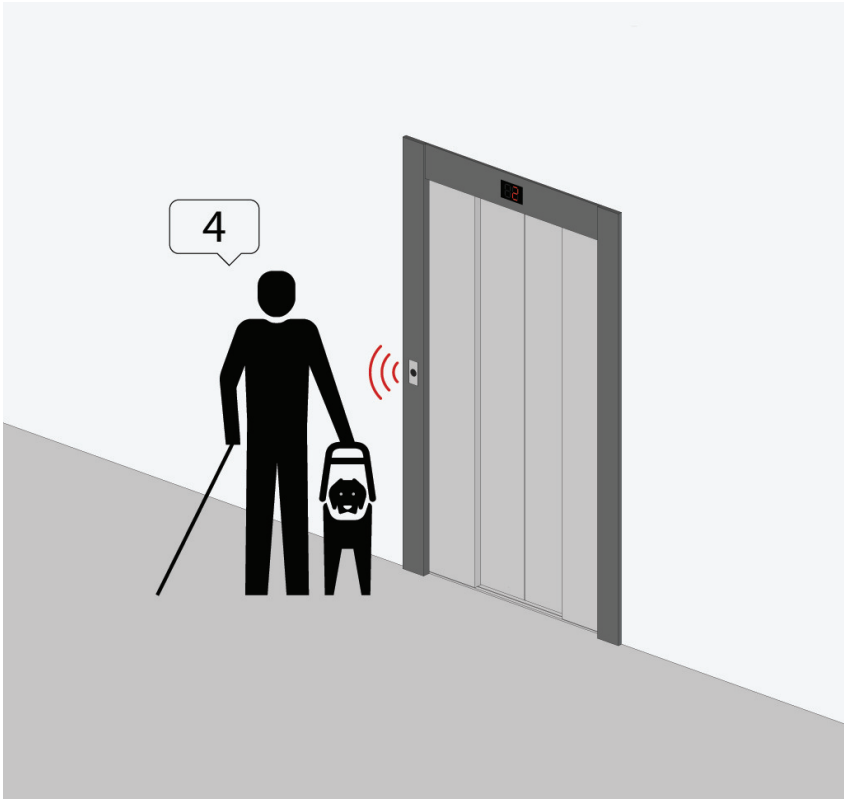
CHALMERS



Smartlift

Voice-Activated Elevator

Oskar Edwertz
Johanna Fredriksson
Ana Ortega Lozano
Åsa Rogenfelt

Bachelor thesis
Department of Signals and Systems
Chalmers University of Technology
Gothenburg, 2015

Smartlift
Voice Activated Elevator

O. Edwertz
J. Fredriksson
A. Ortega Lozano
Å. Rogenfelt

Examiner: Paolo Falcone

Chalmers University of Technology
Department of Signals and Systems
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000
Department of Signals and Systems
Göteborg, Sweden May 2015

Front page picture courtesy of Martin Widerström©

Abstract

Although general accessibility for disabled people has improved during the recent years, the operation of standard elevators can still be enhanced. This report presents an upgrade in the way of requesting destinations, which facilitates the operation for the passengers. The system is implemented into a downscaled prototype, which uses a speech recognition unit based on the methods of Mel Frequency Cepstral Coefficients (MFCC) and Dynamic Time Warping (DTW). The design also includes a back-up option of using buttons. An original design feature of this project is the fact that the selection of destinations is made while the passengers are waiting for the elevator. This voice-operated elevator uses a novel algorithm for sorting the requested destinations, a state machine to control the general logic and a stepper motor to move the elevator car. The report concludes that the designed system can be implemented into a full-scale elevator with some minor adjustments.

Keywords: Speech Recognition, Elevator Control, Voice-operated, MFCC, DTW, Stepper Motor, State Machine

Sammanfattning

Kan vi göra det enklare för personer med funktionshinder att anända hissar? Medan allmän tillgänglighet för funktionshindrade har förbättrats under de senaste åren, finns det fortfarande mycket rum för förbättring kring användandet av hissar. Denna rapport presenterar en nedskalad prototyp som förändrar hur vi efterfrågar våningsdestinationer. Systemet använder sig av en röstigenkänningsenhet, som bygger på metoder så som Mel Frequency Cepstral Coefficients (MFCC) och Dynamic Time Warping (DTW). Dessutom inkluderar designen valmöjligheten att använda knappar. Ytterligare ett nytt inslag i detta projekt är det faktum att valet av destinationer görs medan passagerarna väntar på hissen. Denna röst manövrerade hiss använder en egendesignad algoritm för att sortera de begärda destinationerna, en tillståndsmaskin som styr den allmänna logiken och en stegmotor som förflyttar hisskorgen. I rapporten dras slutsatsen att det avsedda systemet kan implementeras i en fullskalig hiss, med vissa justeringar.

Nyckelord: Röstigenkänning, Hisskontroll, Röststyrning, MFCC, DTW, Stegmotor, Tillståndsmaskin

Acknowledgement

We would like to thank,

Hakan Köroğlu for supporting us throughout the whole project and providing good feedback

Göran Stigler for helping out with the design and construction of the prototype

Erik Ström for giving advice on the speech recognition system

The Department of Signals and Systems at Chalmers University of Technology for providing the necessary resources for finishing the project

The Language and Communication Division at Chalmers University of Technology for giving advice on the language and overall structure of the report

Comillas Pontifical University for answering technical questions about the project

Arduino^{TM} for developing the hardware needed in projects like this

Dustin for the programmable pulse generator, Copyright (C) 2008, Dustin. All rights reserved. Permitted use

Lastly, thank you to the microcontroller board company Raspberry Pi, a trademark of the Raspberry Pi Foundation.

Contents

Acronyms

DFT Discrete-Time Fourier transform. 21

DTW Dynamic Time Warping. , 17, 40

ECU Elevator Control Unit. , 4, 5, 6, 7, 9, 11, 12, 13, 16, 36, 38

FBE Filter Bank Energy. 22

GaAs Gallium arsenide. 30

IDCT Inverse Discrete Cosine Transform. 22

IR Infrared. 30

LED Light-Emitting Diode. , 30, 31, 32, 33

LPC Linear Predictive Coding. 17

MFCC Mel Frequency Cepstral Coefficients. , 17, 18, 40

NPN Negative-Positive-Negative transistor. 30

PID Proportional-Integral-Derivative. 36

RMS Root Mean Square. 33

SMD Surface Mount Device. 29

SRU Speech Recognition Unit. , 4, 5, 7, 12, 16, 17, 18, 22, 24, 26, 35, 37, 38

VAD Voice Activity Detection. 19

1 Introduction

The aim of this project is to develop a voice-activated elevator. To be able to show the elevator algorithm and the speech recognition system, a prototype is designed and built. This section contains the background, the problem descriptions and the boundaries set for this project.

1.1 Background

Despite many advantages provided by modern elevator systems, operating an elevator can be a challenging task for certain groups of people. The control of the elevator panel might be difficult to use if the person is for example blind or physically impaired. In such cases, it would be convenient if the elevator could be controlled with voice commands.

The invention of a voice-activated elevator has already been made before. Nowadays, it is even possible to buy a complete voice-activated control unit and install it on an existing elevator system. One example of this is talk2lift[1]. However, the Smartlift project differs from other voice-controlled elevators in the sense that it is a downscaled prototype. For that reason, the integration of speech recognition with the logic and control of the elevator in low-cost hardware is a challenging problem. Also due to downscaling, some issues such as safety and motor control, will vary from a real elevator. Moreover, the designed system has new features if compared to other voice-activated elevators. In particular, the passenger chooses the desired floor before entering the elevator. This makes it necessary to develop a novel elevator control algorithm.

1.2 Problem Description

In this subsection, the project task and requirements are described. The task and the requirements have been defined based on the areas of knowledge of the project group members.

1.2.1 Task

The aim of this project is to build and control a downscaled prototype of an elevator with ten floors. The position and speed of the elevator should automatically be adjusted to sustain a nice and smooth ride for the hypothetical passengers. Also, the target floors have to be sorted in a way to minimise the travelled distance of the elevator car, which thereby reduces both the energy consumption and the waiting times for the passengers. Furthermore, the elevator should be able to be operated with a speech recognition unit, as well as standard buttons that could replace this unit when desired.

1.2.2 Requirements

The following requirements are established for the prototype:

- Recognition of the numbers 1-10
- Recognition of voices from both men and women (children not included)
- Function without voice operation, by using buttons
- A position accuracy of 0.5 cm
- Smooth movement of the elevator to make a pleasant ride for the hypothetical passengers

[1] A company that produces speech control systems for elevators, http://www.talk2lift.com

- Minimising the distance travelled of the elevator car by sorting the requested floors in an intelligent way
- Ability to keep ten requests at the memory at all times
- A construction cost below 5000 SEK

1.3 Boundaries

Since the goal of the project is to build a downscaled prototype of an elevator, some physical boundaries need to be taken into consideration. In a normal-sized elevator system, the different floors are separated from each other. Therefore, one microphone would be required in each floor. However, due to the small size of the prototype and the wide scope of a microphone, only a single microphone is used in the project. For that reason, only one guest in a specific floor can speak at a time. It is also important to mention that the designed system can only receive one requested floor each time a proximity sensor is activated.

Speech recognition is a challenging task, which requires deep knowledge in the field. Most of the complication in speech recognition is connected to the complexity of the language. In this project, the language part is narrowed down. Hence, only numbers are treated by the system and only the English language is recognised.

The differences in frequencies and vowel space between adults and children make it difficult to create an accurate system for all ages. It has been shown that, when using a model designed for adults, the speech from a child has up to five times higher failure rate [4]. It is hence decided that the system only focuses on speech from adults.

2 System Architecture

This section describes the implemented design in the project Smartlift, including an overall picture of the different units developed.

2.1 Overall Design

To be able to meet the requirements and solve the tasks defined for this project, a voice-controlled elevator system has been developed. The system consists of a downscaled prototype of an elevator that is motion controlled. Two major units can be identified as the 'brain' of the overall system: the elevator control unit (ECU) and the speech recognition unit (SRU). The speech recognition unit is responsible for identifying the passenger requests and delivering them to the elevator control unit. The elevator control unit then properly places the requests in an array and sends commands to a stepper motor to realise them. A flowchart that briefly illustrates the operation of the whole system is provided in Figure 1. The overall system operation will be explained in the sequel in some detail by referring to this figure.

Design choices have been made regarding the downscaled prototype. The construction is built with ten floors, which is considered to be good enough to clearly demonstrate the functionality of the speech processing and the elevator control algorithm. It was also decided to not attach operated doors on each floor. Instead, two LEDs with different colours are used to indicate the operation of the doors.

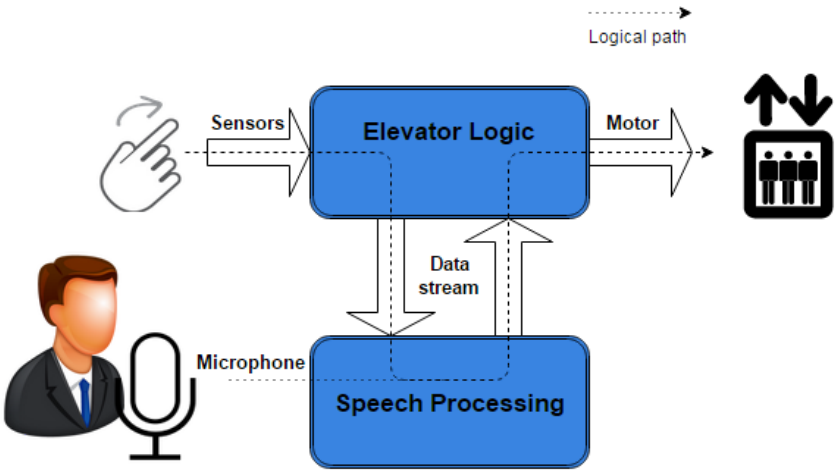


Figure 1: The flowchart of Smartlift

2.2 Design and Implementation

The prototype contains ten floors and is approximately 1.7 m high. A drawing of the construction can be found in Appendix A. On each floor, a proximity sensor is mounted to detect the approaching passengers together with their positions (i.e. the floor that they are at). To simulate the operation of doors, two LEDs are installed on each floor: green for opening and red for closing. A stepper motor is used to move the elevator car, the position of which is determined by counting the steps of the motor. When the system is turned on, the position of the elevator car is unknown

and therefore a reference sensor is needed. The sensor is placed at the first floor. When the system is turned on, the elevator car starts moving down until the sensor gets activated and the position is then known. Ten buttons are also used to facilitate the operation of the elevator without the SRU. Because of downscaling, only one microphone and speaker are used. A display is used to show the array of the elevator. The construction is placed and stabilised on a wooden platform, where the hardware is also located.

The brain of the overall system is formed by two microcontroller boards: Arduino™ Mega ADK (in which the ECU is implemented) and a Raspberry Pi 2 Model B (in which the SRU is implemented)[2]. The input pins of Arduino™ are connected to the sensors and the buttons, while the output pins are connected to the LEDs, the motor (via a motor driver) and the display. As input ports of RPi, a microphone and four digital pins are used, while a speaker and eight digital pins are used as outputs. Arduino™ and RPi are connected in a way to exchange information about the entry and target floors. When a proximity sensor is activated, Arduino™ sends the corresponding floor number (i.e. the entry floor) to RPi. When the RPi determines the passenger's target floor, it sends back both the entry floor and target floor to the Arduino™ (see Figure 2).

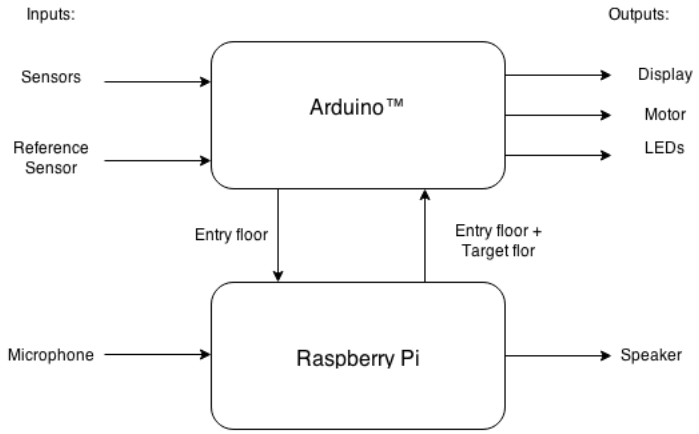


Figure 2: Flowchart of the system including inputs and outputs

The speech processing algorithm is written in MATLAB with the intention to implement it in RPi by using Simulink. However the implementation of speech recognition in RPi (read more in Section 4.5) has not been completed. So the elevator will only be controlled by the buttons and the speech part will be presented in a computer during the demonstration. Therefore, in Figure 2 only the Arduino™ will be connected.

2.3 System Operation

The system start-up operation is designed in the following way. When the elevator is turned on, the elevator car starts to move down until a reference sensor is activated at the ground floor. During this time, it is assumed that there will be no passenger requests. The position of the elevator car is thus ensured to be at the ground floor some time after the system starts.

Once the start-up phase is completed, the system waits for approaching passengers. When a passenger approaches the elevator at a particular floor, a proximity sensor is activated. Then the ECU sends the relevant floor number (i.e. the entry floor) to the SRU. The SRU then activates a speaker and asks the passenger to pronounce the requested target floor (from one to ten). Thereafter, a microphone is activated by the SRU to record the speech for three seconds. The recorded speech is processed by the SRU for recognition purposes. This is done by comparing the

[2]The microcontroller boards will be referred to as Arduino™ and RPi throughout the report

processed signal to a library of pre-recorded signals and finding the best match. When the spoken number is identified in this fashion, the SRU sends the entry and target floor information to the ECU. If the system is to be operated by buttons rather than the SRU, the entry floor information is stored inside the ECU, instead of sending it to the SRU. In this manner, the selected number is combined together with the target floor input obtained from the buttons. The ECU places the entry and target floors at proper positions in an array of destinations that can store a maximum of ten numbers. By processing this array, the ECU controls the movement of the elevator.

The elevator car is programmed to go to the first floor in the array of destinations. When the elevator car arrives at this floor, the ECU commands the elevator car to wait while the door opens and closes for passengers' entry and exit. Thereafter, the array is updated by deleting its first entry. The system then carries on with the remaining floors stored in the array. When the array is empty (i.e. all zeros), the elevator car stays at the last destination that was reached, while waiting for new passenger requests.

3 Elevator Control Unit

This section focuses on the specific unit of the elevator that is responsible for the logical operation of the elevator. Three major tasks can be identified for this system:

- Receiving information about which floor the requesting passengers are at and where they want to go
- Sorting the passenger requests in a proper way
- Deciding what the elevator car should do, i.e. moving, waiting or operating the doors

When the elevator car is moving, the system should control the position and velocity accordingly.

As the elevator control unit (ECU) is the central part of the system, it has to connect the speech recognition with all the other parts of the construction. In the following subsections, first the design and operation as well as the implementation of the ECU are described. An example is then provided to explain how the elevator logic works. Finally, the results of a set of tests and verification procedures are presented.

3.1 Operation of the Elevator Control Unit

It is needed to highlight a general design decision that has been made in order to minimise the distance travelled by the elevator car. It consists of asking the approaching people where they want to go, when they are still outside the elevator car. This offers three advantages if compared to the case in which the passenger requests are received inside the elevator car:

- The elevator would not stand still during the processing of the SRU, since the destination of the passenger would then be known before the passenger enters the elevator.
- Having received the target floor beforehand, the system would be able to decide when it is most suitable to pick up the passenger. This means prioritising the passengers that want to go in the same direction as the elevator car, which results in the fact that the elevator car will not change its direction frequently.
- There would then be no need for a microphone inside the elevator.

In order to have a microcontroller that is able to fulfil the desired tasks and requirements, a state machine that runs in parallel with three other units have been designed. The inputs, outputs and connections between these three units and the state machine can be visualised as in Figure 3. The operation of each unit and the state machine are described in the following subsections with reference to this figure.

3.1.1 Conversion of signal from an activated sensor to a number for the SRU

Unit 1 connects ten proximity sensors with the SRU. These sensors inform the system about where the passenger is and when to activate the microphone. Each sensor is placed at a separate floor of the prototype to detect the approaching passengers. Being integrated in a downscaled prototype, the chosen proximity sensors are optical reflex sensors that will be activated by the reflected light. When one of these sensors is activated, *Unit 1* sends the number of the floor where the person is to the SRU. Due to downscaling, the prototype has a single speaker and a single microphone to communicate with the people. A real product would clearly need as many microphones as the number of floors in the building. For this reason, in order to be able to upscale the project and activate only one microphone and one speaker at the correct floor, it is needed to send the activated floor to the SRU. Further discussion about upscaling is presented in Section 7.2.

