
**Politechnika Śląska**
**Wydział Automatyki, Elektroniki i Informatyki**

---

Sterowanie Procesami

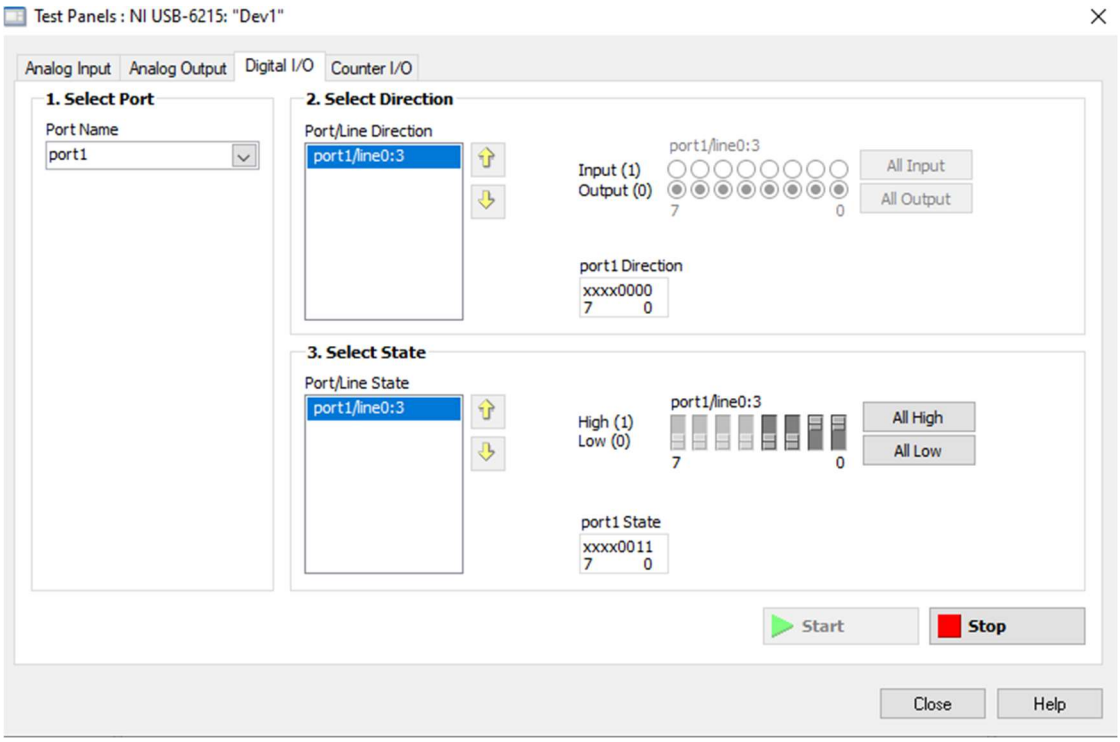
# Monitorowanie i sterowanie układu termicznego – projekt nr 14

**Autorzy: Krystian Kulik, Łukasz Woźniak, Damian Pilny**
**Rok I, semestr 1 stopnia II**

Gliwice, 28.03.2020r

# 1. Tydzień 23.03.2020 – 27.03.2020

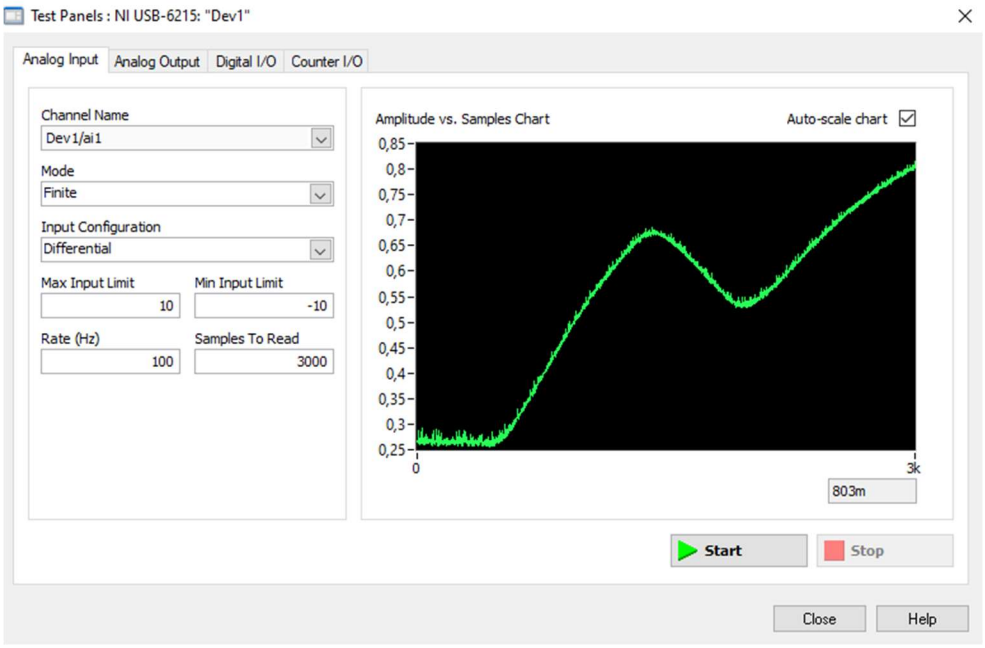
Ten tydzień został poświęcony na zaznajomienie się ze stanowiskiem badawczym, jak i przetestowanie poprawności połączeń za pomocą programu NI MAX. Poniżej ukazany jest widok panelu testowego.




*Rysunek 1: Wejścia/Wyjścia cyfrowe NI USB-6215*

Dwa bity ustawione na stan wysoki na Rysunku 1.1 sterują obiektem. Port1/line0 odpowiada za uruchomienie wentylatora, a Port1/line1 odpowiada za uruchomienie grzałki.

Rysunek 1.2 przedstawia zapis 3000 próbek z częstotliwością 100Hz. Kanał Dev1/a1 odpowiada za pomiar temperatury, aby uzyskać wskazania w stopniach Celsjusza należy przemnożyć te wartości razy 100. Bit odpowiadający za wentylację był w stanie wysokim podczas testu. W fazie narastania bit odpowiadający za załączenie grzałki był w stanie wysokim, w przeciwnym przypadku był ustawiony na stan niski.

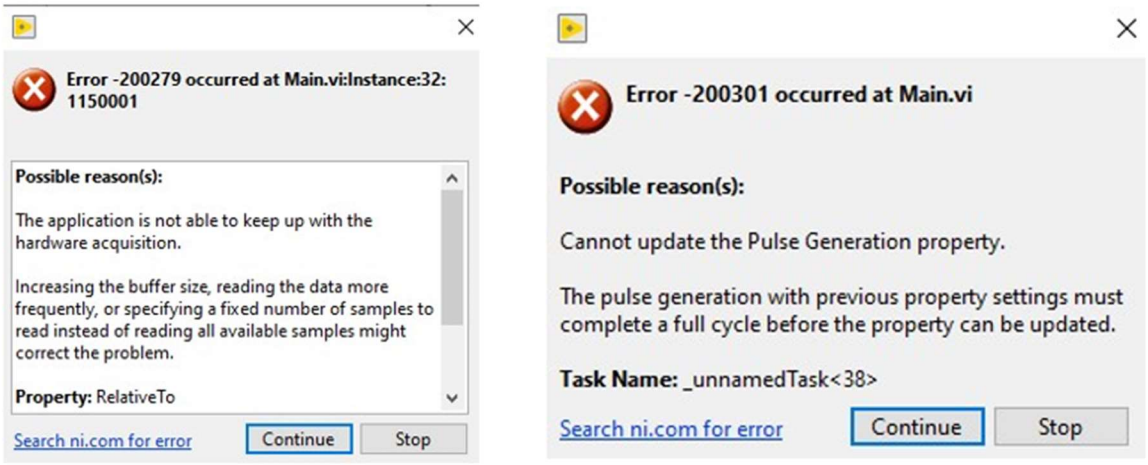



*Rysunek 2: Wynik sterowania wejściami cyfrowymi*

Po zaznajomieniu się z sygnałami w aplikacji NI MAX dokonaliśmy testów dostarczonego oprogramowania. Rysunek 1.3 przedstawia regulację temperatury wyjściowej na poziomie 50 stopni Celsjusza. Aplikacja oferuje również sterowanie ręczne wentylatorami, zmianę typu zakłócenia, oraz zmianę nastaw regulatora PID.


*Rysunek 3: Panel programu sterowania LabView*

Aplikacja od czasu do czasu przejawia tendencję do wyświetlania błędów (Rys. 1.4 i 1.5) związanymi ze zbyt szybką akwizycją sygnału, nie jest zaimplementowany mechanizm kontroli zakończenia zadania przed utworzeniem nowego.



