team name:** MBT inc.

**Author(s):**

*Miroslav Denkov*

*Plamen Tsokov*

*Simeon Stoilov*

*Veselin Atanasov*

*Mykyta Matiushchenko*

**Document date/version:** *2024-03-01 version 1.0*

**Document status:** *Draft*

# Content

[1 Content 1](#__RefHeading___Toc2932_2186009179)

[2 List of tables 1](#__RefHeading___Toc2934_2186009179)

[3 List of figures 1](#__RefHeading___Toc2936_2186009179)

[4 General info 3](#__RefHeading___Toc2938_2186009179)

[5 Introduction 4](#__RefHeading___Toc2940_2186009179)

[6 System overview 4](#__RefHeading___Toc2942_2186009179)

[6.1 System Context Diagram(s) 4](#__RefHeading___Toc2944_2186009179)

[6.2 Problem analysis 5](#__RefHeading___Toc2946_2186009179)

[7 Project requirements 5](#__RefHeading___Toc2948_2186009179)

[7.1 Actors 5](#__RefHeading___Toc2950_2186009179)

[7.2 Scenario’s 5](#__RefHeading___Toc2952_2186009179)

[7.3 Use cases 5](#__RefHeading___Toc2954_2186009179)

[8 Communication protocol 6](#__RefHeading___Toc2956_2186009179)

[9 Hardware design 7](#__RefHeading___Toc2958_2186009179)

[10 Embedded software Design 7](#__RefHeading___Toc2960_2186009179)

[11 PC software Design 8](#__RefHeading___Toc2962_2186009179)

[12 Appendix A detailed use cases 10](#__RefHeading___Toc2964_2186009179)

[13 Appendix: relevant tools 10](#__RefHeading___Toc2966_2186009179)

# List of tables

[Table 1: Document history 2](#_Toc125839984)

[Table 2: Terms & abbreviations 2](#_Toc125839985)

[Table 3: main features 3](#_Toc125839986)

[Table 4: example commands overview, replace your command set here 4](#_Toc125839987)

[Table 5: Tools to create various diagrams 6](#_Toc125839988)

# List of figures

[Figure 1: an example of a “system context” diagram, replace your diagram here 4](#_Toc125839977)

[Figure 2: example or command syntax, replace your syntax here 4](#_Toc125839978)

[Figure 3: an example of a “sequence” diagram, replace your diagram here 5](#_Toc125839979)

[Figure 4: an example of a “system hardware modules” diagram, replace your diagram here 5](#_Toc125839980)

[Figure 5: an example of a “state machine” diagram, replace your own diagram here 6](#_Toc125839981)

# General info

Table 1: Document history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Description** | **Status** |
| **0.1** | Plamen | 2023-01-28 | Sprint 0 add information | Proposed |
| **0.2** | Miroslav | 2023-03-18 | Add context diagram,  component diagram,  use cases(from user perspective) | Proposed |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0.3** | Miroslav | 2023-04-10 |  | Proposed |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0.4** | Miroslav | 2023-05-06 |  | Proposed |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0.5** | Miroslav | 2023-05-09 | Added final design diagram | Proposed |
| 0.6 | Miroslav | 2023-05-31 | Added testing strategies | Proposed |

Table 2: Terms & abbreviations

|  |  |
| --- | --- |
| **Terms & abbrev’s** | **Description** |
| **Aka** | Also known as |
| **DOTF** | Development Oriented Triangulation Framework |
| **Duration** | Number of hours/days it will last until your activity is completed, aka lead time |
| **Effort** | Number of hours/days you will really spent to execute an activity |
| **Ex.** | Example |
| **MoM** | Minutes of Meeting / Meeting notes |
| **MoSCoW** | Acronym which defines priorities: Must - Could - Should - Would |
| **PID** | Project Initiation Document (=PP) |
| **PoC** | Proof of Concept (=prototype) |
| **PP** | Project Plan (=PID) |
| **SDD** | System Design Document |
| **Sprint** | SCRUM period of 2 or 3 weeks |
| **Status - Draft** | The user story is still under construction, no intention to be complete/final |
| **Status - Proposed** | The user story is assumed to be complete/final but still needs approval from customer |
| **Status - Accepted** | The user story is complete/final and approved by customer |
| **US** | User Story |

