



**AKADEMIA GÓRNICZO - HUTNICZA
im. Stanisława Staszica w Krakowie**

**WYDZIAŁ
ENERGETYKI I PALIW**



Praca dyplomowa inżynierska

Imię i nazwisko: **Mateusz Urbańczyk**

Kierunek studiów: **ENERGETYKA**

Temat pracy dyplomowej - inżynierskiej:

Opracowanie programowalnego termostatu grzejnika elektrycznego

Elaboration of a programmable electric heater thermostat

Ocena:

Opiekun pracy:
dr inż. Łukasz Pyrda

Kraków, rok 2016/2017

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór; artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godność studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej " sądem koleżeńskim ””, oświadczam, że niniejszą pracę dyplomową wykonalem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

.....
podpis autora pracy

Strzeszczenie

Celem niniejszej pracy było stworzenie programowalnego termostatu. Na kolejnych stronach będą omówione działania podjęte do realizacji założonego celu. Zostaną opisane podstawy teoretyczne realizowanego zagadnienia wraz z wykorzystanymi w projekcie komponentami. W kolejnych rozdziałach przedstawiono podjęte kroki oraz zaimplementowane rozwiązania prowadzące do odpowiedzi na postawione problemy. W dalszej kolejności będzie opisane działanie układu oraz objaśnione zostaną jego zasady funkcjonowania. W końcowej części zaprezentowane są rezultaty testów przeprowadzonych na skonstruowanym układzie oraz wyciągnięte na ich podstawie wnioski.

Summary

The purpose of thesis was to elaborate a programmable thermostat system. Following pages will elaborate on actions taken to fulfill the purpose. Theoretical basis will be described along with components used in the project. Subsequently, ways of implementing solutions to the problems wil be presented. Next chapters present how the system works and all its features are explained. The ending part shows results of tests performed on a constructed configuration and conclusions based on them.

Spis treści

1. Wstęp.....	1
2. Cel pracy.....	2
3. Wstęp teoretyczny.....	3
3.1. Arduino.....	3
3.2. Termostat.....	6
3.3. Termopara.....	7
3.4. Regulator PID.....	7
4. Projekt.....	9
4.1. Użyte podzespoły.....	9
4.2. Użyte biblioteki.....	11
4.3. Utworzone klasy.....	16
5. Implementacja.....	20
5.1. Schemat połączeń układu.....	20
5.2. Algorytm działania programu.....	21
5.3. Przeprowadzone testy, wyniki.....	25
6. Podsumowanie.....	27
Bibliografia.....	28

1. Wstęp

Rozwój elektroniki sprawia, że jest ona coraz bardziej powszechna w wielu dziedzinach życia. Pozwala na znaczne usprawnienie wielu procesów oraz przyspieszenie uciążliwych obliczeń. Wynikiem rosnącego zapotrzebowania i zainteresowania tą tematyką są projekty takie jak Arduino – otwarte platformy z bogatą dokumentacją i bardzo aktywną społecznością dzielącą się własnymi pracami. Ze względu na swoją uniwersalność i popularność zostało wybrane właśnie to środowisko. Pozwala ono na stosunkowo proste programowanie oraz posiada bardzo wiele dedykowanych urządzeń peryferyjnych ułatwiających realizację własnych projektów.

W wielu dziedzinach przydatne lub nawet konieczne jest utrzymywanie pewnych wielkości na stałym poziomie poprzez regulowanie procesów wpływających na te wielkości, np. wysokość lotu śmigłowca poprzez zmianę szybkości obrotów wirnika czy utrzymywanie wilgotności gleby dzięki dostarczeniu odpowiedniej ilości wody. W niniejszej pracy zajęto się problemem podtrzymywania stałej temperatury poprzez regulowanie mocy dostarczanej do ogrzewanego układu.

Pracę nad realizacją projektu rozpoczęto od zapoznania się z dokumentacją wcześniejszej wspomnianego środowiska Arduino. Dodatkowo konieczne było znalezienie rozwiązań dotyczących odczytu temperatury oraz dostarczania energii cieplnej do ogrzewanego elementu. Wybrane narzędzia oraz sposoby ich obsługi zostaną przedstawione w rozdziale poświęconym etapowi projektowania układu. W kolejnej części przedstawiono sposoby implementacji wybranych rozwiązań prowadzących do stworzenia kompletnego systemu. W ostatnim rozdziale przytoczono wyniki testów.

