CLASSPERT

MAKING EMBEDDED SYSTEMS COURSE BY ELECIA WHITE

FINAL PROJECT REPORT

SMART MICRO PUMP

ANDRÉ A. M. ARAÚJO

FEBRUARY 2022

TABLE OF CONTENTS

1. INTRODUCTION

Micro pumps are commonly seen in various applications including medical, pharmaceutics and laboratory/analytical instruments, gas detectors, air quality and particulate matter analyzers, among other gas sampling devices. Frequently, these products or processes require very precise flow control to work correctly.

As the final project presented to the Classpert MAKING EMBEDDDED SYSTEMS course by Elecia White, a ***Smart Micro Pump*** was developed and a working prototype was built. The following report details its conception, development and testing, concluding with thoughts on how to further develop the idea.

2. APPLICATION DESCRIPTION

The proposed embedded system includes a microcontroller, a flow sensor (with an optional inlet filter), a micro pump (powered by a DC motor), DC motor controller and driver, a button for mode selection and a serial communication port for setting configurations and receiving data for further analysis. The microcontroller features a PID controller for accurate flow output.

This could be supplied as an OEM module: an all-in-one gas sampling solution for manufacturers to integrate into their products, reducing their development time and cost.

3. HARDWARE DESCRIPTION

The system is based on the STM32L152RB microcontroller that is included in the 32L152CDISCOVERY evaluation board, from ST Microelectronics. This microcontroller features 128 KB of flash memory, 16 KB of RAM, an internal oscillator capable of producing a 16 MHz clock (1% accuracy), 12-bit ADC, 16-bit timers, among other common peripherals such as UART, I^{2}C, SPI, DMA, etc.

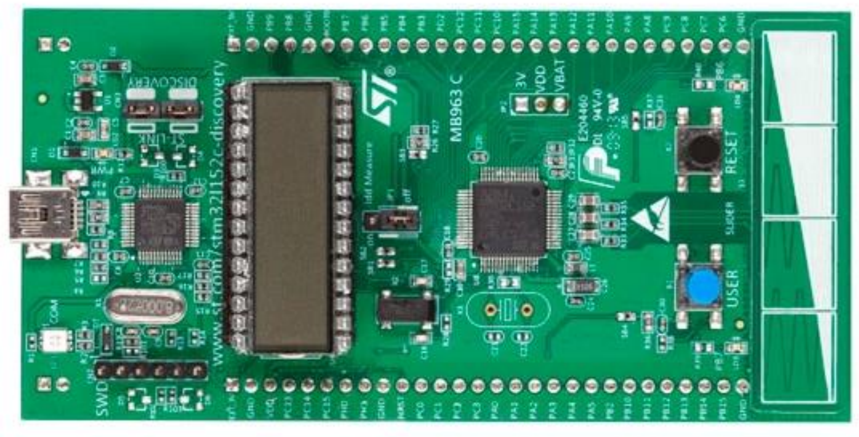


Figure 3-A – 32L152CDISCOVERY development board

The DISCO board also features a button and 2 LEDs that were used as part of the prototype's interface. The LCD and capacitive touchpads, although very interesting, were not used in this project.

The flow sensor used is a D6F-P0010A1, made by OMRON, capable of measuring gas flow from 0 to 1 L/min, and requiring only a 5V, 15mA power supply.

Figure 3-B – Flow sensor (D6F-P0010A1 from OMRON)

For the pump, a UNMP 09KPDC-M micro diaphragm pump, from KNF, was selected. This pump can reach a flow of 0.8 L/min and up to 0.6 bar of positive pressure (and 0.5 bar of negative pressure). The pump is powered by a DC motor that can be driven from a power supply of up to 6V, 160mA.

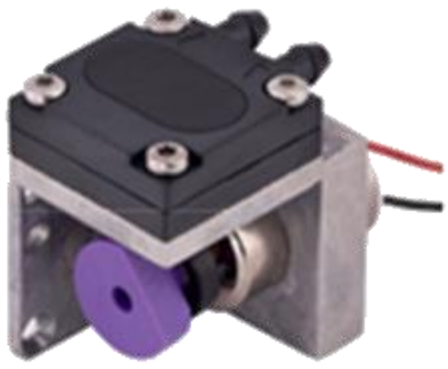


Figure 3-C – Micro diaphragm pump (UNMP 09KPDC-M from KNF)

In order to drive this small motor, a single transistor was used (2N3904). For pumps requiring higher power, a recommendation would be to use a voltage controlled current source, based on a power transistor (such as the TIP120) and an operational amplifier. The latter allows for better current control, but the simpler approach was good enough for the small pump.

The pump current, measured using a 1 Ω shunt resistor, and flow sensor voltage are fed to the MCU. These two signals pass through anti-aliasing filters, to remove noise and any lower frequency oscillations, before going into the ADC.

The schematic of the pump driver and flow sensor, including the anti-aliasing filters, is presented in the next figures:

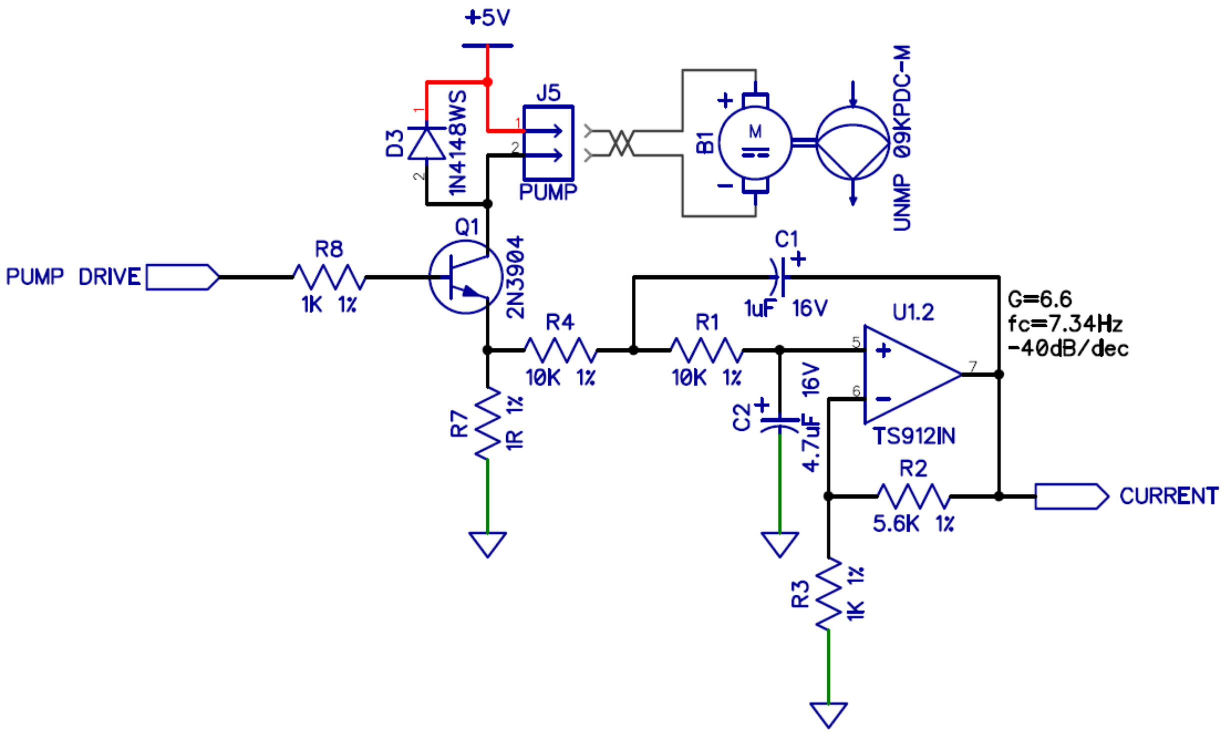


Figure 3-D – Pump driver and current amplifier and filter

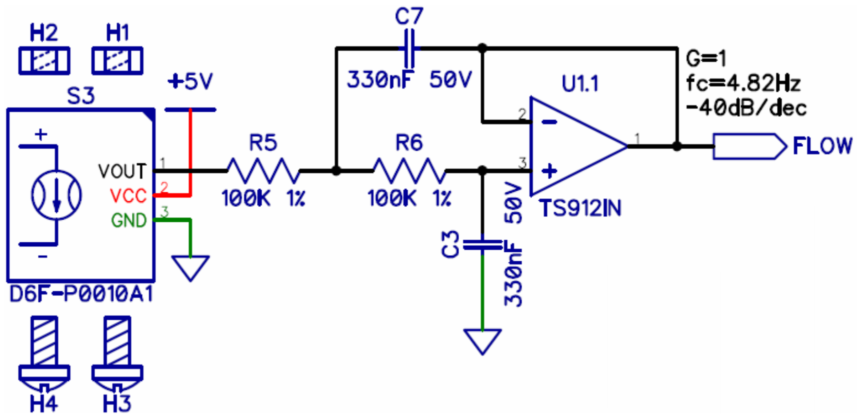


Figure 3-E – Flow sensor and filter

Another input for this prototype is a trimpot that is used to control the desired pump flow (in MANUAL MODE, it sets the PWM duty cycle; in AUTO MODE, adjusts the reference flow).

A hardware block diagram is show in the next figure, detailing the parts and their basic connections. Please notice that elements powered directly from 5V are highlighted in red; other components are powered from a 3V voltage regulator.

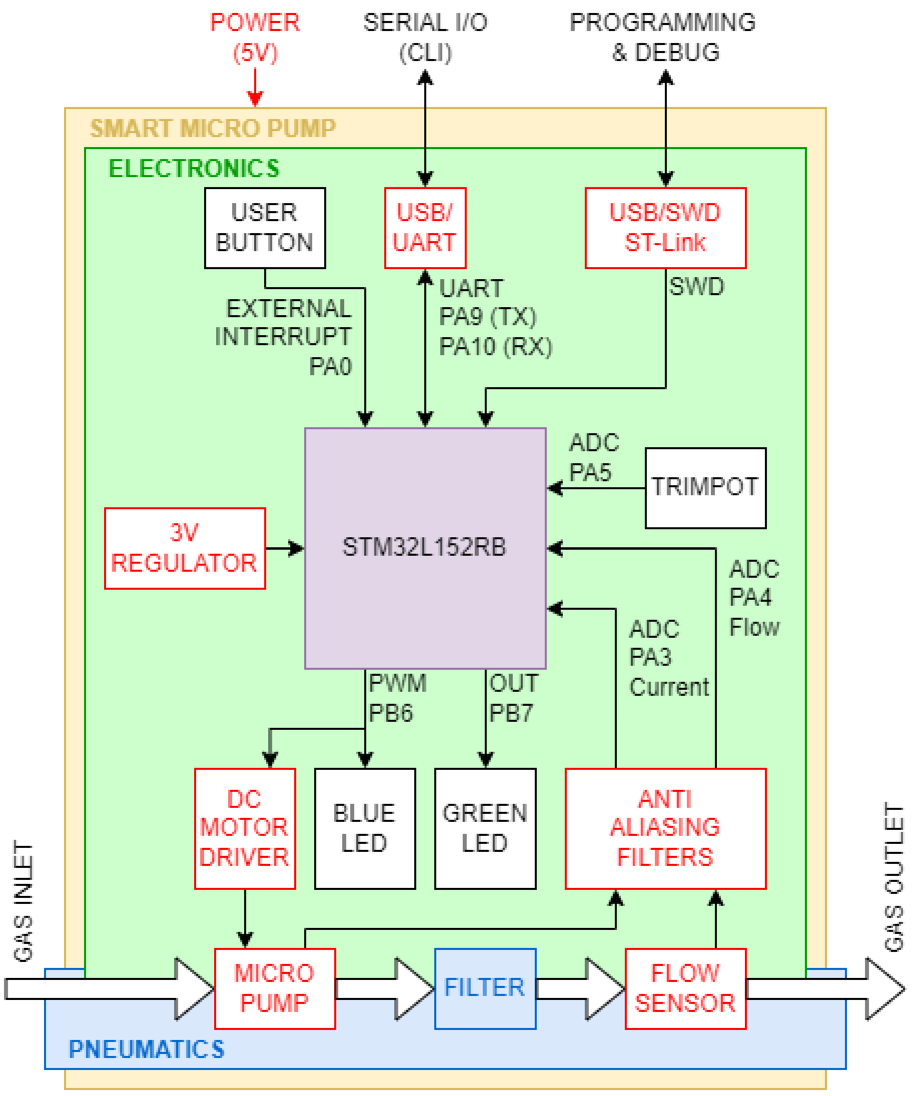


Figure 3-F – Hardware block diagram

The assembled prototype is showed in the following figure:

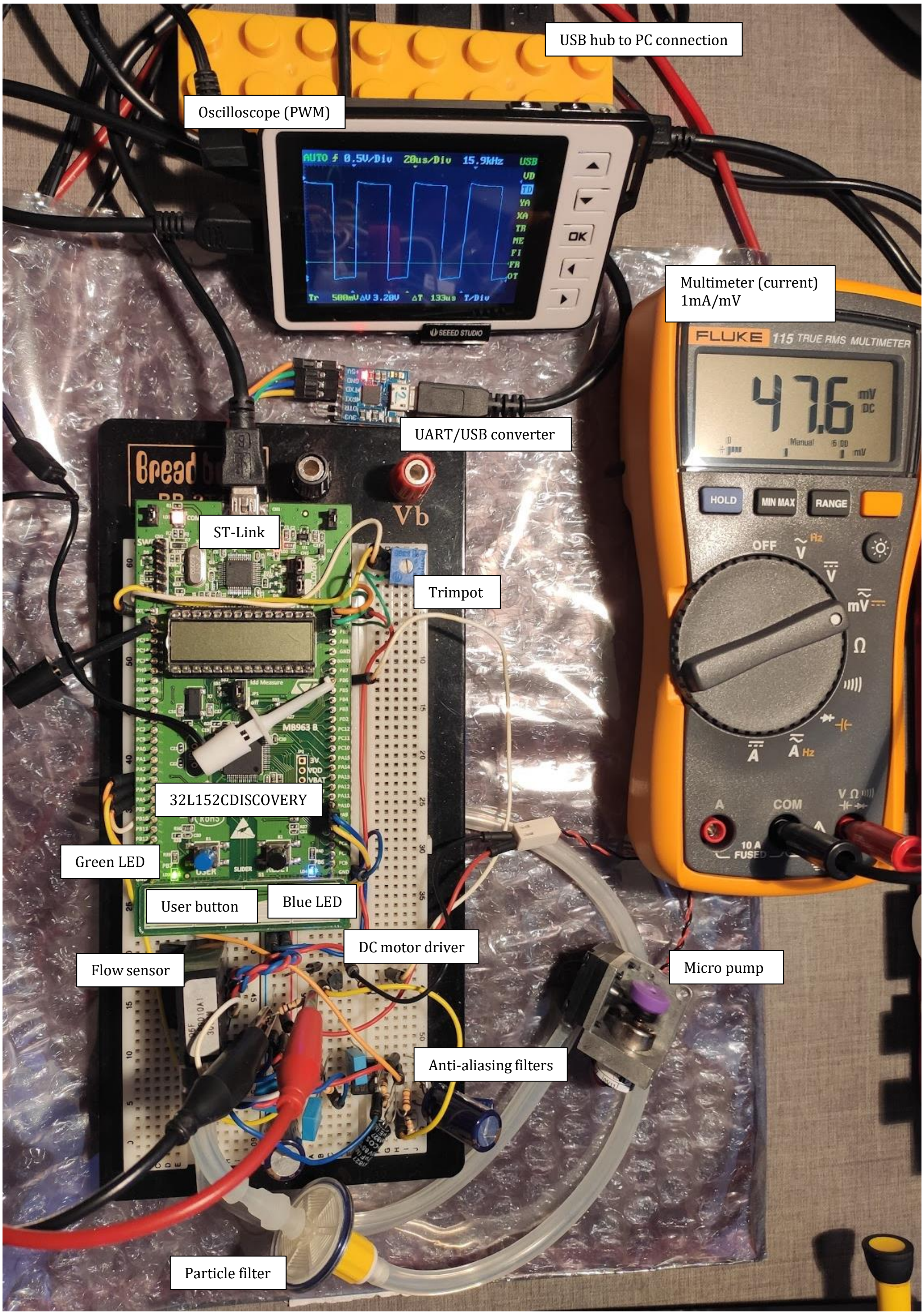


Figure 3-G – Smart Micro Pump prototype

4. SOFTWARE DESCRIPTION

The software was designed to be flexible and easy to read, allowing for future expansions. Using the STM32 CubeMX configuration tool helped the development by automatically generating the heavier parts of configuration code. By the extensive application of façades, the main program is clean and easy to understand – the lower level parts of the HAL are separated in source and header files related to each peripheral or aspect of the main code.

4.1. Block Diagrams

A software block diagram shows the main parts of the code:

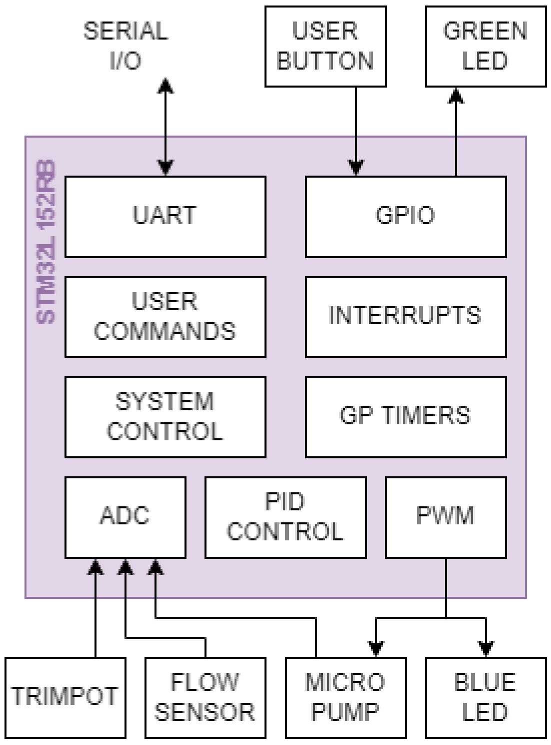


Figure 4-A – Software block diagram

Also, a state machine diagram was planned:

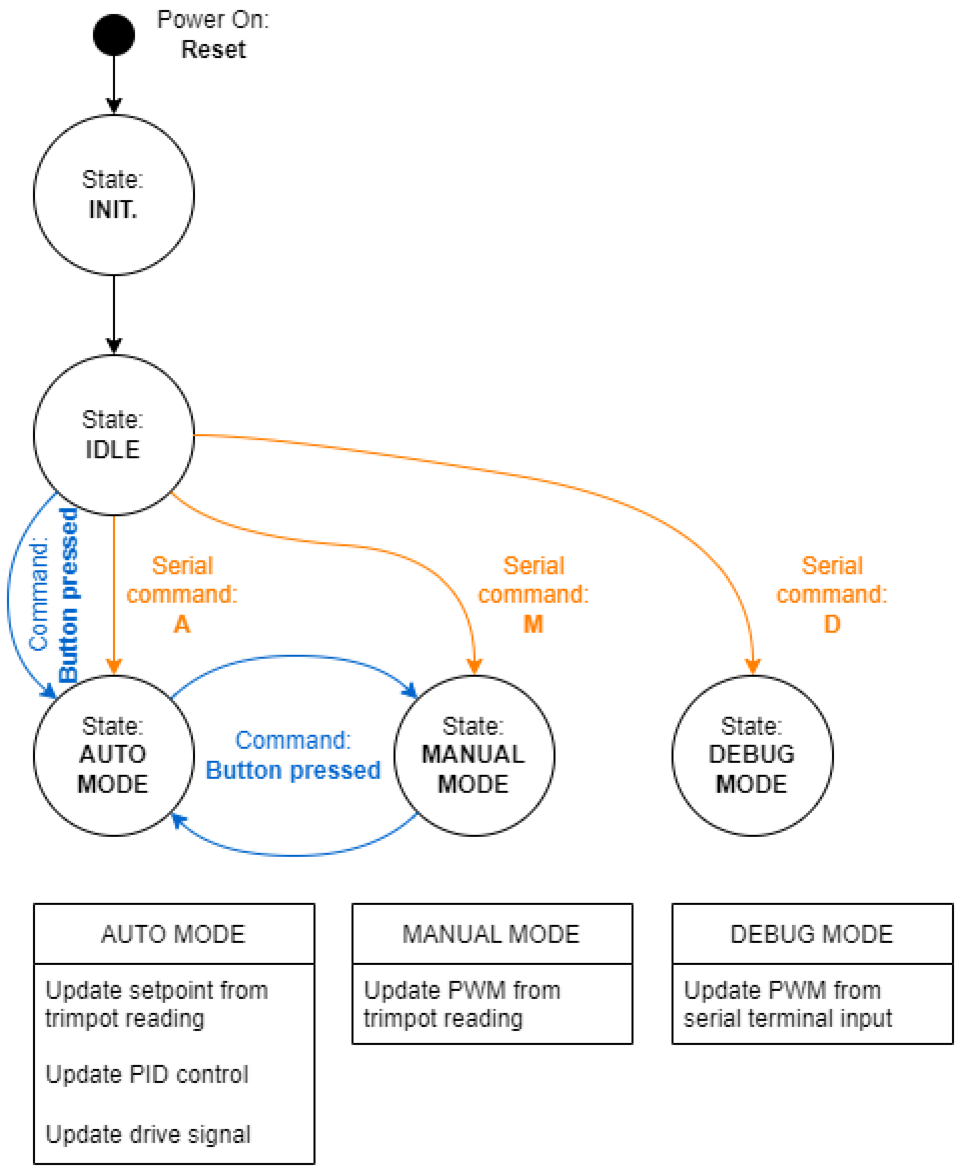


Figure 4-B – State machine diagram

Table 4-A – State machine table

STATES	**ACTION**		**EVENTS**			
	PUMP	**GREEN LED**	**Button pressed**	**Trimpot adjusted**	**Serial command**	**Reset**
INITIALIZATION	***Off***	***Off***	-	-	-	INITIALIZATION
IDLE	***Off***	***Off***	AUTO MODE	-	A: AUTO MODE M: MANUAL MODE D: DEBUG MODE	INITIALIZATION
AUTO MODE	***On***	***On***	MANUAL MODE	MANUAL MODE	A: AUTO MODE M: MANUAL MODE D: DEBUG MODE	INITIALIZATION
MANUAL MODE	***On***	***Off***	AUTO MODE	AUTO MODE	A: AUTO MODE M: MANUAL MODE D: DEBUG MODE	INITIALIZATION
DEBUG MODE	***On***	***Blinking at 1 Hz***	AUTO MODE	-	A: AUTO MODE M: MANUAL MODE D: DEBUG MODE	INITIALIZATION

4.2. Software modules

Next, a there is brief presentation of each software section (in alphabetical order of the source files):

4.2.1. ADC (adc.c/adc.h)

The internal 12-bit ADC is used to obtain the flow reading from the sensor as well as pump motor current. The reference voltage for the ADC in this board is 3.0V. Additionally, the trimpot voltage is measured and later used to configure PWM duty cycle or controller set point. Finally, two internal values are also measured, the MCU internal reference voltage and temperature sensor.

Since 5 channels are used, the ADC was configured to make the conversions sequentially. The results are automatically picked up via DMA, then put into RAM. Finally, an interrupt occurs and the callback function `HAL_ADC_ConvCpltCallback()` sets a flag so the main program can deal with the new set of ADC results.

4.2.2. Command Line Interface (cli.c/cli.h)

A simple command line interface was programmed in order to let the user select among the three operation modes (AUTO, MANUAL or DEBUG); while in DEBUG mode, the user can set the PWM duty cycle by sending the desired percentage (0 to 100) via the serial console.

This CLI is rudimentary but serves as a template to be extended in the future.

4.2.3. Direct Memory Access (dma.c/dma.h)

The DMA captures data from the ADC and transfers to RAM. These files were completely generated by CubeMX.

4.2.4. General Purpose Input and Output (gpio.c/gpio.h)

This source code configures all GPIO pins of the device. The button interrupt (EXTI) triggers the debouncing algorithm. Both rising and falling edges are detected (in this hardware, the button is active high). After successful debounce, a flag is set and the main loop toggles the operating mode, also changing the green LED to indicate which mode is active:

- MANUAL –PWM duty cycle adjusted by the trimpot; green LED off;
- AUTOMATIC – PID controller active, green LED on.

An additional output, with the aid of the oscilloscope, was used for testing and profiling execution times of code sections.

4.2.5. Global variables (global.c/global.h)

These files contain declarations and definitions of the global variables used in the program, which makes it easier to include these variables in other source files.