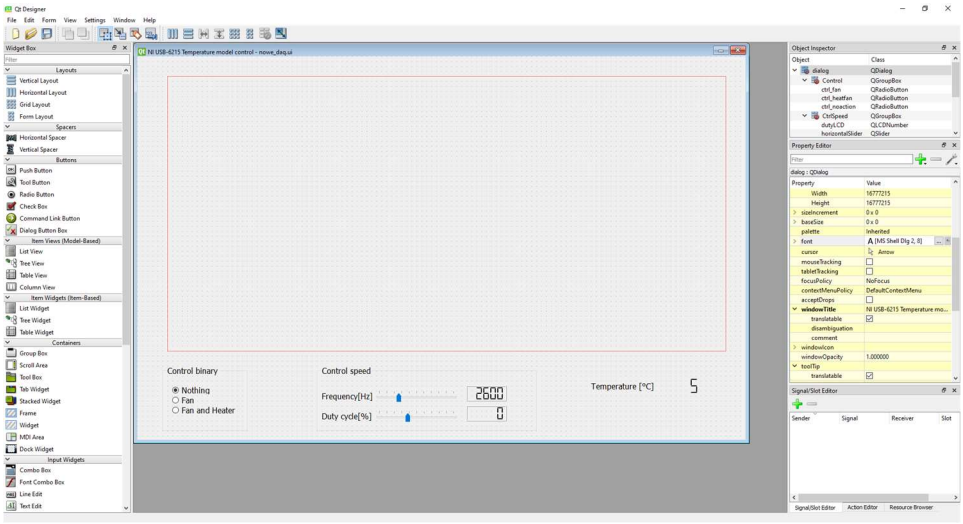

*Rysunek 4: Komunikaty błędów aplikacji*

## 2. Tydzień 30.03.2020 – 03.04.2020

W tym tygodniu postanowiliśmy przetestować funkcjonalność biblioteki w języku Python nidaqmx, która jest dedykowana do obsługi kart DAQ firmy National Instruments. Dokumentację tej biblioteki można znaleźć pod adresem: https://nidaqmx-python.readthedocs.io/en/latest/.

Owocem pracy było wykonanie aplikacji okienkowej umożliwiającej odczyt temperatury wylotowej, oraz sterowanie grzałką i wentylatorem. Na ten tydzień zostało wykonane tylko i wyłącznie wystawienie sygnałów binarnych na wyjścia (Włącz/Wyłącz), nie można w precyzyjny sposób zmieniać wypełnienia oraz częstotliwości sygnału. W przyszłym tygodniu zostanie zaimplementowana funkcjonalność umożliwiająca wystawienia sygnału jako Clock, czyli taktowanego zegarem.

Wygląd GUI i zarazem funkcje jego obiektów zostały wygenerowane za pomocą aplikacji Qt Designer. Aplikacja ta wygenerowała obiekty pewnych klas o pewnych atrybutach i funkcjach, do których możemy się odnosić w skrypcie.

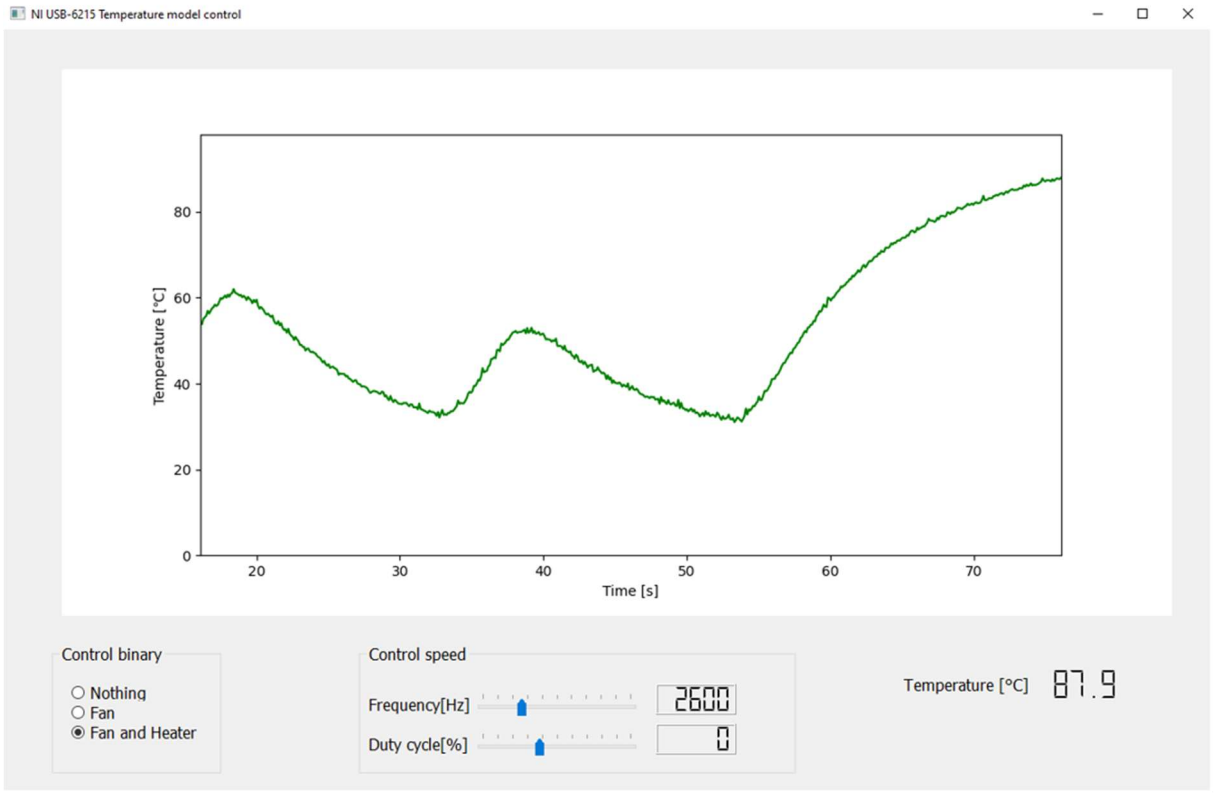



*Rysunek 5: Program Qt Designer*

Kod programu jest odpowiedzialny za utworzenie obiektu typu GUI, stworzenie zadań dla karty DAQ, oraz integrację GUI ze skryptem. Struktura wykonywania zadań jest określona jako Master-Slave. Zadaniem typu master jest odczyt temperatury wylotowej 10 razy na sekundę, a typu slave jest zmiana sterowania. Zadanie akwizycji danych jest wykonywane z częstotliwością 10 razy na sekundę i pobierana jest tylko jedna próbka. Zadanie akwizycji danych ma wbudowaną funkcję typu callback, gdzie można ustawić po ilu próbkach ma być wykonywana jakaś funkcja (ustawiono ten parametr na jedną próbkę). Funkcji callback

wykonywany jest odczyt, ale i wykonane jest nadpisanie stanu wyjść, czyli drugie zadanie. Czyli czas reakcji na zmianę ustawień GUI jest równy 0,1s.

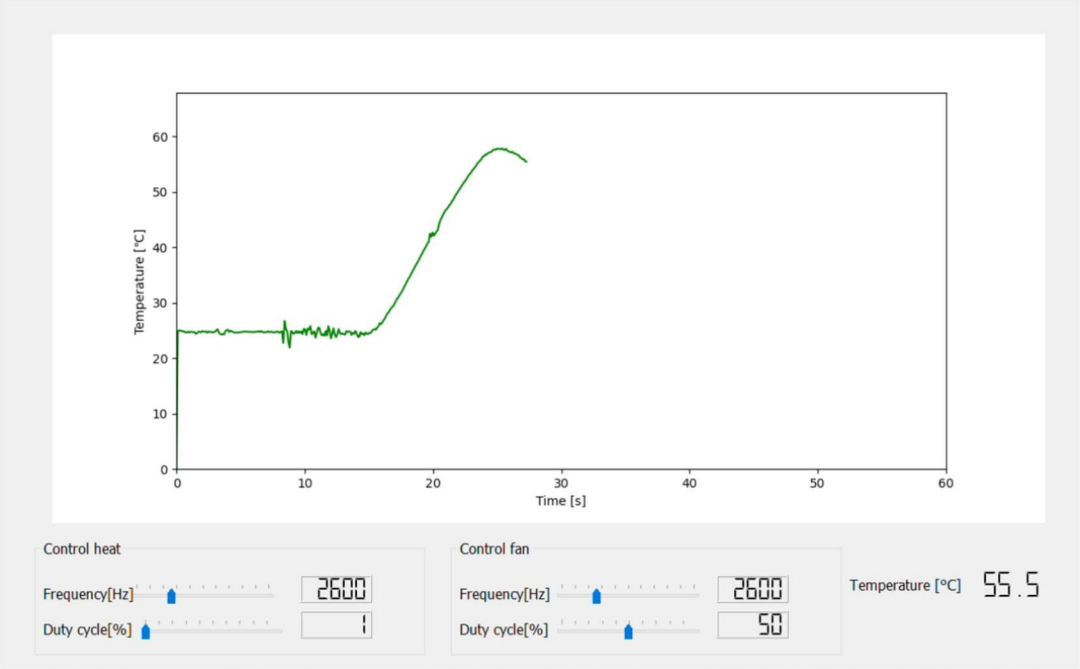
Na rysunku nr 6 poniżej zaprezentowana została zmiana temperatury wyjściowej na odpowiedni ciąg sterowań w zakładce Control binary zgodnie z opisem. Działanie aplikacji zostało ukazane w załączonym filmie.




*Rysunek 6: Wygląd GUI programu*

## 3. Tydzień 06.04.2020 – 14.04.2020

Program z poprzedniego tygodnia został wzbogacony o możliwość sterowania wyjściami karty w trybie wykorzystującym zegar. Panel aplikacji został wzbogacony o suwaki i o zintegrowane z nimi wyświetlacze informujące o aktualnej częstotliwości sygnału oraz jego wypełnieniu. Działanie ten funkcjonalności jest ukazane na rys. 7 i w załączonym filmie.