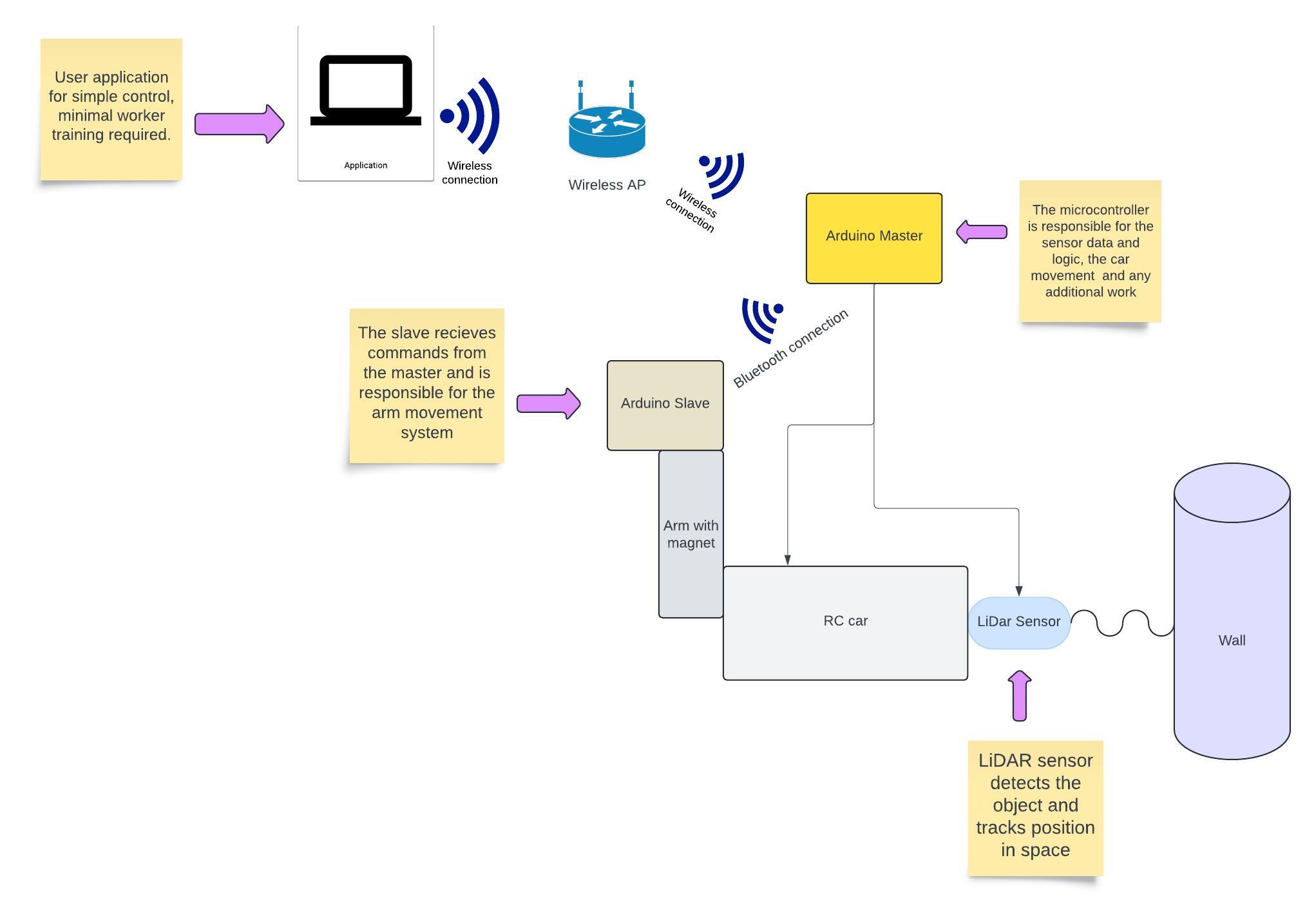
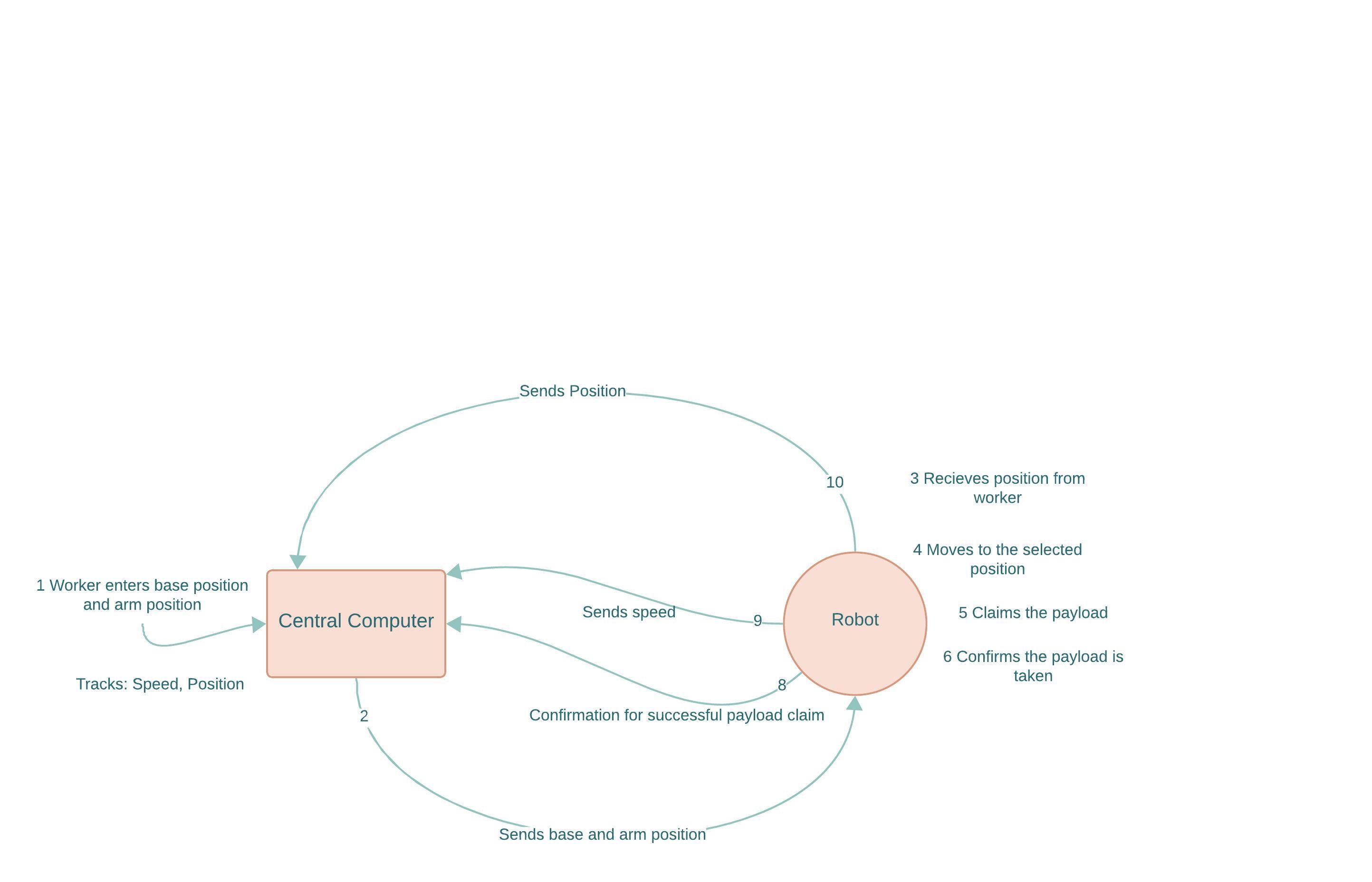
# Introduction