4.2.6. Initialization (init.c/init.h)

This source code is executed as the main program starts, and configures all peripherals for the application. The `MCU_init()` is a good example of a façade, which contains a series of HAL initialization functions but makes the code much cleaner on the main file.

4.2.7. Main program (main.c/main.h)

The main file contains the logic algorithm that executes the state machine according to flags that are set synchronously and asynchronously by interrupts. This allows for very short interrupt service routines, as the heavy lifting is made on the main loop.

4.2.8. PID controller (pid.c/pid.h)

To keep things as easy to understand as possible, the PID controller was implemented based on a simple algorithm available at Wikipedia, and also took inspiration from an implementation made by "Phil's Lab" YouTube channel. A struct was created to store all data pertinent to the controller, and at each sample time, the control algorithm is updated by reading the sensor signal and using it as feedback to close the control loop; then the output is calculated and later used to update the PWM output.

4.2.9. Pulse Width Modulation (pwm.c/pwm.h, tim.c/tim.h)

The PWM is generated by TIM4. This signal drives the DC motor transistor. The frequency used was 16 kHz, to avoid noise in the audible range. The façade function `PWM_setPulse()` updates the pulse width with the per mille (‰) of the duty cycle (0 to 1000).

4.2.10. General purpose timer (tim.c/tim.h)

Two general purpose timers are used. Firstly, TIM2 generates an interrupt every 10 ms (100 Hz). This interrupt event triggers the start of ADC conversions. After conversions are finished, another interrupt is generated by the ADC, leading to an ISR that sets a flag (`flag_EOC` for "END OF CONVERSIONS") that is later handled in the main loop (for PID control step calculation and PWM duty cycle update). This kind of handling is preferable since it keeps the interrupt service routine as short as possible and the heavy processing is done in the main loop.

Secondly, TIM6 is used to allow for a nom blocking debounce algorithm for the user button. When the button is pressed, an external interrupt is triggered and its servicing routine starts this timer; 1 ms later, this timer triggers its interruption and one step debouncing routine is executed. After the button is successfully debounced, this timer is paused.

4.2.11. UART (usart.c/usart.h)

UART is used to receive user command. The data received generates an interrupt so the program can handle the input commands:

- ‘A’ (AUTO MODE):
  PID controller active: set point set by trimpot voltage (0-3 V corresponds to 100-500 mL/min;) and automatic duty cycle update according to PID controller output;
- ‘M’ (MANUAL MODE):
  PWM Duty Cycle set by trimpot voltage: 0-3 V corresponds to 0-100 %;
- ‘D’ (DEBUG MODE):
  User can enter desired PWM duty cycle via the serial console using the command “#p” where # is a number from 0 to 100 and ‘P’ executes the pulse width update. If an incorrect command is sent, the program responds with an alert message.

Also, UART is used to transmit data periodically at 100 Hz. A set of data points is sent by the prototype:

- PID Controller:
  - Set Point
  - Feedback (flow sensor reading, in mL/min)
  - Error (Set Point – Feedback)
  - Proportional term
  - Integral term
  - Derivative term
  - Output (normalized, from 0.000 to 1.000)
- Pump Current (mA)
- MCU internal temperature (°C)
- MCU internal reference voltage (V)

4.2.12. Remaining source and header files

The remaining source and header files were generated automatically by the CubeMX application and were not modified.

4.3. Software licensing

STM32 HAL is licensed under BSD 3-Clause license:
https://opensource.org/licenses/BSD-3-Clause

4.4. Software versioning control

Git and GitHub were used for versioning control. A header file (version.h) contains the code version. The project is publicly available on GitHub at the following address:
https://github.com/andremdaraujo/SMART-MICRO-PUMP

5. BUILDING INSTRUCTIONS

5.1. Building the hardware

The prototype was assembled in a breadboard using an assortment of components recovered from other projects. Besides the DC motor driver and anti-aliasing filters shown earlier, all components are connected directly to the ports of the microcontroller, as documented in the software comments:

```
//TARGET:
//      STM32L152RB ("STM32L-Discovery" Board)
//
//INPUTS:
//      User Button:     PA0     (Active high, rising and falling edges detection)
//      Pump Current:    PA3     (Micro pump electrical current)
//      Pump Flow:       PA4     (Micro pump flow)
//      Trimpot:         PA5     (Potentiometer to adjust PWM duty cycle/set point on MANUA/AUTO modes
//      UART RX:         PA10    (Commands reception)
//
//OUTPUTS:
//      UART TX:         PA9     (Data transmission)
//      DC Motor Drive:  PB6     (PWM)
//      Blue  LED:       PB6     (PWM indication)
//      Green LED:       PB7     (Mode indication)
//      Test output:     PC0     (Timing verification using an oscilloscope)
//
//PERIPHERALS:
//      ADC:                     Internal ADC (3 channels)
//      Sampling period timer:   TIM2    (fS    = 100  Hz)
//      PWM generation:          TIM4    (fPWM =  16 kHz)
//      Debounce timer:          TIM6    (fDEB =   1 kHz)
//      UART:                    USART1  (baud = 115200 bps, 8N1)
```

5.2. Building the software

Code was built with STM32CubeIDE version 1.8.0 and the HAL for L1 series.

5.3. Testing and debugging

Testing and debugging were done with the aid of some hardware and software tools that are detailed next:

5.3.1. Hardware tools

A digital multimeter was used to measure voltages, resistances and especially the pump current (a shunt resistor of 1 Ω makes it possible to obtain the current with a ratio of 1 mA/mV). A 1 channel mini oscilloscope (seeed Studio DSO Nano v2) was also used to check for PWM signals, timer configurations and serial communication debugging.

For programming the device, the ST-Link debugger included on Discovery board was the obvious choice.

Finally, a low cost UART to Virtual COM USB converter (based on the Silicon Labs PC210x chipset) allowed the connection of the prototype to the PC via the CLI.

5.3.2. Software tools

Most of the work was done using the STM32CubeIDE and included STM32CubeMX configuration tool.

For sending and receiving data from the serial port, two applications were used: the CoolTerm serial terminal, for sending commands to the prototype; and Serial Oscilloscope for receiving data and plotting in graphs.

The hardware schematic was made using the free version of DipTrace.

Hardware and software diagrams were drawn using diagrams.net.

6. SYSTEM ANALISYS

6.1. Hardware aspects

The hardware is not complicated because an amplified sensor was used and the DC motor driver is based on a single transistor.

Although the flow sensor reading has little noise, the alternating nature of the diaphragm pump causes oscillation in the flow that are captured by the sensor readings and might disturb the control algorithm. Therefore, a low pass filter was used after the sensor output to filter this oscillation.

For the pump current sense, a low pass filter was also used, but this included a gain of 6.6 times, because the voltage on the 1 Ω shunt was in the order of around 50mV. Also, this signal is directly related to the PWM, so it is an oscillating wave with the same 16 kHz frequency of the PWM signal. The low pass filter removes the AC element and delivers a quasi-DC signal to the ADC input.

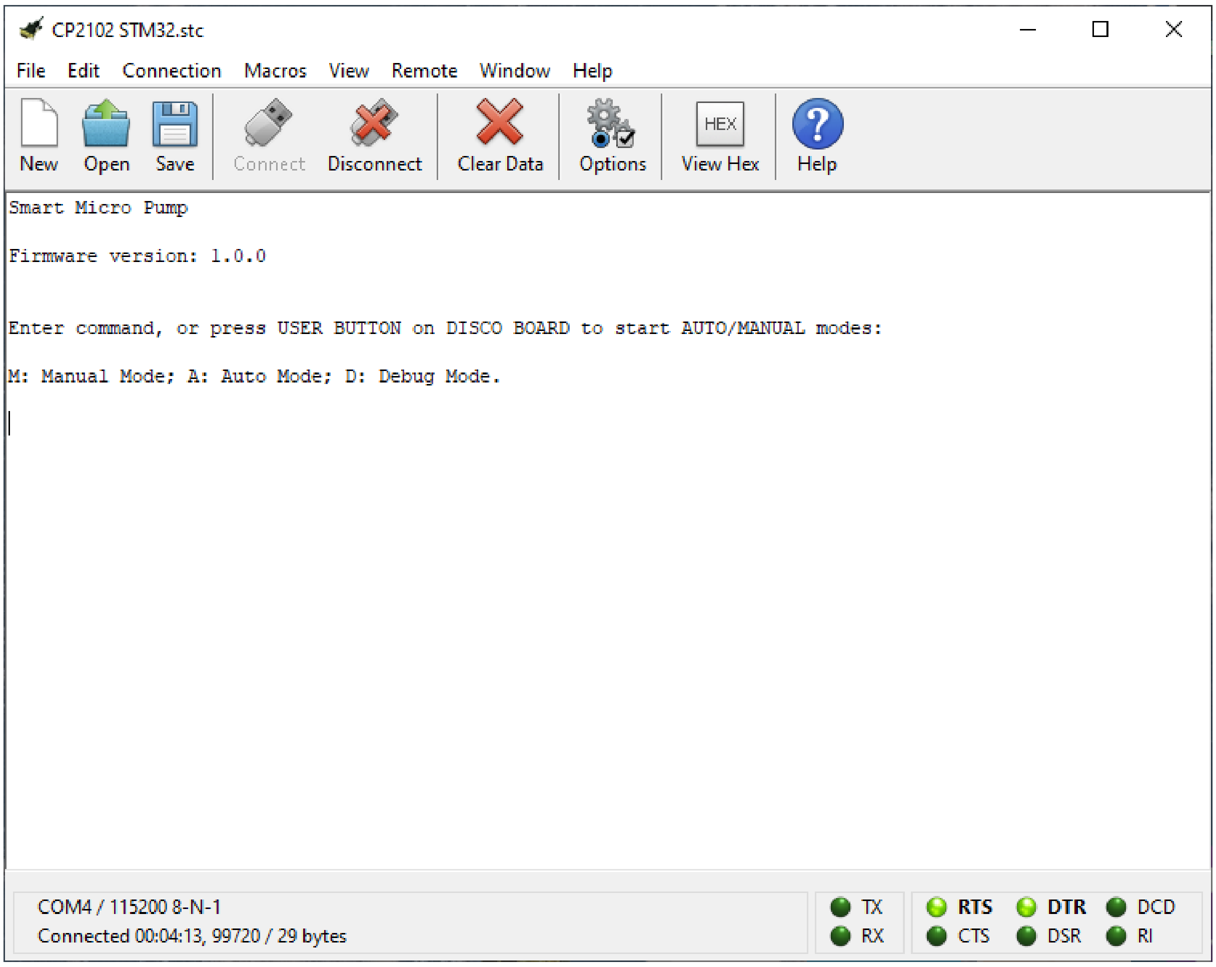
Despite the presence of the low pass filters, the breadboard used suffered from poor connections that introduced a lot of noise in the system. For a proper evaluation of the performance of the system, a custom PCB should be developed and tested.

The topology of the filters is the Sallen-Key, which result in 2nd order Butterworth filters. A great tool for designing such filters is available at:
http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm

6.2. Command Line Interface and DEBUG mode

The Command Line Interface can be used to set the PWM duty cycle from 0 to 100. Here, CoolTerm was used as the serial console.