*Rysunek 7: Panel aplikacji używającej wyjść zegarowych*

Implementacja tego rozwiązania znacząco się różni od sterowania binarnego, ponieważ używana biblioteka nie umożliwia modyfikacji wartości częstotliwości i wypełnienia dla istniejącego zadania. Zadanie utworzone do sterowania w trybie binarnym umożliwiało wystawienie sygnału wysokiego, bądź niskiego bez resetu zadania. Aby obsłużyć nową funkcjonalność należało zaimplementować mechanizm kontroli polegający na zakończeniu zadania i utworzeniu nowego w przypadku zmiany wypełnienia, bądź częstotliwości.

W tym celu wykorzystano ponownie strukturę master-slaves i wcześniej napisaną funkcję callback wykonywaną co 0,1s. Zadaniem nadrzędnym jest pomiar temperatury, a podrzędnymi zadanie sterujące wentylatorem i grzałką. Użycie zmiennych globalnych w funkcjach do określenia zadań umożliwia podtrzymanie ustawionej konfiguracji na zewnątrz bloku funkcyjnego, w innym wypadku zadanie zostałoby wykonane tylko raz.

# 4. Tydzień 14.04.2020 – 21.04.2020

W tym tygodniu stanowisko badawcze zostało przekazane pomiędzy członkami zespołu badawczego. Kolejny tydzień, a więc pierwszy tydzień w nowych rękach został poświęcony na zaznajomienie się ze stanowiskiem badawczym, jak również na skonfigurowanie odpowiednio środowiska do dalszego rozwoju. Aby była możliwa dalsza analiza, jak i implementacja rozwiązań technicznych wymagana jest konfiguracja środowiska Python, Qt Designer oraz karty DAQ firmy National Instruments.

Jako pierwszą konfigurację wybrane zostało środowisko Python. Aby była możliwa praca nad projektem wymagane są odpowiednie biblioteki. Instalację warto przeprowadzić za pomocą rozszerzenia języka Python PIP. Pierwsze zainstalowane biblioteki dotyczyły obsługi tablic jak również umożliwiały możliwość graficznej reprezentacji danych

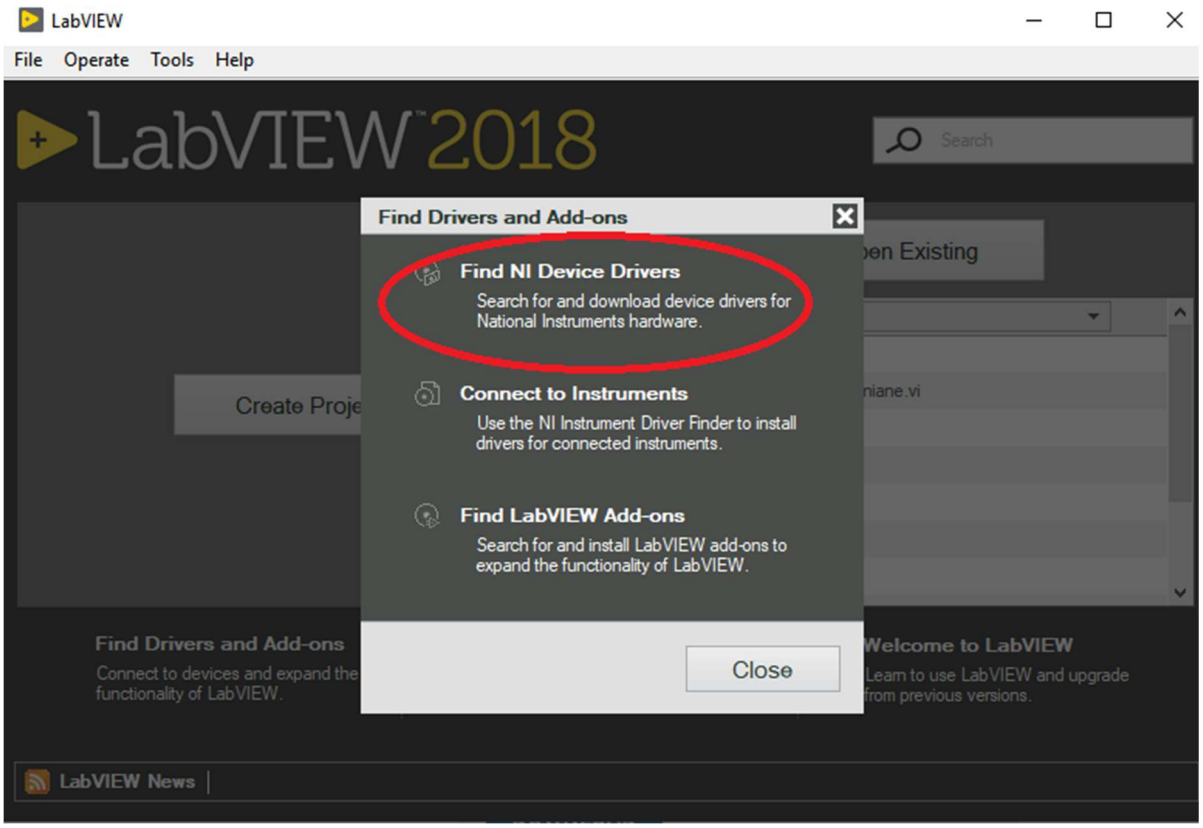
| |
|---|
| *pip install numpy* |
| *pip install matplotlib* |

Następne biblioteki dotyczyły już bezpośrednio pracy z platformą National Instruments. Zainstalowana biblioteka PyQt5 służy do tworzenia GUI, wraz z biblioteką dostępny jest program Qt Designer, który znajduje się bezpośrednio w ścieżce zainstalowanej biblioteki. Natomiast biblioteka nidaqmx wspomaga praco z urządzeniami firmy National Instruments. Biblioteka ta natomiast pracuję tylko i wyłącznie pod systemem operacyjnym Windows.

| |
|---|
| *pip install PyQt5* |
| *pip install PyQt5-tools* |
| *pip install nidaqmx* |

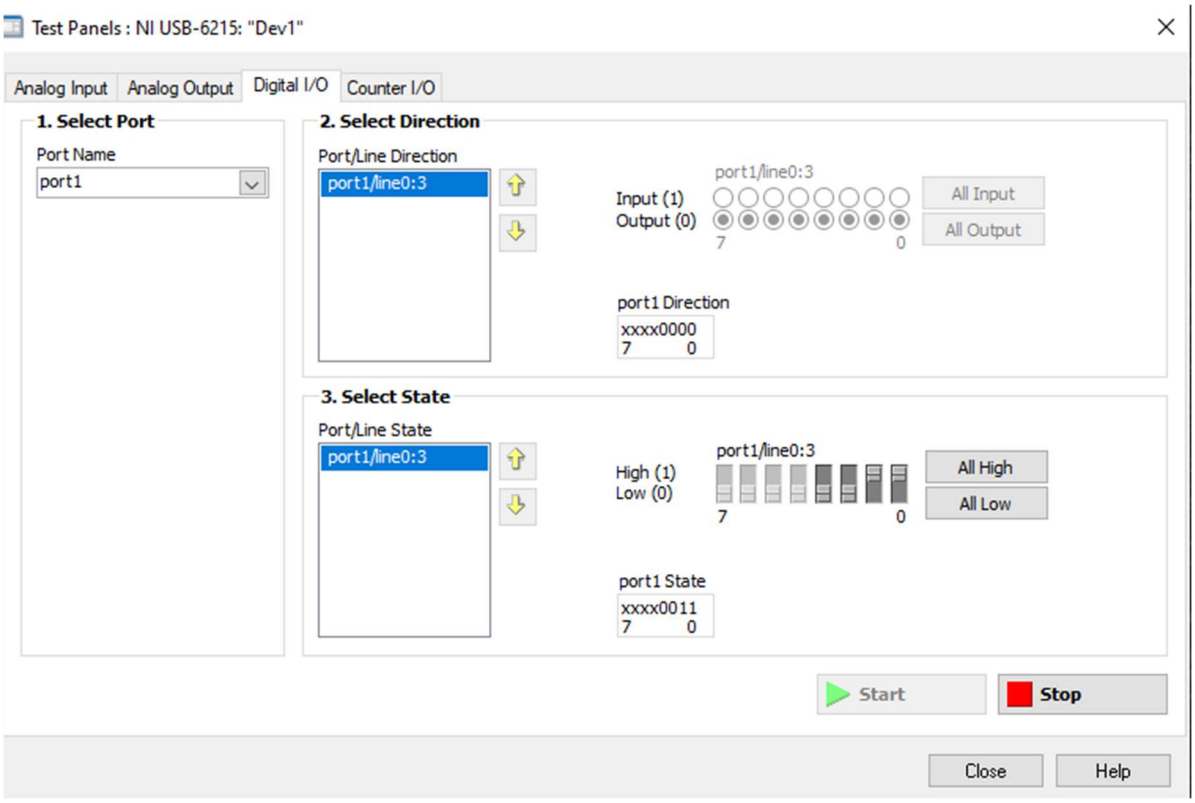
Ostatnią rzeczą jaką należało skonfigurować była karta National Instruments. Aby była możliwość pracy z rozszerzeniem niezbędna była instalacja sterowników urządzenia. Najlepszą drogą do instalacji sterowników okazał się program LabView, który pozwala na

instalację bezpośrednio z panelu powitalnego. Na Rysunku 8 został zilustrowany sposób instalacji sterowników.

