Group 5

Members:
Olga Sharma
Upeksha Sanjeewani
Lihini Hewage

Garage Door Opener

Project Report

Second-year IOT Project
School of ICT
Metropolia University of Applied Sciences
17 March 2025

Contents

1 Introduction

The Garage Door Opener Project is implemented during the second-year second semester at Metropolia University of Applied Sciences. The aim of the project is to develop an automated system for controlling and monitoring a garage door remotely and locally. The system ensures secure and efficient operation using a stepper motor, rotary encoder, limit switches, and MQTT communication for remote access.

1.1 Goals of the project

The primary objective of this project are:

- To design a motor-driven garage door opener that can be controlled locally via physical buttons and remotely through MQTT messages.
- To incorporate a rotary encoder to detect door movement and direction.
- To implement limit switches to prevent damage by detecting the door's end positions.
- To ensure the system communicates status updates via MQTT for remote monitoring.

1.2 Added values

This system enhances security and convenience for users by offering:

- Remote Monitoring & Control: Users can operate the door from anywhere using MQTT.
- Safety Measures: Prevents damage by stopping the motor when a limit switch is reached or if an error is detected.
- Error Handling: Automatically detects and reports faults, such as a stuck door, to enhance reliability.

2 Theoretical Background

A garage door opener is a motorized system designed to open and close a door using automation. In this project, a stepper motor is used for precise control, a rotary encoder tracks movement direction, and limit switches detect the end positions.

The system operates in two modes:

1. Local Control: Buttons connected to the microcontroller allow users to open, close, or stop the door.

2. Remote Control: The system subscribes to an MQTT topic to receive commands and publishes status updates.

To ensure safety, the system stops the motor when the door reaches its limits and reports errors via MQTT. The state of the door (open, closed, or in-between) is preserved even after power cycles.

3 Methods and Material

The following chapter is dedicated to the explanation of the materials utilized for the project. Firstly, the topic is introduced from a hardware point of view in the first subchapter. Secondly, the software aspect is discussed in the second subchapter along with the applied methods.

3.1 Hardware Requirements

The garage door opener consists of the following components:

- Stepper Motor: Controls door movement precisely.
- Rotary Encoder: Detects movement and direction of rotation.
- Limit Switches: Detect when the door reaches the top or bottom position.
- Microcontroller (RP2040-based board): Controls the system logic and MQTT communication.
- LEDs: Indicate door status and error conditions.
- Buttons: Allow manual operation of the door.

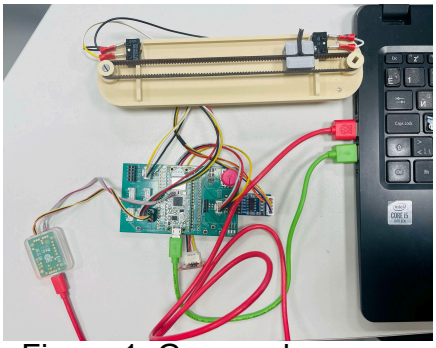


Figure 1: Garage door opener

3.2 Software Requirements and Methods

The software is developed using C++ OOP language within the CLion IDE.
It consists of:

- Motor Control Algorithms: Handles precise movement of the stepper motor.

- MQTT Communication: Implements message-based control and reporting.

- Interrupt Handling: Uses GPIO interrupts to process rotary encoder and button signals.

- State Management: Stores calibration and door position in non-volatile memory (EEPROM).

4 Implementation

4.1 Operating principles of the software

The system operates as follows:

- Calibration: Pressing SW0 and SW2 together runs the door up and down to determine step counts using the limit switches and rotary encoder.

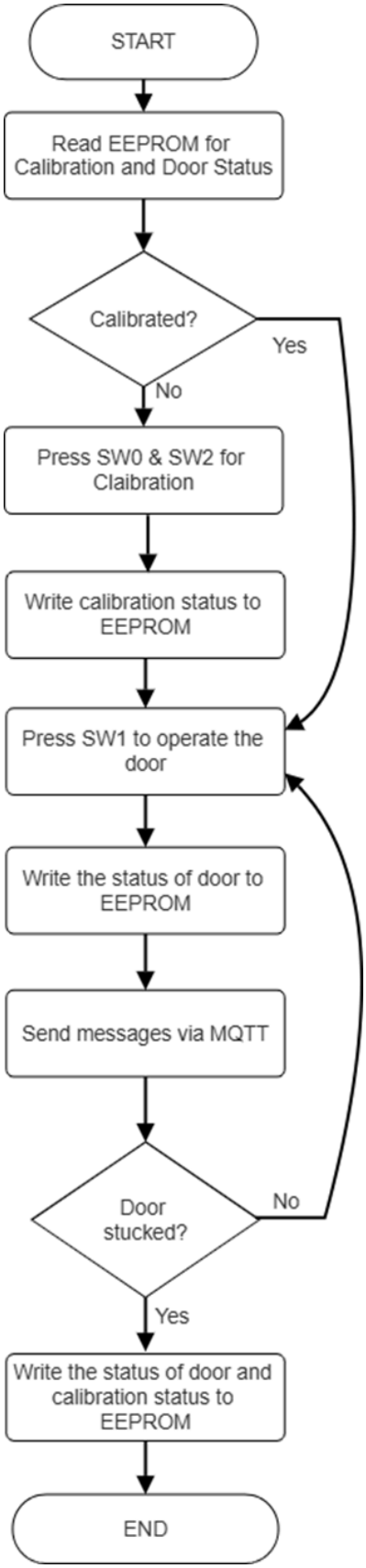
- Door Movement: The motor moves the door up or down based on button presses only.