As soon as the prototype's software starts, it sends the following message:

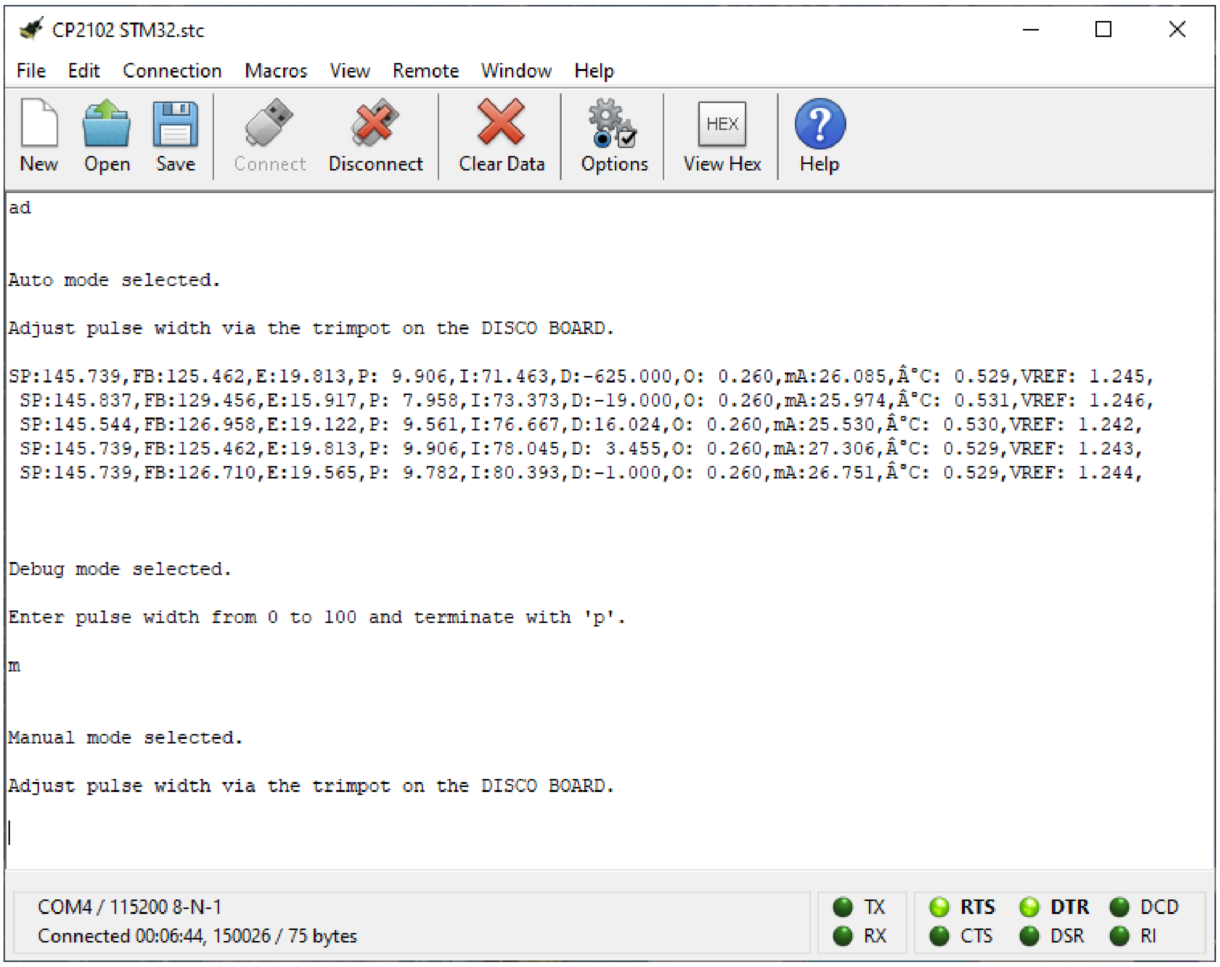

CP2102 STM32.stc
File Edit Connection Macros View Remote Window Help
New Open Save Connect Disconnect Clear Data Options View Hex Help
TX RX RTS CTS DTR DSR DCD RI

```
Smart Micro Pump

Firmware version: 1.0.0


Enter command, or press USER BUTTON on DISCO BOARD to start AUTO/MANUAL modes:

M: Manual Mode; A: Auto Mode; D: Debug Mode.
```

COM4 / 115200 8-N-1
Connected 00:04:13, 99720 / 29 bytes

Figure 6-A – CLI startup message, including firmware version

The user can select 'A' for AUTO mode, 'D' for DEBUG mode and 'M' for MANUAL mode. When AUTO mode is selected, the data logging starts immediately. The other modes do not send data and wait for user commands to be sent via CLI or trimpot adjustments.


CP2102 STM32.stc
File Edit Connection Macros View Remote Window Help
New Open Save Connect Disconnect Clear Data Options View Hex Help

```
ad


Auto mode selected.

Adjust pulse width via the trimpot on the DISCO BOARD.

SP:145.739,FB:125.462,E:19.813,P: 9.906,I:71.463,D:-625.000,O: 0.260,mA:26.085,Â°C: 0.529,VREF: 1.245,
 SP:145.837,FB:129.456,E:15.917,P: 7.958,I:73.373,D:-19.000,O: 0.260,mA:25.974,Â°C: 0.531,VREF: 1.246,
 SP:145.544,FB:126.958,E:19.122,P: 9.561,I:76.667,D:16.024,O: 0.260,mA:25.530,Â°C: 0.530,VREF: 1.242,
 SP:145.739,FB:125.462,E:19.813,P: 9.906,I:78.045,D: 3.455,O: 0.260,mA:27.306,Â°C: 0.529,VREF: 1.243,
 SP:145.739,FB:126.710,E:19.565,P: 9.782,I:80.393,D:-1.000,O: 0.260,mA:26.751,Â°C: 0.529,VREF: 1.244,



Debug mode selected.

Enter pulse width from 0 to 100 and terminate with 'p'.

m


Manual mode selected.

Adjust pulse width via the trimpot on the DISCO BOARD.
```

COM4 / 115200 8-N-1
Connected 00:06:44, 150026 / 75 bytes

Figure 6-B – CLI response after ‘A’, ‘D’ and ‘M’ commands

6.3. PID controller tuning and testing

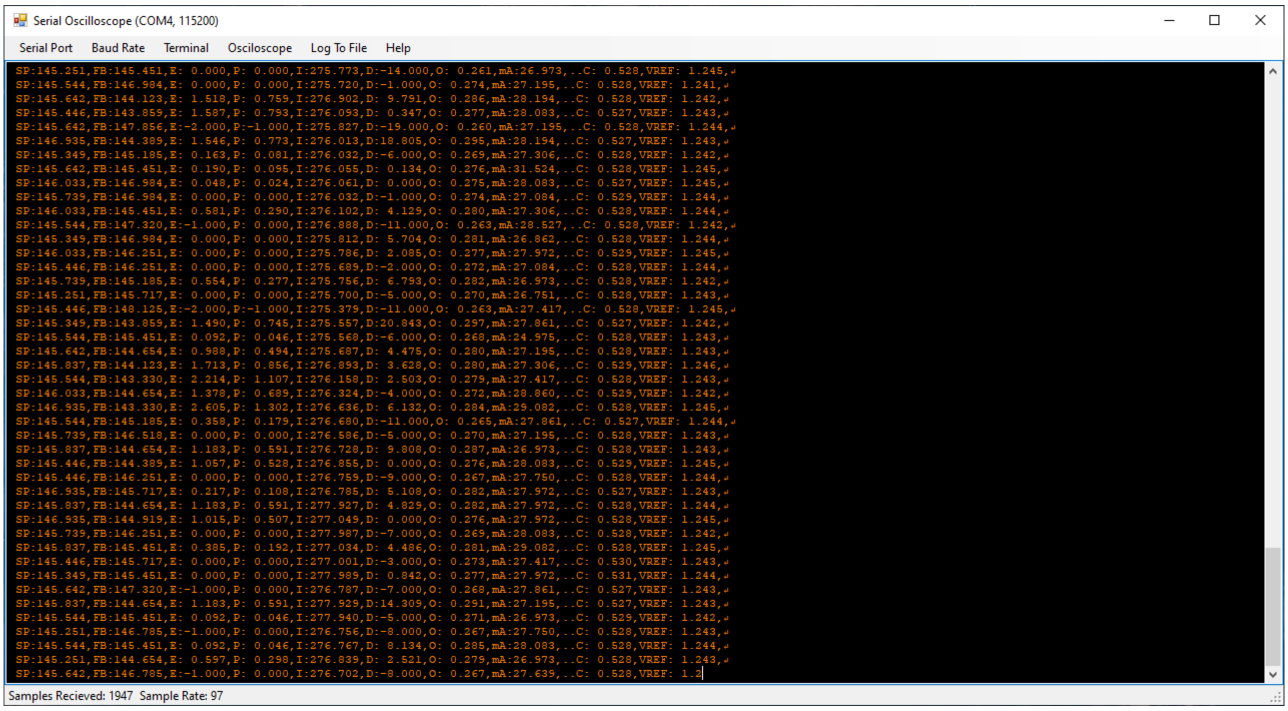
6.3.1. Tuning procedure

The tuning of the PID controller was done empirically. First, proportional gain was gradually increased until the system started to oscillate. Then, it was reduced back to the point when oscillations did not occur – however, stationary error was still large. That’s when the integral term was introduced (as this terms function is to remove stationary error). After increasing this term to achieve a satisfactory response, a small amount of derivative gain was included, as this term serves to dampen the oscillations (but also suffer from noise in the measurement, so it has to be carefully adjusted).

The final response still present a bit of oscillation when abrupt changes to the set point are made, but the system settles in approximately 1 s, which is appropriate for this application. If no overshoots were allowed, lower gains would be necessary, but the response would be slower. There's always a compromise between response time and stability.

Furthermore, to improve the system response, a higher sample rate might be necessary, as well as a proper circuit board for the sensor and filters (the breadboard introduces a lot of noise to the system, limiting the effective resolution of the ADC and the overall system response and stability).