Our goal is to massively improve the efficiency of a warehouse while reducing operating costs. Our idea to achieve this task we have decided to develop a package retrieval robot. The worker in the warehouse will now only have to input which package they need into the computer instead and the robot will go into the warehouse, find the right package, and carry it back. Our device will cut the time it takes to navigate and manage a warehouse and will reduce the cost by replacing work that is usually done by humans. Our system is a step towards the inevitable future of fully automated industry.

Table 3: main features

|  |  |  |
| --- | --- | --- |
| **Id** | **Feature** | **Short description** |
| **1** | Control Robot | Set desired destination. |
| **2** | Robot pick-up | Pick up package from desired destination. |
| **3** | Robot drop-off | Return and drop package off at home location |

## System Context Diagram(s)



## Problem analysis

Rationale: this section is about understanding the technical challenges your team needs to solve.

Answer questions such as: What is problem is the customer facing? What are the sources and consequences of those problems? You could use techniques such as as root-cause analysis here. Tools like five-why's work well.

<your text>

# Project requirements

In this project our main idea is to develop a package retrieval robot that is easily controlled by the user. To achieve this task, we are required to deliver:

1. A robot that can go to a given location in the warehouse, retrieve a payload and return it to its starting point.
2. The code and logic behind the robot's actions.
3. A way for the user to input the location of their wanted package.
4. A shelf that is suitable for the demo.
5. Rail-like system

We will not deliver:

1. Long-term support
2. A fleet of robots
3. Full-sized implementation

## Actors

1. Warehouse worker
2. Warehouse owner

## Scenarios

1. The warehouse worker needs the robot to take cargo from the warehouse.
2. The warehouse owner wants to check the efficiency of the system.

## Use cases

Using our scenarios, we have uncovered which use cases are going to be supported by the system.