- State Management: The door's and calibration state are stored in memory for persistence.

- Error Handling: If the door gets stuck, the motor stops, an error state is reported, and calibration is reset.

- Status Reporting: The system publishes the door's calibration states and error conditions to the MQTT broker.

4.2 Flow Chart



Note:

At any time of the operation process, if the system was rebooted it will start from the beginning.

4.3 Software algorithms

Firstly, to execute the programme successfully, the required libraries are defined. Below table 1 there is an explanation of each library.

Library	Description
hardware/gpio.h	Provides functions for GPIO pin initialization, setting directions, and operations like reading and writing to pins.
LEDs.h	A custom header file that defines the LEDs class, responsible for controlling the LEDs connected to the GPIO pins.
pico/stdlib.h	Contains standard functions for Raspberry Pi Pico such as stdio_init_all(), sleep_ms(), etc.
hardware/cyw43_arch.h	Provides Wi-Fi functionality for the Raspberry Pi Pico W, enabling Wi-Fi connection.
MQTTClient.h	Provides the necessary classes and functions for implementing MQTT communication.
IPStack.h	Defines the stack used for network communication and IP configuration for MQTT.
Countdown.h	A utility class used by the MQTT client to manage the connection timeout and message delay.
pico/util/queue.h	Provides a queue implementation used for managing events (e.g., rotary encoder events).
stdio.h	Standard input/output library used for printing messages to the console.
pico/time.h	Provides functions for time management and delay in the code, such as sleep_ms().
eeprom.h	Custom header for handling EEPROM read and write operations for system state storage.
RotEncoder.h	Custom header that defines the RotaryEncoder class, which interfaces with the rotary encoder to track rotations.
Buttons.h	Custom header for handling button interactions, controlling the garage door with button presses.
Door.h	Custom header for managing the state of the garage door, e.g., open, close, and calibration.

MqttManager.h	Custom header for the MQTT manager that handles MQTT connections, subscriptions, and publishing.

Table 1: The libraries of the programme

In addition, the programme contains overall 28 functions and the main programme. The former is described in detail in Chapter 4.3.1 and the latter one in Chapter 4.3.2.

4.3.1 Functions

As mentioned prior, the programme consists of overall 29 functions, 9 classes, which are listed in Table 2 and Table 3.

Function Name	Description
LEDs::LEDs()	Constructor that initializes a GPIO pin as input or output, optionally setting pull-up/down and invert.
LEDs::read()	Reads the current state of the GPIO pin (input) and returns a boolean value.
LEDs::operator()	Overloaded function call operator that calls read() and returns the pin state.
LEDs::write()	Writes a boolean value to the GPIO pin (output).
LEDs::operator()(bool)	Overloaded function call operator that calls write() to set the pin state.
LimitSwitches::SetSwitches()	Initializes the limit switches by setting the GPIO pins as input and enabling pull-up resistors.
LimitSwitches::LowTriggered()	Checks if the left limit switch (LOW) is triggered.
LimitSwitches::UpperTriggered()	Checks if the right limit switch (UPPER) is triggered.
Motor::SetupMotor()	Initializes GPIO pins for controlling the motor and sets their direction to output.
Motor::RotateMotor()	Rotates the motor in the current direction (clockwise or counterclockwise) using the stepper motor sequence.
Motor::StopMotor()	Stops the motor and sets all control pins to low (turning off the motor).
Motor::SetMotorDirection()	Sets the direction of the motor (1 for clockwise, -1 for counterclockwise).
Motor::MoveMotor()	Moves the motor a specified number of steps in the current direction.
Motor::MotorCalib()	Calibrates the motor by moving it to the upper and lower limit switches and recording the step counts.
Motor::MoveDoor()	Moves the door based on the current direction, using limit switches and encoder to check and update the door state.
Motor::GetDirection()	Returns the current direction of the motor (clockwise or counterclockwise).
Motor::SetStopFlag()	Sets or toggles the stop flag for stopping the motor.
Motor::GetStopFlag()	Returns the current status of the stop flag (whether the motor should stop).
MqttManager::connect()	Connects to the MQTT broker and subscribes to the command topic.

