Ventilation controller project specification

Introduction

The goal of the project is to implement a ventilation controller system that can be controller both from local user interface – buttons and display and by MQTT messages that are sent from a web UI. The controller can be operated in two modes: automatic and manual. In manual mode user sets the speed of the ventilation fan (0 – 100%) from local UI or the remote UI. In automatic mode the user sets target pressure in the ventilation duct and the controller adjusts the fan speed as needed to keep the pressure stable.

Hardware platform

Fan speed is controlled by Produal MIO 12-V, which is a Modbus controlled IO-device that has its 0 – 10V output connected to the ventilation fan control input. MIO output voltage determines the speed of the fan: 0V is off (0%) and 10V is maximum speed (100%). Ventilation fan has a pulse output that gives pulses when the fan is turning, and it is connected digital pulse counter input of MIO. The counter is self-clearing, which means that the counter is reset to zero after the count is read through Modbus. When read periodically if the value is zero after two reads it means that the fan is stopped.

Produal MIO 12-V

- Modbus address: 1
- 0-10V output controls the ventilation fan speed
- Digital input counts rotation pulses from the fan. Counter value is used so check if the fan is running. Counter is self-clearing

The system is equipped with three Modbus sensors. All modbus devices operate at 9600 bps speed. See data sheets of the sensors in the course workspace for more information.

- Vaisala GMP252 CO2 probe
  - Modbus address: 240
- Vaisala HMP60 relative humidity and temperature sensor
  - Modbus address: 241
- Sensirion SDP610 – 120Pa pressure sensor
  - I^2C bus
    - Address: 0x40
    - SCL → Port 0, Pin 22
    - SDA → Port 0, Pin 23
  - Differential pressure sensor which connected to measure pressure difference between room and the ventilation duct
  - Sensor returns a signed 16 bit value
  - The value must be converted to physical value (pascals)
    - Scale factor (see data sheet chapter 2)
    - Altitude correction (see data sheet chapter 5)
    - Hose length compensation (see data sheet chapter 8) – not needed for hose length up to 1 m

Test system


RPI running Robot Framework test bench. Operated from the other side of the test system. Automated tests are not in use at the moment.
Test bench controlled vent
Manually adjusted vent
RPI Pico w/RS-485 interface
Pressure sensor
Vaisala GMP252
Vaisala HMP60
Air inlet
Fan
Produal MIO 12-V

Miniature test system

Miniature test system consists of similar components as the “big” test system with the exception that some of the components are simulated by Arduino software. The physical parts are much smaller which has an impact on the performance of the miniature test system. For example, the miniature system’s fan is not able to produce as high a pressure difference as the actual system which may have an impact on the controller performance on miniature system and vice versa.

However, from the Pico software perspective the two test systems are identical, and the software should run on both systems without any modification.

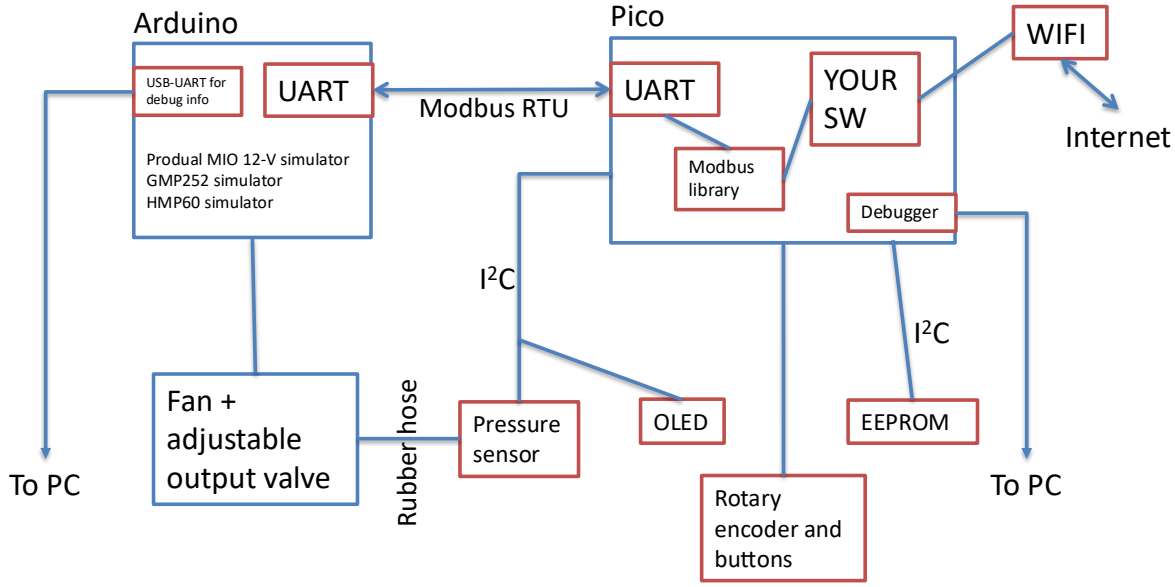
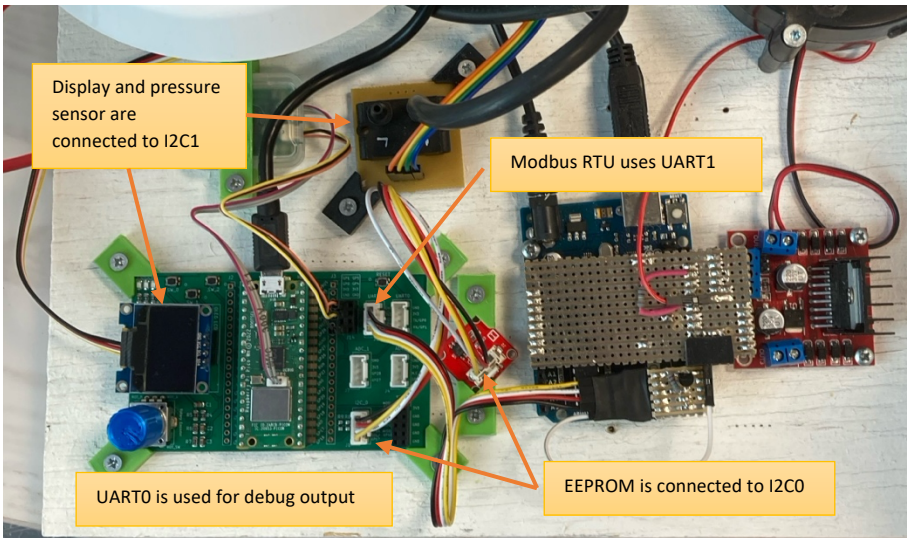