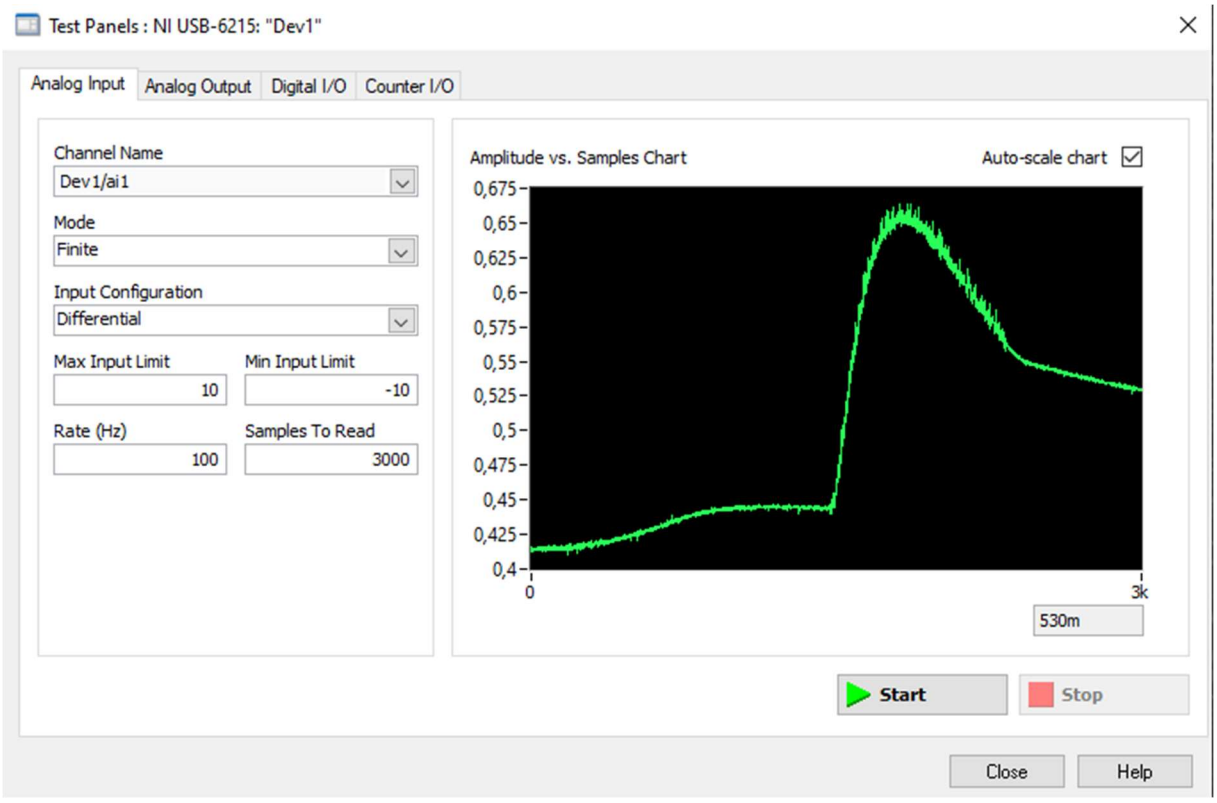



*Rysunek 8. Aktualizacja sterowników z programu LabView*

Po kliknięciu okna instalacyjnego sterowników, otwarte zostało okno wyszukiwarki z możliwymi sterownikami. W naszym przypadku interesuje nas sterownik NI-DAQmx. Po zainstalowaniu sterownika, komputer należało zrestartować. Po uruchomieniu komputera stanowisko zostało przetestowane za pomocą programi NI MAX. Rysunek 9 oraz Rysunek 10 obrazują pomyślne testy. Na Rysunku 9 do poprawnej pracy stanowiska potrzebne jest uruchomienie dwóch bitów. Podobnie jak to zostało opisane na Rysunku 1.

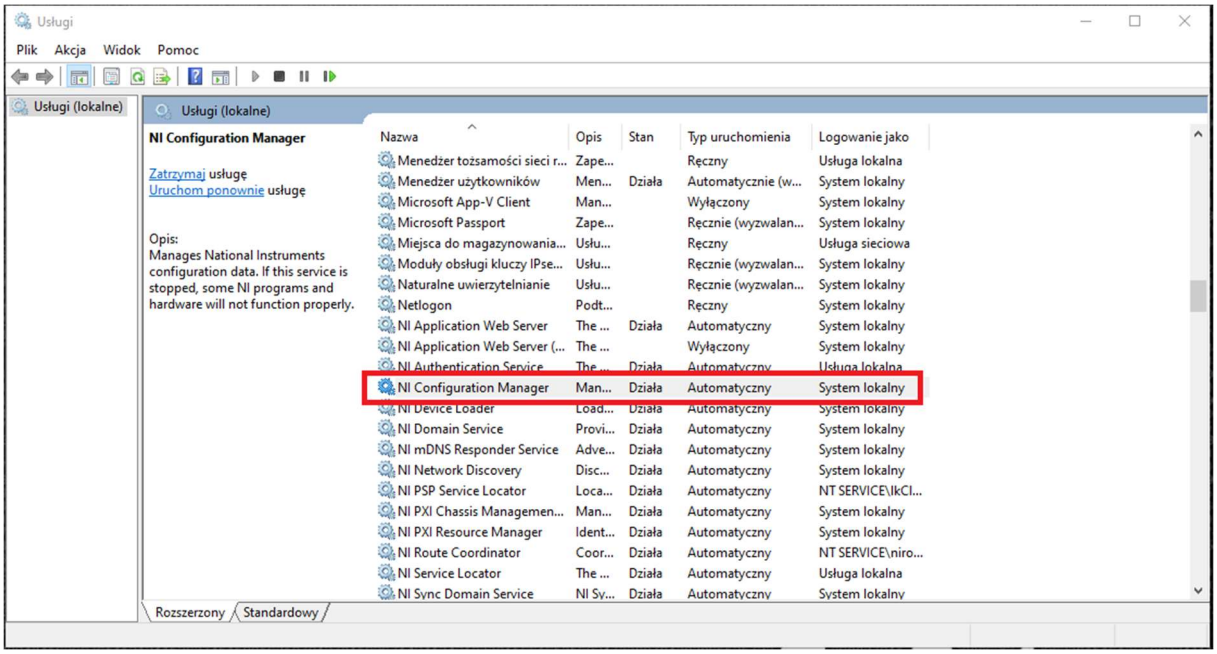



*Rysunek 9. Test portów cyfrowych.*



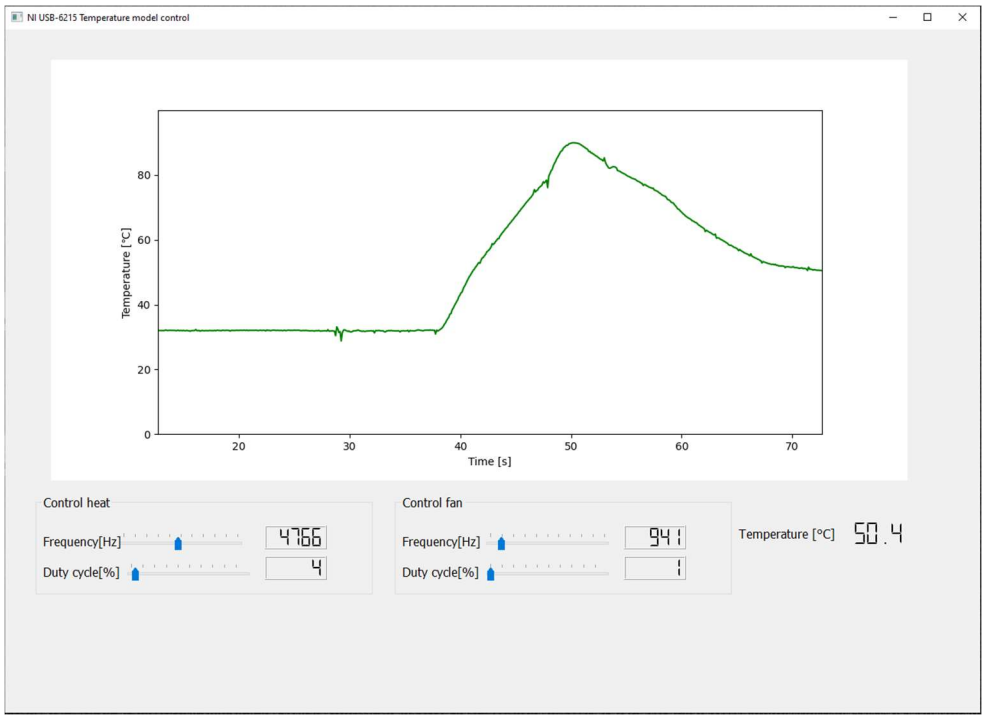

*Rysunek 10. Test wejść analogowych wraz z odczytaną temperaturą na wylocie rury.*

Po każdorazowym wyciągnięciu kabla USB z komputera, do poprawnej pracy urządzenia należy uruchomić ponownie usługę NI Configuration Manager dostępną w aplikacji Usługi środowiska Windows. Obrazuje to Rysunek 11




*Rysunek 11. Aktualizacja sterownika NI Configuration Manager*

Tak przygotowane sterowniki wraz ze środowiskiem są gotowe do dalszego rozwoju. Rysunek 12 przedstawia uruchomiony program z tygodnia poprzedniego.