1. The employee requires a package. They get onto the computer and enter the package's location. The robot enters the warehouse and arrives at its destination. After picking up the cargo, the robot returns to where it was before.
2. The owner of the warehouse wants to monitor his investment. He examines the robot's statistics, which include delivery time, daily uses, and distance traveled. To decide if it is worthwhile, he then evaluates it against the amount of money he would have to pay the person to complete the task and the amount of time it would take.

# Communication protocol

*Rationale: When two or more sub-systems must interact, they shall use a communication protocol that is bidirectional and is able to cope with communication errors. Three aspects are important when defining a protocol: Command syntax, list of all available commands, dynamic behavior when sending and handling commands and responses.*

Figure 1: example or command syntax, $MOVE|A1|checksum#

The robot and the central server will communicate via wifi, and the microcontrollers that govern the robot's mechanical components will communicate with one another via Bluetooth. Commands are sent by the central computer in the format #COMA0, where A0 stands for "coordinates" and COM for "command." For instance, the server will transmit the order #MOVB1 if I, the user, desire to move the robot to position B1. If the robot receives a command, it will check the checksum(we are doing this to validate the commands), on a success the master recieves an ACK packet, if the checksum is wrong a NACK packet is sent.

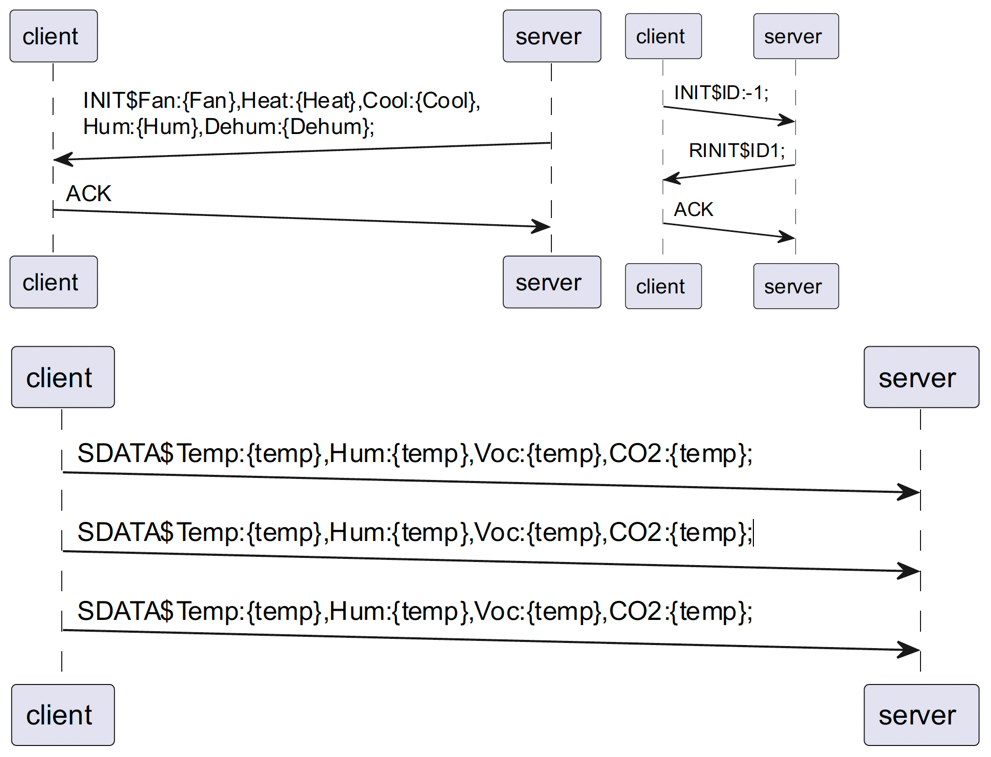


Table 4: example commands overview, replace your command set here

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Purpose** | **Syntax** | **Response** | **Details** |
| ***RES*** | *Reset a sub-module to it default settings* | *#RESET%* | *#ACK%* |  |
| ***MOV*** | *Move an axis to a certain position* | *#MOVE:ID,POS%* | *#ACK%*  *#NACK%* | *ID: identification of axis to be moved*  *POS: target position in [mm]* |
| ***MEASURE*** | *Get the measurement value of a sensor* | *#SENSOR:ID%* | *#ACK,VAL%*  *#NACK%* | *ID: identification of sensor*  *VAL: measurement value returned to sensor* |
| **<your text>** | <your text> | <your text> | <your text> | <your text> |