Figure 1 Miniature test system diagram



Controller software requirement specification

Software must have two operating modes: manual and automatic. The controller can be operated by local UI or by sending MQTT messages to the controller. Status changes and measurements must be displayed on the local UI and be sent to subscribers over MQTT. MQTT message format is JSON.

Automatic mode

User can set the pressure level in the ventilation duct (0 – 120 pa) in the UI or through MQTT. Pressure level is pressure difference between the room and the ventilation duct. Controller measures the pressure level and keeps it at the required level by adjusting the fan speed. If required level can't be reached within a reasonable time user is notified on the UI and over MQTT.

Manual mode

User can set the speed of the fan in UI (0 – 100%) or through MQTT. System displays current fan setting and pressure level in the UI and over MQTT.

MQTT messages

Ventilation system uses two topics: `controller/settings` and `controller/status`. Ventilation controller subscribes controller/settings and acts based on the received messages or values set in the UI. The latest setting always overrides older setting. For example, if fan speed is set to 30% with an MQTT message and shortly after that speed is set to 40% in the local UI then the local UI setting will be used. Whenever settings are changed both UI and MQTT subscribers must be notified of the changes.

Settings messages contain two fields: auto – a boolean that determines the operating mode, and another field which depends on the mode. In manual mode the second field is speed which a number that specifies the speed of the fan (0 – 100%). In automatic mode the second field is pressure which is number in the range of 0 – 120.

Settings messages examples:

Manual mode:

```
{
"auto": false,
"speed": 18
}
```

Automatic mode:

```
{
"auto": true,
"pressure": 25
}
```

Status messages are published to topic controller/status when measured values or controller settings change and periodically to ensure that the current state of the system is known to the subscribers even if they have to restart. Status messages contain fields that report the current state of the controller and the measurement values from the sensors. The fields are:

- nr – a sequence number that is incremented after each message. The first sequence number after system restart is 1.
- speed – the current fan speed.

- setpoint – current controller target value. Same as speed in the manual mode. In the automatic mode this is the target pressure.
- pressure – current pressure measurement.
- auto – a boolean which indicates the controller mode.
- error – a boolean which is set to true if the controller can't reach the target pressure within reasonable time.
- co2 – current CO2 ppm value.
- rh – current relative humidity.
- temp – measured temperature.

Status message example:

```
{
"nr": 96,
"speed": 18,
"setpoint": 18,
"pressure": 5,
"auto": false,
"error": false,
"co2": 300,
"rh": 37,
"temp": 20
}
```

Arduino sensor/IO simulator

Arduino software simulates Vaisala sensors and implements same functionality as Produal MIO 12-V. You can open Arduino's serial port with speed 9600 to see some simulator status message and to set values to sensors. There is no local echo, the characters you type are not echoed back to you, but you will see values change if the command was accepted.

Vaisala sensor values can be set by simple commands: `co2`, `rh`, and `t` followed by value you want the sensor to report. The value that is read from the command is written to the terminal when you press enter after the command. **Note that relative humidity has a default value 0.** If Arduino reboots RH is reported as zero until you set a new value with a command.