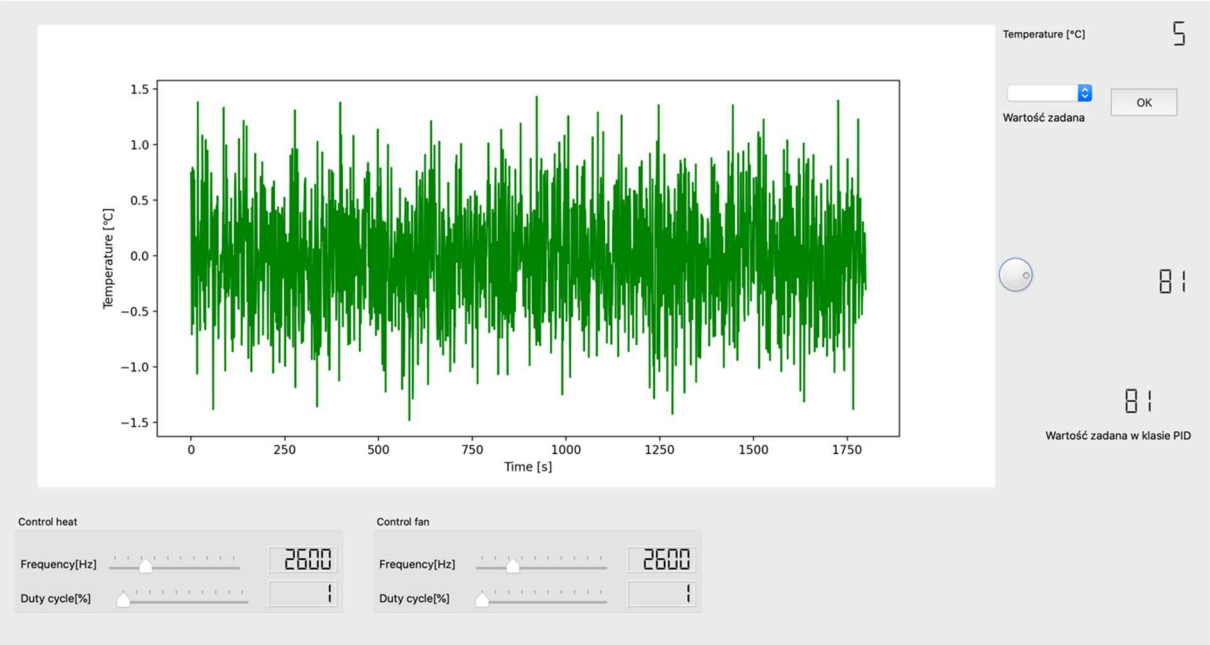

*Rysunek 12. Panel aplikacji.*

# 5. Tydzień 21.04.2020 – 28.04.2020

W tym tygodniu odbył się test funkcjonalności stworzonego oprogramowania oraz wstępna implementacja regulatora PID. W tym tygodniu implementacja nie miała miejsca jeszcze na rzeczywistym obiekcie, jednak stworzona została struktura programu umożliwiającą łatwą implementację regulacji na obiekcie rzeczywistym. Głównym zadaniem na ten tydzień była integracja między uprzednio stworzonymi klasami. Na Rysunku 13 została przedstawiona modyfikacja GUI.


*Rysunek 13. Zmodyfikowane GUI*

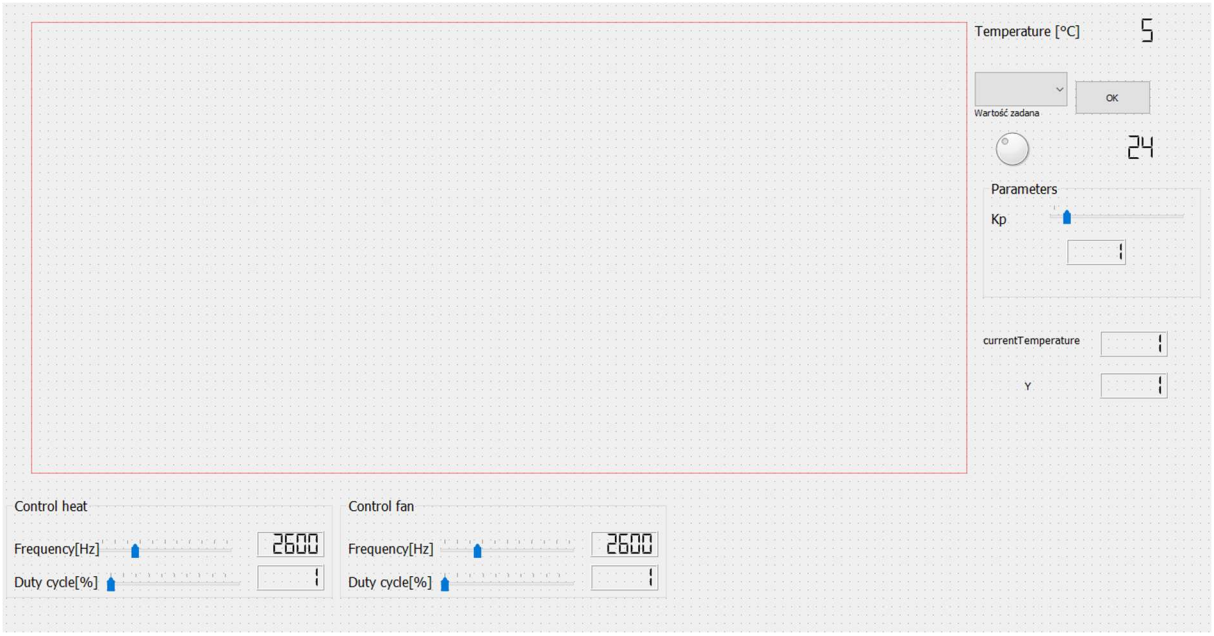
GUI umożliwia nastawienie parametru wartości zadanej w różny sposób. W następnej fazie rozwoju stworzona zostanie możliwość nastaw parametrów regulatora PID. Stworzone nastawy w tym tygodniu miały na celu zbadanie możliwości przesyłu danych między dwiema oddzielnymi klasami programu sterującego.




*Rysunek 14. Test przesyłu danych*

## 6. Tydzień 28.04.2020 – 05.05.2020

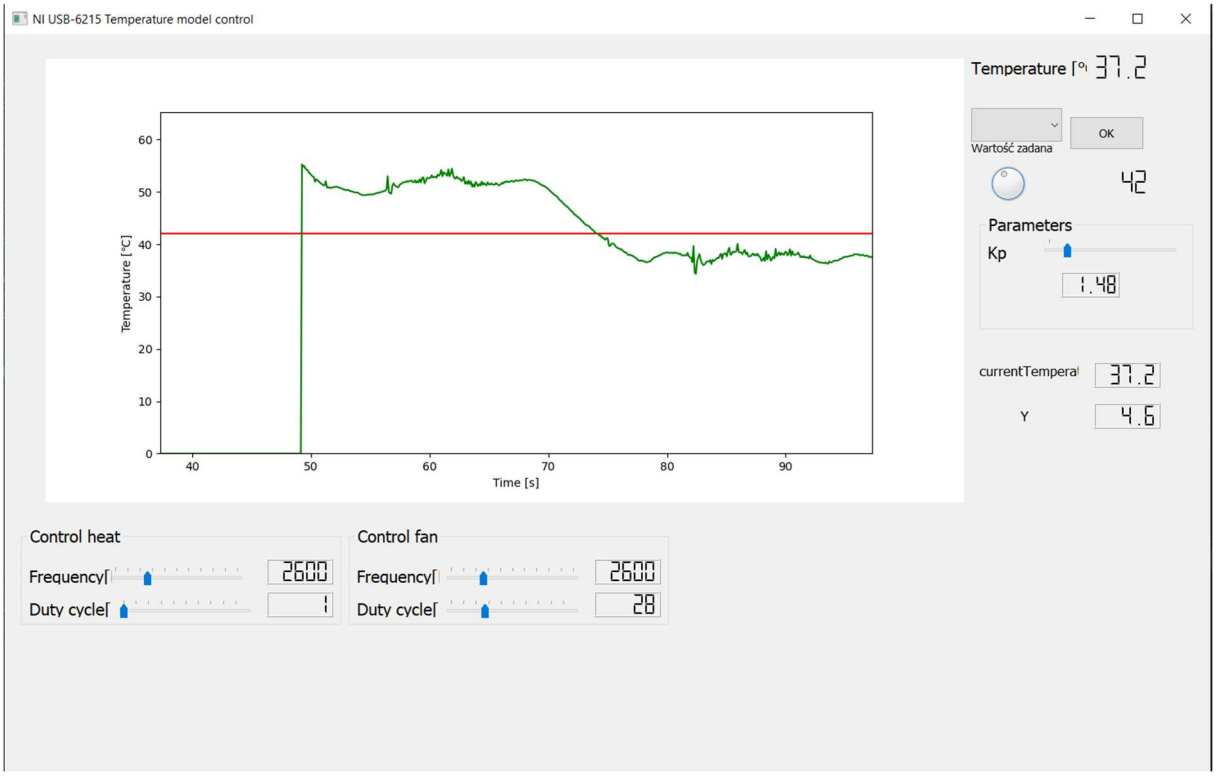
W tym tygodniu nastąpiło przetestowanie stworzonego oprogramowania na rzeczywistym obiekcie. Bardzo szybko okazało się, że symulacje programu bez wykorzystania sprzętu rzeczywistego odbiegają znacząco od założonych. Program wymagał zmiany, a wewnętrzna komunikacja po uruchomieniu Tasków biblioteki nidaqmx przestaję odgrywać znaczącą rolę w całej aplikacji. W GUI stworzona została możliwość wyboru parametru Kp.