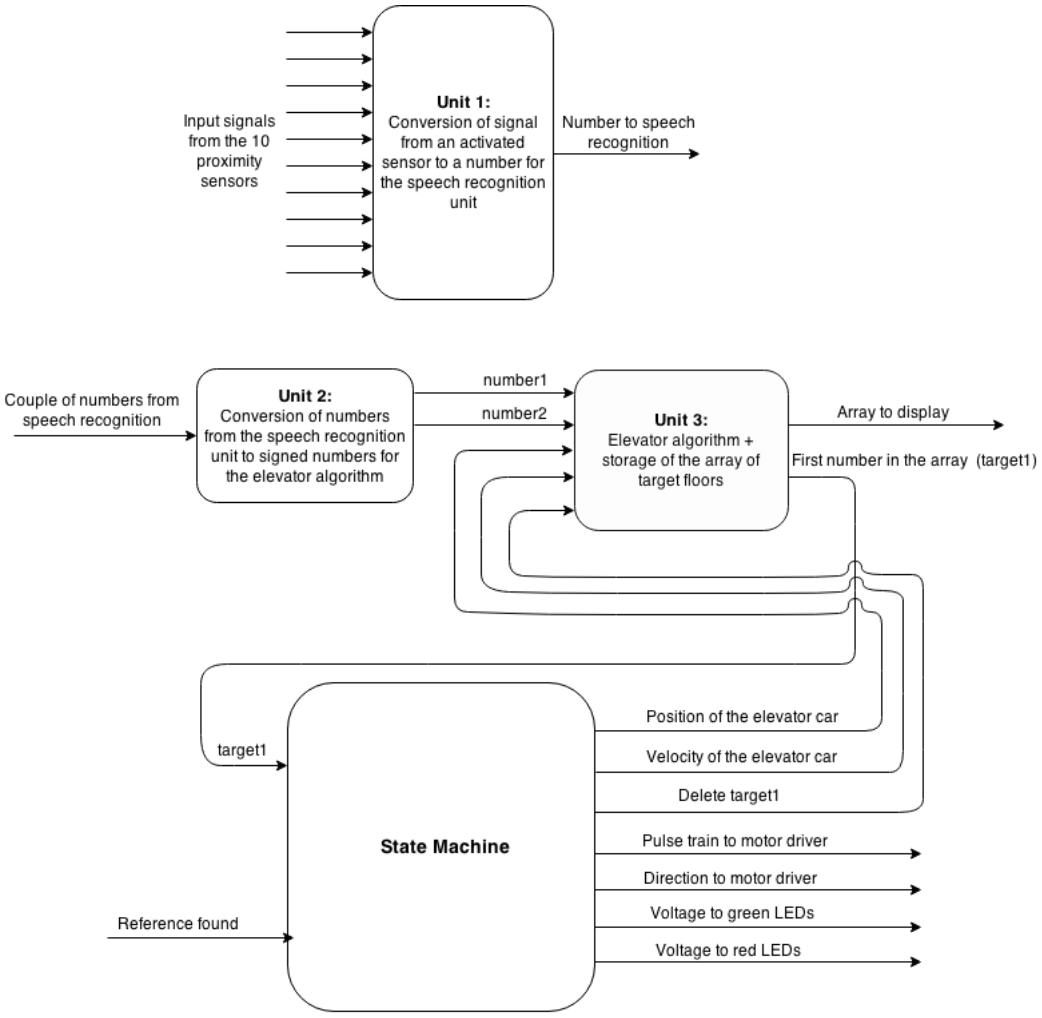


Figure 3: Design of Elevator Control

3.1.2 Conversion of numbers from the SRU to signed numbers for the elevator algorithm

Unit 2 receives both the floor where the person is waiting and the desired target floor from the SRU. It was decided to send these two floors together, even if it could be possible to store the position of the person from the data received from the proximity sensor. Thanks to this choice, it will be easy to modify the system in a way to listen from several floors at the same time. This unit also calculates the direction of the requested movement, since this is needed for the logic of the elevator algorithm. The floor data together with the direction of movement is then sent to *Unit 3* in the form of two numbers with sign (*number1* and *number2*). By convention, *number1* is the floor where the person is waiting and *number2* is the desired target floor. The sign is equal for both numbers, being "+" if the desired direction is upwards and "-" if it is downwards.

3.1.3 Elevator algorithm and storage of the array of target floors

Unit 3 is responsible for placing the received passenger request at a suitable place in the array, which is a major task of the elevator algorithm. It is also in charge of storing the array of target

floors. In order to minimise the travel time and thereby be energy-efficient, the main strategy of the elevator algorithm is decided to be as follows:

Continue in the current direction as long as there are more users that want to go in the same direction, placed in floors that have not been reached yet.

In accordance with this strategy, a "+" or "-" sign is sent, to easily sort all positive floors together in an increasing order and all negative floors together also in an increasing order. The algorithm uses the current position (*position*) and velocity (*velocity*) as inputs from the elevator car in order to determine if there is enough time to change the first target floor in the array, if needed.

The first target floor (*target1*) is an input for the state machine. The size of the array is ten, i.e. maximum ten target floors will be stored in the array, which is enough to demonstrate the operation of the elevator. The first position in the array is sent to a display for the purpose of showing the operation of the elevator algorithm. The signal called *delete target1* is activated by the state machine when the first target floor in the array is reached. When it is activated, the elevator algorithm deletes the first target floor in the array and moves the remaining target floors one place forward. The vacant position is then filled with the number *0*, because it will be interpreted as *"for the moment there are no pending requests"*.

3.1.4 State Machine

The last unit that can be seen in Figure 3 is the state machine. It has two inputs: *target1* and *Reference found*. The position of the elevator car is calculated and stored in it, as will be explained in Section 3.2.3. Based on the inputs (*target1* and *Reference fund*), current positon, the current state and time passed from the beginning of the state, the outputs are selected. These are the signals to delete the first target floor in the array, the position and velocity of the elevator car, the pulse train and direction to the motor driver and the voltages for the green and red LEDs. In order to decide the correct outputs, the state machine is made of four different states, as shown in Figure 4.

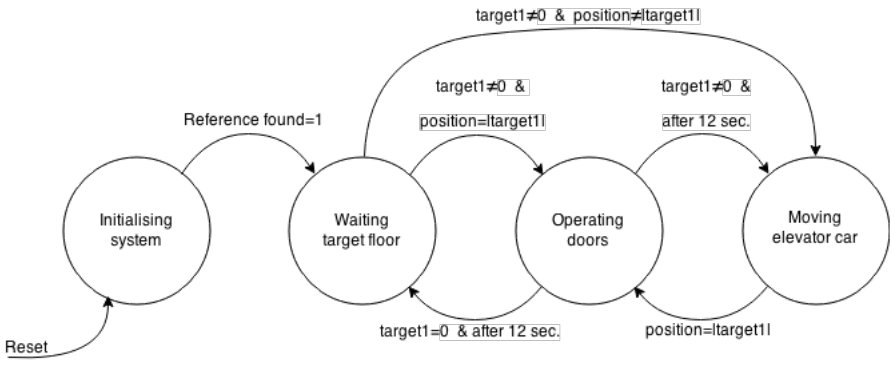


Figure 4: State machine of the ECU

- **Initialising System State:** This is the first state seen in Figure 4. It is only used right after powering on the whole system. It moves the elevator car downwards until a proximity sensor detects the car (by which *Reference found* signal is activated) and lets the system know that the reference position is reached. This allows the system to start with the elevator car placed anywhere.
- **Waiting Target Floor State:** In this state, nothing is done other than waiting with the elevator car stopped and the LEDs turned off until *target1* changes and is different from *0*. When this happens, the elevator has to start the state *Operating doors* if the elevator car is

already in *target1*. Otherwise, if the current position is not the target floor, it has to go to state *Moving elevator car*.

- **Operating Doors State:** Every time this state is reached, the elevator car is in the desired *target1*. First of all, the system waits for one second in order to have some safety margin before opening the doors. Then, it opens the doors during 3 seconds at the floor *target1* (activation signal to the green LED), waits for 2 seconds and closes the doors during 3 seconds (activation signal to the red LED). After this is executed, it sends a signal to *delete target1*. Then, it is checked if deleting another floor is needed (i.e same target floor). In that case, another activation signal is sent in *delete target1*, which may be needed if the elevator car is at the border of changing direction. In this case, the floor that was deleted just after closing the doors may be the same as the new target floor but with opposite sign. As the floor has already been reached, it also has to be deleted from the array. After 12 seconds from the beginning of *Operating doors* state, the system will go to *Moving elevator car* state or to *Waiting target floor* state, depending on whether the new target floor (*target1*) is *0* or not, respectively.
- **Moving Elevator Car State:** This state includes a proportional control of position and velocity to meet the requirements previously described. The position reference is defined as the target floor converted into steps that the motor has to take to reach that floor. The error to the desired position will be calculated by subtracting the current position of the elevator car in steps too. The details of the design of the motor control are explained in Section 3.1.5. When the target floor is reached, the state machine changes to *Operating doors* state.

3.1.5 Motion Control

As motivated in Sections 5.5 and 5.6, a stepper motor and the A4988 motor driver were chosen to drive the elevator car. The chosen motor driver requires only two outputs from the microcontroller board: a pulse train signal and a direction signal. Each rising-edge of the pulse train signal will make the motor move one discrete step in the direction indicated by the direction signal.

The main disadvantage of using a stepper motor if compared to a servo motor is that it does not have a feedback signal. Therefore, an external device (such as an encoder) would be required to get a feedback signal. It was decided not to use this extra device due to three main reasons:

- An extra device would introduce additional complexity to the system, which is not desirable in view of the present level of complexity on the Smartlift project.
- The mass of the elevator car and the inertia of the bobbin are insignificant to the system dynamics, which means that the design objectives can be achieved with a standard stepper motor in the prototype.
- A simulated feedback inside the microcontroller would help realise some features of genuine feedback control.

A simulated feedback inside the microcontroller, instead of a usual feedforward, allows for achieving a smooth stopping of the elevator car. A smooth acceleration is achieved at the beginning thanks to the system dynamics. In case there is an unexpected delay in the arrival time of the elevator car to a floor, caused by the system dynamics, 1.5 seconds are waited until the signal to open the doors is active.

Having all the previously described decisions in mind, it is possible to explain in depth the motion control design, which can be seen in Figure 5. The reference floor signal comes from *Unit 3*, as shown in Figure 3. The blue parts of Figure 5 are the output signals from the microcontroller to the motor driver. In the same figure, the green block and arrows form the simulated feedback. As can be observed from Figure 5, the motion control performs the following main tasks (numbered in the figure):

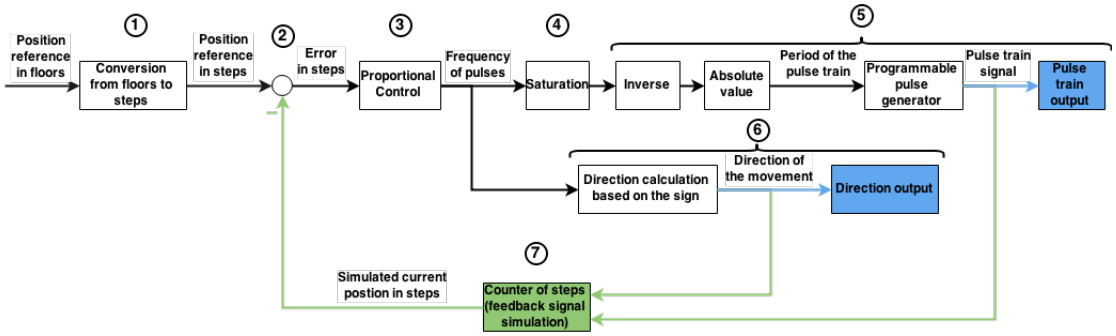


Figure 5: Motion control design

1. The reference position in floors is converted into the amount of motor steps needed to reach the reference floor.

2. The error in steps related to the reference is obtained by subtracting the simulated signal of the current position in steps from the reference position.

3. A proportional controller calculates the frequency of a pulse train signal. As can be seen in Figure 11, the integral branch of the controller is not needed because there would be no position error in steady state. The derivative branch of the controller has no effect on the performance, since the feedback signal is not real but simulated.

4. The maximum desired speed is limited through saturation, which corresponds to 5cm/s. As explained in Section 5, each floor of the prototype is 15cm high, which implies that it takes 3 seconds to reach each floor at maximum speed. In the same section, it is explained that 100 motor steps correspond to one floor. Therefore, the frequency of the train of pulses has to be

$$100 steps/3s = 33.3Hz.$$

   This is why the salutation limits are set to +33.3 and -33.3, where the negative values correspond to a pulse train that moves the elevator car downwards.

5. The period of the desired pulse train is calculated in order to use the Programmable pulse generator, developed by Dustin, permitted to use [3]. This pulse train is an output from the microcontroller to the motor driver.

6. The direction of the movement is calculated from the sign of the frequency of pulses, which corresponds to the sign of the error previously calculated. It is also an output from the microcontroller to the motor driver.

7. In order to simulate a feedback of the current position of the elevator, the rising-edges of the pulse train signal are counted, since each of them corresponds to a step in the motor. If the direction is upwards, the counted rising-edges are added to the current position in steps; otherwise they are subtracted. Thanks to the calculation and storage of the position in steps of the elevator car, we can use the designed proportional controller.

It has to be highlighted that the action of the proportional controller only affects the movement during the last steps before reaching the desired floor. In order to stop in a smooth way, the value of the proportional gain was experimentally adjusted to 2. Further explanation about the implementation of the motor control is provided in Section 3.2.4.

[3]Copyright (C) 2008, Dustin. All rights reserved. Permited use.
http://www.mathworks.com/matlabcentral/fileexchange/19600-simulink-programmable-pulse-generator/content//ProgrammablePulseGenerator.mdl

3.2 Implementation

The algorithms that constitute all the subunits of the ECU are implemented in MATLAB and Simulink. The state machine uses a very powerful tool of Simulink called StateFlow, which permits to use Simulink blocks and MATLAB functions inside a state. For instance, this allows the implementation of the proportional controller in an easy way. Simulink and MATLAB were chosen for algorithm implementation since they have support packages for Arduino^{TM} Mega ADK board, which allows the development of complex systems .

For the case of implementation using the SRU, a general picture of the top-level of the Simulink program is shown in Appendices C and D. For the case of using buttons inputs instead, the top-level Simulink model of the *Unit1*, *Unit2* and *Unit3* is shown in Appendix E. In the following sections, the implementation of the different parts of the ECU is explained in some detail.

3.2.1 Implementation of Unit 1 and Unit 2

Both units are triggered rising-edge subsystems in Simulink, which contain MATLAB functions that are triggered by their own input signals. In this manner, the MATLAB functions will only be executed when there is a new input.

For the case of implementation with SRU, *Unit 1* converts the activated sensor signal from a floor into the corresponding four-bit number of the floor, which is sent to SRU. *Unit 2* receives numbers a and c in binary representation from SRU and converts them into natural numbers. Then, the first number a (entering floor) and the second number c (target floor) are subtracted. Depending on the sign of this result, a "+" or a "-" sign will be added to the calculated natural numbers if the passenger wants to go upwards or downwards, respectively. This integer numbers are outputs of *Unit 2*, being *number1* the corresponding integer number to the entering floor and *number2* the corresponding integer number to the target floor. The MATLAB codes of the functions of *Unit 1* and *Unit 2* can be seen in Appendix G and H respectively.

In the case of button input signals instead of the SRU, *Unit 1* converts the activated sensor signal from a floor into the corresponding natural number of the floor. This number is stored in a memory block. *Unit 2* receives the number stored in the block memory (a) and the signals from the buttons. Depending on the button that is pressed, the natural number c is calculated. As in the previous case, number a (entering floor) and number c (target floor) are subtracted, and depending on the sign of this result the corresponding sign is added to the natural numbers. The outputs are also *numeber1* and *number2*. For this case of input through buttons, the MATLAB codes of the functions of *Unit 1* and *Unit 2* can be seen in Appendix I and J respectively.

3.2.2 Implementation of Unit 3

Unit 3 has two main parts: the elevator algorithm that sorts the requested floors and the storage of the array of target floors. The elevator algorithm is implemented as a MATLAB function that is inside a triggered subsystem just as *Unit 1* and *Unit 2*. The trigger signals are *number1* and *delete target1*, since the array of target floors has to be refreshed each time the signals change values. *Number2* is not a trigger signal since *Unit 2* ensures that there will always be a rising-edge in *number1* and in *number2* at the same time. The storage of the array of target floors is done through Simulink blocks called Data Store Memory, Data Store Write and Data Store Read.

The operation of the elevator algorithm is performed by the MATLAB code in Appendix K and can be explained as follows:

1. If *number1* and *number2* are different from zero, they have to be added to the array. In that case, if there are at least two empty positions in the array, suitable places are searched for both numbers. The variables *x1* and *x2* contain the positions of *number1* and *number2*

respectively, related to the still non-updated array of target floors. The general logic is shown in the block diagram of Figure 6 and explained as follows:

- The current direction of the elevator car is checked through the sign of the first number in the array.
- If *number1* and *number2* have the same sign as the first number in the array, it has to be checked if there is time to change position in case it is needed. If there is enough time, *number1* and *number2* have to be placed in the first part of the array. If there is no time to change the first number (because the elevator has already passed by), the second part of the array with the same sign as *number1* and *number2* has to be found. The positions of the numbers will be, in that case, in this second part.
- If the sign of the new numbers is different from the sign of the first number of the array, the end of the first part of numbers has to be found and the numbers have to be placed in the following part with the correct sign.

2. The values of *number1* and *number2* are added to the array of target floors in the positions *x1* and *x2*
3. If *delete target1* is active, the first number in the array is deleted and the rest are moved one position forward. An extra zero is added at the end.

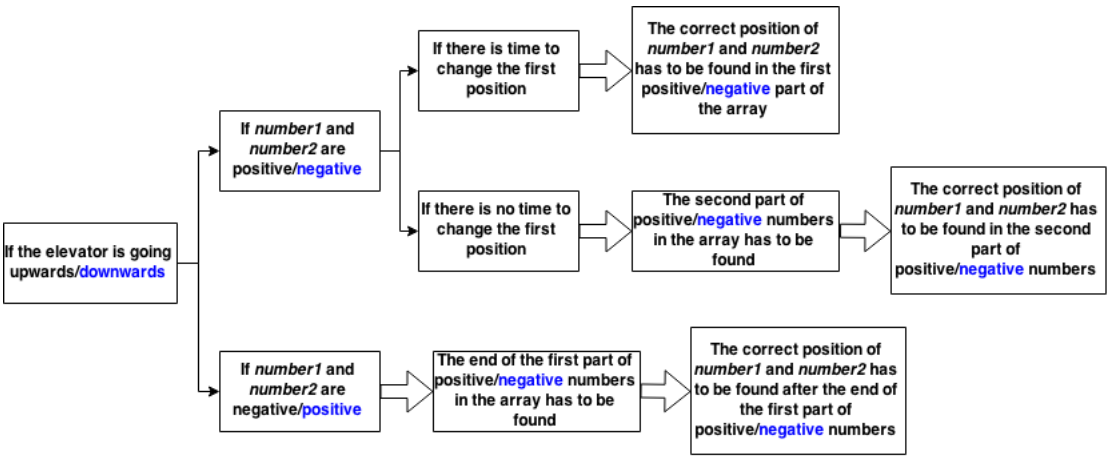


Figure 6: Block diagram showing how the new numbers are added to the array of target floors

The output to the display is done thanks to HCMAX7219 library for Arduino™, which allows an easy control of it. An S-Function Builder block of Simulink is used to work with this library.

3.2.3 Implementation of State Machine

The complete state machine is made of five main parts:

1. A chart diagram in StateFlow
2. A triggered subsystem to reset the position
3. An enabled subsystem to initialise the ECU
4. An enabled subsystem which contains the motion control
5. The corresponding inputs and output pins of the Arduino™

As Appendix D shows, the state machine chart has a clock signal, the *Reference found* signal, *target1* and the position of the elevator car in steps as inputs. Its outputs are the signal to delete the current target1, the voltage to the green and red LEDs, the reset position signal and the enables for motion controls during the states *Initialising system* and *Moving elevator car*. This chart is in charge of the transitions between states and some of the actions inside them.

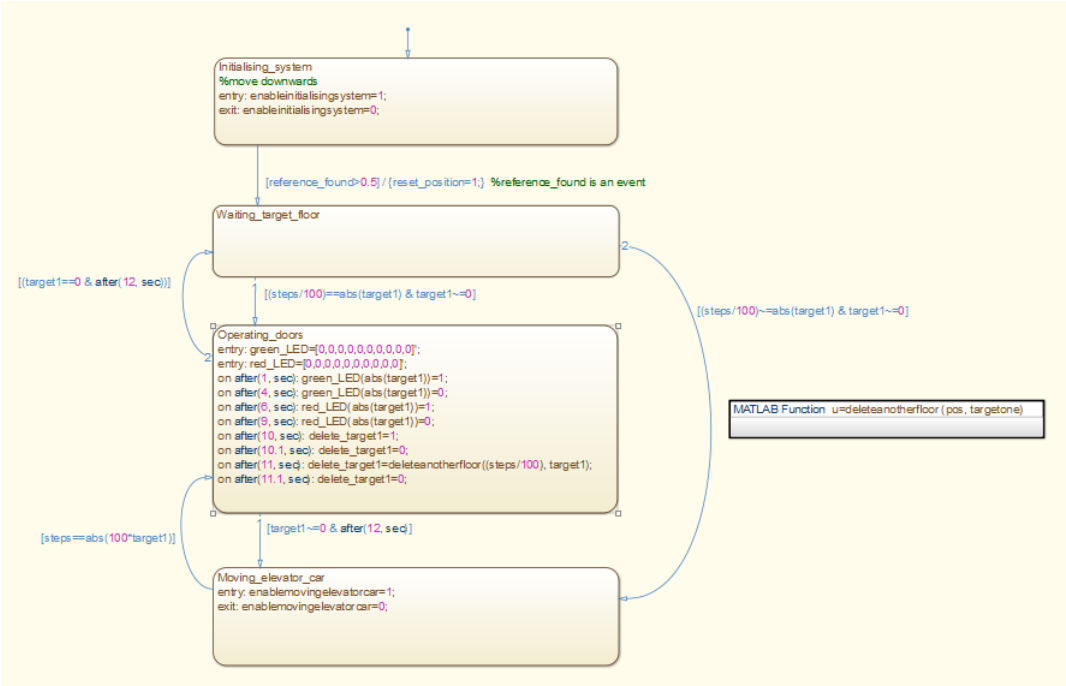


Figure 7: Chart of the state machine in Simulink (see Appendix L for the code of the MATLAB function).