Wynikiem przeprowadzonych prac jest wykonanie użytecznego narzędzia, prostego w obsłudze, skutecznie realizującego postawione przed nim zadanie.

2. Cel pracy

Za cel niniejszej pracy obrano zaprojektowanie i wykonanie z pomocą środowiska Arduino odpowiedniego oprogramowania oraz wybranie podzespołów, których wspólne działanie będzie spełniało funkcję termostatu. Gotowy układ musi odczytywać aktualną temperaturę oraz tak dobierać moc przeznaczoną na ogrzewanie, aby osiągnąć i utrzymać temperaturę na zadanym przez użytkownika poziomie. Do realizacji tego celu wybrano rozwiązanie na zasadzie regulatora PID.

Interfejs skonstruowanego urządzenia powinien być przyjazny i umożliwiać łatwe wybranie ustawień oraz szybkie rozpoczęcie pracy.

Dodatkowym atutem będzie pozostawienie miejsca na rozwijanie systemu pod kątem zastosowania w różnorakich układach wymagających wykorzystania termostatów.

Zakres pracy obejmuje:

- zapoznanie się z literaturą i dokumentacją wykorzystywanych elementów i podzespołów,
- wybór odpowiedniego algorytmu, który będzie spełniał stawiane przed projektem zadanie,
- stworzenie oprogramowania sterującego układem ogrzewania oraz umożliwiającego łatwą interakcję z użytkownikiem,
- testy układu.

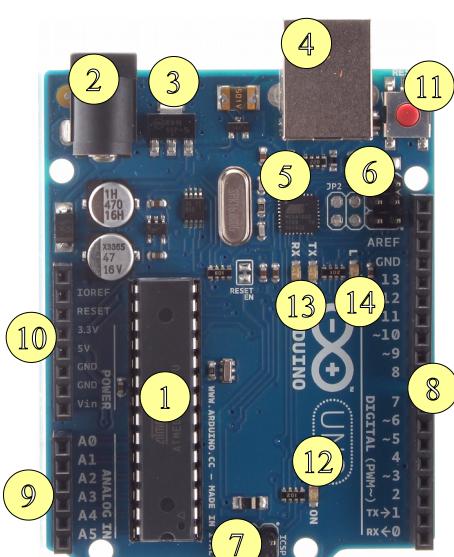
3. Wstęp teoretyczny

3.1. Arduino

Płytki Arduino

Projekt został zrealizowany w oparciu o płytę z rodziny Arduino (*Ilustracja 1*). Arduino to platforma open-source'owa (z ang. *open source* – wolne oprogramowanie) powstała w *Interaction Design Institute* w mieście Ivrea we Włoszech, prace nad nią rozpoczęto w 2005 roku. Schemat sprzętowy układu jest dostępny na licencji *Creative Commons*, a kod źródłowy oprogramowania Arduino IDE – na licencji *GNU General Public License*. Otwartość sprzętu i oprogramowania przyczyniła się do dużej popularności Arduino – dzięki dostępnej dokumentacji każdy może stworzyć swoją wersję płytki zgodną ze standardem Arduino. Platforma jest rozwijana w celu konstrukcji łatwo dostępnego, taniego i prostego w obsłudze narzędzia, które będzie stanowić alternatywę dla zaawansowanych kontrolerów wymagających skomplikowanych narzędzi. W założeniach jest przeznaczony zarówno dla początkujących jak i bardziej zaawansowanych użytkowników. Przykładowe projekty jakie są realizowane przez hobbystów za pomocą Arduino to proste roboty, czujniki ruchu czy systemy nawadniania^[1].