*Rysunek 15. Wygląd GUI*

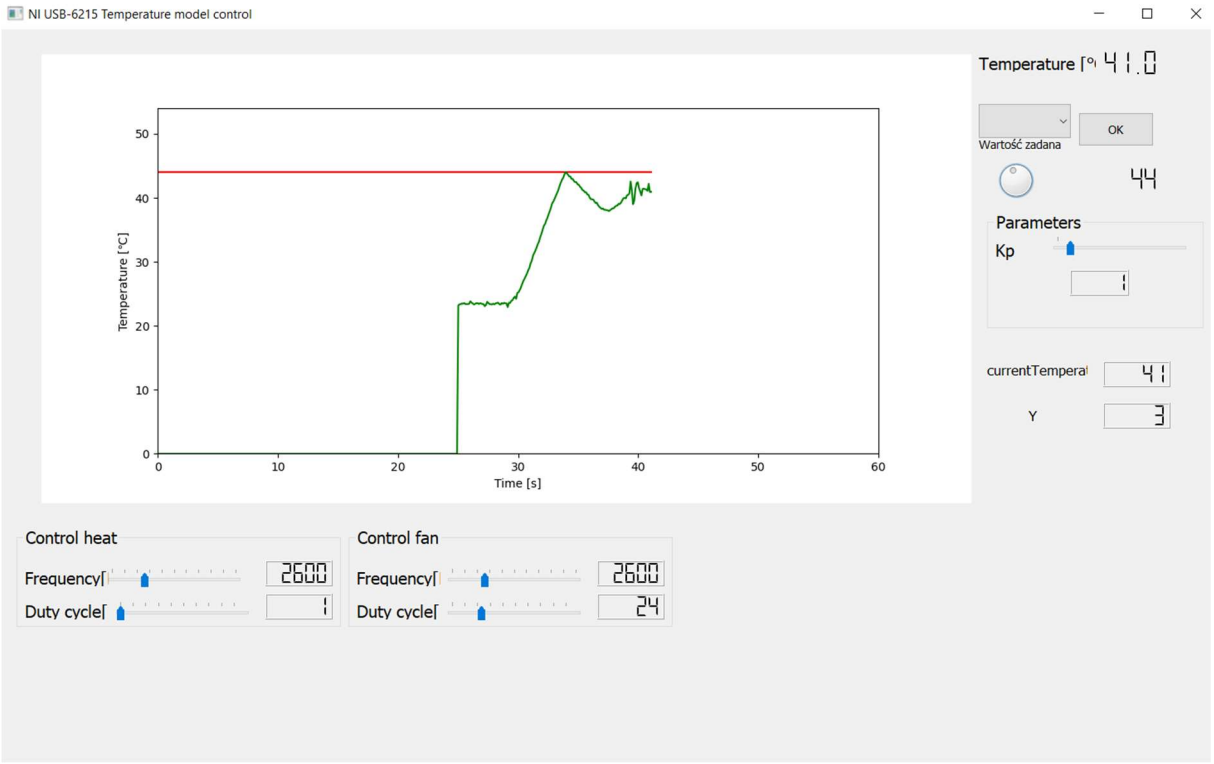
Regulator PID realizuje równanie:

$$Y(s) = Kp + Ki\frac{1}{s} + Kds$$




*Rysunek 16. Realizacja regulatora typu P*

W minionym tygodniu, sporo czasu upłynęło na integracje algorytmu z rzeczywistym obiektem, co ostatecznie skutkowało realizacji regulatora typu P. Ponadto ciekawym problemem okazał się fakt osiągania wartości regulowanej wartości zadanej. Gdy sygnał ten został osiągnięty, zawieszany zostawał cały program. Ponowna realizacja programu mogła być kontynuowana dopiero po zejściu temperatury poniżej wartości zadanej. Problem został rozwiązany poprzez ustawienie dolnej wartości wielkości regulującej na wartość minimalną.




*Rysunek 9. Błąd wyświetlania wartości temperaturowej*

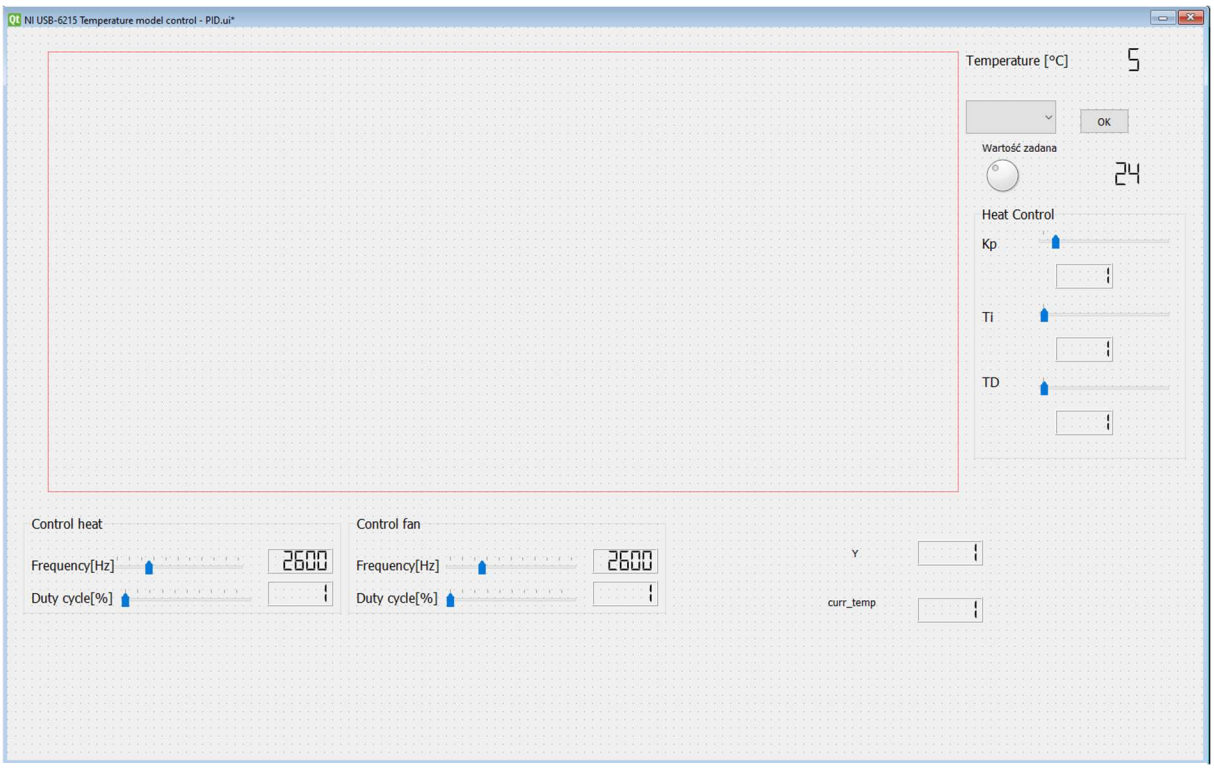
## 7. Tydzień 01.06.2020 – 08.06.2020

W tym tygodniu został rozbudowany regulator PID o wartości członów I oraz D. Regulator prawidłowo realizuje swoje równanie:

$$Y(s) = Kp + Ki\frac{1}{s} + Kds$$

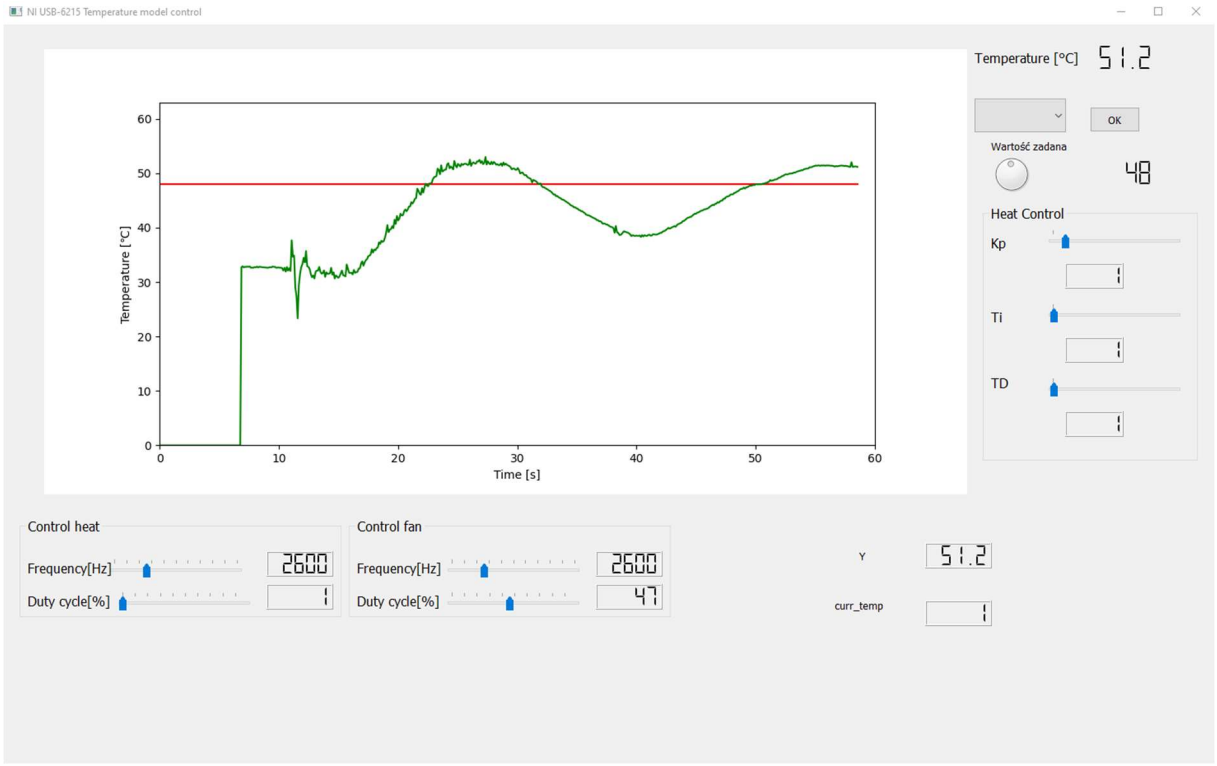
Ponadto pojawił się problem z opóźnieniem, które pojawiło się przy wyświetlanej wartości regulacji. Regulator do prawidłowego działanie wymaga znajomości aktualnej