To aid the tuning process, a freely available tool called "Serial Oscilloscope" was used. It consists of a serial terminal that parses numeric data and is able to display up to 9 channels graphically.



```
SP:145.251,FB:145.451,E: 0.000,P: 0.000,I:275.773,D:-14.000,O: 0.261,mA:26.973,..C: 0.528,VREF: 1.245,↵
SP:145.544,FB:146.984,E: 0.000,P: 0.000,I:275.720,D:-1.000,O: 0.274,mA:27.195,..C: 0.528,VREF: 1.241,↵
SP:145.642,FB:144.123,E: 1.518,P: 0.759,I:276.902,D: 9.791,O: 0.286,mA:28.194,..C: 0.528,VREF: 1.242,↵
SP:145.446,FB:143.859,E: 1.587,P: 0.793,I:276.093,D: 0.347,O: 0.277,mA:28.083,..C: 0.527,VREF: 1.243,↵
SP:145.642,FB:147.856,E:-2.000,P:-1.000,I:275.827,D:-19.000,O: 0.260,mA:27.195,..C: 0.528,VREF: 1.244,↵
SP:146.935,FB:144.389,E: 1.546,P: 0.773,I:276.013,D:18.805,O: 0.295,mA:28.194,..C: 0.527,VREF: 1.243,↵
SP:145.349,FB:145.185,E: 0.163,P: 0.081,I:276.032,D:-6.000,O: 0.269,mA:27.306,..C: 0.528,VREF: 1.242,↵
SP:145.642,FB:145.451,E: 0.190,P: 0.095,I:276.055,D: 0.134,O: 0.276,mA:31.524,..C: 0.528,VREF: 1.245,↵
SP:146.033,FB:146.984,E: 0.048,P: 0.024,I:276.061,D: 0.000,O: 0.275,mA:28.083,..C: 0.527,VREF: 1.245,↵
SP:145.739,FB:146.984,E: 0.000,P: 0.000,I:276.032,D:-1.000,O: 0.274,mA:27.084,..C: 0.529,VREF: 1.244,↵
SP:146.033,FB:145.451,E: 0.581,P: 0.290,I:276.102,D: 4.129,O: 0.280,mA:27.306,..C: 0.528,VREF: 1.244,↵
SP:145.544,FB:147.320,E:-1.000,P: 0.000,I:276.888,D:-11.000,O: 0.263,mA:28.527,..C: 0.528,VREF: 1.242,↵
SP:145.349,FB:146.984,E: 0.000,P: 0.000,I:275.812,D: 5.704,O: 0.281,mA:26.862,..C: 0.528,VREF: 1.244,↵
SP:146.033,FB:146.251,E: 0.000,P: 0.000,I:275.786,D: 2.085,O: 0.277,mA:27.972,..C: 0.529,VREF: 1.245,↵
SP:145.446,FB:146.251,E: 0.000,P: 0.000,I:275.689,D:-2.000,O: 0.272,mA:27.084,..C: 0.528,VREF: 1.244,↵
SP:145.739,FB:145.185,E: 0.554,P: 0.277,I:275.756,D: 6.793,O: 0.282,mA:26.973,..C: 0.528,VREF: 1.242,↵
SP:145.251,FB:145.717,E: 0.000,P: 0.000,I:275.700,D:-5.000,O: 0.270,mA:26.751,..C: 0.528,VREF: 1.243,↵
SP:145.446,FB:148.125,E:-2.000,P:-1.000,I:275.379,D:-11.000,O: 0.263,mA:27.417,..C: 0.528,VREF: 1.245,↵
SP:145.349,FB:143.859,E: 1.490,P: 0.745,I:275.557,D:20.843,O: 0.297,mA:27.861,..C: 0.527,VREF: 1.242,↵
SP:145.544,FB:145.451,E: 0.092,P: 0.046,I:275.568,D:-6.000,O: 0.268,mA:24.975,..C: 0.528,VREF: 1.243,↵
SP:145.642,FB:144.654,E: 0.988,P: 0.494,I:275.687,D: 4.475,O: 0.280,mA:27.195,..C: 0.528,VREF: 1.243,↵
SP:145.837,FB:144.123,E: 1.713,P: 0.856,I:276.893,D: 3.628,O: 0.280,mA:27.306,..C: 0.529,VREF: 1.246,↵
SP:145.544,FB:143.330,E: 2.214,P: 1.107,I:276.158,D: 2.503,O: 0.279,mA:27.417,..C: 0.528,VREF: 1.243,↵
SP:146.033,FB:144.654,E: 1.378,P: 0.689,I:276.324,D:-4.000,O: 0.272,mA:28.860,..C: 0.529,VREF: 1.242,↵
SP:146.935,FB:143.330,E: 2.605,P: 1.302,I:276.636,D: 6.132,O: 0.284,mA:29.082,..C: 0.528,VREF: 1.245,↵
SP:145.544,FB:145.185,E: 0.358,P: 0.179,I:276.680,D:-11.000,O: 0.265,mA:27.861,..C: 0.527,VREF: 1.244,↵
SP:145.739,FB:146.518,E: 0.000,P: 0.000,I:276.586,D:-5.000,O: 0.270,mA:27.195,..C: 0.528,VREF: 1.243,↵
SP:145.837,FB:144.654,E: 1.183,P: 0.591,I:276.728,D: 9.808,O: 0.287,mA:26.973,..C: 0.528,VREF: 1.243,↵
SP:145.446,FB:144.389,E: 1.057,P: 0.528,I:276.855,D: 0.000,O: 0.276,mA:28.083,..C: 0.529,VREF: 1.245,↵
SP:145.446,FB:146.251,E: 0.000,P: 0.000,I:276.759,D:-9.000,O: 0.267,mA:27.750,..C: 0.528,VREF: 1.244,↵
SP:146.935,FB:145.717,E: 0.217,P: 0.108,I:276.785,D: 5.108,O: 0.282,mA:27.972,..C: 0.527,VREF: 1.243,↵
SP:145.837,FB:144.654,E: 1.183,P: 0.591,I:277.927,D: 4.829,O: 0.282,mA:27.972,..C: 0.528,VREF: 1.244,↵
SP:146.935,FB:144.919,E: 1.015,P: 0.507,I:277.049,D: 0.000,O: 0.276,mA:27.972,..C: 0.528,VREF: 1.245,↵
SP:145.739,FB:146.251,E: 0.000,P: 0.000,I:277.987,D:-7.000,O: 0.269,mA:28.083,..C: 0.528,VREF: 1.242,↵
SP:145.837,FB:145.451,E: 0.385,P: 0.192,I:277.034,D: 4.486,O: 0.281,mA:29.082,..C: 0.528,VREF: 1.245,↵
SP:145.446,FB:145.717,E: 0.000,P: 0.000,I:277.001,D:-3.000,O: 0.273,mA:27.417,..C: 0.530,VREF: 1.243,↵
SP:145.349,FB:145.451,E: 0.000,P: 0.000,I:277.989,D: 0.842,O: 0.277,mA:27.972,..C: 0.531,VREF: 1.244,↵
SP:145.642,FB:147.320,E:-1.000,P: 0.000,I:276.787,D:-7.000,O: 0.268,mA:27.861,..C: 0.527,VREF: 1.243,↵
SP:145.837,FB:144.654,E: 1.183,P: 0.591,I:277.929,D:14.309,O: 0.291,mA:27.195,..C: 0.527,VREF: 1.243,↵
SP:145.544,FB:145.451,E: 0.092,P: 0.046,I:277.940,D:-5.000,O: 0.271,mA:26.973,..C: 0.529,VREF: 1.242,↵
SP:145.251,FB:146.785,E:-1.000,P: 0.000,I:276.756,D:-8.000,O: 0.267,mA:27.750,..C: 0.528,VREF: 1.243,↵
SP:145.544,FB:145.451,E: 0.092,P: 0.046,I:276.767,D: 8.134,O: 0.285,mA:28.083,..C: 0.528,VREF: 1.244,↵
SP:145.251,FB:144.654,E: 0.597,P: 0.298,I:276.839,D: 2.521,O: 0.279,mA:26.973,..C: 0.528,VREF: 1.243,↵
SP:145.642,FB:146.785,E:-1.000,P: 0.000,I:276.702,D:-8.000,O: 0.267,mA:27.639,..C: 0.528,VREF: 1.2
```

Samples Recieved: 1947 Sample Rate: 97

Figure 6-C – Terminal window of Serial Oscilloscope
(Notice the sample rate of 97 S/s, compatible with the interrupt design value of 100 Hz)

6.3.2. PID controller response tests

The first test recorded was a gradual ramp up and down of the set point. Here it is possible to see that the controlled variable followed closely to the reference signal. (Notice that on the following graphs, each horizontal division corresponds to 200 ms, or 5 divisions per second.)

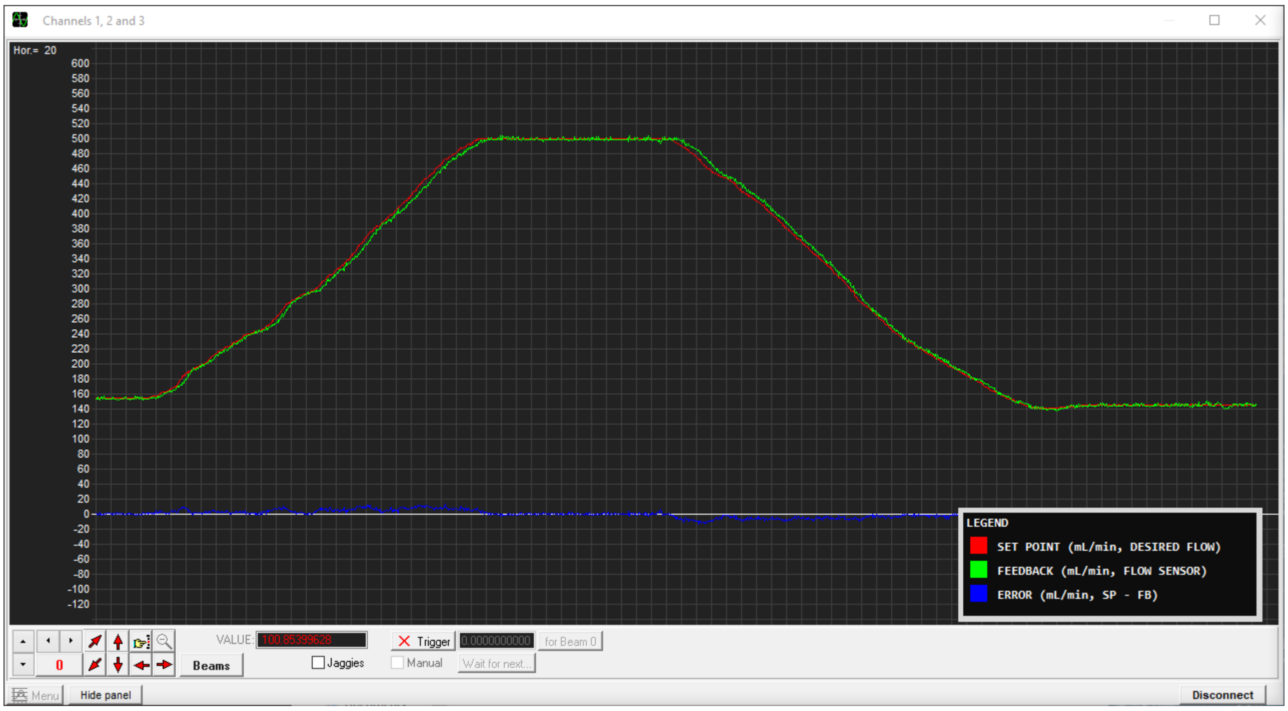


Figure 6-D – Ramp test: set point (red), feedback (green) and error (blue)

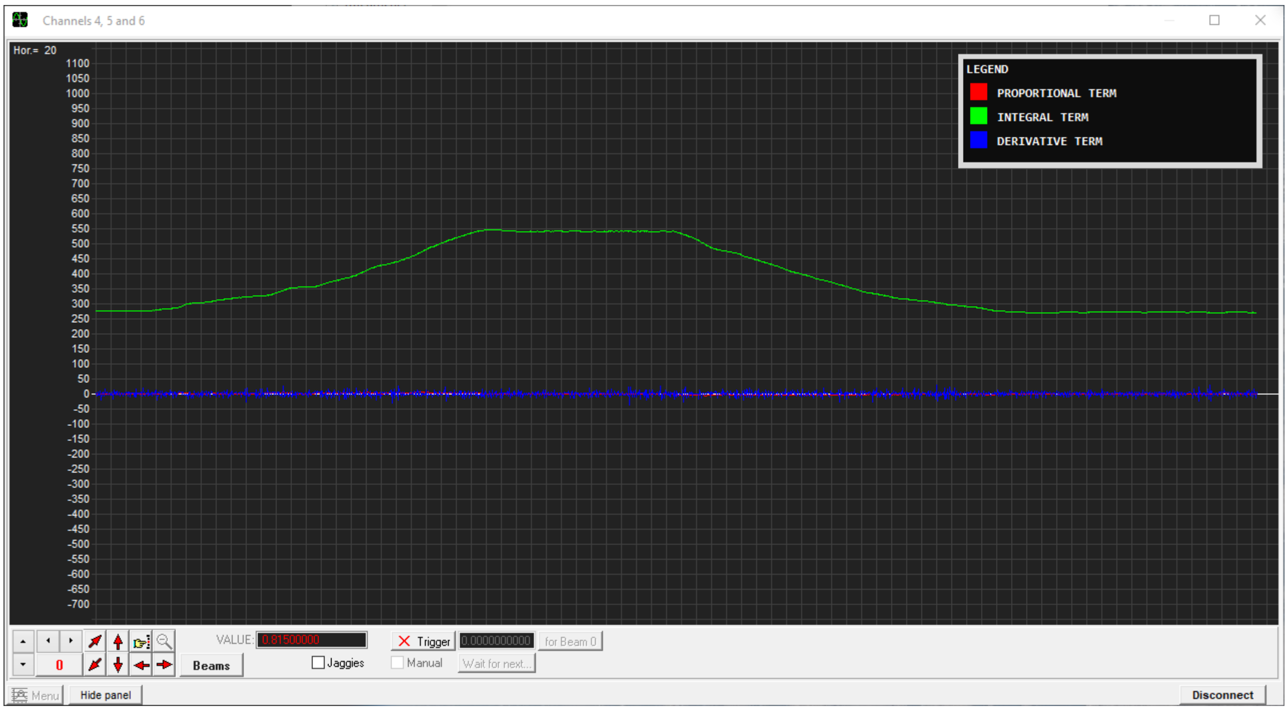


Figure 6-E – Ramp test: proportional (red), integral (green) and derivative (blue) terms