Sercem płyt Arduino są mikrokontrolery AVR firmy Atmel. W zależności od modelu płytki mogą to być układy 8-, 16- lub 32-bitowe. Płytki oficjalnego producenta są wyposażone w jednostki z serii megaAVR takie jak: ATmega8, ATmega168, ATmega1280 czy ATmega2560. W projekcie wykorzystano jeden z najpopularniejszych modeli – Arduino UNO, który korzysta z 8-bitowego układu ATmega328P (**1**) pracującego z częstotliwością 16 Mhz. Jest to chip z dwudziestoma ośmioma stykami umieszczony w gnieździe znajdującym się na środku płytki. Układ ten ma 32 kB pamięci flash^[2], w której są przetrzymywane instrukcje programu (szkice). Dane w pamięci flash są przechowywane cały czas, również po przerwaniu zasilania układu. Oprócz programu może w nich być zapisany program rozruchowy (ang. *Bootloader*). Życotność pamięci flash jest ograniczona do pewnej ilości operacji zapisu/odczytu.



Ilustracja 1: Arduino Uno, źródło ilustracji

<https://commons.wikimedia.org/>

Instrukcje zawarte w pamięci flash pobierane są przez jednostkę CPU (z ang. *Central Processing Unit*) a następnie wykonywane. CPU czyli procesor to cyfrowy układ sekwencyjny. Cyfrowy, to znaczy taki, który wykonuje operacje jedynie na dwóch stanach napięcia: wysokim i niskim. Tym poziomom przypisane są wartości 0 i 1 i na nich są realizowane obliczenia zgodnie z algebrą Boole'a. Słowo "sekwencyjny" oznacza że stan wyjść procesora zależy od stanu wejść oraz od poprzedniego stanu. Zadaniem CPU jest wykonywanie programu zapisanego w pamięci flash. Program jest podzielony na rozkazy, które są wysyłane do procesora w odpowiedniej kolejności i wykonywane. W pojedynczym rozkazie zawarta jest informacja o rodzaju operacji oraz na jakich wartościach ma być ona wykonana. Typowe rozkazy wykonywane przez procesor to kopiowanie danych, działania arytmetyczne (dodawanie, odejmowanie, porównywanie liczb, zmiana znaku liczby) czy działania logiczne na bitach (negacja, iloczyn i suma logiczna). Częścią procesora odpowiedzialną za działania arytmetyczne jest ALU (z ang. *Arithmetic Logic Unit*)^[3].

Mikrokontroler jest również wyposażony w 2 kB pamięci RAM (ang. *Static Random Access Memory*), która przechowuje dane tylko wtedy gdy jest on włączony a po każdym resecie zostaje oczyszczona. Jej zaletą jest to, że dostęp do niej jest dużo szybszy dlatego są w niej przechowywane aktualnie wykonywane

programy, dane dla tych programów oraz wyniki ich pracy. W przeciwnieństwie do pamięci flash nie ma ograniczenia dopuszczalnych operacji zapisu/odczytu. Pamięci RAM dzielą się na pamięci dynamiczne oraz pamięci statyczne (SRAM) – ten właśnie rodzaj jest stosowany w mikrokontrolerach AVR. Pamięć statyczna jest szybsza i nie wymaga częstego odświeżania, jak to ma miejsce w pamięci dynamicznej, która szybko traci swoją zawartość.

Oprócz samego mikrokontrolera na płytce znajdują się inne podzespoły umożliwiające użytkownikowi Arduino pracę z układem. Złącze zasilania (2) umożliwia łatwe podłączenie płytki do źródła energii za pomocą wtyczki 2.1 mm. Optymalne napięcie wejściowe mieści się w zakresie 7V – 12V. Logika mikrokontrolera pracuje na napięciu 5V stąd bezpośrednio obok złącza zasilania umieszczony jest stabilizator napięcia (3). Jego rolą jest obniżanie napięcia ze złącza do 5V oraz utrzymywanie go na stałym poziomie niezależnie od obciążenia układu oraz waahań napięcia zasilającego^[3].

Element oznaczony numerem (4) to złącze USB. Może ono zostać wykorzystane jako alternatywne źródło zasilania oraz przede wszystkim służy do programowania mikrokontrolera oraz komunikacji z komputerem za pomocą interfejsu szeregowego UART (ang. *Universal Asynchronous Receiver and Transmitter*) zaimplementowanego dzięki adapterowi *USB-to-serial* FTDI FT232 (chip z numerem (5)). Mikrokontroler FTDI może zostać przeprogramowany za pomocą dedykowanego dla niego interfejsu ICSP (6) (ang. *In-Circuit Serial Programming*)^[4].