temperatury. Aktualna temperatura (curr_temp) jest odczytywana bezpośrednio z wykresu. Jednak niefortunna lokalizacja wyświetlania, a także złe buforowanie otrzymywanych próbek sprawił, że w całym układzie pojawiało się około 10s opóźnienie. Jednak analiza programu pozwoliła zdiagnozować usterkę. A określenie lepszego warunku pozwoliła zminimalizować otrzymane opóźnienie.



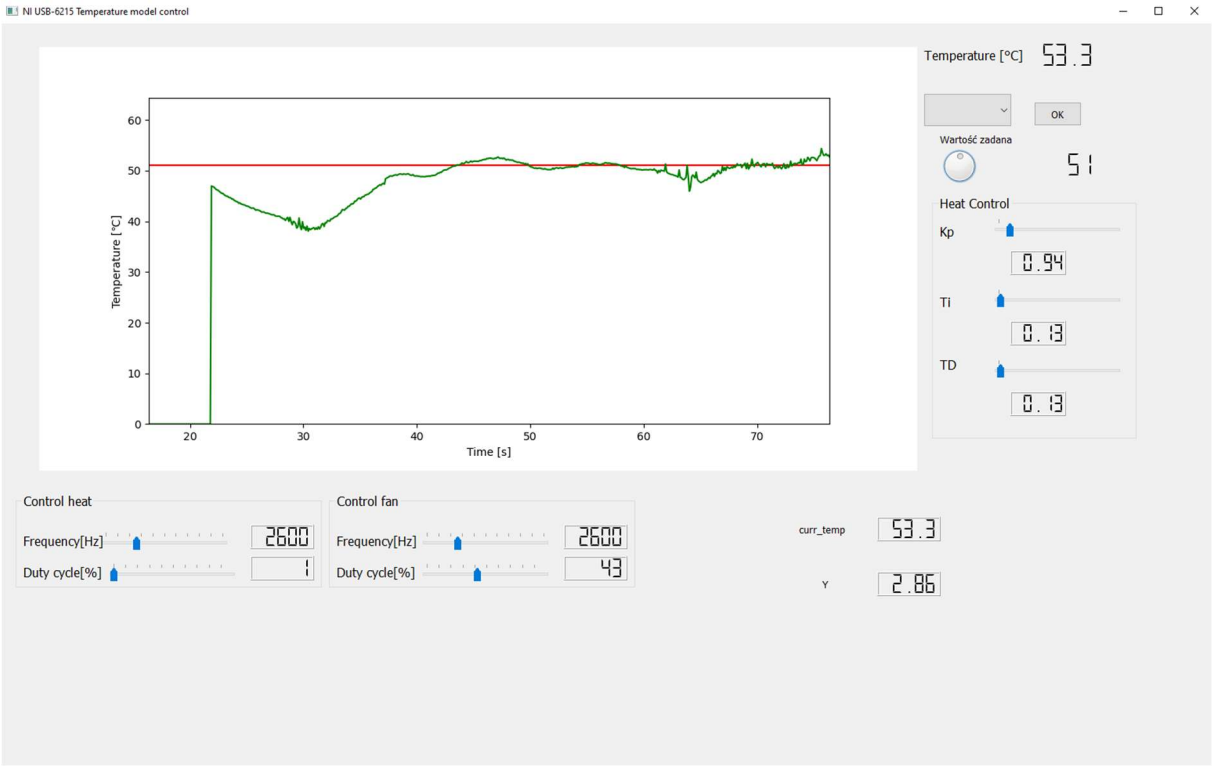

*Rysunek 10. Realizacja regulacji w programie QT Designer*

W pierwszej wersji programu odczytywana była tylko jedna próbka. Modyfikacja oraz odczyt wszystkich możliwych próbek w danym momencie pozwoliła uniknąć opóźnienia w wyświetlaniu i prawidłowo realizować sterowanie regulatorem PID. Na Rysunku 11 jest przedstawiona regulacja z opóźnieniem i odczytywaniem jednej próbki, na Rysunku 12 odczytywane są wszystkie możliwe próbki.




*Rysunek 11. Realizacja regulatora PID - opóźnienie*

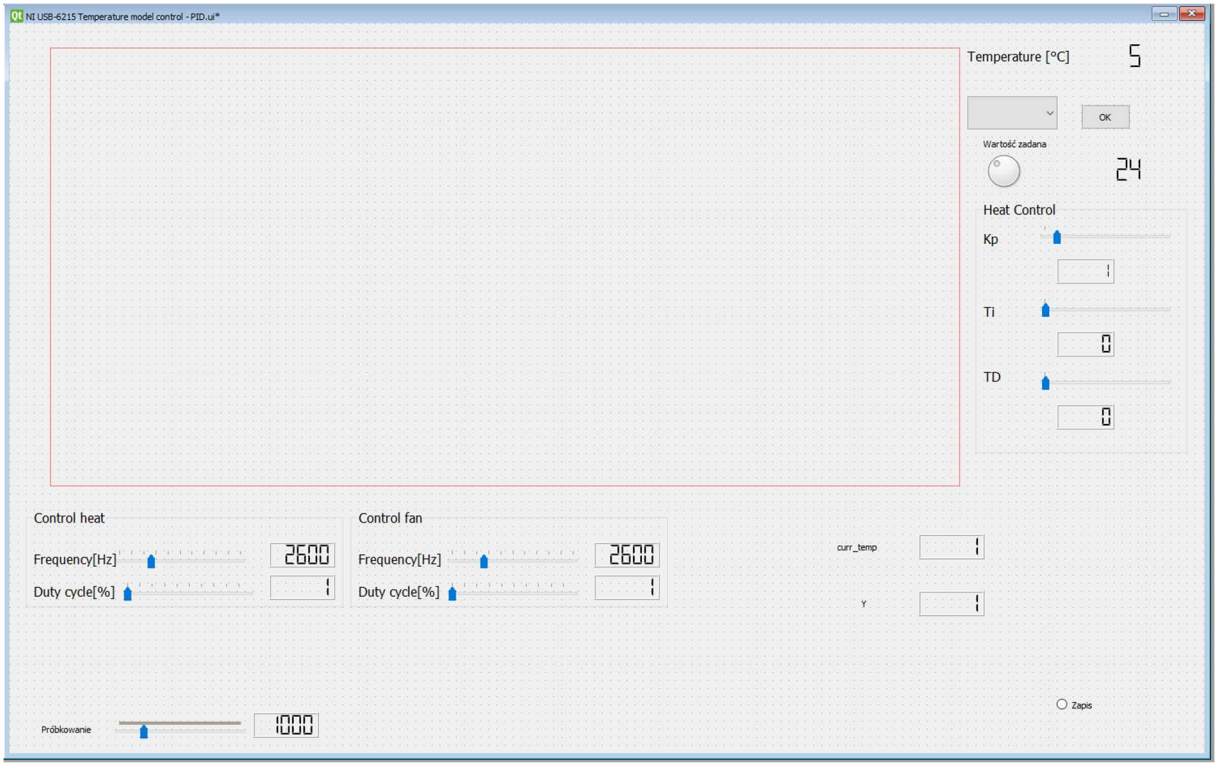
Na Rysunku 11 zmiana wartości zadanej nastąpiła w chwili t=10s. Po początkowym zaburzeniu temperatury, musiało upłynąć trochę czasu, zanim regulator rozpoczął swoją prawidłową pracę. Ponadto przeregulowanie na załączonym obrazku jest dość znaczne.




*Rysunek 12. Regulacja PID, bez opóźnienia*

# 8. Tydzień 08.06.2020 – 15.06.2020

W tym tygodniu prace odbyły się na zmianie możliwości próbkowania regulatora PID, jak i zapisem danych do dedykowanego pliku. Aby zrealizować zapis, dodana została biblioteka XlsxWriter (https://xlsxwriter.readthedocs.io/). Zapis odbywa się cyklicznie co każdorazowe wywołanie funkcji callback, a więc po zdefiniowanym czasie w parametrze rate odpowiadającej za częstość odczytu Taska, a więc co 0.1s. Ponadto stworzony został Slider, odpowiedzialny za ustawienie próbkowania. Na Rysunku 13 przedstawiony został projekt w programie QtDesigner.