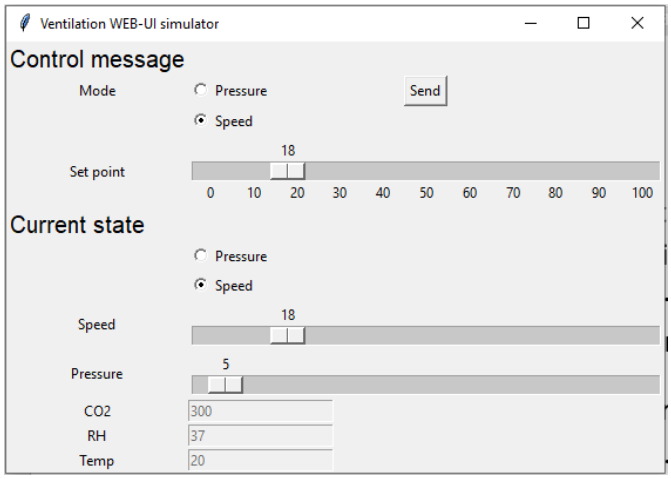
Python test software

To make testing easier two python programs are provided. The programs require some external packages to be installed:

- Numpy
  `pip install numpy`
- MQTT
  `pip install paho-mqtt`
- tk
  should be installed by default – if not
  `pip install tk`

MQTT based UI

MQTT-based user interface can set the operating mode of the controller and to receive and display the current state of the controller and the measured environmental parameters.

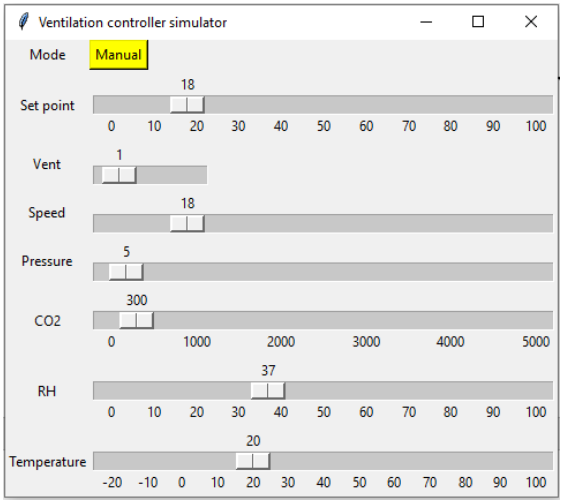


Control message side is used to change the state of the controller.

Current state displays the state of the controller as reported in the last status message. Note that current state and control message may display different values if the state of the controller is changed in the local UI after sending MQTT message from Python UI.

MQTT based ventilation system simulator

This software simulates the functionality of the ventilation system and the ventilation controller. This software serves as a reference design for messages that the ventilation controller should send (and receive) if online reporting/control is implemented.



You adjust all sliders except speed and pressure. They are controlled by the simulation algorithm in the software. Changes made to mode and set point are reported to the web UI using MQTT messages the same way as your controller software should do.

Both programs can take up to four parameters. The parameters are not mandatory, and they are positional parameters so setting the third parameter requires the first two to be given as well. The parameters are:

1. Broker IP address
2. Broker port
3. Publish topic
4. Subscribe topic

If no parameters are given, then defaults that are set in the source are used.

The values in the distributed versions are:

1. 192.168.1.10
2. 1883
3. controller/settings
4. controller/status

If you wish to use other default values feel free change them in the source.

Project template

Template project to use as a starting point comes with Modbus and MQTT-libraries. The core of both libraries is written in C. C++-wrapper classes are provided for both libraries to improve their usability in a C++ program.

You can check out the library with:

```
git clone --recurse-submodules https://gitlab.metropolia.fi/lansk/pico-modbus.git
```

Note the additional parameter –recurse-submodules that is needed to fetch submodule repository during checkout. Paho.mqtt.embedded-c is included as a submodule which means that the sources are pulled from an external repository.

Display (ssd1306) example can be found in pico-examples/i2c/ssd1306_i2c/ssd1306_i2c.c. For time being there is no C++ library for the display, but the example can be easily ported to C++ as PicoSDK is fully compatible with C++. Note: Do not adopt the awful use of goto from the example.

The template project comes with example code showing the basic usage of Modbus and MQTT libraries. The example read relative humidity using Modbus. Note that the default value for relative humidity in the simulator is zero. You need to set a non-zero value though Arduino console as instructed earlier to see non-zero values.

Minimum requirements

To have the project approved you need to implement at least the manual mode so that it can be operated from the local interface. All sensors on the system must be read and the values displayed on the OLED screen. You also need to provide the following documentation:

- User manual
    - How to operate your system through the UI
- Program documentation
    - Class diagrams, block diagrams, flow charts etc.
    - Implementation principles

For higher grade the following will be considered in the order given. Meaning that you must complete the requirements in the order given for them to count.

1. Automatic mode
2. Status reporting using MQTT with fixed network credentials and broker IP.
3. Remote control using MQTT with fixed network credentials and broker IP.
4. Setting MQTT broker IP from the local UI and storing it to persistent storage
5. Setting network credentials from the local UI and storing them to persistent storage