Na płytce znajduje się oprócz tego jeszcze jedno wyprowadzenie ICSP (7) – służy ono bezpośrednio do programowania głównego mikrokontrolera AVR ATmega328P.

Wzdłuż dłuższych krawędzi płytki umieszczone są wyprowadzenia złącz. Wśród nich znajduje się 14 złącz I/O (wejścia/wyjścia) opisanych numerami 0-13 (8). Są to piny cyfrowe co oznacza że możemy na nich ustawić jedynie stan wysoki lub niski (jeżeli ustawimy dany port jako wyjście), możemy je zmieniać dowolnie z poziomu szyciu lub odczytywać analogiczne sygnały (gdy dany port jest zdefiniowany jako wejście) i mogą posłużyć np. do odczytywania informacji z prostych czujników lub przycisków. Pierwsze dwa piny – 0 i 1 oznaczone odpowiednio RX (*receive* - odbiór) i TX (*transmit* – transmisja) są stosowane do transmisji danych. Są one pośrednio połączone ze złączem USB służącym do komunikacji z komputerem. Przez omawiane złącza cyfrowe może przepływać maksymalny prąd o natężeniu 40 mA przy napięciu 5V. Sześć z pinów cyfrowych oznaczonych znakiem tyldy "˜" to wyprowadzenia PWM (ang. *Pulse-Width Modulation*). Wyjścia te dają możliwość generowania sygnału prostokątnego o zmiennym wypełnieniu.

Oprócz pinów cyfrowych w tym rzędzie są jeszcze piny zasilania GND (masa), AREF (*Analog Reference*), oraz SDA i SCL czyli wyprowadzenia magistrali I²C wykorzystywanej np. do komunikacji z bardziej zaawansowanymi czujnikami.

Po drugiej stronie płytki znajduje się kolejny rząd pinów: są to wejścia analogowe (9) oraz wyjścia zasilania (10).

Sześć złącz analogowych o oznaczeniach od A0 do A5 można zastosować do pomiaru przyłożonego do nich napięcia. Wynik pomiaru może zostać wykorzystany w szkicu, zmierzone napięcie jest odczytywane jako liczba z zakresu 0 -1023 (co odpowiada 10 bitom). Piny analogowe odpowiednio skonfigurowane mogą służyć jako zwykłe piny cyfrowe.

Złącza zasilania znajdujące się powyżej złącz analogowych jako jedyne na płytce Arduino nie są programowalne i odpowiadają za zasilanie układu. Pierwsze złącze – "Reset" ma taką samą funkcję jak przycisk "Reset" umieszczony na płytce (11). Po zwarciu tego pinu do masy (lub po naciśnięciu przycisku) program zapisany w pamięci mikrokontrolera jest wykonywany od samego początku. Kolejne piny zgodnie z oznaczeniami dostarczają napięcia 5V i 3.3V, kolejno 2 piny GND (masa) oraz Vin (napięcie wejściowe).