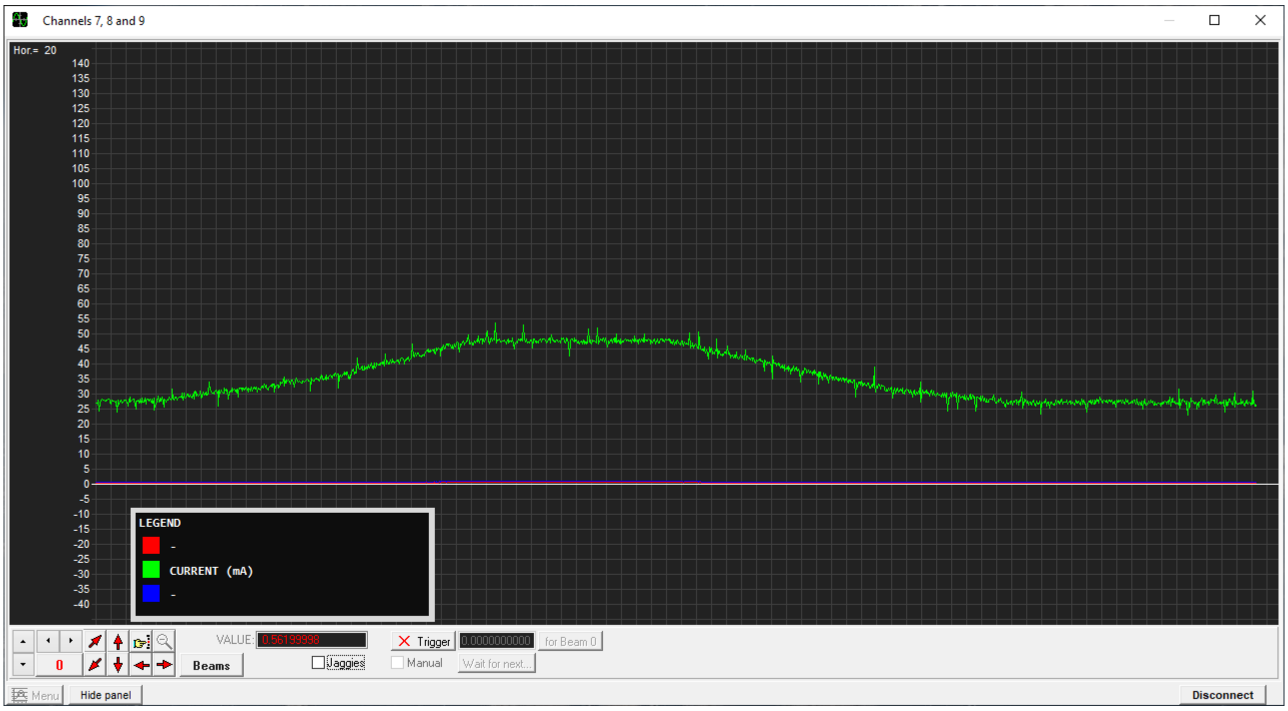


Figure 6-F – Ramp test: pump current (green)

Next, quicker set point changes were made:

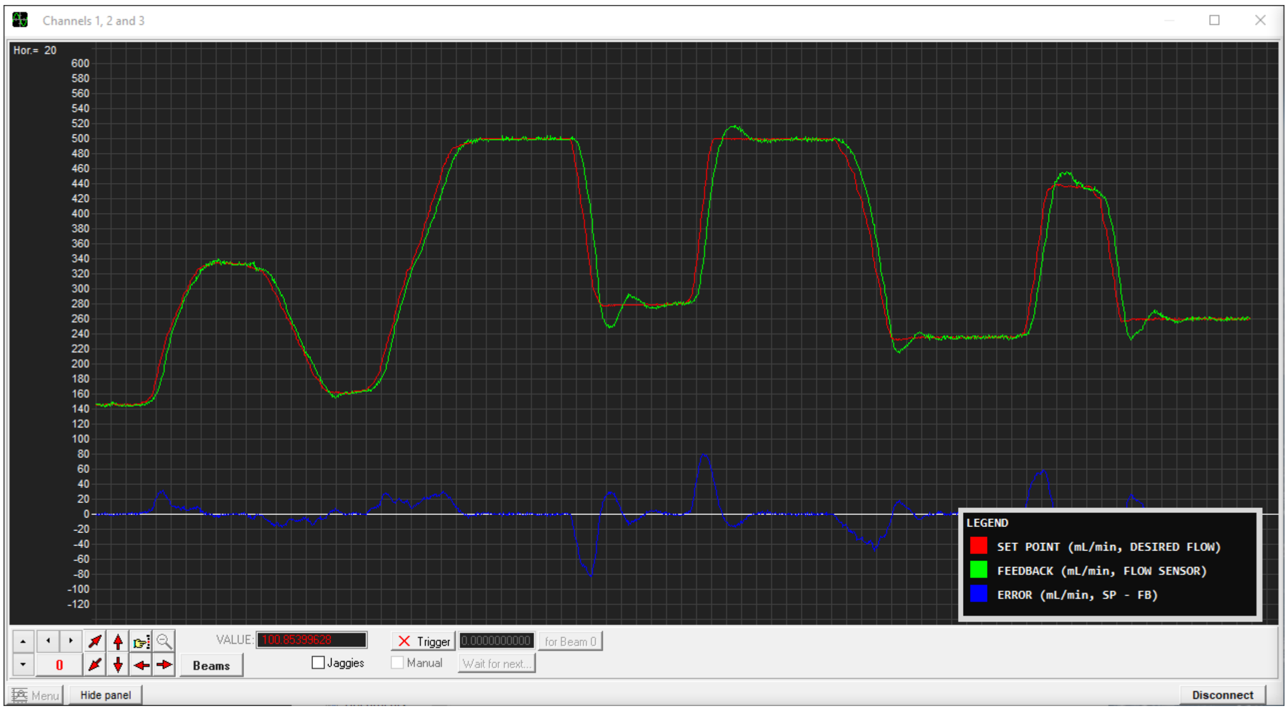


Figure 6-G – Varying set point test: set point (red), feedback (green) and error (blue)

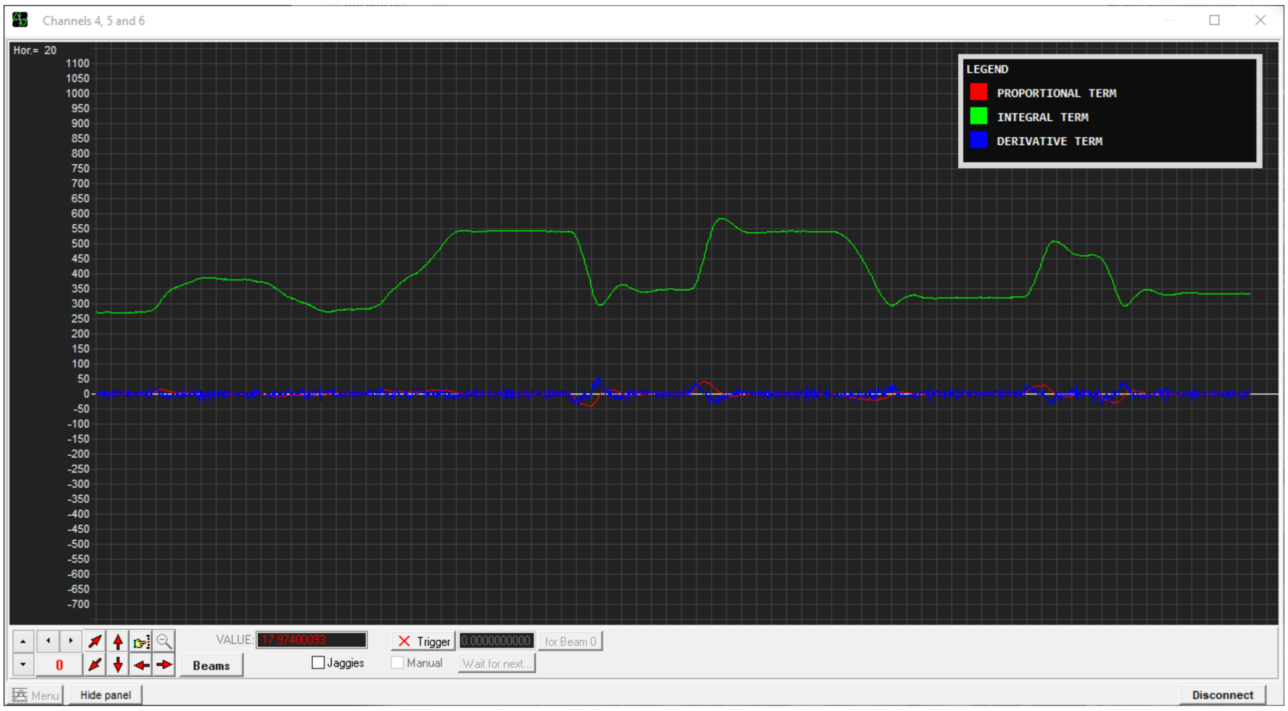


Figure 6-H – Varying set point test: proportional (red), integral (green) and derivative (blue) terms

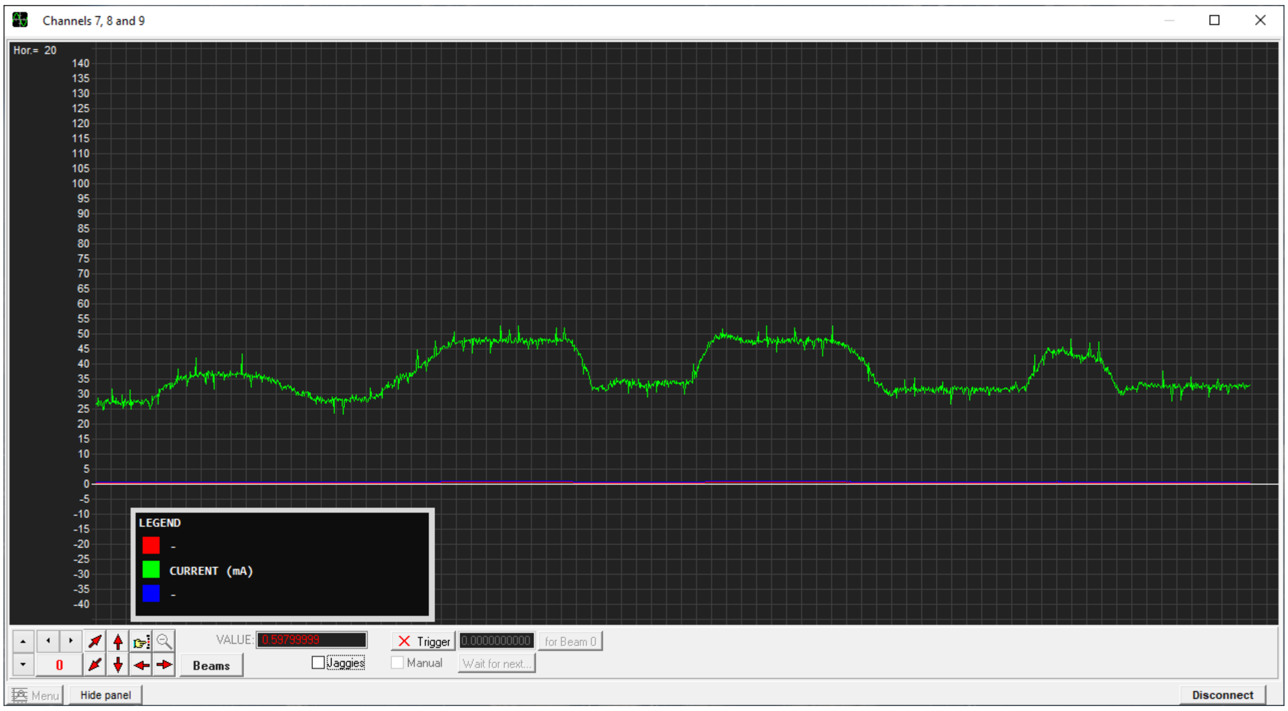


Figure 6-I – Varying set point test: pump current (green)

In the following test, the pump inlet was gradually blocked. Here, it is possible to see the controller in action, gradually increasing the integral term; pump current also increases gradually. As the pump is completely blocked (flow drops to zero), the integral term saturates at 1000 and the controller output reaches its maximum value (Controller output from 0 to 1000 corresponds to the PWM Duty Cycle from 0 to 100%). Then, the inlet is unblocked and the controller quickly recovers to the desired flow.

