Technical documentation CirquitCraft

By Abe Bruinsma, Bart Vink, Sybren Zijlstra, Tom Stuurman and Werner Meihuizen

Contents

Introduction

This report serves as an account of the technical aspects involved in the development of our project. It aims to provide a clear understanding of the technologies used, and the thoughts behind the decisions made during the course of the project.

The objective of this report is to document the technical details for reference purposes but also to ensure transparency and accountability in our work process.

In the following sections, we will discuss everything from the initial design phase to the final implementation. We hope that this report provides a thorough and insightful look into our technical endeavours.

Technical Specifications of Raspberry PI:

This chapter outlines the technical details of the Raspberry PI, specifically the Raspberry PI 4 Model B, which is utilized in this project. The Raspberry PI 4 Model B is equipped with a Broadcom BMC2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz processor. It boasts 4GBs of LDPDDR4-3200 SDRAM and a 2.4GHZ/5.0GHZ IEEE 802.11ac wireless, Bluetooth 5.0, BLE Gigabit Ethernet chip. The device features multiple HDMI and USB ports, along with an SD card slot for operating system installation and data storage.

Installed software

The Raspberry PI has Node RED installed for data visualization and SQLite for data storage. The operating system in use is the headless version of latest version of Raspberry PI OS.

OS

Choosing the most recent version of Raspberry Pi OS has several advantages. Firstly, the latest version contains the most recent security updates and patches, which are essential for protecting your system against potential threats. Secondly, it offers the latest features and improvements implemented by the developers, which can lead to better performance and more functionality. Running Raspberry Pi OS in headless mode means that you operate the operating system via the command line interface (CLI) instead of through a graphical user interface (GUI). This has several advantages, especially for server applications or for systems that are managed remotely. A headless system consumes fewer system resources because it does not need to support a graphical interface. This can lead to better performance, especially on systems with limited hardware resources, such as a Raspberry Pi. Moreover, working with a CLI can be more efficient for certain tasks and can be easier to automate and script than working with a GUI. It can also be operated remotely via SSH, which is convenient for managing servers or other systems that are not directly accessible. In summary, choosing the most recent version of Raspberry Pi OS and running it in headless mode can lead to a safer, more efficient, and more flexible system, depending on your specific needs and applications.

ESP32 specifications

The ESP32 boards were the centrepiece of the project, the ESP32 board collected all the data such as humidity, temperature and air pressure. All ESP32 boards were connected in a Painless mesh network, this allowed the whole system to operate without an existing WiFi network.

Painless Mesh network

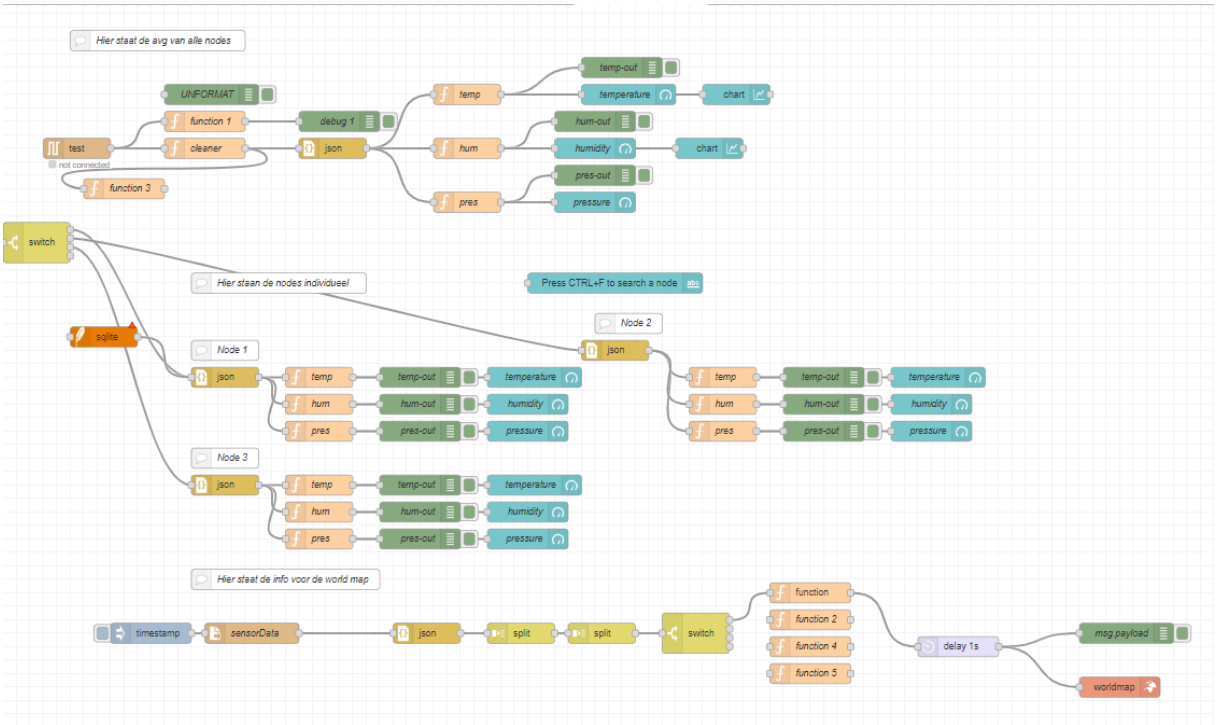
A Painless Mesh network is a form of an ad-hoc network. This means that no routers or central controllers are needed. A collection of two or more nodes will autonomously create a fully functioning mesh network.