A general overview of the states and their transitions can be seen in Figure 7, where the content of the StateFlow chart is shown. The system always starts in *Initialising system* state, which enables the subsystem *Initialising system motor control*. This subsystem is further explained in Section 3.2.5. When the *Reference found* signal is active, the transition to *Waiting target floor* state is made. During that transition, the signal *Reset position* is activated, which resets the position of the elevator car as it is explained in Section 3.2.6.

The system is in *Waiting target floor* state every time there are no more pending destinations for the elevator car. This is indicated through a "0" in the signal *target1*. Therefore, when finishing the *Operating doors target1* signal has a value different from 0, the state changes to *Operating doors* or to *Moving elevator car*.

For the activation of *Operating doors* state, it is needed that the elevator is already in the position of *target1*. This is checked by comparing the absolute value of *target1* with the value of the position in steps divided by 100 (since the change in position for one floor to the following one corresponds to a change in position of 100 steps). The code that is executed inside *Operating doors* states follows the designed actions that where specified in Section 3.1.4.

For activating *Moving elevator car* state, two things may occur. If coming from *Waiting target floor* state, *target1* signal has to contain a different value from 0, as it has just been explained, and the position of the elevator in steps does not correspond to the value of *target1*. If coming from *Operating doors* state, the condition of 12 seconds passed from the beginning of this state has to be fulfilled as well as the condition of having a *target1* signal different from "0". While *Moving elevator car* is active, an enable signal to the *Moving elevator car motion control* subsystem is

sent. A further explanation of this subsystem is explained in Section 3.2.5. When the target floor is reached, *Operating doors* state will be active, since the position of the elevator car in steps will be equal to the absolute value of 100 times the number contained in *target1*.

3.2.4 Implementation of Motion Control

The Simulink model of the *Moving elevator car motion control* subsystem is shown in Figure 8. This model fulfils the designed actions described in Section 3.1.5.

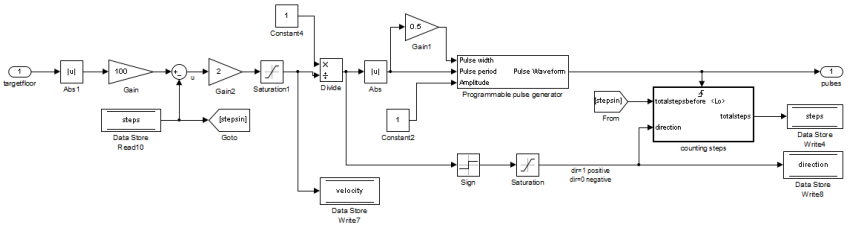


Figure 8: Motion control unit in Simulink, where the MATLAB code of the subsystem *counting steps* can be visualised in Appendix M

3.2.5 Implementation of Motion Control in Initialising System State

In order to move the elevator car downwards until the position reference is found, during *Initialising System* state, a variation of the motion control subsystem is activated. The exact model is shown in Figure 9. It does not have a position reference, but a constant velocity downwards. In concrete, the constant block of -16.7 sets the linear speed of the elevator to -2.5cm/s, since it is half of the maximum speed in the motion control.

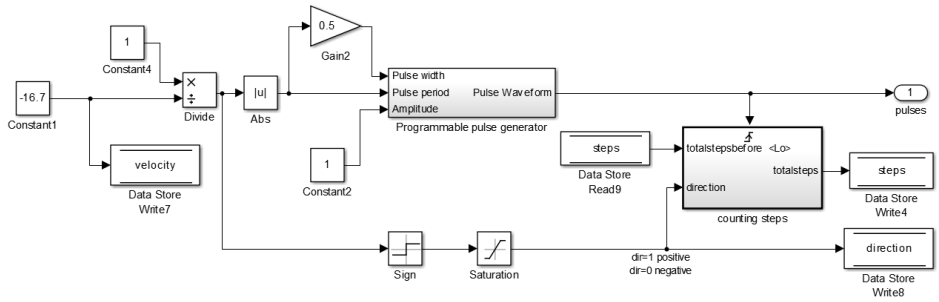


Figure 9: Simulink model of the motion control for *Initialising system* state

3.2.6 Reset Position Block

In order to reset the position of the elevator car when the system is initialised and stopped at floor one, the block *Reset position to 100 steps* in Appendix D is activated. It resets the position of the elevator car to 100 motor steps. The content of this block is shown in Figure 10.

3.3 Example of Operation with SRU

In this subsection, the operation of the elevator control algorithm is described with a particular example.

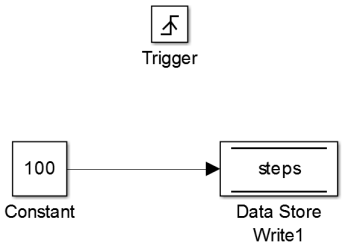


Figure 10: Simulink subsystem to reset the position when the system is initialised

Suppose the elevator car is at floor 2, just after a new person has entered the elevator, starting to go up because the target floor is 5. In the state machine, the state *Moving elevator car* is active, with a reference position of 500 steps, corresponding to floor 5. As the direction is upwards, *target1*=+5. It is also supposed that the array of target floors is as follows:

$$[+5, +7, +9, -10, -2, +1, +4, 0, 0, 0]$$

At this time instant, a person is detected by the proximity sensor in floor 3. This is translated to the number *3* and is sent to the SRU. The SRU would decode that the desired floor is *7*. The couple of numbers *3* and *7* is received in the ECU and they are then sent to the elevator algorithm as *number1=+3* and *number2=+7*, both positive as the direction is upwards. As the elevator car is nearly in floor 3 already, there is no time to update *target1*. Therefore the elevator algorithm should place them in the positions of the array as follows:

$$[+5, +7, +9, -10, -2, +1, +3, +4, +7, 0]$$

They have to be in the correct place according to increasing order in the next part that has positive numbers.

When floor five is reached, the state will change to *Operating doors* state. The green LED will light for 3 seconds and then the red LED. After that, *delete target1* will be active and the array will be updated as:

$$[+7, +9, -10, -2, +1, +3, +4, +7, 0, 0]$$

Then, the elevator will visit all the remaining floors in the queue in a similar way, until it is empty.

3.4 Verification of the ECU

Unit 1, *Unit 2*, *Unit 3* and the state machine, were tested separately through simulation in the computer and in external mode in Arduino™. The external mode allows to check each signal in Simulink while the program is running in Arduino™. Once all the units were verified separately, they were also tested together.

Most of the verification tests were made to check and improve *Unit 3*, through simulation in Simulink running on the computer. An exhaustive test bench was made and input to this unit in order to check as many different cases possible.

In order to test the motion control, several destinations were input to it, both in simulation in the computer and running in external mode from Arduino™. The motion control follows the reference in a proportional way to the one shown in Figure 11.

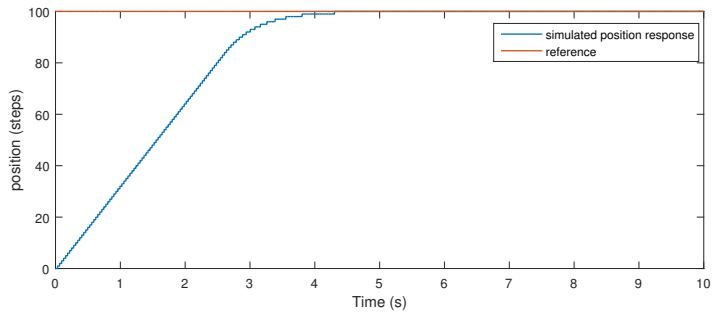


Figure 11: Step response of the motion control system for one floor-change in reference position

4 Speech Recognition Unit

This section provides a brief background on speech recognition systems and presents the method designed for this project. First of all, an introduction to speech processing is presented by a brief section about the general concept of speech recognition, as well as basic signal theory. Thereafter, in Section 4.2 and 4.3, the method used in this project is explained in detail before presenting its results in Section 4.4. Lastly, the implementation of the system into the hardware is explained in Section 4.5.

4.1 State of the Art and Basic Signal Processing Theory

Speech processing is a relatively old area of research. The first successful system of speech was developed around 1950 [1]. Already in the first half of the 20^{th} century, the relation between a given speech spectrum and the characteristic sound was documented. Today, most speech recognition systems are still based on the speech power spectrum. In the 80s, the Hidden Markov Model (HMM) was introduced as a method in the area of speech recognition, which was a big breakthrough that led to remarkable improvements. HMM is a method that is commonly used in the classification process.[2] Over the years, the speech recognition systems have improved from those that can only respond to a small set of words to those that can perform continuous speech recognition.This has allowed for an increase in the number of speech recognition system applications, such as in health care equipment, military products, in-car systems and cellular devices.

Nowadays, there are several different methods that can be used to successfully build a system for speech recognition [3]. In continuous speech recognition, the language is often divided in different phonemes by an acoustic model. In English, about 40 of these phonemes are required and then the word is determined by using a Hidden Markov Model (HMM). HMMs are commonly used in SRU, and in general in systems that can be in a number of different states and switch among them. But it is not possible to determine the current state without seeing the previous states (therefore the phrase "hidden")[4]. However, in the Smartlift project, an isolated speech recognition system is used. A speech recognition system is referred to as isolated when only single commands are used. It can be one word or more, but they are always predetermined. The command is recognised by listening to the word and then comparing it to the reference library that contains the possible commands. This is a simpler process and dividing words into phonemes is not necessary.

However, since all speech recognition systems can fail occasionally, different methods are investigated depending on the particular application. The methods are often based on feature extraction (i.e. representing the information of an analog signal with a set of numbers), where the most common are Mel Frequency Cepstral Coefficients (MFCC) and Linear Predictive Coding (LPC). As the method of MFCC is used in the Smartlift project, it is further explained in Section 4.2.

Once the features are extracted, a method of classification is needed. Frequently the methods used are either HMM or Dynamic Time Warping (DTW).

Some of the challenges remaining in the Smartlift project after the SRU has been limited is:

- Additive noise to the source signal
- Speech accents
- Different frequency levels between women and men
- Similarity between phrases that are used

In the signal processing for speech recognition systems, Fourier Transform has a central part; therefore it will be briefly described here. Fourier Transform is a function that is used to transform a signal from the time domain to frequency domain. The variability in the time domain when handling speech is high and it is known that different tones correspond to different levels of energy in the frequency domain. Therefore applying FT to get the magnitude frequency response will be very useful when handling with speech recognition. However this is not enough and the signal has to be processed further for a successful classification process.

4.2 Isolated Speech Recognition Algorithm

As mentioned before, there are many different methods for implementing speech recognition systems. One concept frequently used in speech processing systems (e.g. mobile phones) is the method of MFCC [5]. This method of feature extraction transforms the frequency spectrum to the so called "mel scale" [6], which is based on pitch comparisons. It has been experienced that the mel scale provides more accuracy to speech recognition algorithms, which is why it has been successfully implemented in various speech systems [7]. Observe that the mel scale is near-linear until 1000 Hz, where it takes on a logarithmic form. This is also visible Figure 12, which shows the mel scale versus the Hz scale.

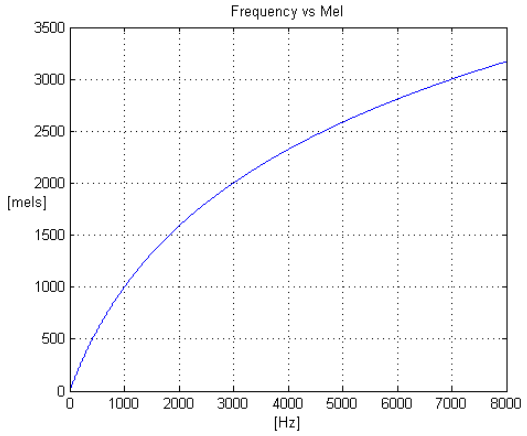


Figure 12: Showing the relationship between frequency and mel.

While MFCC plays an important role in the speech recognition algorithm, it is also crucial to design the steps both before and after the signal is converted to the mel scale. In the first step, the sampled signal is filtered, both to cancel out the frequencies outside the speech range and to improve the signal-to-noise ratio. Then the signal is significantly shortened by finding the sections that potentially contain a number. This is done by using a Voice Activity Detection algorithm. After this step, the signal is segmented, overlapped and windowed. By undergoing these operations, the signal is prepared for feature extraction using the MFCC algorithm. Lastly,

the feature data is compared to each class stored in the library and thereby classified as a number. This is done by using a Dynamic Time Warping algorithm, which produces a minimal distance matrix. All the steps are shown in Figure 13. In the following subsections, each block in the figure will be explained and motivated.

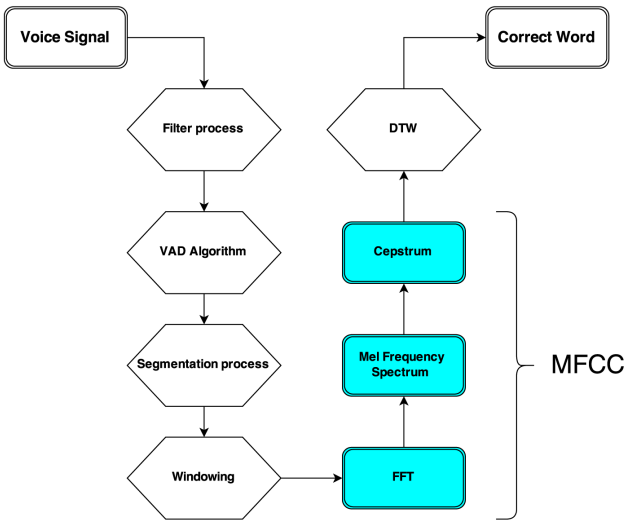


Figure 13: The speech recognition system.

4.2.1 Filtering

The first logical step is to pass the signal through two different filters: band- and high-pass. The reason for using a band-pass filter is that the signal components which are contained in human speech are limited to the range 300-3400 Hz [8]. By using a band-pass filter, all the frequencies outside this range can be discarded. The filter is created by using MATLAB's DSP-toolbox, which provides a variety of functions. As the bandwidth is hereby determined, to avoid aliasing and thereby fulfil the Nyquist–Shannon sampling theorem, it is reasonable to set the sampling rate to at least $F_s = 8000$ Hz.

After passing the signal through the band-pass filter, the signal is pre-emphasised by using a high-pass filter.

```
pre_emp = filter([1 -alpha], 1, signal); %alpha=[0.9 1]
```

The reason for applying pre-emphasis to the signal is that the lower frequency region contains most of the signal's energy. The high-pass filter is basically used to compensate for this. Thereby, the frequency spectrum gets equalised and the signal-to-noise ratio is improved [9]. The effects of these filter operations are illustrated in Figure 14. As can be seen from this figure, the pre-emphasis method improves the signal-to-noise ratio substantially, while the band-pass filter takes care of the high-pitched frequencies.

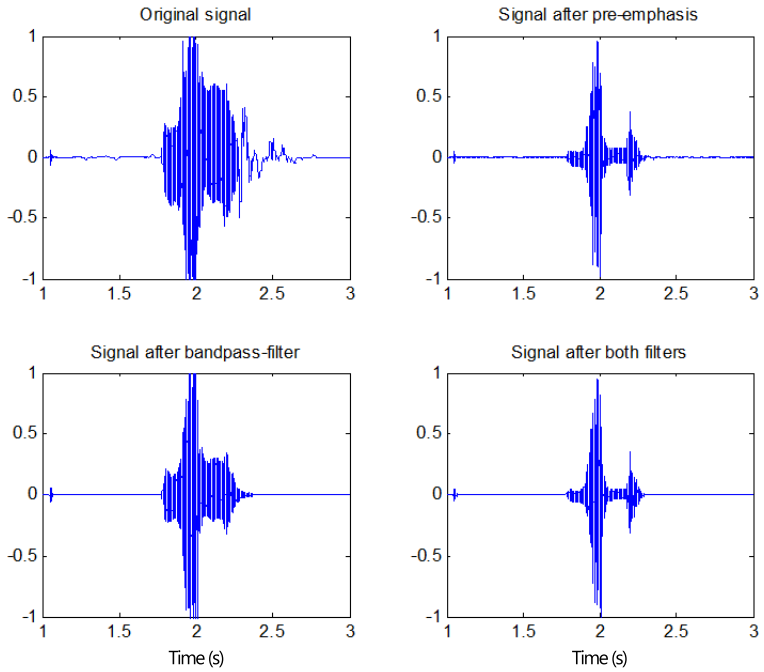


Figure 14: The effects of the filtering operation.

4.2.2 Voice Activity Detection

By integrating a Voice Activity Detection (VAD) algorithm to the system, it is possible to reduce both the processing time and the error probability [10]. There are many versions of this detection process, such as energy threshold processing and waveform analysis. The latter looks for parts of the signal that are harmonic, which is a typical characteristic of voices [11]. However, as the background noise may very well be voices from people, energy-based VAD is used in this project.

Energy-based VAD is done by splitting the signal into different segments (20 ms each) and then determining the energy contained in each segment. An energy threshold is defined based on the background noise by considering the first few milliseconds of the signal as noise. This method functions well, as it is not possible that the significant segment of the signal (i.e. the number) is being pronounced by the passenger during this period. If the packet energy is less than or equal to this threshold, it will be removed from the final sample vector. The VAD-process reduces the signal duration significantly, as exemplified in Figure 15.

4.2.3 Segmentation Process

The signal shortened by the VAD algorithm is then segmented into small packages, as it is much more efficient to handle many short samples if compared to a long one [12]. Furthermore, this does simplify the forthcoming processes as these short packages can be considered to be stationary and not change significantly in time. The package length is therefore set to 25 ms, which is considered to be a suitable value for obtaining enough samples that will produce an accurate spectral analysis [13, 14].

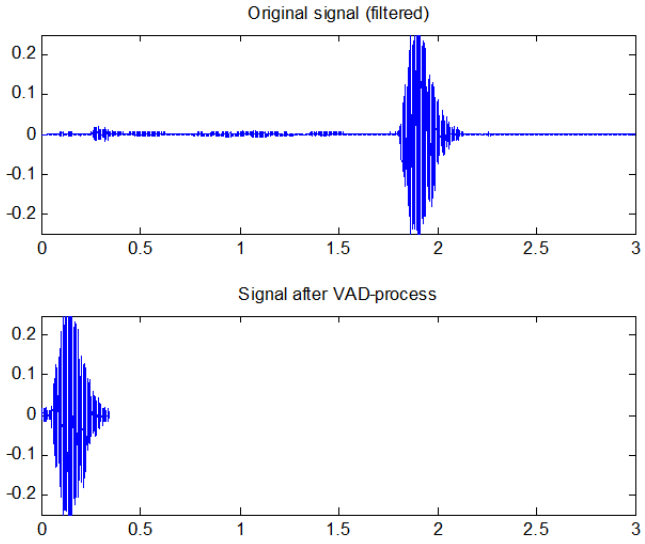


Figure 15: Recorded speech signal for the word "five".

4.2.4 Windowing

Dividing a signal in this manner corresponds to applying rectangular windows to it. This is not very desirable in regards to the resulting effect in the frequency domain. As a matter of fact, the sharp edges of the rectangular shape result in a wide frequency spectrum, which gives the sinc-function periodic peaks of high magnitude. To prevent this effect, each package is overlapped by 50% and multiplied with the well-known Hamming window (see Figure 16). Thereby the frequency domain of each package will be more distinctive, as the Fourier transform of the Hamming window greatly suppresses the signal outside the central point.

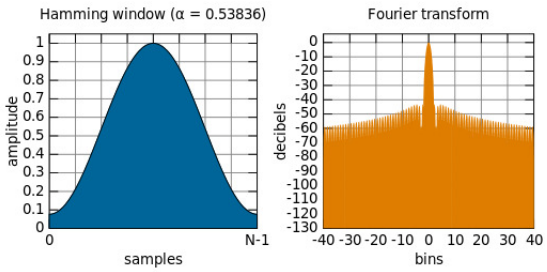


Figure 16: The Hamming window and its Fourier transform.