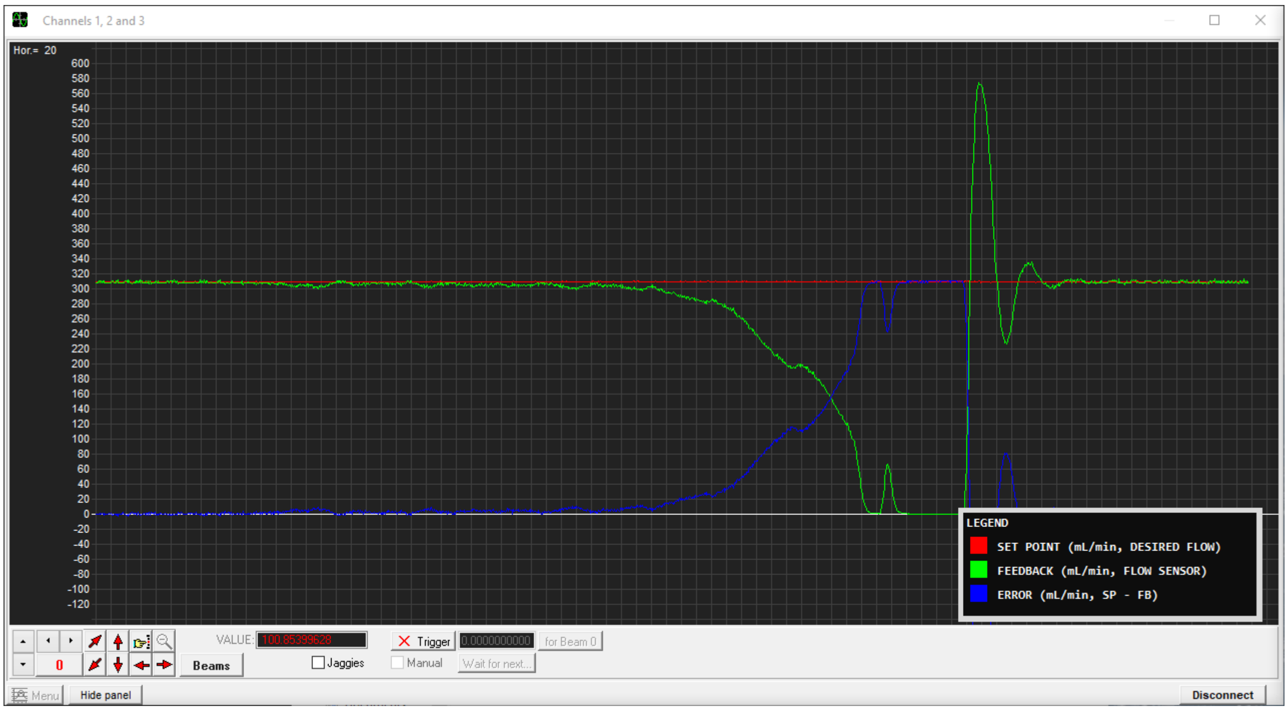


Figure 6-J – Progressive inlet block test: set point (red), feedback (green) and error (blue)

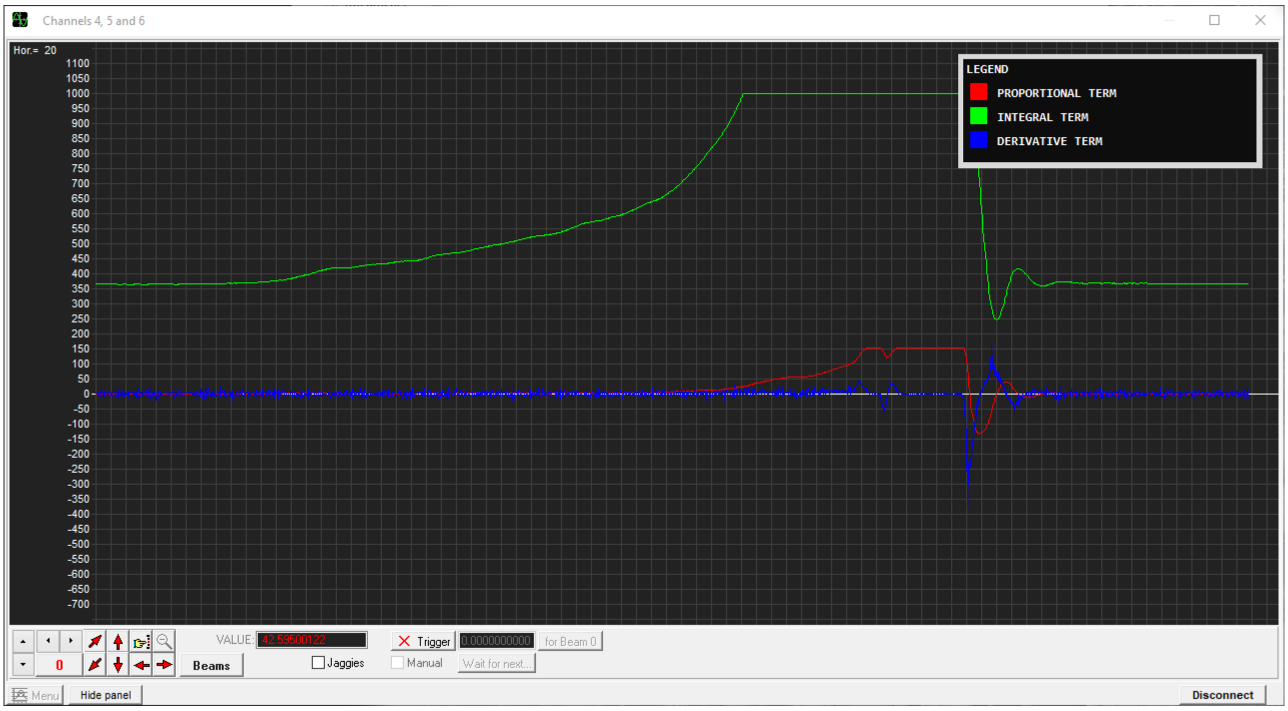


Figure 6-K – Progressive inlet block test: proportional (red), integral (green) and derivative (blue) terms

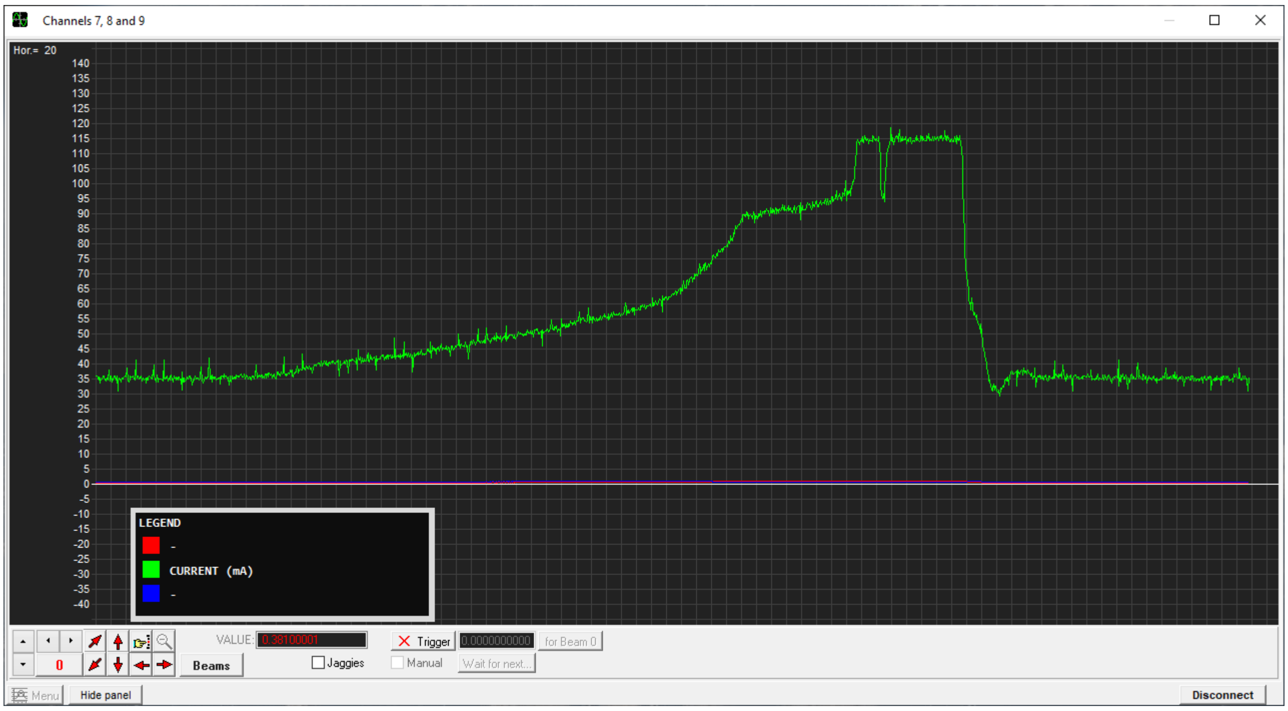


Figure 6-L – Progressive inlet block test: pump current (green)

The final test was to quickly block the pump inlet and then release it (2 times in this data recording).

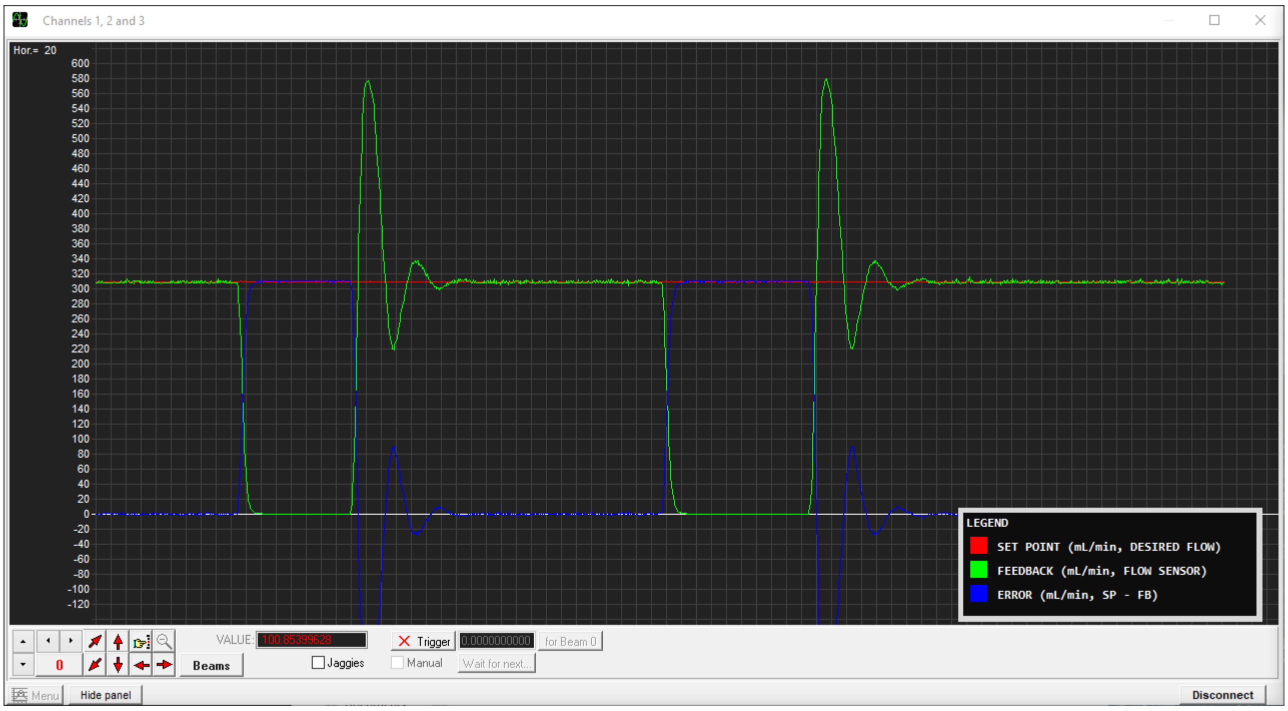


Figure 6-M – Sudden inlet block test: set point (red), feedback (green) and error (blue)

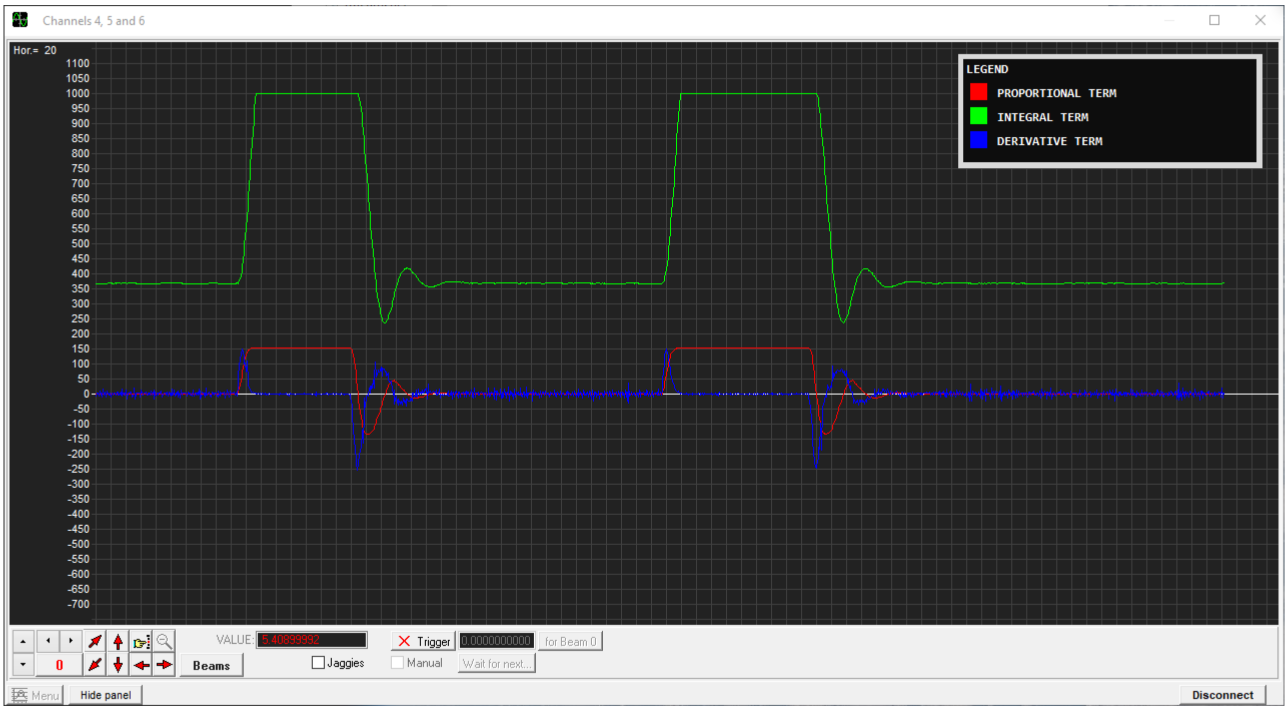


Figure 6-N – Sudden inlet block test: proportional (red), integral (green) and derivative (blue) terms