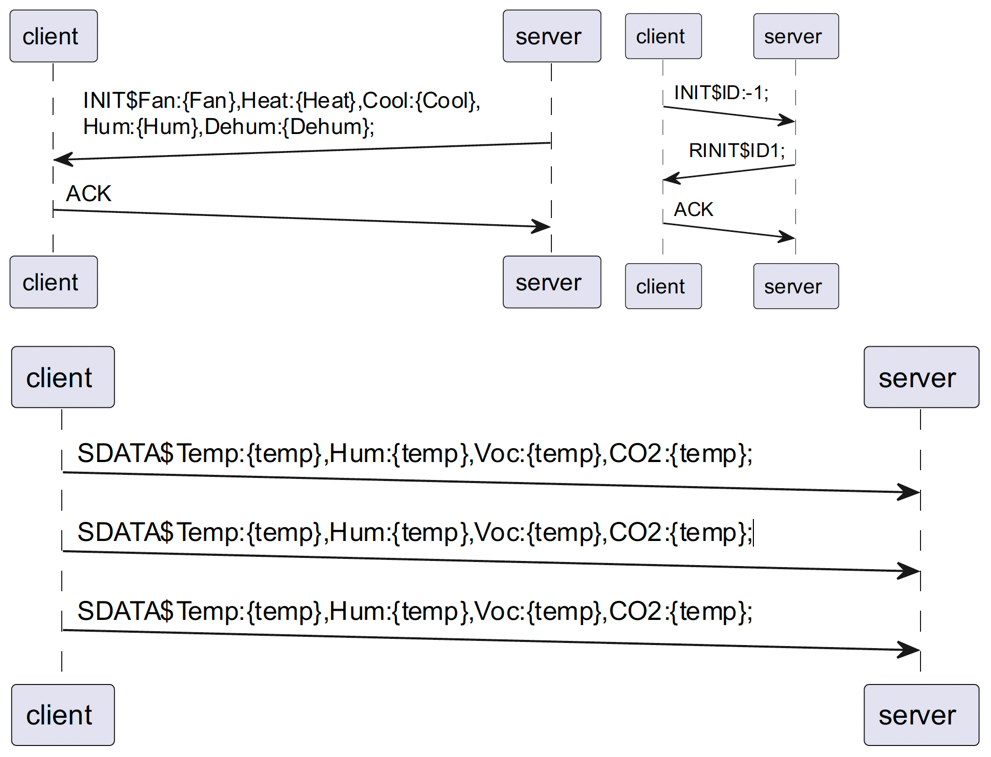


Figure 2: an example of a “sequence” diagram, replace your diagram here

# Hardware design

*Rationale: creating hardware diagrams gives and overview of all components and how they are interconnected. Also add a table to list all pins and IO-types. This indicates whether system is feasible from HW point of view.*

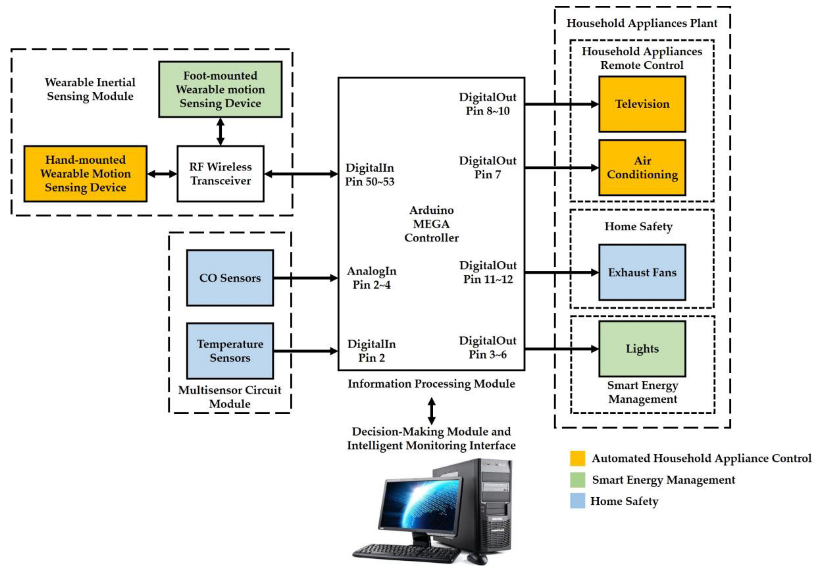


Figure 3: an example of a “system hardware modules” diagram, replace your diagram here

Table 5: an example of a connection table, replace your table here

Afbeelding met tafel

Automatisch gegenereerde beschrijving

# Embedded software Design

Rationale: Proper embedded software design increases the likelihood of a working end-product. Designing flow charts (and state machines if applicable) before creating code is a good engineering practice.