4.2.5 MFCC Generation

The signal is now referred to as $x_i(n)$, where $i = 1, ..., t$ and $t =$ total number of packages. Each of the t packages are now prepared for undergoing feature extraction in form of the MFCC algorithm. It consists of the following five steps:

1. **Take the Discrete-Time Fourier transform (DFT)**. For each package the DFT is

applied as,

$$X_i(k) = \sum_{n=1}^{N} x_i(n) e^{-j\omega_k n}, \qquad k = 1, ..., N \tag{1}$$

where $\omega_k = \frac{2\pi}{N}(k-1)$ is in radians and N = total number of frequency bins for each package i. If needed, the length is padded with zeros to reach N. To obtain a high resolution in the frequency domain N is set to 512 bins.

2. **Obtain the power spectrum** as

$$P_i(k) = \frac{1}{N}|X_i(k)|^2, \tag{2}$$

where X_i is the sample vector returned from the DFT.

3. **Calculate the filter bank energies (FBE)**. A filter bank is a way of merging together a group of band-pass filters [15]. In this case, a filter bank is used to estimate the energies along the various regions in the power spectrum. Each filter is formed as a triangular window (Figure 17) and its end-points are determined from the mel-scale which is defined as

$$\mu = \frac{1000}{\log(2)} \cdot \log(1 + \frac{f}{1000}), \tag{3}$$

where $f = \frac{\omega}{2\pi}$ represents the frequency in the usual Hz unit. By inserting the start and end points of the frequency range (i.e 300 and 3400 Hz) into Equation 3, the corresponding mel-points are set to 378.5 and 2137.5 mels. Thereafter, the remaining points are evenly distributed along the mel spectrum. As a final step, each point is transformed back to the frequency domain, by breaking out f from Equation 3. As can be observed in Figure 17, the filters are getting wider as the frequency increases. The higher order of filters is presumed to lead to redundant information [16], which is why a filter number between 20 and 40 is preferred [17]. Each of the 26 filters are multiplied with the power spectrum, and then the sum of the resulting signal is obtained, which determines the FBE contained in each filter.

4. **Take the logarithm of the result**. By taking the logarithm of the M number of energies, the resulting feature vectors become more distinguishable. This is motivated by the fact that the human ear perceives loudness logarithmically [18].

5. **Inverse-transform the features (cepstrum)**. Lastly, the logarithmic FBEs are inverse-transformed by using the Inverse Discrete Cosine Transform (IDCT) as,

$$C_i(k) = \frac{1}{2}c_0 + \sum_{n=1}^{N-1} c_n cos[\frac{\pi}{N} n(k + \frac{1}{2})], \qquad k = 1, ..., N, \tag{4}$$

where c_n is the logarithmic FBEs and $n = 1, ..., N-1$. The vector elements obtained at the end of this process are referred to as the Mel Frequency Cepstral Coefficients.

4.2.6 Dynamic Coefficients

The error rate of the SRU is significantly improved by adding time derivatives to the otherwise static MFCCs [19]. These are referred to as delta (Δ) and acceleration ($\Delta\Delta$) coefficients. The

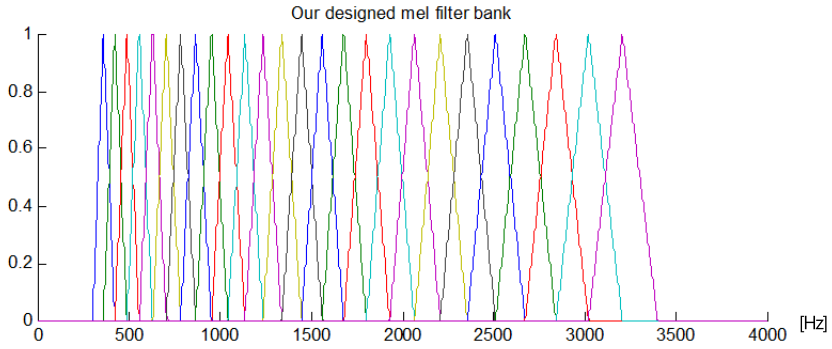


Figure 17: The 26 triangular windows forming the mel-filterbanks

equation for calculating these trajectories is as follows,

$$\Delta C^t(k) = \frac{\sum_{n=-N}^{N} nC^{t+n}(k)}{\sum_{n=-N}^{N} n^2}, \tag{5}$$

where $C^t(k)$ corresponds to coefficient (MFCC) number k evaluated within package t and N is the number of packages taken into consideration for each Δ computation. Take notice that to calculate the acceleration coefficients the resulting matrix of Equation 5 should undergo the same procedure twice. Thereby, the final matrix which is used for classification has the size MFCC + Δ + $\Delta\Delta$.

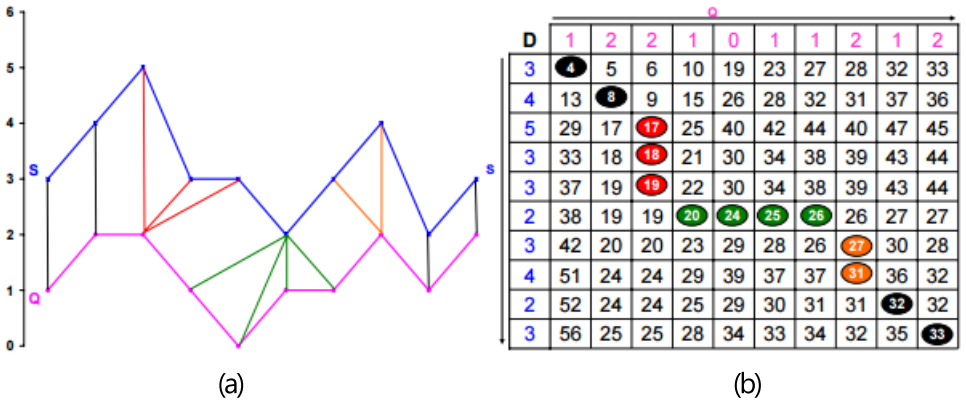
4.3 Classification Process

One of the most challenging tasks in the Smartlift project is to successfully match the analysed signal with the correct number from the library. As mentioned before, DTW is being used for this purpose, since it has proven to be equally successful as HMM when considering small databases [19]. One of the key features of DTW is that it allows for comparing signals that vary in time and speed [12].

To further motivate the use of DTW, its method is compared with the Euclidean distance method, which is linear. Assume two signals, S and Q, whose samples are s_i and q_i respectively. The Euclidean method would calculate the distance along the two signals, i.e. $D_i = |s_i - q_i|$. On the other hand, the DTW distance is calculated from each point s_i to the closest point in the signal Q. This implies that if two signals are visually similar to each other, but differ in speed, they might be classified as a mismatch when using the Euclidean distance, but not when using DTW [15].

For this reason, the DTW algorithm is used for comparing the MFCCs from the input speech and the MFCC matrices stored in the library. As seen in Figure 18(a), each point in signal S connects to the closest point in Q. All points in Q must have at least one connection. By comparing the signals a minimum distance map is formed, as seen in Figure 18(b). The coloured line represents the warping path to take for minimising the distance cost for going from the first to the last cell. The minimal distance cost is referred to as C.

For each comparison between the input signal and the library data a distance map is generated. One of these may be observed in Figure 19, where the darker areas correspond to a lower distance.



D	1	2	2	1	0	1	1	2	1	2
3	4	5	6	10	19	23	27	28	32	33
4	13	8	9	15	26	28	32	31	37	36
5	29	17	17	25	40	42	44	40	47	45
3	33	18	18	21	30	34	38	39	43	44
3	37	19	19	22	30	34	38	39	43	44
2	38	19	19	20	24	25	26	26	27	27
3	42	20	20	23	29	28	26	27	30	28
4	51	24	24	29	39	37	37	31	36	32
2	52	24	24	25	29	30	31	31	32	32
3	56	25	25	28	34	33	34	32	35	33

Figure 18: (a) The DTW distances between S and Q. (b) The minimal distance map relating to figure a. Copyright (C) Javid Taheri et al., permitted to use.

The red line is the optimal warping path (W) to take for minimising C. The calculation of this path is done as [15],

$$C = \frac{\sum_{k=1}^{K} \sqrt{(w_{ki} - w_{kj})^2}}{|S| + |Q|}, \tag{6}$$

where K is the element length of W and w_i and w_j are the optimal path to take for signal S and Q respectively.

The classification procedure looks for the lowest valued C among the library comparisons, which is considered to correspond to the correct number pronounced[4].

The complete speech recognition code can be seen at Appendix N.

4.4 Verification Process

The verification of the speech algorithm was made with both dependent speaker and independent speaker systems. For the classification process to be able to match the results a library containing the demands (1-10) are required.

To test the success rate for the speaker independent system, the library that was created, contained recordings of the numbers from 5 people of each gender. A person that was not included in the library tested the SRU, it was both tested with mixed library from the both genders and with only the library containing the same gender as the test person.

When testing speaker dependent system, the library contained 10 recordings from the same person for each demand. The speaker dependent system had a much higher success rate than speaker independent system. The success rate in speaker dependent system was even so high as 90%. Since it has only been tested by one person for each system before the final report submission, further testing should be done to get a more reliable verification of the speech algorithm. The library created for the speaker independent system should also be extended, by adding more recordings from different persons.

[4]The implementation of this section into MATLAB was heavily inspired by the work of Daniel P. W. Ellis (Dynamic Time Warp in Matlab, Copyright (C) 2003).

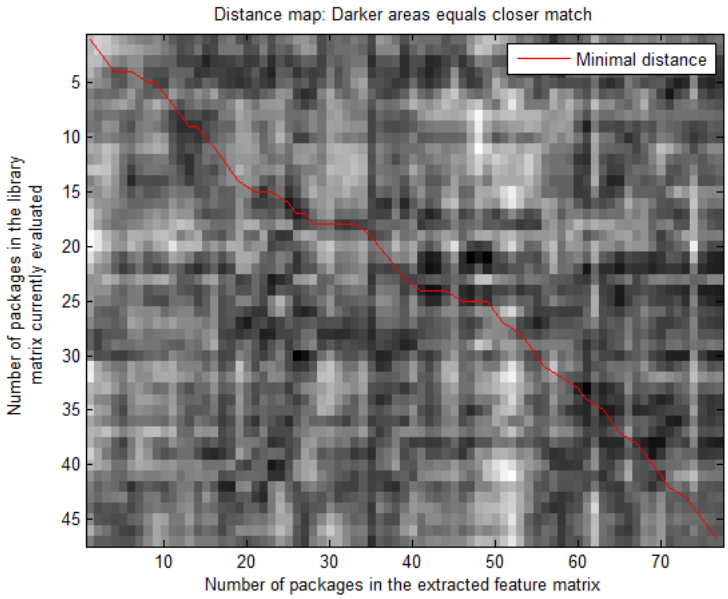


Figure 19: Distance map of speech vs a pre-recorded sample.

4.5 Implementation Into Hardware

The speech recognition algorithm described above is implemented in a Raspberry Pi Model B. RPi can not work with MATLAB code, therefore the created algorithm has to be converted into a language that RPi is compatible with. There are different ways that this can be done and one way is by using Simulink. In Simulink, blocks can be used that can transform MATLAB code into C code, which are a language RPi is compatible with. To make this implementation a state machine was created, as in Figure 20.

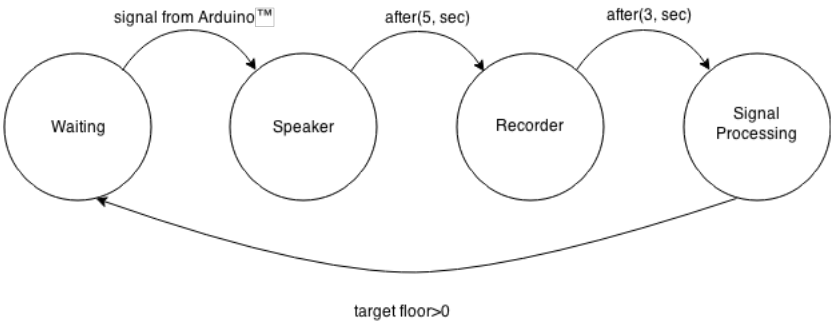


Figure 20: State machine for the speech recognition implemented in Simulink.

- **Waiting State** is the first state seen in Figure 20. In this state, nothing is done; it just waits until a signal is received from the Arduino™ then it goes to Speaker State.
- **Speaker state** is started when the RPi receive the signal from the Arduino™. When activated it starts the speaker to announce to the passenger that he or she should say the desired floor. After the speaker is done (5 seconds), it goes to the next state.
- **Record State** starts the microphone and records for three seconds.

- **Signal Processing State** is the main state, here the signal is processed. The output from this state is the target floor for the passenger.

The MATLAB code was implemented in "Signal Processing state" as a MATLAB function block. Here there occurred problems regarding the implementation. Many of the functions that were used in MATLAB are not suported in Simulink; so they had to be changed. But the major problem that could not be solved was that MATLAB function blocks can not handle matrices with changing sizes as outputs. The problem with changing sizes occurs in the VAD function but this is such a significant role in the quality of the SRU that it could not be removed. So the code was converted to C by using an app in Simulink, and then implemented in "Signal Processing state" as a S-function block in Simulink. Here there also occurred problems that could not be solved.

Then the C-program was also implemented directly into the RPi but here problems were faced when compiling the files. Many errors occurred regarding finding files. So probably something had gone wrong when converting it to a C code or the proper libraries were not implemented in the RPi.

It was not only the implementation itself that caused problems when implementing the code in the RPi. There were problems with connecting the RPi to the computer so the RPi needed to be reformatted; this happened twice. Because of all the problems that occurred during this process and the lack of time, this part of the Smartlift project has not been completed before the final report submission.

5 Physical Construction and System Components

In order to demonstrate the performance of the speech recognition and the elevator control, a downscaled prototype of an elevator has been built. In this section, the construction of the prototype and the applied components are presented.

5.1 Construction of Prototype

The construction needs to have enough floors to fully demonstrate the elevator algorithm and the speech recognition. It also has to be transportable. To meet the requirements, the elevator car was to be positioned stably and precisely by a motor, with an accuracy of 0.5 cm. The elevator car has to move with a maximum velocity of 5 cm/s. The mentioned requirements is used to calculate the measurements and the dimensions of the prototype.

The height of each floor was chosen as 15 cm and the total height of the construction is 170 cm. The width of the elevator is 10 cm and the sides of the elevator were bent to create a more stable construction. To create a 1.5 cm wide rail for the elevator car to slide on, the sides were bent again, at 4 cm. This part of the construction is made in 1.5 mm sheet aluminium. The aluminium has been cut in a water cutter and is then bent. To make sure that the construction is steady, a couple of L-squares were made and attached to a compact base plate. The details of the prototype can be seen in Appendix A. The elevator car has the form of a box with the dimensions 8cm x 8cm x 11cm and is printed in a 3D printer. The weight of the elevator car is 50 g. The design has a track that fit the construction's rail, in which the elevator car will slide.

The wire attached between the elevator car and the motor have a perimeter of 0.03 cm and a length of 170 cm. The wire should be attached to a bobbin where all the wire can fit without being placed in several layers on top of each other. To make sure that the wire is wrapped in the right way, a motor bobbin has been designed and printed in a 3D printer.

The stepper motor has a precision of 48 steps on each rev (see Section 5.2 for more information). To meet the requirement of a precision of 0.5 cm, the bobbin had to be designed to make sure that one step on the motor corresponds to a movement of less than 0.5 cm. The perimeter of the bobbin is decided based on the Equation 7, where the steps/rev is 48 from the used motor and the precision is 0.5 cm set from the requirements.

$$perimeter < \frac{steps}{rev} \cdot precision \tag{7}$$

This gives a maximum value on the perimeter of 24 cm. The length of the bobbin needed is calculated through Equation 8, where the length of the wire is 170 cm, the perimeter of the bobbin is 24 cm and the diameter of the wire is 0.03 cm.

$$Length_{bobbin} > \frac{Length_{wire}}{Perimeter_{bobbin}} \cdot Diameter_{wire} \tag{8}$$

This gives a minimum length of 0.3 cm. The bobbin was designed with a perimeter of 7.3 cm and a length of 2.2 cm. It has a cylindrical form and is attached on the motor axis.

5.2 Motor

In the project, a stepper motor is used to move the elevator car in the prototype. A stepper motor functions like a DC motor, but moves in discrete steps. The motor has multiple coils organised in groups. By energising one group at a time, the motor moves in fixed steps. Thanks to the

discrete steps, a stepper motor is suitable for positioning and velocity control. A stepper motor is controlled by moving one step at a time and can therefore be used with a simple controller. A servo motor also functions like a DC motor but with a built-in encoder and would therefore be suitable for the project. As a servo motor only move in a range of degree, for example 180 degrees, the servo motor was not chosen, as the height of the construction requires continuous rotation of the motor.

5.2.1 Precision

The number of steps/rev on the motor needed for the precision is calculated by the perimeter of the motor bobbin.

$$\frac{Steps}{rev} = \frac{perimeter}{precision} \tag{9}$$

With the set value of the perimeter on the bobbin of 7.3 cm, it is required for the motor to have a precision of at least 15 steps/rev. The motor used in the prototype have a number of 48 steps/rev, which with the value of the bobbin gives a precision of 0.15 cm on each step for the motor. With a length of 15 cm for each floor, a movement of 100 steps by the motor corresponds to a transportation of one floor.

5.2.2 Torque

With the previously mentioned dimensions of the bobbin, the holding torque needed to hold the elevator car can be calculated by Equation 10. F is the gravitational force that the elevator car experiences, r is the radius of the bobbin and m is the mass of the elevator car and the passengers. The gravitational acceleration (g) was assumed to be $9.82 m/s^2$.

$$T = F \cdot r = m \cdot g \cdot r \tag{10}$$

With a weight of the elevator of 50 g and a radius of the bobbin on 1.165 cm, the equation gives a value for the maximum holding torque (T) needed to 0.58 Ncm. Another way in describing the strength of the motor is simply to divide the weight of the elevator car with the length of the radius of the motor bobbin. This term gives a value of 43 g/cm.

5.2.3 Rotational speed

The rotational speed needed for the motor is calculated by Equation 11, being ω the rotational speed, v the velocity of the car and p the perimeter of the bobbin.

$$\omega = \frac{v}{p} \tag{11}$$

To keep the velocity of the car at 5 cm/s, the motor should have a rotational speed of 0.683 rev/s. With 48 steps per rev, this corresponds to 33 steps/s.

Based on the presented calculations, a small stepper motor with the proper specifications was chosen. The motor is a bipolar stepper motor and have a rated voltage and current on 12 V and 400 mA. It has as mentioned 48 steps/rev and a holding torque on 100 g/cm [20].

5.3 Motor driver

The connection between the Arduino™ microcontroller and the stepper motor is done with a motor driver called A4988. The driver offers adjustable current limiting, overcurrent protection and five different microstep resolutions. The driver operates from 8-35 V and can deliver up to 2 A per coil, with additional cooling. The driver has a built-in translator making it easy to operate. Simply by inputting one pulse to the driver, the motor moves one step. The different functions and the layout of the motor driver can be seen in Appendix B.

Thanks to the current limiting, the driver is able to run the motor with a higher voltage than required from the motor. The higher voltage will ramp up the current faster and then also magnetise the coils faster, making the movement from one step to another faster than with a lower voltage level. The current limitation in the driver prevents that the motor is damaged during the drive.

To set the current limitation on the driver, the voltage on the "ref" pin is to be measured and then the current limitation can be calculated. The current limit is related to the reference voltage times 2.5. The current limit on the stepper motor is 0.4 A, which corresponds to a reference voltage of 0.16 V. But the current through the coils is limited to 70 % of the current limitation. Therefore, the reference voltage should instead be set to 0.22 V. By this setup, the motor gets a coil-current of 0.4 A.

The different inputs to the driver are step, direction, reset and MS1, MS2, MS3. MSx sets the microstepping, which can differ from full stepping to 1/16 steps. The driver needs to be connected to a power supply, which is chosen as the 5 V pin on the Arduino™ microcontroller. To be able to increase the voltage supplied to the motor, the driver also needs an external power supply for the motor; the power supply is set to 12 V. The two motor coils are connected directly to the driver. The connection to the Arduino™ and the stepper motor can be seen in Figure 21 [21].