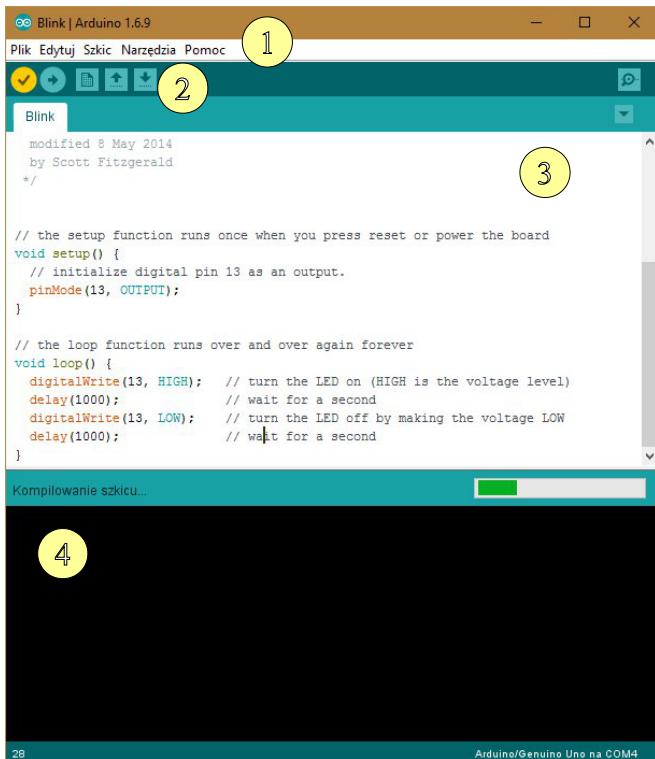
Na płytce znajdują się też 4 diody LED dostarczające użytkownikowi pewnych informacji. Dioda ON (12) działa gdy mikrokontroler jest podłączony do zasilania i układ pracuje. Diody TX i RX (13) sygnalizują transmisję danych od powiednio z i do mikrokontrolera za pośrednictwem portu szeregowego i złącza USB. Ostatnia dioda oznaczona literą "L" (14) jest pozostawiona do dyspozycji użytkownika i jest domyślnie podłączona do pinu 13. Może ona zostać dowolnie zaprogramowana z poziomu Arduino IDE poprzez zmianę

stanu na pinie 13^[3].

Oprogramowanie Arduino IDE

Arduino IDE (ang. *IDE* – *Integrated Development Environment* – zintegrowane środowisko programistyczne) – aplikacja dedykowana dla płyt Arduino z mikrokontrolerami AVR stworzona w języku Java dostępna dla wszystkich znaczących systemów operacyjnych (Windows, Linux, Mac OS). Wywodzi się z IDE dla języka *Processing* oraz projektu *Wiring*^[4]. Stworzone jest z myślą o łatwości obsługi nawet dla osób początkujących. Edytor kodu zawiera takie funkcje jak podświetlanie składni, dopasowywanie nawiasów czy automatyczne wcięcia w kodzie.

Menu oprogramowania (1) zawiera następujące opcje (*Ilustracja 2*):



Ilustracja 2: Okno programu Arduino IDE (Windows)

Plik – pozwala na utworzenie nowych szkiców, zapisywanie i otwieranie już utworzonych. Znajduje się tam także zbiór przykładowych szkiców oraz opcja *Preferencje* zawierająca opcje środowiska.

Edycja – opcje kopiowania, wklejania i przeszukiwania kodu.

Szkic – daje możliwość weryfikacji kodu i jego komplikacji. Poza opcją wgrania gotowego programu na płytę jest tu jeszcze możliwość dołączenia potrzebnych bibliotek.

Narzędzia – zawiera opcje umożliwiające wybór używanego typu płytki Arduino i portu COM do którego jest podłączona. Jest tam również opcja otworzenia *Monitora szeregowego*, który pozwala w wygodny sposób komunikować się z płytą Arduino za pomocą interfejsu UART. Monitor szeregowy posiada pole wprowadzania tekstu, do którego można wpisywać komendy oraz pole tekstowe, w którym wyświetlane są dane otrzymane z płytki. Ponadto można w tym oknie ustawić prędkość przesyłu danych (*baud-rate*). Kolejna opcja (*Monitor portu szeregowego*) ma podobną funkcję, jednak otrzymane dane może prezentować w formie wykresu.

Pomoc – zawiera odniesienia do dokumentów objaśniających działanie programu oraz informację o aktualnej wersji oprogramowania.

Na pasku ikon (2) umieszczone są najważniejsze funkcje – weryfikacja kodu, możliwość wgrania programu na płytę, utworzenie, zapisanie i wczytanie pliku. Ikona po prawej stronie oznacza *Monitor szeregowy*.

W obszarze tekstu (3) można edytować otwarty szkic. Istnieje również możliwość otwarcia wielu plików naraz w osobnych kartach.

W dolnej części okna znajduje się obszar gdzie wyświetlane są komunikaty środowiska Arduino IDE (4). Zawierają one głównie informacje o weryfikacji szkiców, aktualizacjach stanu środowiska, itp. W prawym dolnym rogu wyświetlana jest nazwa używanej płytki Arduino oraz numer portu COM do której jest podłączona.