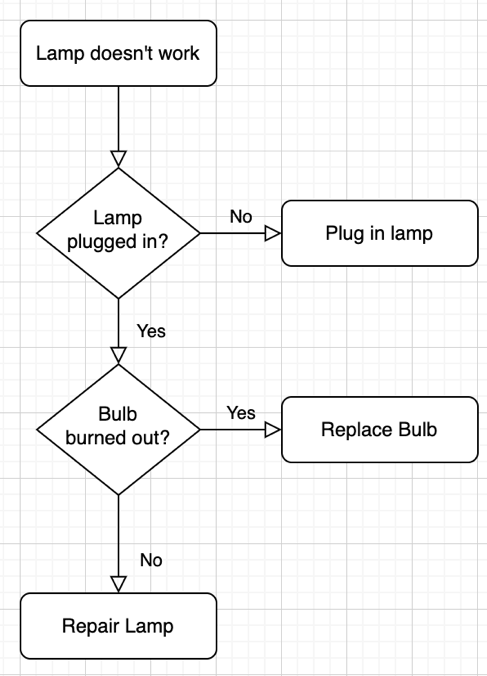


Figure 4: an example of a flow-diagram, replace your own diagram here

<your text to explain details about the flow of your embedded software>

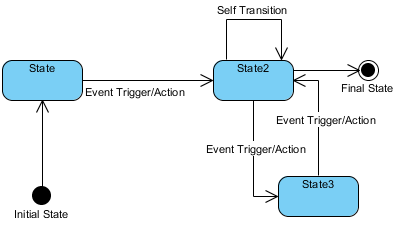


Figure 5: an example of a “state machine” diagram, replace your own diagram here

<your text to explain details about state machine behavior of your embedded software, if applicable>

# PC software Design

Rationale: Proper PC software design increases the likelihood of a working end-product. Designing GUI designs and software class diagrams before creating code is a good engineering practice.