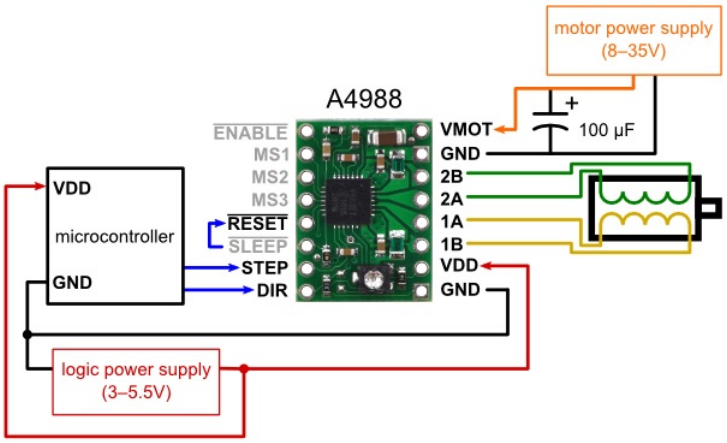


Figure 21: Connection between Arduino™, motor driver and motor

The control of the driver from the Arduino™ is made mainly through two pins: step and dir. A pulse on the step moves the motor one step. A high level on the dir moves the elevator car up and a low level moves the elevator car down.

5.4 Sensors

On each floor is a proximity sensor attached to detect the presence of the approaching passengers. The sensor that is used for the project is an optical reflex sensor, Reflex Coupler SMD. The sensor works in the infrared-light range and has a sensing distance of 1 mm. Because of the short sensing distance it is almost needed to touch the sensors to activate them. This prevents that the wrong sensor is activated on the small prototype. It also makes sure that only one pulse is sent to the microcontroller, instead of a flickering pulse activation that could happen when a finger is moved back and forth over the sensor.

The sensor consists of two different components mounted together in a plastic package. The two components are a GaAs IR-LED transmitter and an NPN photo-transistor (see Figure 22) [22].

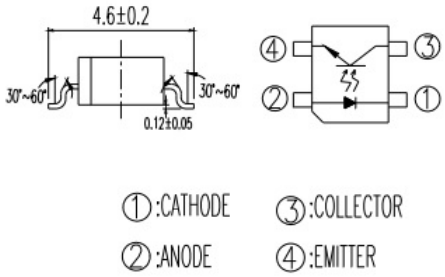


Figure 22: Graphical layout of the sensor, with permission from Everlight[22]

The sensors are interfaced with the Arduino^{TM} microcontroller. The cathode of the IR-LED is connected to ground and the anode is connected to the 5 V pin. The collector of the NPN photo-transistor is connected to the 5 V pin and the emitter is connected to an analog input pin. The cathode and the emitter are connected through the resistance R_1. The anode and the collector are connected through the resistance R_2, see Figure 23 for the connection. The same connection was done on the breadboard for each of the separate sensors.

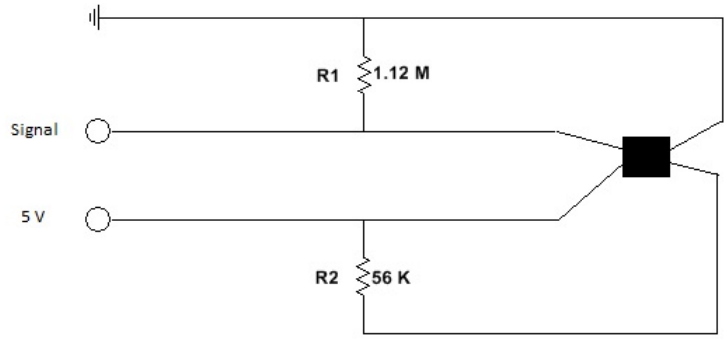


Figure 23: Connection of the sensor

When not activated, the sensors will give a voltage level of 4.5 V to the analog input on the Arduino^{TM} microcontroller. When a sensor is active, the voltage level drops and changes to 1 V. The voltage is measured relative to a reference voltage, 5 V, in the Arduino^{TM}. The input block

Table 1: Specifications of the LEDs

	Red	Green
Drive Voltage	2.2 V - 2.4 V	3.1 V - 3.6 V
Rated Current	20 mA	20 mA
Wavelenght	625 nm	525 nm
Brightness	7000 mcd	9000 mcd

used in Simulink uses 10 bits for the analog input, which gives a value from 0 to 1023. When the input voltage corresponds to the reference voltage, the input block gives a value of 1023 in Simulink. When the input voltage is equal to the ground voltage, a value of 0 is set.

5.5 LEDs

Two LEDs with different colours are used at each floor to display the function of the doors moving. The green LED indicates that the door is opening and the red LED indicates that the door is closing.

The LEDs are also connected to the Arduino™ microcontroller (see Figure 24). The anodes of each LED are connected to one digital output pin and the cathodes are connected to the ground on the microcontroller. To set the desired level of brightness and current through the LED, a resistance is connected in series. The connection of the LEDs and the resistance is done on a breadboard and only one LED is activated by the Arduino™ at a time.

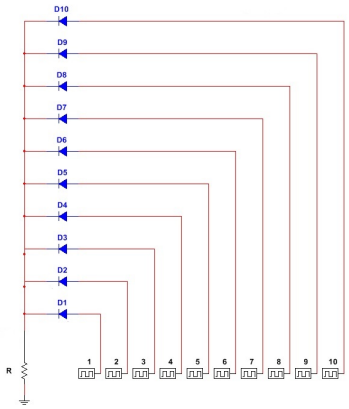


Figure 24: Connection of the LEDs, similiar for both red and green LEDs

The specifications of the LEDs used in the prototype design are shown in Table1, [23] [24]. Since the brightness and intensity of the used LEDs are high, a lower current is needed to not sense the light as too strong. By lowering the current, the intensity of the light is decreased. Only one LED will be fed by an analog high at a time, therefore the calculations can be based on a series connection of a single LED with the resistor. A resistance with a value of R = 1.2 kΩ is used, giving a current of 1.3 mA:

$$I = \frac{5V - 3.4V}{1.2k\Omega} = 1.3mA \tag{12}$$

This leads to a light with a pleasant brightness accordingly to our observations.

For the red LEDs, the voltage is given by 2.3 V. With the same value for the resistance, the current is obtained as:

$$I = \frac{5V - 2.3V}{1.2k\Omega} = 2.25mA \tag{13}$$

The green LED has a higher level of brightness. Therefore the current through the green LED should be lower.

5.6 Buttons

A system of buttons is constructed as a back-up for the speech recognition and has also been used for testing the separate units in the elevator control.

If the speech recognition is not able to function properly, the requested floor can instead be given to the system by buttons, each of them corresponding to one floor.

A small pushbutton switch is used. The buttons have a voltage rating of 12 V and a current rating of 50 mA. The on resistance is smaller than 50 mΩ and the off resistance is more than 100 MΩ [25].

The diagram over the connection of the buttons can be seen in Figure 25.

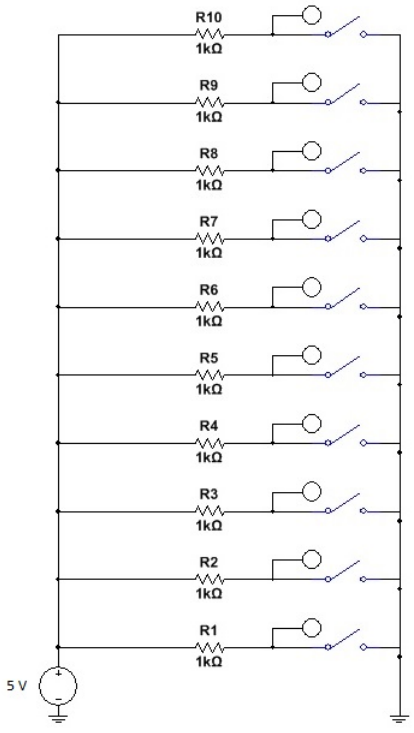


Figure 25: Connection of the buttons

5.7 Microphone

The microphone used in the project is called AKIRO USB MICROPHONE and it is suitable for the project because of several reasons. First of all the microphone has a good sound quality with minimal background noise, which is important for the speech recognition. The microphone is also connected by a USB-port, allowing a connection with the Raspberry Pi board. Lastly, the microphone does not require any drivers to be installed before use [26].

5.8 Speaker

The speaker used in the project is a small speaker of 3 W (RMS). It has a built in battery with a playtime of four hours and a range of 8 m. It has a wired connection with a 3.5 mm jack [27]

5.9 Microcontrollers

The speech recognition algorithms are very demanding in terms of data calculations. It is therefore necessary to run the algorithm on a CPU with a high clock rate. The Raspberry Pi (RPi) 2 Model B microcontroller board is equipped with a 900 MHz quad-core ARM Cortex-A7 CPU, which is why it is used for this project [28]. The code is compiled with MATLAB, since its language features implemented functions suitable for signal processing, such as standardised Fourier transform methods and filters. The program is however created in Simulink, where the code is running through MATLAB Functions. Hence, both the Simulink and MATLAB hardware-support packages are used. The choice of using Simulink is justified by the fact that pure MATLAB-code does not run on the ARM processor (as it is too heavy). Since Simulink converts the MATLAB-code to C, it is regarded a good option.

Arduino^{TM} Mega ADK was chosen for several reasons. First of all, the output voltage of its general-purpose-input-outputs (GPIOs) is 5V, which makes it easier to connect to sensors and actuators, compared to for example a Raspberry Pi, which has some GPIOs at 3.3V. Secondly, it has more pins for inputs and outputs than most of the Arduino^{TM} boards, which is needed due to the large number of sensors, LEDs, etc. Lastly, it has an USB connection, which is an option to facilitate the connection with Raspberry Pi (which holds the speech recognition unit) [29].

An inventory list of all the products used in the project can be seen in Appendix F.

6 Environment and Sustainability

When constructing a prototype, it is important to thoughtfully select the materials to make a sustainable design. In addition to making sure that the prototype fulfils the requirements regarding the performance, it should also be taken into consideration that certain material choices affect the environment differently.

The main part of the prototype is built in aluminium. Recycling of aluminium saves 95 % of the total energy required to manufacture new metal. With the requirements set up for the prototype, this choice of material was good from sustainability aspect, assuming the metal is going to be recycled.[30]

The elevator car is printed with a 3D printer in plastic. This due to the complex design of the elevator car. Plastic is mostly made out of oil and when producing 1 kg of plastic, 2 kg of oil is required. When recycling plastic, it is almost unavoidable that different plastics are mixed, as a result of which the mechanical properties often decreases and there are many residual products in this process[31]. However, the plastic used for the elevator car is polylactic acid, which contains biodegradable raw materials such as corn or sugarcane and are renewable.

Batteries are made by various chemicals, but always one hard metal. Some of these are toxic such as nickel and cadmium. These materials can cause damages on both humans as well as the environment and many of them are not possible to recycle. Therefore, the system was chosen to be powered by cables instead of using batteries.

Nowadays, all real elevators use counter weight to minimise the energy consumption. However, in the downscaled prototype it can be questioned how much impact a counter weight would have on the energy consumption, since the weight of the elevator car is only 50g. Real elevators also have more advanced pulley systems, with more pulleys. This could also be implied in the prototype and would probably increase the precision of the elevator car but would probably not affect the energy consumption remarkably.

7 Improvements and Issues in Upscaling

Through the process, additional features of the prototype have not been included. Some of them have been added to the list of boundaries and some have not been considered because of the time limit. Additional features and improvements can be considered in future works, as detailed in the following subsections.

7.1 Required Improvements in the Downscaled Prototype

The prototype algorithms and functions can be improved for better performance and extended with additional features. The speech recognition system can be improved from many aspects. It can for example be improved by developing the classification process to increase the recognition rate.

The SRU is activated after a proximity sensor is triggered. The system then records a speech signal from the passenger for 3 seconds. After this time period, the microphone is no longer active and the algorithm for the recognition is operating. This is an isolated speech recognition system in which only one word can be recognised for each activation of the proximity sensors. An improvement of the system is to instead create a continuous system which listens for a longer time and is able to recognise several numbers and longer sentences. This way, the sensor would be activated once, but different people can leave their requests on the same activation. This requires a change in the speech recognition activation, as well as a modification in the elevator control. The control unit that sorts the queue is expecting only two numbers: one for where the passenger will enter and one for where the passenger wants to exit the elevator. With several numbers as an input, the algorithm for sorting the requested floors must function in a different way.

An additional operation for the SRU is to add a function with the possibility to change the recorded number. This would be useful if the speech recognition did not work properly, if the background noise corrupted the signal or simply if the passenger would like to change the requested target floor. The SRU should thereby ask the passenger if the registered target floor is correct, to confirm that this is the number that should be sent to the elevator control.

To improve the voice control, short words could be included in the SRU. These words can be "open" and "close" but also shorter phrases such as "open doors", "close doors" and for example "floor two". With phrases included in the recognition system, the system can handle a larger spectrum of situations. It is not possible to control the passenger to only say the direct number. If the passenger is to say "floor two" instead of only "two", the SRU in its present form, would not be able to recognise the requested floor.

The function of the sensors can be improved, allowing the user to flicker the finger in front of the sensor without sending several activation pulses to the input of the elevator control. This could be done by only activating the input signal when the sensor has been active for a number of sample points. The verge between not active and active on the sensor is then not as sharp and would give a more accurate registration of the passengers. The sensors should not be activated by people that walk past the elevator door without attempting to enter the elevator car. This could be done in the same way, by neglecting a voltage drop over the sensor if the level is not low for a certain number of sample times.

The total weight of the elevator car has not been taken into consideration during the project. The elevator system should be developed to sense the total weight of the passengers and only move the elevator car if the upper weight limit is not reached. Also the elevator car should not stop and let passengers enter, if the weight is close to the upper limit. The design of taking the orders outside of the elevator allows even more advantages in the design of the elevator control. By knowing how many people are waiting outside of the elevator car it is possible to decide if the elevator car should stop or not. If the total weight is close to the upper limit and there are several people

waiting to enter, it would be a better solution to not stop at that floor, and instead move on to the floors where the current passengers want to exit.

A situation that can occur during the run of the elevator is the fact that a requesting passenger might make an order to the elevator system, but later when the elevator arrives, is no longer interested in going with the elevator. This can be avoided by using the sensors. If the sensor is still active, the requesting passenger is still standing by the elevator and thereby wants to go with the elevator. With this design system, the elevator will not stop and open the doors when no one wants to enter.

The elevator control is only designed to control one elevator. The system could be changed to include operations of several elevator cars, moving in parallel. With more elevator cars in the system, an optimisation of the power consumption can be taken into consideration in a larger scale. The decision of which car should take the different orders has to be done. The decision should be based on how the lowest power consumption could be reached. It should also be based on the optimised path for the passengers, with as few stops at possible during the ride. The ordering of the queues requires a more advanced structure of the ECU.

Since a stepper motor is used in the project, an advanced motor control was not needed to be implemented. By choosing a different motor and motor driver a more advanced motor control can be implemented. With the use of an encoder, a feedback signal would be available, allowing a PID-controller to be implemented. This improvement is also crucial when aiming for the design of a real elevator system.

7.2 Upscaling for a Real Elevator System

As the prototype is downscaled, some functions have been adapted to fit in a smaller scale. In order to upscale the prototype and implement it to a real-sized system, some parts need to be changed. These include the parts of the construction and the elevator algorithm.

When upscaled, the system goes through some physical changes. The elevator shaft would be bigger and the elevator car would be heavier. This leads to a greater demand on the motor and the pulley system. The motor needs to be strong, reliable and precise.

Because of the bigger scale and heavier elevator car, it is important to think about how to reduce the energy consumption for the elevator system. First of all, the pulley system is to be constructed in a different way. It is not enough to only have one single bobbin. The pulley system should be assembled with a number of pulleys to ease the power needed from the motor and it also helps to increase the precision. Furthermore, a counterweight can be attached to the pulley system and, together with the weight, the pulley system will decrease the power needed to move the elevator car even more. An additional gear system can be implemented in the pulley system, where the gear system increases the precision and velocity control and decreases the torque needed.

When transporting human beings, it is important that the system is safe and accidents are avoided. The system should therefore be equipped with security breaks that can prevent the elevator car to fall down the shaft.

A central function of the downscaled system is the use of a single microphone. The implementation is done in this fashion because of the small area and the wide scope of the microphone. In a real system, the floors are separated and one microphone is needed on each floor. The upscaled version of the system would also require the proximity sensors to have a larger sensing distance and a wider scope. The sensors should be activated when a person is standing in front of the elevator door.

To implement the construction in a real system, doors are to be mounted on each floor. This includes an algorithm to control the doors in the microcontroller and motors on each floor to

operate them. The doors also need sensors to detect if the opening is cleared before closing the door. The doors should only open when the elevator car is standing still.

To add extra security to the system, an emergency break operated inside of the elevator car by the passengers should be implemented. The emergency break should stop the elevator car as fast as possible and yet in a safe way. A voice-operated emergency break requires some extra thought. It is to be decided which words are needed to be said. If the SRU does not function correctly, a physical button might be needed to enhance the safety function.

8 Discussion and Evaluation of the Achievements

Since it is the first time the Smartlift project is carried out in the Department of Signals and Systems at Chalmers University of Technology, a start-up phase of research was required. Furthermore, a prototype needed to be both designed and constructed. For this reason, most of the time in the first two months was spent in researching about which hardware and software to use. It was necessary to do an extensive research about microcontroller boards, since we had a number of crucial requirements. These relate to high memory capacity and clock rate to hold the signal processing, as well as the number of inputs and outputs for the large number of components. Deciding which software programs and motion actuators were more suitable to use was also an important task. Thanks to the wide research, it was possible to design the overall system. Also, during the first two months, significant amount of time was spent in the design of the construction, even though it was not the main focus of the project.

When implementing the designed system, we faced challenges which were not predicted during the research phase. The SRU code was written in MATLAB, both since it is quite convenient for signal processing and since it is a programming language that the group members are familiar with. The basic idea was to use Simulink to implement the code in RPi, but unexpected problems occurred in this phase. Since some MATLAB functions are not supported by Simulink, it was not possible to embed the code to RPi directly from MATLAB. After facing this problem, it was tried to convert the code to C, which RPi can compile. However, this did not work either, as it was not possible to obtain the proper libraries required. This shows the necessity of having at least one group member with more advanced knowledge in a high level programming language such as C. This would have made the implementation process possible.

It is important to note that a more advanced control could be designed with a feedback signal from an encoder. Having prioritised the whole system integration as the major target, a simple stepper motor is chosen and a relatively simple approach is adopted for the control design.

The major achievements of the Smartlift project are summarised below:

- The speech recognition system was a challenging unit to develop, due to the lack of significant background knowledge. The methods of the SRU are based on previous research papers in this field, from where they have been adapted to the project necessities. A large variety of sources were investigated and their contents were thoughtfully selected to fit the project task. The outcome of this work was successful and the SRU has an average speaker dependent success rate of 90 % .
- Regarding the ECU, a major concern throughout the design process has been the realisability of the implementation, due to the fact that several units have to run simultaneously in the same program in Arduino^{TM}. In order to achieve this, it was crucial to trigger and enable all the subsystems in the correct way. Moreover, the initial conditions to outputs and memory blocks needed to be set correctly. Thanks to this, the final operation of this unit functions as expected.
- Writing the elevator algorithm code, i.e. the one that sorts the requested floors, was time consuming. It was difficult to take into account all the possible cases that can occur when adding two new numbers to the array of destinations. After many tests, the algorithm was successfully completed and its operation has been verified.
- The implemented motion control permits to demonstrate the general operation of the elevator. Nevertheless, since a stepper motor is used within the prototype, the movement of the elevator car is not as smooth as it should be in an up-scaled system.
- The prototype includes a large number of components attached to the construction. Therefore, it was time consuming to make sure that all of them were placed properly and connected in the right way. The connection of the sensors demanded additional workload as they had

to be connected to individual circuits on the breadboard. The system with the different components was successfully achieved, including the connection of the motor driver, sensors and LEDs.

The Smartlift project is completed without the integration of the speech recognition unit into the elevator control unit. It has been a challenging and fruitful learning and experience for the group members. It is hoped that the implementation would be completed together with improved designs in the coming years.