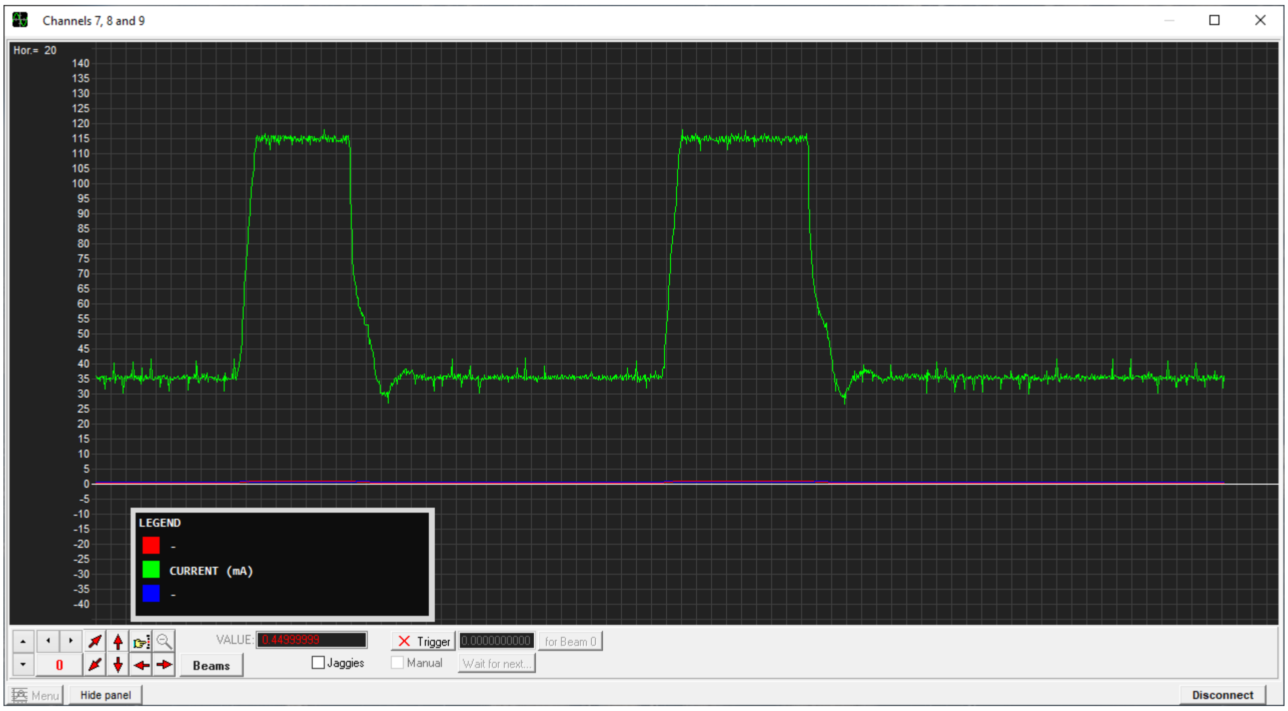


Figure 6-O – Sudden inlet block test: pump current (green)

6.4. System profiling

A system profiling of execution times for AUTO MODE was held by using an output pin and embracing code sections of interest by setting and resetting this output.

Code section	**Δt**	
ADC sampling (5 channels)	66 µs	187 µs
ADC conversions from counts to voltages (5 channels)	56 µs	
ADC voltages to physical units (5 channels)	65 µs	
PID Control and PWM update	80 µs	
UART data transmission	5.5 ms	
Total time used to execute AUTO MODE	**5.8 ms**	

From this data, it is clear that the UART data transmission is consuming most of the resources of the processor, but since the sampling period is 10 ms, there's still enough time to make the computations (about 4 ms). However, the UART is already on its maximum standard speed of 230400 bps.

In order to reduce power consumption, a sleep mode could be enabled after the ADC and control algorithms were executed; UART conditional enabling could also be used to reduce most of the processing usage.

7. FUTURE DEVELOPMENTS

7.1. Preparing for production

This prototype was quickly built upon a development board and some spare parts put together on a breadboard. Evidently, in order to get the project ready for production in a more viable manner, a custom hardware design would be necessary. This would be essential to reduce the final product cost and size. In addition, a more refined design allows for selecting an ideal MCU with appropriate I/O count, memory, clock frequency, etc.

7.2. Extending the idea

The more direct adaptation is to use other types of pumps and flow sensors. The board is powered from a 5V power supply, so probably any flow sensor with this rating could be used. However, the MCU is powered from a 3V power supply, so attenuation of the sensor output might be necessary if it goes above that value.

This prototype was thought to work with pumps but it would not be hard to adapt it to work with valves as well. If the pump is replaced by a proportional valve and a pressurized source, the rest of the system could control the output flow as well. The PID controller would probably need to be tuned to the new system, or perhaps a new control architecture would be necessary, since proportional valves are not as linear and easy to control as DC motors; however, the essence of the product is very similar. Another idea would be to change the flow sensor to pressure sensor, allowing the design of a noninvasive blood pressure monitor.

Power consumption could be improved by using the sleep modes while de processor is not doing useful calculations.

8. GRADING

8.1. Fulfillment of minimum requirements

As per the project requirements, the following items must be achieved:

a. Use a Cortex-M processor;
b. Have a button that causes an interrupt;
c. Use at least three peripherals;
d. Have a serial port output;
e. Implement an interesting algorithmic piece;
f. Implement a state machine.

Also, using a HAL is encouraged.

The microcontroller used was the STM32L152RB, which features a Cortex-M3 processor. There is a button that causes an external interrupt to later change the active mode of the device (a flag is set after a debounce algorithm is executed). Peripherals used include:

- GPIO external interrupt
- General purpose timer (non-blocking debounce algorithm)
- General purpose timer (non-blocking periodic sampling, triggers the ADC)
- ADC (sensors measurement)
- DMA (data transfer from the ADC to RAM)
- PWM timer (pump drive signal)
- UART (serial I/O)

The PID controller is the most interesting algorithmic piece in this project. Also, a state machine was used to change between different modes (denoted by the `op_mode` enumeration).

Therefore, all minimum requirements were fulfilled by the prototype.

The STM32CubeMX and STM32 HAL were widely used to configure peripherals, despite code being eventually modified to fit the needs of the project and finally cleaned to remove comments that make the code feel very heavy and disorienting. The final code is cleaner and easier to navigate.

8.2. Self-assessment

CRITERIA	GRADE	COMMENTS
Project meets minimum project goals	2.5	▪ Additional sensors and peripherals were used (internal temperature and reference voltage acquired; DMA used to get ADC data; non-blocking timed debounce algorithm for the button interrupt). ▪ A simple state machine was documented and implemented. ▪ A simple command line over serial port was implemented, but needs expansion of the commands and functionality.
Completeness of deliverables	3	▪ Code is readable on its own by the extensive use of comments. ▪ Report addresses all points required in the guideline document, and a bit more. ▪ The project was presented in live class, replacing the video.
Clear intentions and working code	3	▪ The system performs as described in the report in a manner that is professionally polished. The code shows how it works in a way that is easy for a maintainer to see.
Reusing code	3	▪ Versioning of reused code was included along with a license document that describes the license for the student's code and the reused code as well as shipping implications. Reader is confident they could rebuild the student's system
Originality and scope of goals	2.5	▪ Not a completely novel solution; OEM all-in-one modules controlled by serial communication are available for a multitude of applications. Although I've never seen one for flow control, it is possible that it is already available on the market. The project achieved all the minimum goals and, in my opinion, turned out to be a very comprehensive mechatronics project (although still in prototype phase).
Bonus: Power analysis, firmware update, or system profiling	1	▪ System profiling was carried out ▪ I had difficulty to make a proper power analysis due to my rudimentary tools and setup. I also did not investigate in depth how to make a proper firmware upgrade via UART or other means. I would need more time to further investigate these topics.
Bonus: Version control was used	3	▪ Git and GitHub were used to log the process of writing the code. Messages were indicative of the progress on each step.
TOTAL	**18**	A total of 15 + 6 are available

9. CONCLUSION

Working in this project was really challenging, but fun as well. The contents of the book and the course were extremely useful for completing this task. Also, the Discord forum was very helpful, as the interaction with instructor, mentors and students resulted in good ideas and valuable information for developing this project and others, as well.

I feel that completing this project was very rewarding; I certainly learned a lot of new concepts and improved upon prior knowledge. The prototype achieved all the requirements and even some bonus items, but there would be still a long road ahead if a product was to be manufactured based on the project.

Overall, the course was excellent and the final project is a good way to put in everything in practice.

Finally, I would like to thank Elecia White, Jeff Hurd, Daniel Fu, Erin Kennedy and Thomas Fuller for all support during this endeavor, as well as all students for their contributions during these months.

Thank you and best regards,

André A. M. Araújo.

10. REFERENCES

10.1. Hardware tools and components

32L152CDISCOVERY:
https://www.st.com/en/evaluation-tools/32l152cdiscovery.html

STM32L152RB:
https://www.st.com/en/microcontrollers-microprocessors/stm32l152rb.html

Omron D6F-P0010A1 gas flow sensor:
https://omronfs.omron.com/en_US/ecb/products/pdf/en-d6f_p.pdf

KNF UNMP 09KPDC-M micro pump:
https://knf.com/fileadmin/Local_files/USA/Downloads/OEM_Process_downloads/datasheets/Datasheet_UNMP_09_KNF_USA.pdf.pdf

Seeed Studio DSO Nano V2 – Mini oscilloscope (1 channel), no longer available but replaced by the DSO Nano V3:
https://www.seeedstudio.com/DSO-Nano-v2-p-681.html
https://www.seeedstudio.com/DSO-Nano-v3.html

BenF custom firmware for the DSO Nano V2:
https://seeeddoc.github.io/DSO_Nano_v2/res/BenF364_LIB353.zip

Sallen-Key Low-pass Filter Design Tool:
http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm

10.2. Software tools

STM32CubeIDE Integrated Development Environment for STM32:
https://www.st.com/en/development-tools/stm32cubeide.html

DipTrace – EDA software
https://diptrace.com/

CoolTerm – The best terminal application I have found, available from Roger Meier's website:
https://freeware.the-meiers.org/

Serial Oscilloscope – Application that captures data from the serial port and plots on the screen:
https://x-io.co.uk/serial-oscilloscope/

diagrams.net (formerly draw.io) – Online tool for creating block diagrams:
https://app.diagrams.net/

BSD 3-Clause license, used by STM32 HAL:
https://opensource.org/licenses/BSD-3-Clause

10.3. Control

Wikipedia – PID Controller:
https://en.wikipedia.org/wiki/PID_controller

PID Without a PhD (Tim Wescott) – Interesting article with basic information about PID control:
http://wescottdesign.com/articles/pid/pidWithoutAPhd.pdf

PID Controller Implementation in Software (Phil's Lab) – Interesting video about a PID controller using an STM32:
https://youtu.be/zOByx3Izf5U

Digital Control Engineering – Analysis and Design (Fadali, Visioli):
https://www.elsevier.com/books/digital-control-engineering/fadali/978-0-12-814433-6