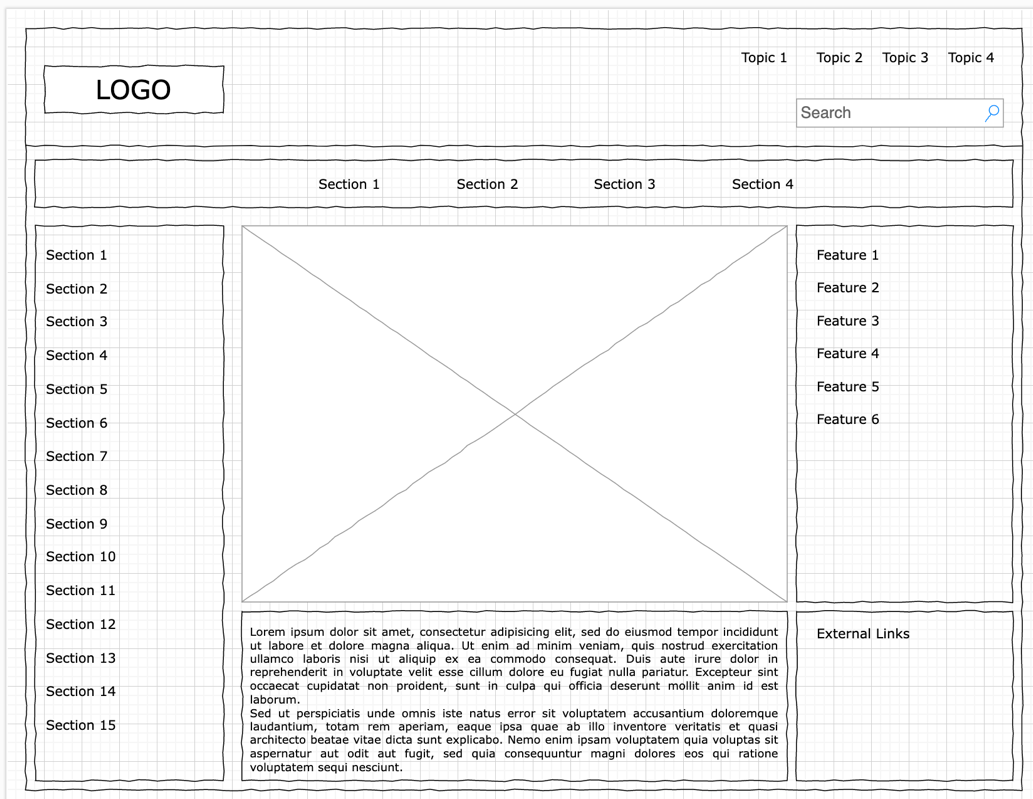


Figure 6: an example of a wire frame diagram, replace your own diagram here

<your text to explain details about the GUI>

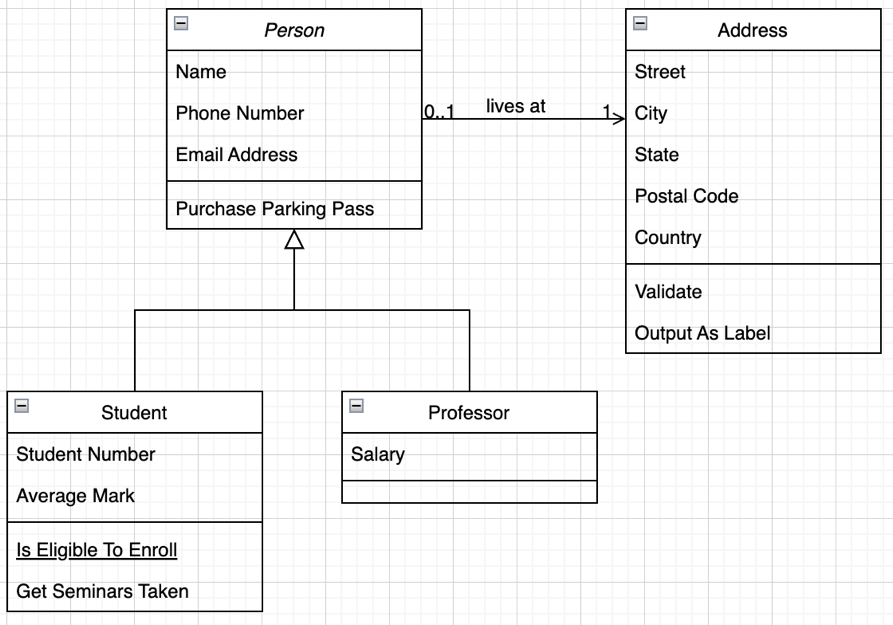


Figure 7: an example of a class diagram, replace your own diagram here

<your text to explain details about the PC software design>

11 Testing Strategies

***C# application:***

1. Ensure that the Application and the client Arduino are able to connect to the same network.
2. Start the server and wait for client’s connection.
3. On a successful connection we get a confirmation in the terminal.
4. Successful command transmission is tested by:
   1. Sending a command from the Application to the Arduino client.
   2. We get a response back from the Arduino client, which may consist of:
      1. ACK response.
      2. Checksum Errors.
      3. If the Arduino has not intercepted the command after it was sent from the server, we expect no response.

Summary:

1. After sending a string from the application, we get an ACK response. This means that everything is connected correctly and the Arduino receives the string.

***RC car:***

1. ***Motor Control Component***

**Objective**: Verify that the motor control code correctly handles motor operations (forward, reverse, stop).

**Steps**:

* 1. Forward Movement Test: Upload a test sketch that continuously runs forwardMotors(). Verify that both motors move the car forward.
  2. Reverse Movement Test: Upload a test sketch that continuously runs reverseMotors(). Verify that both motors move the car backward.
  3. Stop Motors Test: Upload a test sketch that continuously runs stopMotors(). Verify that both motors are stopped.
  4. PWM Control Test: Modify the forwardMotors and reverseMotors functions to control motor speed using PWM signals. Verify that the motors run at different speeds based on the PWM values.

Summary:

1. The forward movement is functional.
2. Reverse movement is functional.
3. The motors can be stopped.
4. PWM control can be done, speed changes.
5. ***LiDAR Component***

**Objective**: Verify that the LiDAR sensor correctly measures distance and reports it via serial communication.

**Steps**:

* 1. Distance Measurement Test: Upload a test sketch that reads distance values from the LiDAR and prints them to the Serial Monitor. Verify that the distance values change appropriately as objects move closer or further from the sensor.
  2. Signal Strength Test: Verify that the signal strength values are reasonable and correlate with the distance measurements.

Summary:

1. The sensor correctly detects an object’s distance and adjusts accordingly based on how far said object is.
2. The strength of the sensor is still under testing. Objects at further distance are hard to read.
3. ***Integration of Motor Control and LiDAR Components***

Objective: Verify that the integrated system correctly uses LiDAR data to control the car's movement.

**Steps**:

* 1. Integration Test: Upload the integrated code. Send a command to move the car to 10 cm from the wall. Verify that the car moves forward and stops when it is 10 cm from the wall. Repeat for 15 cm.
  2. Test Procedure: Place the car at a starting position with no obstacles in front. Send the command $MOVEA2# to the Arduino via the Serial Monitor. Observe that the car moves forward and stops when the LiDAR measures a distance of 10 cm from the wall. Move the car back to the starting position.

Summary:

1. The car moves forward and stops when it is 10 cm from the wall, the same goes for 15cm.
2. The test confirms that the car moves to the desired position. But more work on the connection is needed.

***Embedded Arduino communication:***

1. Conditions
   1. The Bluetooth modules are configured as a master and slave.
   2. The master’s Bluetooth module can only connect to the slave’s MAC address.
   3. Both microcontrollers are working at 38400 baud rate.