9 Concluding Remarks

Within the Smartlift project, a downscaled prototype of an elevator has been built and two units, one for elevator control and one for speech processing, have been implemented. The elevator control consists of a state machine and an algorithm which minimises the distance travelled by the elevator car by sorting the requested floors in an intelligent way. The implementation of the elevator control was successfully achieved. The movement of the elevator car is controlled with a stepper motor. The speech recognition contains the use of MFCC and the classification process DTW.

The main focus of the Smartlift project was to integrate several different parts. When looking at this at a higher level, it is notable that some tasks required more thought than expected. For example the SRU could not be realisable without taking the ECU into consideration.

The speaker dependent SRU functions properly. The algorithm code provides a good basis for further development (e.g. speaker independent and continuous system).

Voice-activated elevators facilitate the operation of the elevator car. The development of this kind of systems also increases the availability for blind or physically impaired people. Therefore, this kind of project can be helpful by improving the ability of transportation by elevator for many people.

References

[1] B. H. Juang and L. R. Rabiner, "Automatic Speech Recognition – A Brief History of the Technology Development," October 2004.

[2] G. I. of Technology and L. R. Rabiner, *Automatic Speech Recognition – A Brief History of the Technology Development*. PhD thesis, Georgia Institute of Technology, Atlanta, United States of America, October 2004.

[3] S. Satapathy *et al.*, *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications*, vol. 1. Springer, 2008.

[4] J. Benesty *et al.*, *Springer Handbook of Speech Processing*. Springer, 2014.

[5] M. R. Hasan *et al.*, "Speaker identification using mel frequency cepstral coefficients," tech. rep., Indian Institute of Technology Kanpur, City University of New York, U.S. Dept. of Defense, Kanpur, India, New York, USA, Ft. Meade, USA, 1999.

[6] S. Umesh *et al.*, "Fitting the mel scale," tech. rep., Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, December 2004.

[7] A. University, "The mel frequency scale and coefficients," tech. rep., Aalborg University, Aalborg, Denmark, 2004.

[8] R. Freeman, "Fundamentals of telecommunications," tech. rep., John Wiley Sons, Inc, Scottsdale, USA, November 1999.

[9] P. Felber, "Speech recognition: Report of an isolated word experiment," tech. rep., Illinois Institute of Technology, Chicago, USA, April 2001.

[10] A. Aibinu *et al.*, "Evaluating the effect of voice activity detection in isolated yoruba word recognition system," tech. rep., International Islamic University, Kuala Lumpur, Malaysia, May 2011.

[11] VOCAL Technologies ltd, "Standard methods of vad." `http://www.vocal.com/voice-quality-enhancement/standard-methods-of-voice-activity-detection-vad`. Accessed: 2015-03-17.

[12] S. Dev Dhingra *et al.*, "Isolated speech recognition using mfcc and dtw," tech. rep., Dept. of ECE, Faridabad, India, August 2013.

[13] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," tech. rep., IEEE, New York, USA, 1989.

[14] Jyh-Shing Roger Jang, "Audio Signal Processing and Recognition." `http://www.cs.nthu.edu.tw/~jang`. Accessed: 2015-04-12.

[15] S. V. Chapaner, "Spoken digits recognition using weighted mfcc and improved features for dynamic time warping," tech. rep., St. Francis Institute of Technology, Mumbai, India, February 2012.

[16] S. Gupta *et al.*, "Feature extraction using mfcc," tech. rep., Universiti Tecknologi PETRONAS, Perak, Malaysia, August 2013.

[17] T. Ganchev *et al.*, "Comparative evaluation of various mfcc implementations on the speaker verification task," tech. rep., University of Patras, Rion-Patras, Greece.

[18] L. Salhi and A. Cherif, "Robustness of auditory teager energy cepstrum coefficients for classification of pathological and normal voices in noisy environments," *The Scientific World Journal*, vol. Volume 2013, May 2013.

[19] A. Pramanik and R. Raha, "Speaker independent word recognition using cepstral distance measurement," tech. rep., Indian Institute of Technology, Kharagpur, India.

[20] Mercury Motor, New Taipei City, Taiwan, *ST-PM35-15-11C Small Stepper Motor*, issue: 1 ed., May 2011.

[21] Allegro MicroSystems, Worcester, Massachusetts, *DMOS Microstepping Driver with Translator And Overcurrent Protection*, revision: 5 ed.

[22] E. Chen, *Technical Data Sheet Opto Interrupter*. Everlight Electronics Co., Ltd., New Taipei City, Taiwan, revision: 2 ed., September 2008.

[23] Kjell & Company, "Led, 3 mm, red." http://www.kjell.com/sortiment/el/elektronik/optokomponenter/lysdioder/lysdiod-3-mm-rod-p90700. Accessed: 2015-04-6.

[24] Kjell & Company, "Led, 3 mm, green." http://www.kjell.com/sortiment/el/elektronik/optokomponenter/lysdioder/lysdiod-3-mm-gron-p90701. Accessed: 2015-04-6.

[25] Pololu Robotics and Electronics, "Mini pushbutton switch." https://www.pololu.com/product/1400. Accessed: 2015-04-20.

[26] Kinobo, "AKIRO USB microphone." http://www.kinobo.co.uk/kinobo/microphone/akiro-51.html. Accessed: 2015-04-01.

[27] Kjell & Company, "Roxcore Cubee Portabel Bluetooth-högtalare." http://www.kjell.com/sortiment/dator-kringutrustning/tillbehor-till-surfplattor/hogtalare/tradlosa-hogtalare/roxcore-cubee-portabel-bluetooth-hogtalare-svart-p95731#ProductDetailedInformation. Accessed: 2015-05-01.

[28] Raspberry Pi Foundation, "RASPBERRY PI 2 MODEL B." https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. Accessed: 2015-02-09.

[29] Arduino, "Arduino MEGA ADK." http://www.arduino.cc/en/Main/ArduinoBoardMegaADK?from=Main.ArduinoBoardADK. Accessed: 2015-02-09.

[30] RagnSells, "Energibesparing med återvinning." http://www.ragnsells.se/sv/Miljokunskap/Miljonyttan-med-atervinning/Energibesparing-med-atervinning/. Accessed: 2015-05-19.

[31] Stockholmsregionens Avfallråd, "Återvinning." http://www.atervinningscentralen.se/web/page.aspx?refid=180. Accessed: 2015-05-19.

Appendices

A Drawing of the construction

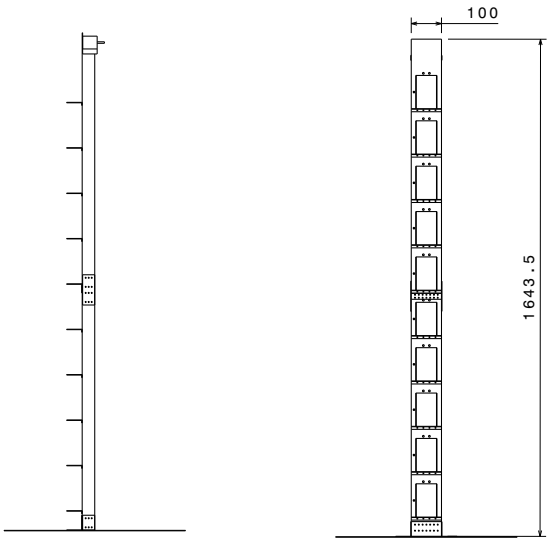


Figure 26

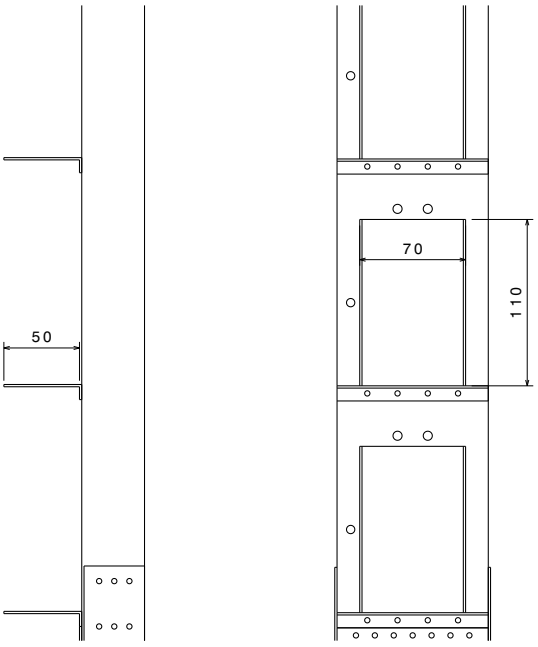


Figure 27

B Layout and functions of the motor driver

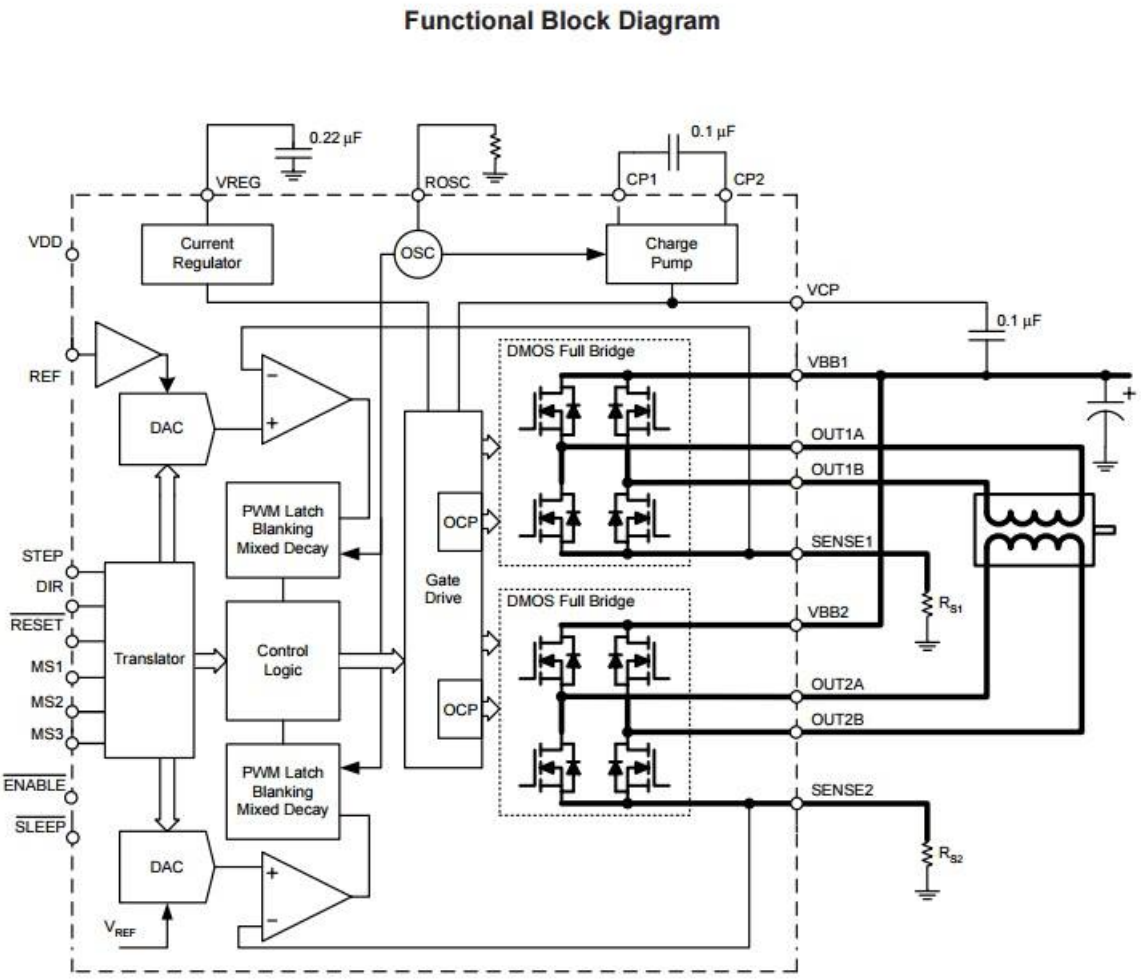


Figure 28

C Unit 1, Unit 2 and Unit 3 Simulink Models adapted for SRU

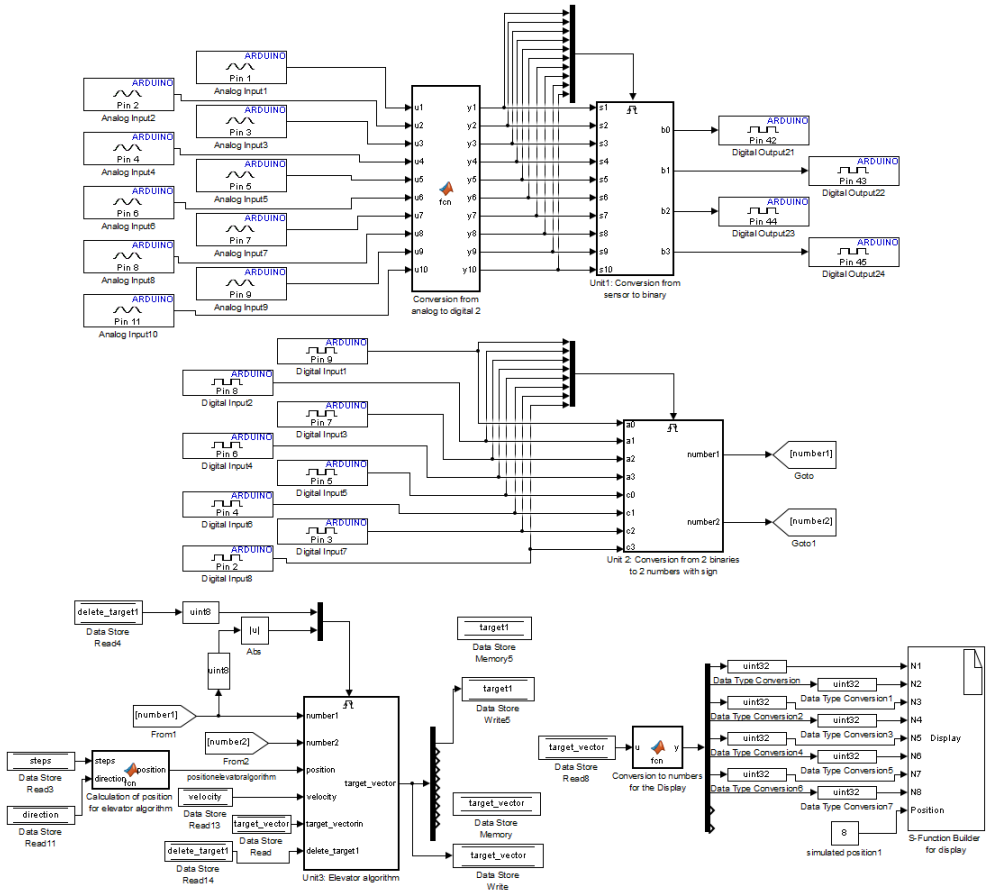


Figure 29

D State Machine Simulink Model

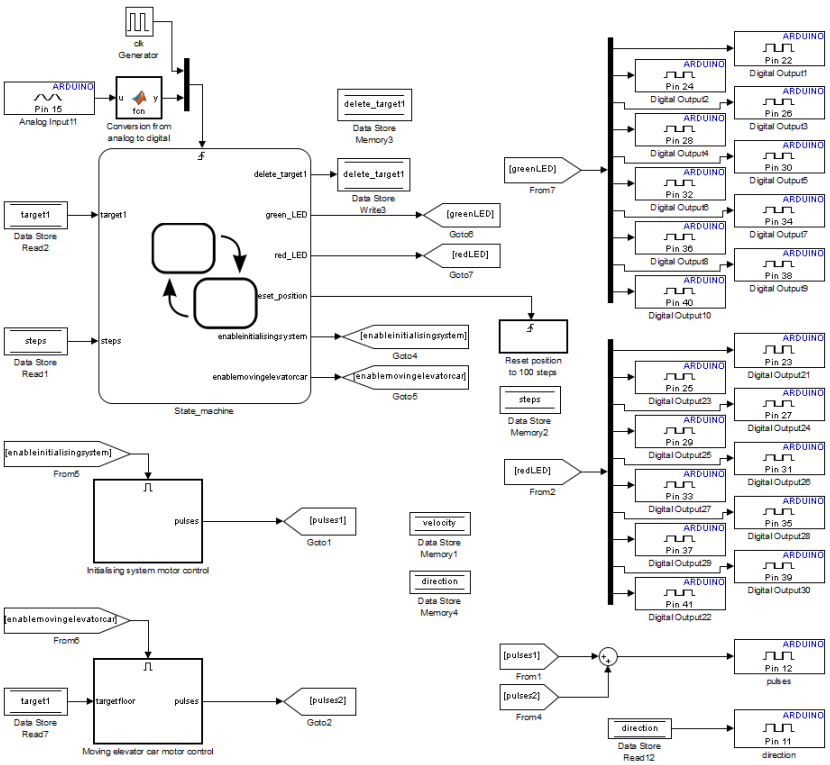


Figure 30

E Unit 1, Unit 2 and Unit 3 Simulink Modelsadapted for Buttons

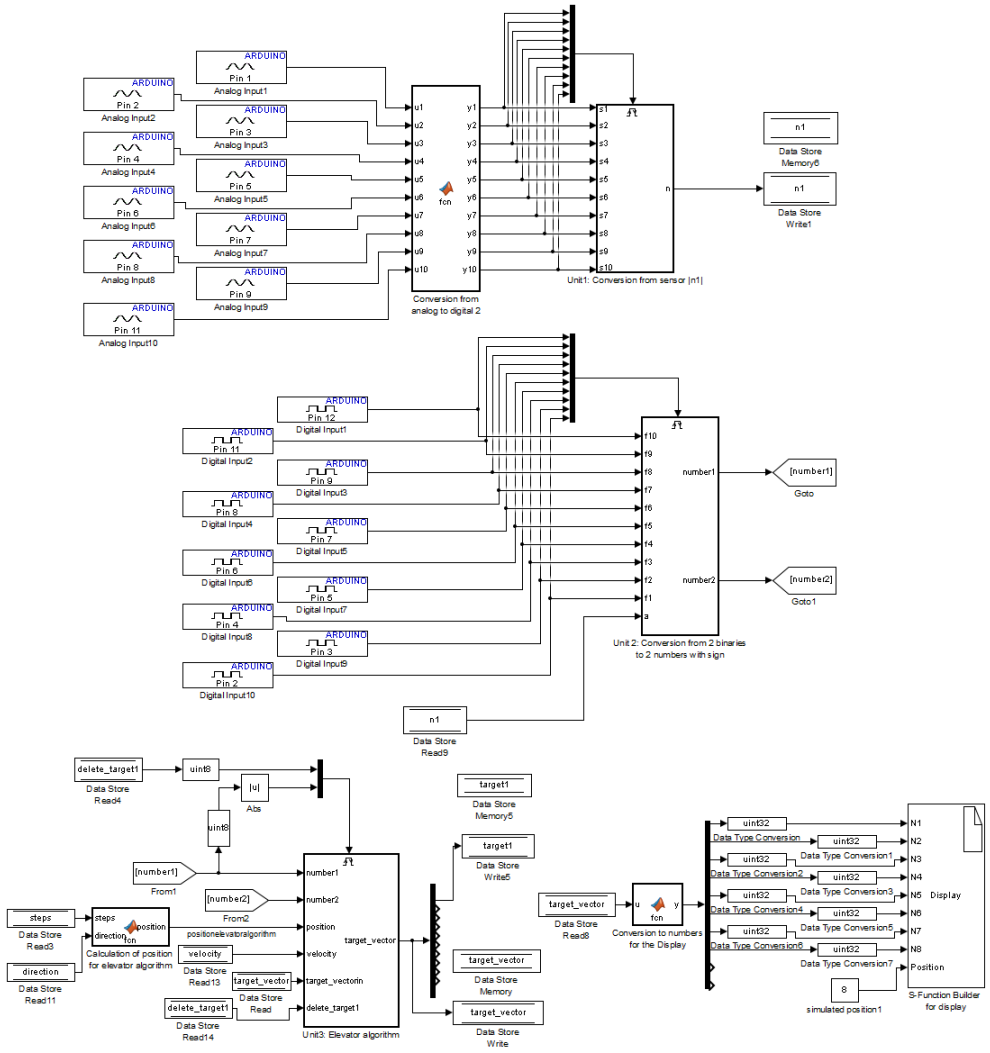


Figure 31

F Inventory list

Amount	Hardware	Model	Cost
1	Stepper Motor	ST-PM35-15-11C, bipolar	250
1	Motor Driver	A4988	138
	Arduino Mega Proto Shield R3	-	55
1	Arduino Board	Arduino Mega ADK	494
1	Rasberry Pi Board	Rasberry Pi Model B	399
1	USB to Power cord	-	108
1	SD-card	MicroSDHC 8 GB Class 6	49
2	Block Terminal	1.5mm² 12-pol	24
2	Screw Terminals	5mm 18-pol	36
-	Wires	-	158
25	Sensors	Reflex detector SMD	288
50	Diodes	5 mm green & red	60
1	Microphone	AKIRO USB Microphone	253
1	Breadboard	-	319
1	USB A/A cable	-	100
1	Segment Display	-	130
10	Pushbotton	-	39
1	Power Cord	-	113
1	Speaker	-	300
-	Cable ties	-	30
1	Fishing Line	-	49
-	Wires male-female	-	50
SUM			3442

