Monarco HAT Report

Adam Wiszniowski-Świder

Smart Industry and IoT Specialization Semester 4

1. Introduction

Monarco HAT is an add-on board which provides input-output interfaces following industrial automation standards for the Raspberry Pi (B+ and newer) minicomputer. It is designed according to the HAT (Hardware Attached on Top) specification. It can be used in various applications thanks to its flexibility in terms of accepted inputs and outputs. It operates on every open source communication, from mqtt, through python to email. Additionally it offers quick and easy communication with cloud and database services. It is limited however by being only a prototype system, which means it will automatically turn off after 2 hours, making it unreliable for usage outside of prototyping.

2. Research

Before this assignment I had no prior experience in setting up first order systems on external devices, nor did I have any idea regarding how Monarco HAT or Rexygen operates. Since prototyping only industrial automation systems are rather niche topic, I've conducted a thorough analysis of both Rexygen Manual and Monarco HAT manual. My main research questions consisted of:

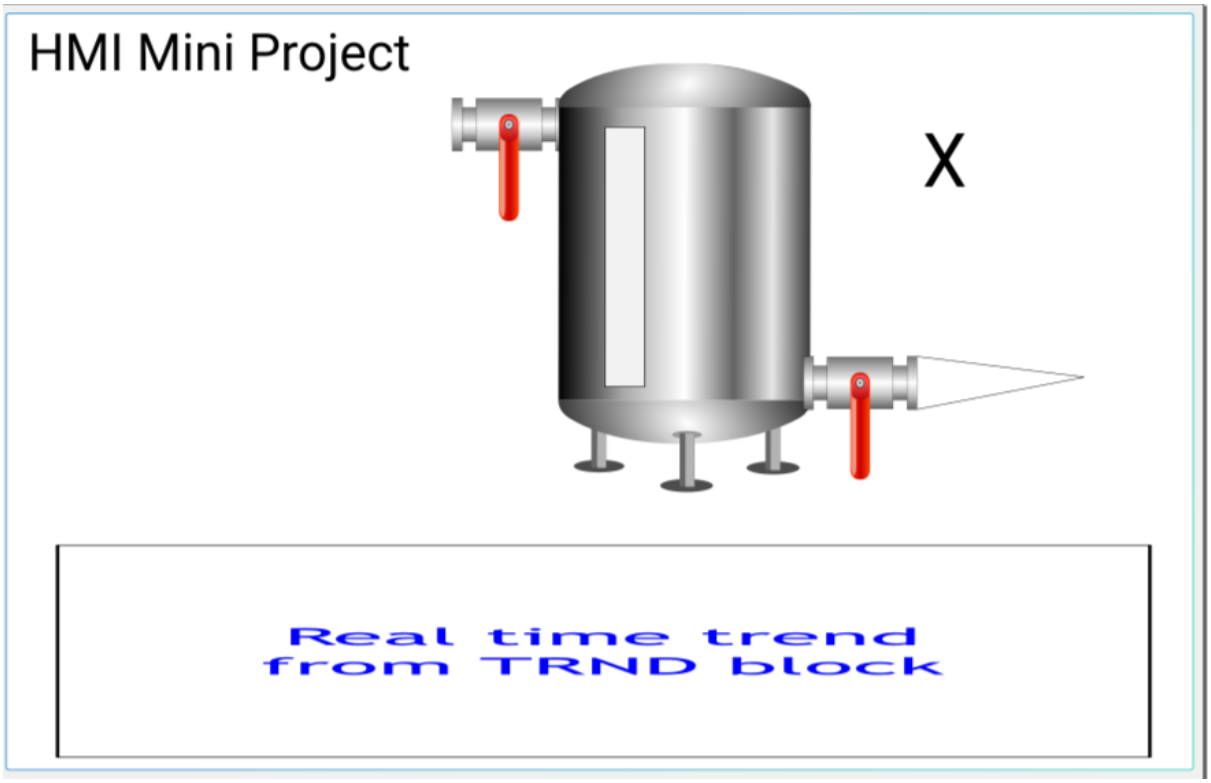
- Which tools can I use to simulate a water tank with valve(s) ?