MqttManager::publish()	Publishes a message to the status topic on the MQTT broker.
MqttManager::checkMessages()	Checks for new messages from the MQTT broker and processes them.
MqttManager::messageArrived()	Callback function to handle received MQTT messages.
RotaryEncoder::rot_interrupt()	Interrupt handler for the rotary encoder, updates step count and adds the direction to the event queue.
RotaryEncoder::GetStepCount()	Returns the current count of steps (rotations) for the rotary encoder.
RotaryEncoder::ResetStepCount()	Resets the step count of the rotary encoder to zero.
Buttons::SetupButtons()	Initializes the buttons and assigns interrupt handlers.
Buttons::HandleCalibration()	Handles the calibration process for the system when a button is pressed.
Buttons::HandleButtonPress()	Handles the action when a button is pressed, such as opening/closing the door.
main()	The entry point of the program. Initializes objects, connects to Wi-Fi, and controls the motor, buttons, and MQTT communication.

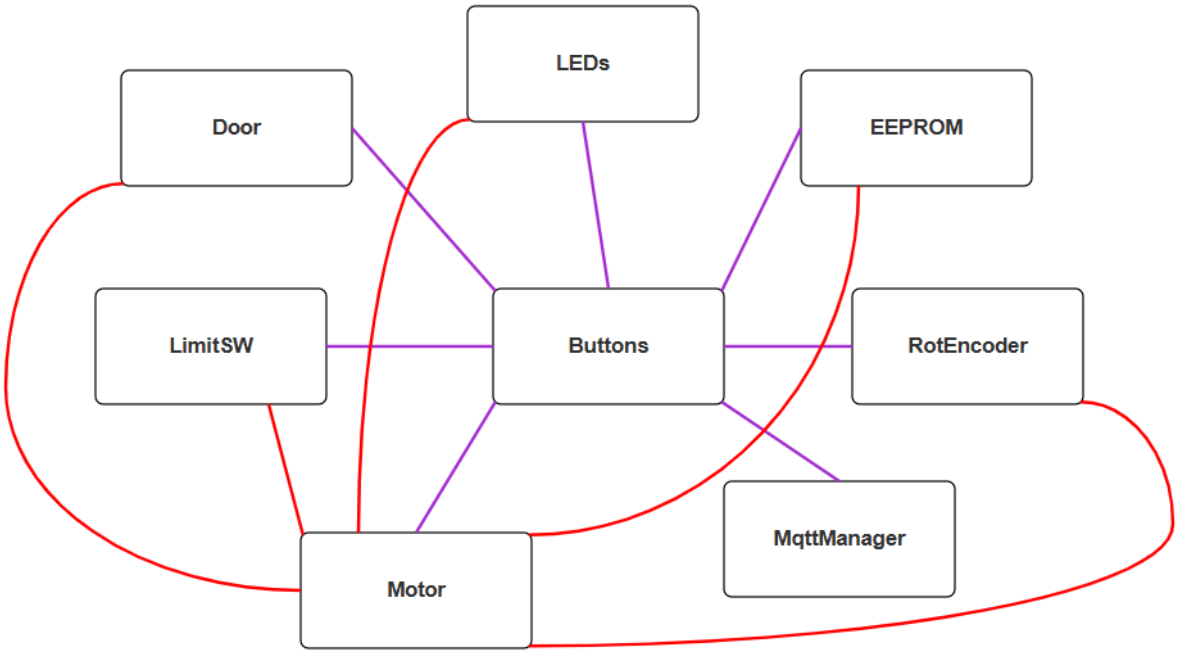
Table 2: The functions of the programme

Classes

Class Name	Description
LEDs	Manages the state of an LED connected to a GPIO pin, allowing reading and writing to the pin.
LimitSw	Manages the state of two limit switches used to detect the position of the garage door.
Motor	Controls the motor driving the garage door, including rotation, direction, and movement based on limit switches and an encoder.
MqttManager	Manages the connection and communication with an MQTT broker, including subscribing to topics and publishing messages.
RotEncoder	Handles the rotary encoder that tracks rotations and provides step counts based on the encoder's movement.
Buttons	Handles button presses for controlling the garage door, including calibration and door movement control.
Door	Represents the state of the garage door, tracking its open/closed status and performing actions like opening and closing.
EEPROM	Custom class for handling EEPROM read/write operations, used for storing system states like calibration and door position.

Table 3: The classes of the programme

4.3.2 Class Diagram



4.3.3 Main Programme

The main() function serves as the central control for the garage door system. It initializes hardware components (motor, buttons, LEDs, limit switches) and sets the system's initial state by reading data from EEPROM. The function first checks the previous state stored in EEPROM, allowing the system to recover from interruptions. It handles calibration if necessary or resumes operation where it left off. The main loop continuously monitors button presses, manages door movement, and communicates with the MQTT server for status updates. This design ensures reliability, system recovery, and persistence of door states across reboots.