The Painless Mesh network uses the ESP-MESH network protocol. This allows multiple devices (nodes) to communicate with each other within a single wireless local network. This protocol is supported on the ESP32 boards.

Unlike a traditional Wi-Fi network, where each node must communicate via a central node (usually the router), the nodes in an ESP-MESH network do not need to be connected to a central node. Nodes are responsible for passing on each other's transmissions, allowing multiple devices to spread over a large physical area. The nodes can organize themselves and dynamically communicate with each other to ensure that the packet reaches its final destination.

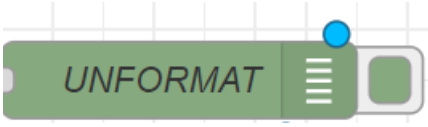
If a node is removed from the network, the network can reorganize itself to ensure that the packets reach their destination. The PainlessMesh library makes it easy to create a mesh network with the ESP32 boards.

Node red



this part of the documentation will describe what each button of the following image does and how it is configured

This connects us to the ESP-32 in our setup



This is a debug node

This is our first function of the board.

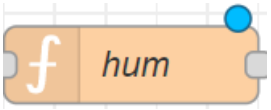
This piece of code is used in a Node-RED Function node to check the content of msg.payload and perform actions based on that content. If msg.payload consists only of spaces or a single dot, it is replaced with “connecting”. If msg.payload contains the substring ‘startHere: Received from’, the message is passed through without further processing. In all other cases, msg is passed unchanged to the next node in the flow.

The JSON file contains the configuration of a Node-RED flow, including nodes such as UI elements, serial communication, data processing with JavaScript, and debug output. It describes the setup of the flow, including the UI layout, data processing, and debug settings, allowing the flow to be loaded and executed within Node-RED. The JSON file also checks if the incoming code is indeed a JSON code.

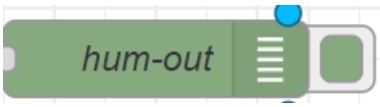
The "temp" function in the given Node-RED flowchart is a Function node responsible for processing temperature data. The function retrieves the temperature directly from the messages and rounds it to a precision of 0.5. Subsequently, a new message is created containing only the rounded temperature and is passed on to other nodes in the flow, including a UI Gauge node named "temperature" for display.

The "temp-out" block is a Debug node in Node-RED responsible for displaying temperature data. When activated, it sends messages containing temperature values to the debug output of Node-RED, which is useful for monitoring processed temperature data during development and testing.

The "temperature" block is a UI Gauge node in Node-RED responsible for displaying temperature measurements in the user interface. It utilizes a gauge-style visualization to visually represent temperature values, with a label in Kelvin and a colour scheme set based on temperature ranges.



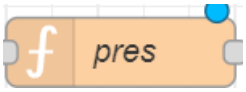
The "hum" function in the given Node-RED flowchart is a Function node responsible for processing humidity data. This function retrieves humidity directly from the messages and rounds it to a precision of 0.5. Subsequently, a new message is created containing only the rounded humidity and is passed on to other nodes in the flow, including a UI Gauge node named "humidity" for display.



The "hum-out" block is a Debug node in Node-RED responsible for displaying humidity data. When activated, it sends messages with humidity values to the debug output of Node-RED, which is useful for monitoring processed humidity data during development and testing.

The "humidity" block is a UI Gauge node in Node-RED responsible for displaying humidity measurements in the user interface. This node uses a gauge-style visualization to visually represent humidity values, with a label in percentage and a colour scheme set based on humidity ranges.

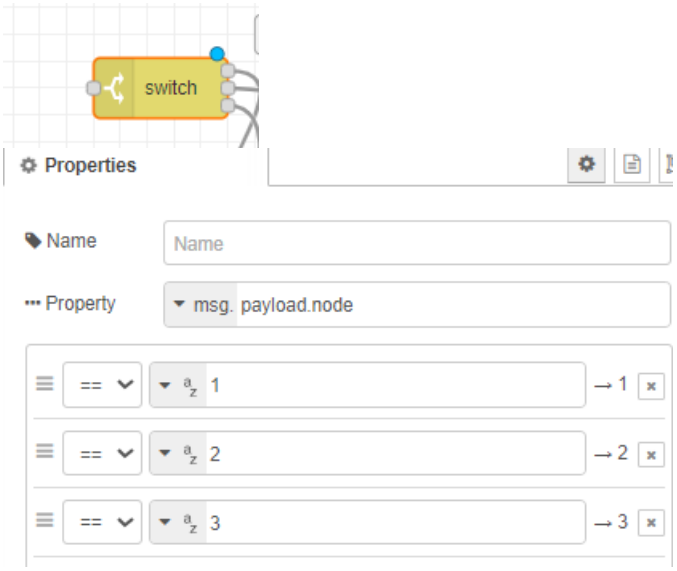
The "chart" block is a UI Chart node in Node-RED responsible for displaying data in a chart in the user interface. It uses a bar chart-style visualization to display data such as temperature, humidity, or other measurements, with customizable settings such as chart labels, interpolation method, and colour scheme.