- What to include in a display to maximize user usability ?

- How to achieve correct flow in/ flow out calculation ?

To answer the first question I have meticulously studied all available tools in Rexygen HMI Designer, as well as manuals to it. I've found a few solutions that would work, however after digging deep enough I've found a representation of a literal water tank, as well as a literal valve, which were perfect for our assignment. Setup containing a water tank and valve was sufficient, however it was still lacking a component that would allow the user to see how water level has been over time. For this purpose we've included a TRND block as well, that registers past water level within a predetermined amount of time, to help users with analysis of gained data. For the last question we've needed help from Oswald, to find some kind of equation that would fulfill our needs. We've found it in the form of the Euler method, which I'll explain in detail in the Coding part.

3. HMI Display



For the HMI part of this assignment I have made the graphical display of our system. Through this display users can control both valves, and receive insight on how much water is currently in the tank.. Inputs are included in form of Valves, the top one is used for pouring water inside the tank, and the bottom valve is used for draining. They both use the same equation, so input and output paces are the same. For a direct insight on current water level, a digit displays on the right side of the tank (the X sign on screenshot), additionally for further data analysis water level over time is also included on the bottom.

4. Coding

To achieve the desired outcome, which is a water tank with the ability of flowing water in and out, we have resolved to a variant of the so called Euler method. Our solution is based on current water level (later referred to as value), 2 inputs which represent each valve, one for flowing water in and one for draining (later referred to as input0 and input1 respectively) and predetermined setpoints to which value seeks to achieve, bottom one being 0 and maximum one being 80. If input0 is activated, then setpoint is changed to 80, then if value is lower or equal to said setpoint, following equation will be executed:

```
value := value + dt;

sy := (value*K/T) - (counter/T);

counter := counter + (sy * dt);
```

This adds a predetermined dt (which is 0.01) to value, at the pace calculated by sy, which itself is calculated by current value multiplied by gain pace (which is K at the value of 10). Thanks to these calculations we're able to achieve a steady flow that is accelerating until it starts to reach the end of its maximum capacity, which is exactly what we were looking for. To eliminate the possibility of an overshoot we're limiting counter by implementing :

```
IF counter > setpoint THEN

        counter := setpoint;

    END_IF;
```

Which limits the maximum counter to setpoint value.

To achieve the same effect with a draining valve, we're using a similar methodology to the input valve. First we're setting setpoint to 0, and then compare the value with said setpoint. If it's lower or equal, then dt is subtracted from value, at the pace that's the same as in the input valve.(value := value - dt;). Similarly as before, it's limited by forcing counters to not go below the minimum limit.

```
IF counter < min_limit THEN

        counter := min_limit;

    END_IF;
```

We had to go through the debugging phase a few times, the most irritating of them being when our minimal value seemed to be stuck at whatever K gain value would be. When we changed it to 20 it was stucking at 20. When we lowered it as low as 0.1 it would go to 1, but extremely slowly, as well as it still wouldn't be so that would just mask a problem. The issue turned out to be an eclipse value which we previously used when calculating sy. Fortunately we were able to switch the counter, and now it works flawlessly.

5. Conclusion

This miniProject has primarily taught me patience. Both MonarcoHAT and Rexygen are pretty niche products, therefore finding helpful tutorials online has turned out to be pointless. Because of that we had to go through tons and tons of manuals Rexygen and Monarco, fortunately we were mostly able to find all we were looking for, and our skill and consultations with teachers has helped with the rest. It was a great opportunity for me to learn more regarding working with dedicated hardware, as well as discovering new useful software. I see a lot of possibilities with this system, and I'll be happy to create other applications for Monarco in later semesters, hopefully more advanced and with using irl inputs and outputs.

6. Diagram