Test

1. To test the connection between the two Arduinos, the master will send strings to the slave and the slave will print them out in the serial monitor. ***If the string printed out is the same as the string that was sent, then the test will be successful.***
2. Integration of the communication between the front-end C# application will be done by sending strings over to the master Arduino(as it is the Arduino that handles socket communication) from the C# application and then the master Arduino prints them to the serial monitor and sends the mover to the slave Arduino which prints the string it receives. ***If the strings and commands match the ones sent from the C# app, then the test is complete.***

Summary:

1. After evaluating, the slave prints the same string as the master.
2. The string sent from the C# server, matches the strings in the Arduino boards.

***Test Strategy for Elevation Code Component***

**Objective**:

Verify that the elevation code for the elevator correctly handles the stepper motor movements and the encoder inputs to reach specified shelf positions and toggle between manual and automatic modes.

**Steps**:

**Connection and Initialization Test:**

**Objective**: Ensure the Arduino is correctly set up and the stepper motor is initialized.

**Steps**:

Upload the code to the Arduino.

Open the Serial Monitor.

Verify that the Serial Monitor prints initialization messages and sets the stepper motor to the home position.

**Expected Result**: The Serial Monitor displays initialization messages and confirms that the stepper motor is at the home position.

Shelf Movement Test:

**Objective**: Verify that the elevator correctly moves to the specified shelf positions.

**Steps**:

Send the command shelf1 via I2C.

Observe the stepper motor's movement to the bottom shelf height.

After a delay, verify the stepper motor returns to the home position.

Repeat the process for the command shelf2 to move to the higher shelf height.

**Expected Result**: The stepper motor moves to the specified shelf positions and returns to the home position after a delay.

Manual Mode Activation Test:

**Objective**: Verify that the elevator can switch between manual and automatic modes using the button.

Steps:

Press the button connected to buttonPin.

Observe the Serial Monitor for mode change messages.

Verify the state of manualMode toggles between true and false with each button press.

**Expected Result**: The Serial Monitor displays the correct mode change messages, and the elevator toggles between manual and automatic modes.

Manual Mode Movement Test:

**Objective**: Verify that the encoder inputs correctly control the elevator movement in manual mode.

**Steps**:

Activate manual mode by pressing the button.

Rotate the encoder and observe the stepper motor movement.

Verify that the elevator moves according to the encoder input.

Confirm that the encoder position resets after movement.

**Expected Result:** The elevator moves in response to the encoder inputs and stops correctly when the encoder position resets.

I2C Command Reception Test:

Objective: Verify that the elevator correctly receives and processes I2C commands.

Steps:

Send various commands (shelf1, shelf2) via I2C.

Observe the Serial Monitor for received command messages.

Verify that the stepper motor moves to the correct positions based on the received commands.

**Expected Result**: The Serial Monitor displays the correct received command messages, and the stepper motor moves to the appropriate positions.

**Summary of Results:**

**Initialization Test**: The Arduino and stepper motor are initialized correctly, with the motor moving to the home position.

**Shelf Movement Test**: The elevator successfully moves to both the bottom and higher shelf positions and returns to the home position.

**Manual Mode Activation Test**: The elevator correctly toggles between manual and automatic modes, with appropriate messages displayed on the Serial Monitor.

**Manual Mode Movement Test**: The encoder inputs effectively control the elevator's movement in manual mode, and the position resets as expected.

**I2C Command Reception Test**: The elevator correctly receives and processes I2C commands, with accurate movement to the specified positions.

Overall, the elevation code for the elevator functions as intended, correctly handling both automatic and manual operations and responding appropriately to I2C commands

# Appendix A detailed use cases

|  |  |
| --- | --- |
| **<Use case title>** | |
| **<Diagram goes here.>** | |
|  | |
| **Use case name** |  |
| **Use case number** |  |
| **System** |  |
| **Actors** | 1. |
| 2. |
| 3. |
| **Use case goal** |  |
| **Primary actor** |  |
| **Preconditions** | 1. |
| 2. |
| 3. |
| **Basic flow** |  |
| **Alternative flows** | 1. |
| 2. |
| 3. |

# Appendix: relevant tools

Table 6: Tools to create various diagrams

|  |  |
| --- | --- |
| **Purpose** | **URL** |
| **Create UML diagram** | [Online UML diagram tool | Lucidchart](https://www.lucidchart.com/pages/landing/uml-diagram-software) |
| **PlantUML** | [Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams. (plantuml.com)](https://plantuml.com/) |
| **Various diagrams and charts** | [Diagram Software and Flowchart Maker](https://www.diagrams.net/) |