G Unit 1 Code for the Case of Operation with SRU

```
function [b0, b1, b2, b3] = fcn(s1, s2, s3, s4, s5, s6, s7, s8, s9, s10)

sth=0.5; %sensor threshold
bth=1; %threshold for the binary communication to the Raspberry Pi

if (s1>=sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<sth
    && s9<sth && s10<sth)
    n=[0,0,0,bth];
elseif (s1<sth && s2>=sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=[0,0,bth,0];
elseif (s1<sth && s2<sth && s3>=sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=[0,0,bth,bth];
elseif (s1<sth && s2<sth && s3<sth && s4>=sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=[0,bth,0,0];
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5>=sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=[0,bth,0,bth];
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6>=sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=[0,bth,bth,0];
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7>=sth && s8<
    sth && s9<sth && s10<sth)
    n=[0,bth,bth,bth];
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8>=
    sth && s9<sth && s10<sth)
    n=[bth,0,0,0];
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9>=sth && s10<sth)
    n=[bth,0,0,bth];
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10>=sth)
    n=[bth,0,bth,0];
else
    n=[0,0,0,0]; %no sensor activated or error
end;

%n=[n(1), n(2), n(3), n(4)]= [b_3, b_2, b_1, b_0]
b3=n(1);
b2=n(2);
b1=n(3);
b0=n(4);
```

H Unit 2 Code for the Case of Operation with SRU

```
function [number1, number2] = fcn(a0, a1, a2, a3, c0, c1, c2, c3)

bth=0.5; %threshold for the binary communication from the Raspberry

if a0<=bth
    a0=1;
else
    a0=0;
end

if a1<=bth
    a1=1;
else
    a1=0;
end

if a2<=bth
    a2=1;
else
    a2=0;
end

if a3<=bth
    a3=1;
else
    a3=0;
end

if c0<=bth
    c0=1;
else
    c0=0;
end

if c1<=bth
    c1=1;
else
    c1=0;
end

if c2<=bth
    c2=1;
else
    c2=0;
end

if c3<=bth
    c3=1;
else
    c3=0;
end


a_natural=a0+a1*2^1+a2*2^2+a3*2^3; %sensor floor
c_natural=c0+c1*2^1+c2*2^2+c3*2^3; %desired floor

if a_natural>0 && c_natural>0 && a_natural<=10 && c_natural<=10
    sus=a_natural-c_natural;
    if sus>0 %it is wanted to go down
        number1=-a_natural;
        number2=-c_natural;
    elseif sus<0 %it is wanted to go up
        number1=+a_natural;
        number2=+c_natural;
    else % error, the person does not want to change floor
```

```
66              number1=0;
67              number2=0;
68          end
69      else %error in a or c
70          number1=0;
71          number2=0;
72      end
```

I Unit 1 Code for the Case of Operation with Buttons

```
function n = fcn(s1, s2, s3, s4, s5, s6, s7, s8, s9, s10)

sth=0.5; %sensor threshold

if (s1>=sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<sth
    && s9<sth && s10<sth)
    n=1;
elseif (s1<sth && s2>=sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=2;
elseif (s1<sth && s2<sth && s3>=sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=3;
elseif (s1<sth && s2<sth && s3<sth && s4>=sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=4;
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5>=sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=5;
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6>=sth && s7<sth && s8<
    sth && s9<sth && s10<sth)
    n=6;
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7>=sth && s8<
    sth && s9<sth && s10<sth)
    n=7;
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8>=
    sth && s9<sth && s10<sth)
    n=8;
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9>=sth && s10<sth)
    n=9;
elseif (s1<sth && s2<sth && s3<sth && s4<sth && s5<sth && s6<sth && s7<sth && s8<
    sth && s9<sth && s10>=sth)
    n=10;
else
    n=0; %no sensor activated or error
end;
```

J Unit 2 Code for the Case of Operation with Buttons

```
function [number1, number2] = fcn(f10, f9, f8, f7, f6, f5, f4, f3, f2, f1, a)

bth=0.5; %threshold for the binary communication to/from the raspberry

if (f1<=bth && f2>bth && f3>bth && f4>bth && f5>bth && f6>bth && f7>bth && f8>bth
    && f9>bth && f10>bth)
    c=1;
elseif (f1>bth && f2<=bth && f3>bth && f4>bth && f5>bth && f6>bth && f7>bth && f8>
    bth && f9>bth && f10>bth)
    c=2;
elseif (f1>bth && f2>bth && f3<=bth && f4>bth && f5>bth && f6>bth && f7>bth && f8>
    bth && f9>bth && f10>bth)
    c=3;
elseif (f1>bth && f2>bth && f3>bth && f4<=bth && f5>bth && f6>bth && f7>bth && f8>
    bth && f9>bth && f10>bth)
    c=4;
elseif (f1>bth && f2>bth && f3>bth && f4>bth && f5<=bth && f6>bth && f7>bth && f8>
    bth && f9>bth && f10>bth)
    c=5;
elseif (f1>bth && f2>bth && f3>bth && f4>bth && f5>bth && f6<=bth && f7>bth && f8>
    bth && f9>bth && f10>bth)
    c=6;
elseif (f1>bth && f2>bth && f3>bth && f4>bth && f5>bth && f6>bth && f7<=bth && f8>
    bth && f9>bth && f10>bth)
    c=7;
elseif (f1>bth && f2>bth && f3>bth && f4>bth && f5>bth && f6>bth && f7>bth && f8<=
    bth && f9>bth && f10>bth)
    c=8;
elseif (f1>bth && f2>bth && f3>bth && f4>bth && f5>bth && f6>bth && f7>bth && f8>
    bth && f9<=bth && f10>bth)
    c=9;
elseif (f1>bth && f2>bth && f3>bth && f4>bth && f5>bth && f6>bth && f7>bth && f8>
    bth && f9>bth && f10<=bth)
    c=10;
else
    c=0; %no sensor activated or error
end;


a_natural=a; %sensor floor
c_natural=c; %desired floor

if a_natural>0 && c_natural>0 && a_natural<=10 && c_natural<=10
    sus=a_natural-c_natural;

    if sus>0 %it is wanted to go down
        number1=-a_natural;
        number2=-c_natural;
    elseif sus<0 %it is wanted to go up
        number1=+a_natural;
        number2=+c_natural;
    else % error, the person does not want to change floor
        number1=0;
        number2=0;
    end
else %error in a or c
    number1=0;
    number2=0;
end
```

K Elevator Algorithm Code

```
function t  = fcn(number1, number2, position, velocity, tprevious,...
delete_target1)
%position has to be the closest floor to the elevator car with + sign if
%the elevator car is going up or - sign if the elevator car is going down.

%% Creation of the array t, that will contain the current requested floors:

t=[0,0,0,0,0,0,0,0,0,0]';

%% Calculation of the margin needed to add new target1 (floormargin) from
% 0 to 1, depending on the velocity of the elevator:

if velocity==0 %if the elevator is stopped
    floormargin=0; %it can go to the current position if requested
elseif position==tprevious(1) %the elevator car is stopping
    floormargin=0; %it can go to the current position if requested
else
    floormargin=1; %it cannot go to the current position
end

%% If there are 2 new requested floors their position in the array has to
% be found and t has to be updated:

if number1~=0 && number2~=0
    if tprevious(9)==0 %the array has at least two empty spaces

        if tprevious(1)>0 %elevator going up
            if number1>0 %the new person also wants to go up, number1 and
                %number2 have the same sign
                nextposition=position+floormargin; %Next position where
                %target1 can be placed in the first part of the array
                if number1>=nextposition %we have time to change first
                    %target floor if needed.
                    %The position of the target 1 is surely going to be in
                    %the first positive part of the array.
                    i=1; %counter for the while loop
                    x1=0; %position of number1, '0' means not found and
                    %'11' means not change needed in the array
                    pos1=0; %position of number1 even if it is already in
                    %the array
                    while i<11
                        if tprevious(i)>0 %still looking in the positive
                            %part of the array
                            if number1<tprevious(i) %place of number1 found
                                x1=i;
                                pos1=i;
                                i=11; % end while
                            elseif number1==tprevious(i)
                                x1=11; % there is no change needed
                                pos1=i;
                                i=11; % end while
                            else
                                if tprevious(i)>tprevious(i+1) %i is the
                                    %last position of the first positive
                                    %part. So number1 has to be placed in
                                    %i+1
                                    x1=i+1;
                                    pos1=i+1;
                                    i=11;
                                else
                                    i=i+1;
                                end
                            end
                        else %looking in the end of positive numbers. So
                            %place of number1 = i
```

```
66                          x1=i;
67                          pos1=i;
68                          i=11; % end while
69                      end
70                  end
71
72                  %number2 is going to be also in the first positive
73                  %part of the array and after the position of
74                  %number1. The exact position is searched:
75
76                  j=pos1; %counter for the while loop
77                  x2=0; %position of number2
78                  while j<11
79                      if tprevious(j)>0 %still looking in the positive
80                          %part of the array
81                          if number2<tprevious(j) %place of number2
82                              %found=j in tprevious
83                              x2=j;
84                              j=11; % end while
85                          elseif number2==tprevious(j)
86                              x2=11; % there is no change needed
87                              j=11; % end while
88                          else
89                              if tprevious(j)>tprevious(j+1) %n is the
90                                  %last position of the first positive
91                                  %part. So number2 has to be placed in
92                                  %j+1
93                                  x2=j+1;
94                                  j=11;
95                              else
96                                  j=j+1;
97                              end
98                          end
99                      else %looking in the end of positive numbers. So
100                         %place of number2 = j in tprevious
101                         x2=j;
102                         j=11; % end while
103                     end
104                 end
105
106
107
108
109             else
110                 %number1 is already passed, we have to search the
111                 %position of number1 in the second part of positive
112                 %numbers:
113                 %First, the end of the first positive part is searched:
114                 i=2;%counter for the while loop, we start in position 2
115                 %because it has been already checked that the first
116                 %position is positive
117                 p1=0;%position of the first number in the array that
118                 %does not belong to the first positive part
119                 while i<11
120                     if (tprevious(i)<=0)||(tprevious(i)<tprevious(i-1))
121                         p1=i;
122                         i=11; %end while
123                     else
124                         i=i+1;
125                     end
126                 end
127                 %Second, the exact positions of number1 and number2 are
128                 %searched:
129                 if tprevious(p1)==0 %position of number1 = p1 and
130                     %position of number2 = p1
131                     x1=p1;
132                     x2=p1;
```

```
133
134                     elseif tprevious(p1)>0
135
136                         i=p1; %counter for the while loop
137                         x1=0; %position of number1, '0' means not found and
138                         %'11' means not change needed in the array
139                         pos1=0; %position of number1 even if it is already
140                         %in the array
141                         while i<11
142                             if tprevious(i)>0 %still looking in the positive
143                                 %part of the array
144                                 if number1<tprevious(i) %place of number1
145                                     %found = i
146                                     x1=i;
147                                     pos1=i;
148                                     i=11; % end while
149                                 elseif number1==tprevious(i)
150                                     x1=11; % there is no change needed
151                                     pos1=i;
152                                     i=11; % end while
153                                 else
154                                     i=i+1;
155                                 end
156                             else %looking in the end of positive numbers. So
157                                 %place of number1 = i
158                                 x1=i;
159                                 pos1=i;
160                                 i=11; % end while
161                             end
162                         end
163
164                         %number2 is going to be also in this positive part
165                         %of the array and after the position of number1.
166                         %The exact position is searched:
167
168                         j=pos1; %counter for the while loop
169                         x2=0; %position of number2
170                         while j<11
171                             if tprevious(j)>0 %still looking in the positive
172                                 %part of the array
173                                 if number2<tprevious(j) %place of number2
174                                     %found = j in tprevious
175                                     x2=j;
176                                     j=11; % end while
177                                 elseif number2==tprevious(j)
178                                     x2=11; % there is no change needed
179                                     j=11; % end while
180                                 else
181                                     j=j+1;
182                                 end
183                             else %looking in the end of positive numbers. So
184                                 %place of number2 = j in tprevious
185                                 x2=j;
186                                 j=11; % end while
187                             end
188                         end
189
190
191                     else %tprevious(p1) is negative, it is needed to search
192                         %where the negative part ends
193                         l=p1+1; %counter for while
194                         n1=0; %position of the array where the new positive
195                         %or zero part starts
196                         while l<11
197                             if tprevious(l)>=0
198                                 n1=l;
199                                 l=11; %end while loop
```

```
                    else
                        l=l+1;
                    end
                end
                if tprevious(n1)==0 %position of number1 = n1,
                    %position of number2 = n1
                    x1=n1;
                    x2=n1;

                else %tprevious(n1)>0
                    i=n1; %counter for the while loop
                    x1=0; %position of number1, '0' means not found
                    %and '11' means not change needed in the array
                    pos1=0; %position of number1 even if it is
                    %already in the array
                    while i<11
                        if tprevious(i)>0 %still looking in the
                            %positive part of the array
                            if number1<tprevious(i) %place of
                                %number1 found = i
                                x1=i;
                                pos1=i;
                                i=11; % end while
                            elseif number1==tprevious(i)
                                x1=11; % there is no change needed
                                pos1=i;
                                i=11; % end while
                            else
                                i=i+1;
                            end
                        else %looking in the end of positive
                            %numbers. So place of number1 = i
                            x1=i;
                            pos1=i;
                            i=11; % end while
                        end
                    end

                    %number2 is going to be also in this positive
                    %part of the array and after the position of
                    %number1. The exact position is searched:

                    j=pos1; %counter for the while loop
                    x2=0; %position of number2
                    while j<11
                        if tprevious(j)>0 %still looking in the
                            %positive part of the array
                            if number2<tprevious(j) %place of
                                %number2 found = j in tprevious
                                x2=j;
                                j=11; % end while
                            elseif number2==tprevious(j)
                                x2=11; % there is no change needed
                                j=11; % end while
                            else
                                j=j+1;
                            end
                        else %looking in the end of positive
                            %numbers. So place of number2 = j in tprevious
                            x2=j;
                            j=11; % end while
                        end
                    end

                end

            end
```

```
267                 end
268             else %new person wants to go down, because number1 and number2
269                 %are negative
270                 %searching where the first positive part of the array ends:
271                 i=2; %counter for while, we start in position 2 because it
272                 %has already been checked that the 1st position is positive
273                 p1=0; %position of the array where the new positive,
274                 %negative or zero part starts
275                 while i<11
276                     if tprevious(i)<=0 || tprevious(i)<tprevious(i-1)
277                         p1=i;
278                         i=11; %end while
279                     else
280                         i=i+1;
281                     end
282                 end
283
284                 if tprevious(p1)>=0 %position of number1 = p1, position of
285                     %number2 = p1
286                     x1=p1;
287                     x2=p1;
288
289                 else
290                     %tprevious(p1) is negative. Positions of number1 and
291                     %number2 have to be found in the same negative part
292                     %from position p1
293
294                     i=p1; %counter for the while loop
295                     x1=0; %position of number1, '0' means not found and
296                     %'11' means not change needed in the array
297                     pos1=0; %position of number1 even if it is already in
298                     %the array
299                     while i<11
300                         if tprevious(i)<0 %still looking in the negative
301                             %part of the array
302                             if number1<tprevious(i) %place of number1 found
303                                 x1=i;
304                                 pos1=i;
305                                 i=11; % end while
306                             elseif number1==tprevious(i)
307                                 x1=11; % there is no change needed
308                                 pos1=i;
309                                 i=11; % end while
310                             else
311                                 i=i+1;
312                             end
313                         else %looking in the end of negative numbers. So
314                             %place of number1 = i
315                             x1=i;
316                             pos1=i;
317                             i=11; % end while
318                         end
319                     end
320
321                     %number2 is going to be also in the negative part of
322                     %the array and after the position of number1. The exact
323                     %position is searched:
324
325                     j=pos1; %counter for the while loop
326                     x2=0; %position of number2
327                     while j<11
328                         if tprevious(j)<0 %still looking in the negative
329                             %part of the array
330                             if number2<tprevious(j) %place of number2
331                                 %found = j in tprevious
332                                 x2=j;
333                                 j=11; % end while
```

```
                            elseif number2==tprevious(j)
                                x2=11; % there is no change needed
                                j=11; % end while
                            else
                                j=j+1;
                            end
                        else %looking in the end of negative numbers. So
                            %place of number2 = j in tprevious
                            x2=j;
                            j=11; % end while
                        end
                    end

                end

            end
        elseif tprevious(1)<0 % elevator going down

            if number1<0 % the new person also wants to go down, number1
               %and number2 have the same sign

                nextposition=position+floormargin; %Next position where
                %target1 can be placed in the first part of the array
                if number1>=nextposition %we have time to change first
                    %target floor if needed.
                    %The position of the target 1 is surely going to be in
                    %the first negative part of the array.

                    i=1; %counter for the while loop
                    x1=0; %position of number1, '0' means not found and
                    %'11' means not change needed in the array
                    pos1=0; %position of number1 even if it is already in
                    %the array
                    while i<11
                        if tprevious(i)<0 %still looking in the negative
                            %part of the array
                            if number1<tprevious(i) %place of number1 found
                                x1=i;
                                pos1=i;
                                i=11; % end while
                            elseif number1==tprevious(i)
                                x1=11; % there is no change needed
                                pos1=i;
                                i=11; % end while
                            else
                                if tprevious(i)>tprevious(i+1) %i is the
                                    %last position of the first negative
                                    %part. So number1 has to be placed in
                                    %i+1
                                    x1=i+1;
                                    pos1=i+1;
                                    i=11;
                                else
                                    i=i+1;
                                end
                            end
                        else %looking in the end of positive numbers. So
                            %place of number1 = i
                            x1=i;
                            pos1=i;
                            i=11; % end while
                        end
                    end

                    %number2 is going to be also in the first negative
                    %part of the array and after the position of
                    %number1. The exact position is searched:
```

```
401
402                     j=pos1; %counter for the while loop
403                     x2=0; %position of number2
404                     while j<11
405                         if tprevious(j)<0 %still looking in the negative
406                             %part of the array
407                             if number2<tprevious(j) %place of number2
408                                 %found = j in tprevious
409                                 x2=j;
410                                 j=11; % end while
411                             elseif number2==tprevious(j)
412                                 x2=11; % there is no change needed
413                                 j=11; % end while
414                             else
415                                 if tprevious(j)>tprevious(j+1) %n is the
416                                     %last position of the first negative
417                                     %part. So number2 has to be placed in
418                                     %j+1
419                                     x2=j+1;
420                                     j=11;
421                                 else
422                                     j=j+1;
423                                 end
424                             end
425                         else %looking in the end of positive numbers. So
426                             %place of number2 = j in tprevious
427                             x2=j;
428                             j=11; % end while
429                         end
430                     end
431
432                 else
433                     %number1 is already passed, we have to search the
434                     %position of number1 in the second part of negative
435                     %numbers:
436                     %First, the end of the first negative part is searched:
437                     i=2;%counter for the while loop, we start in position 2
438                     %because it has been already checked that the first
439                     %position is negative
440                     n1=0;%position of the first number in the array that
441                     %does not belong to the first negative part
442                     while i<11
443                         if tprevious(i)>=0 || tprevious(i)<tprevious(i-1)
444                             n1=i;
445                             i=11; %end while
446                         else
447                             i=i+1;
448                         end
449                     end
450                     %Second, the exact position of number1 and number2 are
451                     %searched:
452
453                     if tprevious(n1)==0 %position of number1 = n1 and
454                         %position of number2 = n1
455                         x1=n1;
456                         x2=n1;
457
458                     elseif tprevious(n1)<0
459
460                         i=n1; %counter for the while loop
461                         x1=0; %position of number1, '0' means not found and
462                         %'11' means not change needed in the array
463                         pos1=0; %position of number1 even if it is already
464                         %in the array
465                         while i<11
466                             if tprevious(i)<0 %still looking in the negative
467                                 %part of the array
```

```
468                             if number1<tprevious(i) %place of number1
469                                 %found = i
470                                 x1=i;
471                                 pos1=i;
472                                 i=11; % end while
473                             elseif number1==tprevious(i)
474                                 x1=11; % there is no change needed
475                                 pos1=i;
476                                 i=11; % end while
477                             else
478                                 i=i+1;
479                             end
480                         else %looking in the end of negative numbers. So
481                             %place of number1 = i
482                             x1=i;
483                             pos1=i;
484                             i=11; % end while
485                         end
486                     end
487
488                     %number2 is going to be also in this negative part
489                     %of the array and after the position of number1.
490                     %The exact position is searched:
491
492                     j=pos1; %counter for the while loop
493                     x2=0; %position of number2
494                     while j<11
495                         if tprevious(j)<0 %still looking in the negative
496                             %part of the array
497                             if number2<tprevious(j) %place of number2
498                                 %found = j in tprevious
499                                 x2=j;
500                                 j=11; % end while
501                             elseif number2==tprevious(j)
502                                 x2=11; % there is no change needed
503                                 j=11; % end while
504                             else
505                                 j=j+1;
506                             end
507                         else %looking in the end of negative numbers. So
508                             %place of number2 = j in tprevious
509                             x2=j;
510                             j=11; % end while
511                         end
512                     end
513
514
515                 else %tprevious(n1) is positive, it is needed to search
516                     %where the positive part ends
517                     l=n1+1; %counter for while
518                     p1=0; %position of the array where the new negative
519                     %or zero part starts
520                     while l<11
521                         if tprevious(l)<=0
522                             p1=l;
523                             l=11; %end while loop
524                         else
525                             l=l+1;
526                         end
527                     end
528                     if tprevious(p1)==0 %position of number1 = p1,
529                         %position of number2 = p1
530                         x1=p1;
531                         x2=p1;
532
533                     else %tprevious(p1)<0
534                         i=p1; %counter for the while loop
```

```
535                             x1=0; %position of number1, '0' means not found
536                             %and '11' means not change needed in the array
537                             pos1=0; %position of number1 even if it is
538                             %already in the array
539                             while i<11
540                                 if tprevious(i)<0 %still looking in the
541                                     %negative part of the array
542                                     if number1<tprevious(i) %place of
543                                         %number1 found = i
544                                         x1=i;
545                                         pos1=i;
546                                         i=11; % end while
547                                     elseif number1==tprevious(i)
548                                         x1=11; % there is no change needed
549                                         pos1=i;
550                                         i=11; % end while
551                                     else
552                                         i=i+1;
553                                     end
554                                 else %looking in the end of negative
555                                     %numbers. So place of number1 = i
556                                     x1=i;
557                                     pos1=i;
558                                     i=11; % end while
559                                 end
560                             end
561
562                             %number2 is going to be also in this negative
563                             %part of the array and after the position of
564                             %number1. The exact position is searched:
565
566                             j=pos1; %counter for the while loop
567                             x2=0; %position of number2
568                             while j<11
569                                 if tprevious(j)<0 %still looking in the
570                                     %negative part of the array
571                                     if number2<tprevious(j) %place of
572                                         %number2 found = j in tprevious
573                                         x2=j;
574                                         j=11; % end while
575                                     elseif number2==tprevious(j)
576                                         x2=11; % there is no change needed
577                                         j=11; % end while
578                                     else
579                                         j=j+1;
580                                     end
581                                 else %looking in the end of negative
582                                     %numbers. So place of number2 = j in
583                                     %tprevious
584                                     x2=j;
585                                     j=11; % end while
586                                 end
587                             end
588
589                         end
590
591                     end
592
593                 end
594             else %new person wants to go up, because number1 and number2
595                 %are positive
596
597                 %searching where the first negative part of the array ends:
598                 i=2; %counter for while, we start in position 2 because it
599                 %has been already checked that the 1st position is negative
600                 n1=0; %position of the array where the new negative,
601                 %positive or zero part starts
```

```
602                 while i<11
603                     if tprevious(i)>=0 || tprevious(i)<tprevious(i-1)
604                         n1=i;
605                         i=11; %end while
606                     else
607                         i=i+1;
608                     end
609                 end
610
611                 if tprevious(n1)<=0 %position of number1 = p1, position of
612                     %number2 = p1
613                     x1=n1;
614                     x2=n1;
615
616                 else
617                     %tprevious(p1) is positive. Positions of number1 and
618                     %number2 have to be found in the same positive part
619                     %from position n1
620
621                     i=n1; %counter for the while loop
622                     x1=0; %position of number1, '0' means not found and
623                     %'11' means not change needed in the array
624                     pos1=0; %position of number1 even if it is already in
625                     %the array
626                     while i<11
627                         if tprevious(i)>0 %still looking in the positive
628                             %part of the array
629                             if number1<tprevious(i) %place of number1 found
630                                 x1=i;
631                                 pos1=i;
632                                 i=11; % end while
633                             elseif number1==tprevious(i)
634                                 x1=11; % there is no change needed
635                                 pos1=i;
636                                 i=11; % end while
637                             else
638                                 i=i+1;
639                             end
640                         else %looking in the end of positive numbers. So
641                             %place of number1 = i
642                             x1=i;
643                             pos1=i;
644                             i=11; % end while
645                         end
646                     end
647
648                     %number2 is going to be also in the positive part of
649                     %the array and after the position of number1. The exact
650                     %position is searched:
651
652                     j=pos1; %counter for the while loop
653                     x2=0; %position of number2
654                     while j<11
655                         if tprevious(j)>0 %still looking in the positive
656                             %part of the array
657                             if number2<tprevious(j) %place of number2
658                                 %found = j in tprevious
659                                 x2=j;
660                                 j=11; % end while
661                             elseif number2==tprevious(j)
662                                 x2=11; % there is no change needed
663                                 j=11; % end while
664                             else
665                                 j=j+1;
666                             end
667                         else %looking in the end of positive numbers. So
668                             %place of number2 = j in tprevious
```

```
                                x2=j;
                                j=11; % end while
                            end
                        end

                    end

                end

            else %elevator is stopped
                x1=1;
                x2=1;

            end

        else %there is not enough space in the array to add new floors
            x1=11;
            x2=11;

        end

else %there are not new requested floors
    x1=11;
    x2=11;

end
%% Update t:

if x1<11 %it is needed to add number1 in the array
    if x1>1
        t(1:x1-1)=tprevious(1:x1-1);
    else
    end
    t(x1)=number1;
    if x2<11 %it is needed to add number2 in the array
        t(x1+1:x2)=tprevious(x1:x2-1);
        t(x2+1)=number2;
        t(x2+2:10)=tprevious(x2:8);

    else %number2 is already in the array
        t(x1+1:10)=tprevious(x1:9);

    end
else %number1 is already in the array
    if x2<11 %it is needed to add number2 in the array
        if x2>1
            t(1:x2-1)=tprevious(1:x2-1);
        else
        end
        t(x2)=number2;
        t(x2+1:10)=tprevious(x2:9);

    else %number2 is already in the array
        t(1:10)=tprevious(1:10);

    end

end

%% If delete_target1 is active, the first number in the array has to be
%deleted and the rest of numbers have to be moved one position forward:
if delete_target1~=0
    t(1:9)=t(2:10);
    t(10)=0;
else
    t(1:10)=t(1:10);
end
```