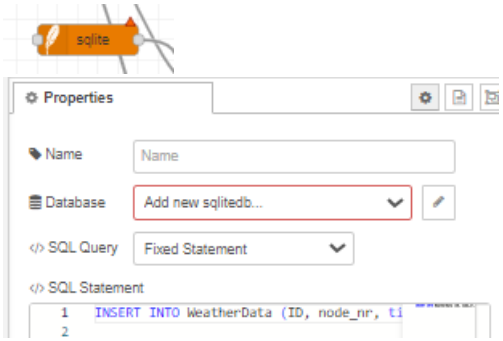
The "pres" function in the given Node-RED flowchart is a JavaScript Function node responsible for processing pressure data. This function extracts the pressure value from the messages and rounds it to a precision of 0.5. Subsequently, a new message is created containing only the rounded pressure value, which is passed on to other nodes in the flow, such as a UI Gauge node named "pressure" for visualization.

The "pres-out" block is a Debug node in Node-RED responsible for displaying pressure data. When activated, it sends messages with pressure values to the debug output of Node-RED, which is useful for monitoring processed pressure data during development and testing.

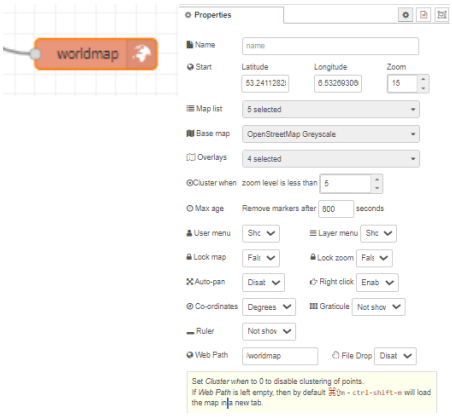
The "pressure" block is a UI Gauge node in Node-RED responsible for displaying pressure measurements in the user interface. It uses a gauge-style visualization to visually represent pressure values, with a label in bar and a colour scheme set based on pressure ranges.



The switch node executes one of several tasks, depending on the rules you provide.



This sqlite node allow us to give commands to a database so we can display that information on our UI



This node allows us to display a map, it is possible to add points to the map that contain sensor location and measured information

Node red dashboard

Node-RED Dashboard Documentation

Node-RED-dashboard is a module that allows you to quickly and easily create interactive dashboards for visualizing data and controlling devices and systems.

Step 1: Installing Node-RED-dashboard

1. Ensure that you have Node-RED installed on your system or device.
2. Open a terminal or command prompt and execute the following command to install Node-RED-dashboard: `npm install node-red- dashboard`
3. Restart Node-RED to apply the changes.

Step 2: Adding Dashboard Nodes

1. Open Node-RED in your browser and go to the Node-RED editor.
2. Drag a UI_tab node to the workspace. This will serve as a tab for your dashboard.
3. Drag a UI_group node to the workspace. This will serve as a group for your dashboard widgets.
4. Drag various UI_ nodes (such as Misbuttons, UI gauge, UI_chart, etc.) to the workspace to add the desired widgets to your dashboard.

Step 3: Configuring Dashboard Nodes

1. Double-click on a node to open the properties window.
2. Configure the settings as desired, such as the title, label, colours, etc.
3. Connect the nodes by drawing wires between the outputs and inputs of the nodes.

Step 4: Designing the Dashboard

1. Drag the nodes around on the workspace to design the layout of your dashboard.
2. Give each element a logical place on the dashboard.
3. Adjust the size and properties of the nodes for optimal display.

Step 5: Saving and Deploying

1. Once you are satisfied with your dashboard, click on the Deploy button in the top right corner of the Node-RED editor.
2. Your dashboard is now live and available via the specified URL. Go to the URL to view and use your dashboard.

Advanced Options

Adding Interactivity: Use the built-in capabilities of Node-RED-dashboard to add interactive features to your widgets, such as controlling devices, executing actions, etc. • Customizing Styles: Customize the styles of your dashboard by applying CSS styles to the widgets and layout elements. • Integrating Data Sources: Integrate external data sources such as databases, APIs, etc., to display real-time data on your dashboard.

Conclusion

With Node-RED-dashboard, you can quickly and easily create powerful dashboards for visualizing and controlling data and systems. Experiment with the different widgets and features to create a dashboard that meets your specific needs.