*Rysunek 13. Widok w programie QtDesigner*

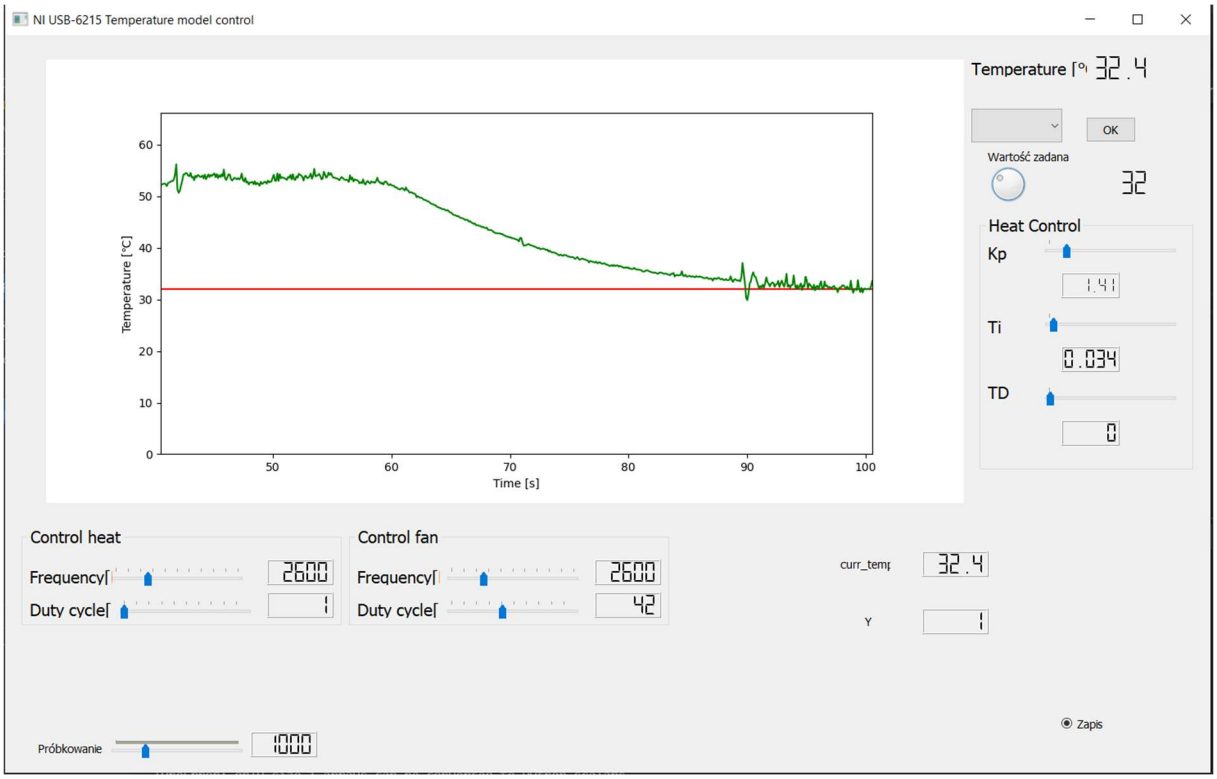
W czasie inicjalizacji danych, zostaje stworzony plik Dane.xlsx. Plik jest stworzony w miejscu na dysku, gdzie został uruchomiony kod. Podczas inicjalizacji zostają opisane

odpowiednie wiersze i kolumny, które będą odpowiedzialne za odczyt danych z urządzenia. Całość zapisu odbywa się w funkcji Write_Data.

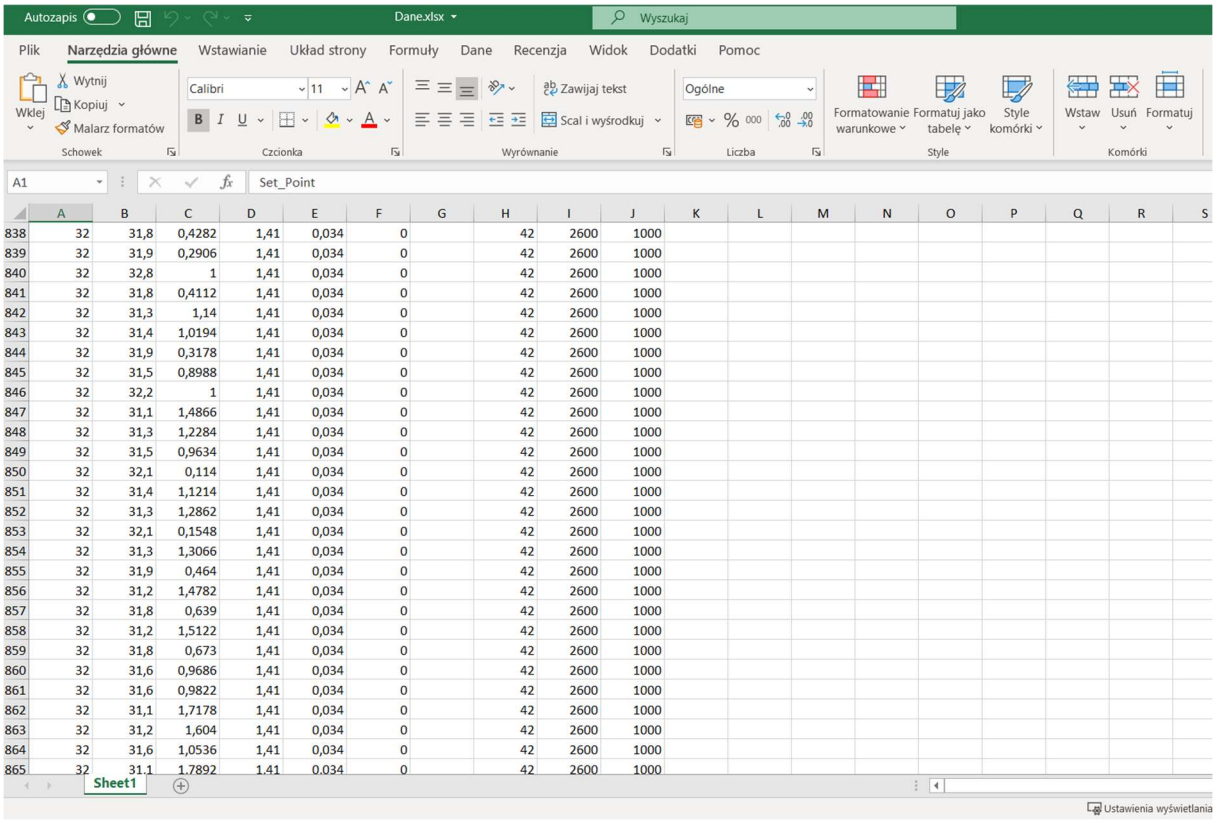
```
def Write_Data(self, curr_temp):
    worksheet.write(self.col,self.row-1, self.SetPoint)
    worksheet.write(self.col,self.row, curr_temp)
    worksheet.write(self.col,self.row+1, self.Y.value())
    worksheet.write(self.col,self.row+2, self.Kp)
    worksheet.write(self.col,self.row+3, self.Ki)
    worksheet.write(self.col,self.row+4, self.Kd)
    worksheet.write(self.col,self.row+6, self.duty1)
    worksheet.write(self.col,self.row+7, self.freq1)
    worksheet.write(self.col,self.row+8, self.probka)
```

*Rysunek 14. Fragment programu odpowiedzialny za zapis danych w pliku Dane.xlsx*

Próbkowanie jest możliwe w zakresie 1-4095 próbek. Ostatnim krokiem było nastrojenie regulatora PID.




*Rysunek 15. Regulacja PID dla Set_Point = 32*




| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 838 | 32 | 31,8 | 0,4282 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 839 | 32 | 31,9 | 0,2906 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 840 | 32 | 32,8 | 1 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 841 | 32 | 31,8 | 0,4112 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 842 | 32 | 31,3 | 1,14 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 843 | 32 | 31,4 | 1,0194 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 844 | 32 | 31,9 | 0,3178 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 845 | 32 | 31,5 | 0,8988 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 846 | 32 | 32,2 | 1 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 847 | 32 | 31,1 | 1,4866 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 848 | 32 | 31,3 | 1,2284 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 849 | 32 | 31,5 | 0,9634 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 850 | 32 | 32,1 | 0,114 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 851 | 32 | 31,4 | 1,1214 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 852 | 32 | 31,3 | 1,2862 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 853 | 32 | 32,1 | 0,1548 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 854 | 32 | 31,3 | 1,3066 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 855 | 32 | 31,9 | 0,464 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 856 | 32 | 31,2 | 1,4782 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 857 | 32 | 31,8 | 0,639 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 858 | 32 | 31,2 | 1,5122 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 859 | 32 | 31,8 | 0,673 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 860 | 32 | 31,6 | 0,9686 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 861 | 32 | 31,6 | 0,9822 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 862 | 32 | 31,1 | 1,7178 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 863 | 32 | 31,2 | 1,604 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 864 | 32 | 31,6 | 1,0536 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |
| 865 | 32 | 31,1 | 1,7892 | 1,41 | 0,034 | 0 | | 42 | 2600 | 1000 | | | | | | | | | |

*Rysunek 16. Zapis danych do pliku Dane.xlsx*