L Delete another floor MATLAB Function Code

```
function u=deleteanotherfloor (pos, targetone)

 if abs(pos)==abs(targetone)
     u=1;
 else
     u=0;
 end
```

M Counting Steps MATLAB Function Code

```
function totalsteps = fcn(totalstepsbefore, direction)

if direction >0
    totalsteps= totalstepsbefore + 1;
elseif direction ==0
    totalsteps= totalstepsbefore - 1;
else
    totalsteps= totalstepsbefore;
end
```

N Speech Recognition Code

```
function out=mfcc_delta()
%Records for 3 seconds and extracts feature vectors (MFCC) from the analog
%speech signal. The matrix is then compared between a large library
%containing pre-recorded speech signals (i.e mfcc matrices). The classified
%number/floor is the variable 'out'. This program can easily be modified
%to generate mfccs from a stored wav file, which was done for generating
%the library.

%_____________________________________________________________
% Main program starts here ~
%_____________________________________________________________
clc
clear all
tic
%------------------
% Initial values
%------------------

 Fs=8000;
 bits=24;
 t=3;
 mono=1;

%------------------
%Recording
%------------------

    r = audiorecorder(Fs, bits, mono); %Cretes a recording object

    recordblocking(r, t); %Records for t seconds

    signal = getaudiodata(r, 'double')* 2^15; %recorded voice

%-----------------
%Filters
%-----------------
     pre_emp = filter([1 -.97], 1, signal); %pre-emphasis filter

     signal = filtering(pre_emp,Fs); %Bandpass-filter using DSP-toolbox


%% VAD
%Recive a signal, look for parts containing high energy and stores it in
%a new, shorter array (which is the output).
%-----------------
% Initial value
%-----------------
length_package_time=20;

%-----------------


 Num_packages=round((t*10.^3)/length_package_time); %for t=3 sec, => #150

 length_package=length(signal)/Num_packages; %160 sampels => 20ms

%----------------------
% Calculation of noise
%----------------------

%The noise is regarded as the first few ms of the signal. We start sampling
%at 200 to avoid interfering with the speaker sound.

    i=0;
```

```
66      for e=1:1800
67          i=abs(signal(e+200))+i;
68      end
69      noise=(i/1800)*4;
70      if(noise>max(signal)/3)
71          noise=noise/3;
72      end
73
74     %If the noise is high, we divide it with a factor
75     %of 3, to ensure that the static is removed.
76
77 %---------------------------------------
78 % Voice activity detection algorithm
79 %---------------------------------------
80
81      a=zeros(length_package,1); %vector that stores one package
82
83      temp=zeros(length(signal),1);%vector that stores all accepted packages
84
85
86          for i=1:Num_packages
87             for n=1:length_package
88                 a(n) = (signal(n+(i-1)*length_package)); %stores the data
89                                                          %from one package
90
91             end
92
93             weight=(sum(abs(a))/length(a))-noise; %stores its weight
94
95             if weight>0 %if the energy>background noise => save
96
97                 for p=1:length_package
98
99                 temp(p+(i-1)*length_package)=signal(p+(i-1)*length_package);
100                end
101
102            end
103
104
105
106         end
107
108
109
110     k=find(temp);   %determine the position of all non-zero elements
111     u=zeros(length(k),1);
112
113         for tot=1:length(k)
114             x=k(tot);
115             u(tot)=temp(x); %store all non-zero elements in a small array
116         end
117
118    signal2=u(1:end);
119
120 %% Windowing
121
122 %Recives a signal, segments it into short overlaped packages and applies
123 %a window to each package. Then puts each package into cells. The output
124 %is a cell array.
125
126 %---------------
127 %Initial values
128 %---------------
129 duration=25; %ms
130 overlap=1.5; %50%
131 %---------------
132 %---------------
```

```
133
134 %Makes sure that the length of the signal divided with total number of
135 %packges is an integer
136 while(ceil(length(signal2)/(round(((length(signal2)/Fs)*10.^3) ...
137 /duration)))~=floor(length(signal2)/(round(((length(signal2)/Fs)*10.^3) ...
138 /duration))))
139     duration=duration-1
140         if duration < 1
141           error('fel i duration')
142         end
143 end
144
145 Num_packages=round(((length(signal2)/Fs)*10.^3)/duration);
146 %Note that this will increase after the overlaping process!
147
148 length_package=length(signal2)/Num_packages; %This variable is however
149                                              %fixed
150
151 %--------------------------------
152 %Overlapping process
153 %--------------------------------
154
155 extension = round(length_package*overlap-length_package); %number of eleme-
156                                                           %ments used as overlap
157
158 x=length(signal2)/length_package; %Gives us how many rows we have
159                                   %(filled and almost filled)
160
161 g=ceil(x);  %total rows
162 y=floor(x); %filled rows
163
164 right=length_package-extension;
165 J=right;
166 i=1;
167 while (length(signal2)-right)>length_package
168     right=right+J;
169     i=i+1;
170 end
171
172 length_new=i*length_package+(length(signal2)-right); %The length of our
173                                                      %padded signal
174
175 dd=ceil(length_new/length_package);
176
177 Z=zeros(dd,length_package); %empty matrix with best size for this vector
178
179 Z(1,1:length_package)=signal2(1:length_package); %The first row can be
180                                                  %inserted as it is, as it
181                                                  %is not padded.
182
183 S=size(Z);
184
185 Num_packages=S(1);
186
187 F=zeros(1,extension); %Temporary vector for storing the padded elements
188                       %for each row
189
190 p=0; %Initialization of counter
191
192
193
194 for i=2:S(1) %algorithm for inserting the elements in correct order
195
196     for u=extension:-1:1
197
198     F(extension-u+1)= signal2((length_package-u+1)+((i-2)* ...
199         (length_package-extension)));
```

```
200     end
201     Z(i,1:extension)=F; %padded elements
202
203     t=0;
204
205     for r=extension+1:length_package %original elements (shifted)
206         t=t+1;
207         Z(i,r)= signal2(length_package+t+(i-2)*(length_package-extension));
208         p=p+1;
209
210
211         if(p==(length(signal2)-length_package)) %safety procedure
212             break
213         end
214     end
215
216
217 end
218
219
220 %------------------------------
221 % Windowing process
222 %------------------------------
223
224 tot = hamming(160);
225
226 a=zeros(length_package,1); %empty vector that temporary stores each package
227 C=zeros(length_package,Num_packages);
228
229 for i=1:Num_packages
230     for k=1:length_package
231
232           a(k)=Z(i,k);
233
234     end
235
236     window=a.*tot;
237
238     C(:,i)=window;
239 end
240
241 %--------------------
242 %Power spectrum
243 %--------------------
244 N=512;
245 [one,two]=size(C);
246
247 temp=zeros(N,two);
248     for i=1:two
249
250         y=fft((C(:,i)),N); %fft of one cell
251         temp(:,i)= real(y.*conj(y)/N); %the power
252
253     end
254
255
256 %%
257 %----------------
258 % Filterbank
259 %----------------
260 %Creates the filterbank triangular windows
261
262 Num_filters=26;
263
264 K=N/2;
265
266
```

```
%All the following variables are in [Hz]
    Range=[300 3400];
    Min = 0;
    Low = Range(1);
    High = Range(2);
    Max = 0.5*Fs;
    F = linspace(Min,Max,K);

    % Determine the start and end positions for each filter in the
    % frequency spectrum
    positions = freq( mel(Low)+[0:Num_filters+1]*((mel(High)-mel(Low))/ ...
        (Num_filters+1)) );


    H = zeros( Num_filters, K ); %Matrix to store the filter data in
    UH= zeros( Num_filters, K );
    for m = 1:Num_filters   %Creates each filter seperatly

        P=zeros(1,length(F));   %vector that will store all the freq
                                %positions between two elements in 'positions'


       for p=1:length(F)        %In this case we look between each filters
                                %start and middle position, i.e the raising
                                %side of the triang.
            if F(p)>=positions(m) && F(p)<=positions(m+1)
                P(p)=1;
            end
        end
        pp=find(P); %Finds which elements are none-zero

        H(m,pp) = (F(pp)-positions(m)) / ((positions(m+2)-positions(m)) ...
            *(positions(m+1)-positions(m))); %finds the corresponding
                                             %y-value for each P.


        P=zeros(1,length(F));
        for p=1:length(F)   %In this case we look between each filters
                            %middle and end position, i.e the decreaing
                            %side of the triang.
            if F(p)>=positions(m+1) && F(p)<=positions(m+2)
                P(p)=1;
            end
        end
        pp=find(P);


      H(m,pp) = (positions(m+2)-F(pp)) / ((positions(m+2)-positions(m))...
          *(positions(m+2)-positions(m+1)));
    end
     H = H./repmat(max(H,[],2),1,K);  %normalize to 1

     vector=H;


%% Calculate Filterbank Energies from previous data

Output=vector * temp(1:K,:); %Where temp is the powerspec results

% Lifting:
    Num_mfcc=13; %Number of MFCCs regarded
    LP=22;


    lifter = expander( Num_mfcc, LP ); %see function below
    LOG = log(Output); %Logarithmic
    DCT=ICT( Num_mfcc, Num_filters ); %See function below
```

```
334     DCT=DCT*LOG;
335     DCT_lift=diag(lifter)*DCT;
336     DCT_lift(1,:)=[]; % Removes first row (represents whole energy)
337
338     Final_matrix=DCT_lift;
339
340 %% deltas
341
342   tot=9; %Number of mfccs taken into consideration
343   [Q,E] = size(Final_matrix);
344   floor = floor(tot/2);
345   tot = 2*floor + 1;
346   win = floor:-1:-floor;
347
348   xx = [repmat(Final_matrix(:,1),1,floor),Final_matrix,repmat...
349       (Final_matrix(:,end),1,floor)];
350   % The mfcc arrays are padded
351
352   filt = filter(win, 1, xx, [], 2);
353
354   Delta = filt(:,2*floor + [1:E]);
355
356   %Courtesy of Daniel P. W. Ellis
357   % http://labrosa.ee.columbia.edu/matlab/rastamat/deltas.m
358
359 %% Turn 2 %same as above but with other tot.
360
361   tot=5;
362   floor = floor(tot/2);
363   tot = 2*floor + 1;
364   win = floor:-1:-floor;
365   xx = [repmat(Final_matrix(:,1),1,floor),Final_matrix,repmat...
366  (Final_matrix(:,end),1,floor)];
367   filt = filter(win, 1, xx, [], 2);
368   Delta_store = filt(:,2*floor + [1:E]);
369   %% acceleration %calculated same as above but with x = Delta_store.
370   tot=5;
371   [Q,E] = size(Delta_store);
372   floor = floor(tot/2);
373   tot = 2*floor + 1;
374   win = floor:-1:-floor;
375   xx = [repmat(Delta_store(:,1),1,floor),Delta_store,repmat...
376       (Delta_store(:,end),1,floor)];
377   filt = filter(win, 1, xx, [], 2);
378   Delta2 = filt(:,2*floor + [1:E]);
379
380  %% Final result
381
382     Final_matrix=[Final_matrix;Delta;Delta2]; %these are the extracted
383                                               %features
384     r=sorting2(Final_matrix);
385    % str=['Desired floor is ', num2str(r)]; %Uncomment when running here
386    %  disp(str)
387
388     out=r; %The desired floor
389
390 end
391
392
393 function Number=sorting2(vector)
394
395 BIB=12; %number of recordings for each number
396
397 temp=zeros(BIB,10);
398
399 for w=1:10
400     for t=1:BIB
```

```
401   n=t+9;
402    V=struct2array(load(sprintf('%d%d.mat',w,n))); %loads the library
403
404
405    Minimun_d_map = map_d(abs(V),abs(vector));
406    [p,q,C] = dtw_calc(1-Minimun_d_map);
407
408    %Functions map_d and dtw_calc courtesy of Daniel P. W. Ellis ...
409    %(Dynamic Time Warp in Matlab, Copyright (C) 2003).
410
411
412
413     A=C(size(C,1),size(C,2)); %The corner value of the line represents
414                               %the total travel cost.
415
416     temp(t,w)=A; %Stores each comparinson
417
418     end
419 end
420
421 [a,b]=find(temp==min(temp(:))); %finds the minimum value and prints its
422                                 %position (i.e floor number)
423
424
425 Number=b;
426
427 end
428
429
430
431  function out_mel = mel(Hz)
432  out_mel=(1000/log10(2)*log10(1+Hz/1000));
433  end
434
435       function out_freq = freq(Mel)
436     out_freq=(1000*(2.^(Mel/1000)-1));
437       end
438
439       function out = expander( Num_mfcc, LP)
440       out = ( 1+0.5*LP*sin(pi*[0:Num_mfcc-1]/LP) );
441       end
442
443       function out = ICT(Num_mfcc, Num_filters)
444       out= ( sqrt(2.0/Num_filters) * ...
445           cos( repmat([0:Num_mfcc-1].',1,Num_filters).* ...
446           repmat(pi*([1:Num_filters]-0.5)/Num_filters,Num_mfcc,1) ) );
447       end
```