Pliki środowiska Arduino IDE mają rozszerzenie .ino, a wspieranym językiem jest C i C++. Do prawidłowego działania programu konieczne jest zdefiniowanie dwóch funkcji: *setup()* i *loop()*^[4]. Kod przykładowego programu:

```

void setup() {
    pinMode(13, OUTPUT); //deklaracja pinu 13 jako pinu wyjścia
}

void loop() {
    digitalWrite(13, HIGH);      // ustawienie stanu wysokiego na pinie 13 (dioda
                                świeci)
    delay(1000);               // oczekanie 1 sekundy przed wykonaniem
                                kolejnego polecenia
    digitalWrite(13, LOW);     // ustawienie stanu niskiego na pinie 13
    delay(1000);               // oczekanie 1 sekundy
}

```

Powyższy program to *Blink* z kolekcji przykładów Arduino IDE. Co sekundę zmienia stan napięcia na pinie 13, do którego podłączona jest dioda LED umieszczona na płytce.

Jak wcześniej wspomniano każdy program musi zawierać funkcję *setup()* oraz funkcję *loop()*.

Funkcja *setup()* zawiera instrukcje wykonywane przez system Arduino tylko raz – zaraz po włączeniu lub resecie mikrokontrolera. W tej funkcji powinny znaleźć się ustawienia początkowe takie jak ustawienie trybu pinów za pomocą funkcji *pinMode()*, rozpoczęcie transmisji danych przez funkcję *Serial.begin()*, czy np. powitanie użytkownika za pomocą komunikatu na ekranie, itp.

Po wykonaniu wszystkich instrukcji startowych program przechodzi do funkcji *loop()*, która jest zapętlana przez cały czas działania mikrokontrolera (do momentu odłączenia zasilania od układu lub wciśnięcia przycisku RESET).

3.2. Termostat

Termostat (z gr. *thermos* – ciepło, *statos* - stały) to urządzenie, którego zadaniem jest aktywne utrzymywanie pożąданej temperatury na podstawie pomiarów wykonywanych przez należący do układu czujnik temperatury. Czasami termostatami nazywa się również urządzenia o charakterze biernym^[9] (czyli takie, które utrzymują stałą temperaturę dzięki bardzo dobrej izolacji cieplnej).

Termostaty to urządzenia ze sprzężeniem zwrotnym, których celem jest zmniejszenie różnicy między temperaturą zmierzoną a zadaną.

Proste termostaty realizują swoją funkcję poprzez okresowe włączanie i wyłączanie przyrządów grzejnych lub chłodniczych (w zależności od tego czy celem jest podwyższenie lub obniżenie temperatury do wskazanego poziomu). To rozwiązanie prowadzi do wahań (oscylacji) temperatury wokół wartości zadanej, co może być zjawiskiem niepożdanym w procesach wymagających utrzymywania określonej stałej temperatury.

Bardziej zaawansowane modele dopasowują moc, którą przeznaczają na ogrzanie układu, w taki sposób aby po zbliżeniu się do zadanej temperatury kompensowała straty energii cieplnej tego układu do otoczenia i utrzymywała temperaturę na jednym poziomie.

3.3. Termopara

Termopary są powszechnie stosowane w przemyśle, aparaturze pomiarowej lub różnych systemach regulacji. O ich popularności decyduje łatwość ich użycia i duży zakres pomiarowy oraz relatywnie niska cena. Ze względu na prostą fizyczną konstrukcję termopar, cechują się one trwałością mechaniczną i wysoką niezawodnością. Termopara jest zbudowana z dwóch różnych przewodników w postaci przewodów, których końce są połączone. Styk dwóch metali powoduje wystąpienie napięcia, które jest funkcją gradientu temperatury^[5]. Efekt ten nosi nazwę zjawiska Seebecka od nazwiska jego odkrywcy. Zależność tego napięcia od temperatury powinna być w zakresie użytkowym termopary liniowa. Konieczny jest odpowiedni dobór przyrządu do pomiaru napięcia termoelektrycznego, ponieważ napięcia występujące na zaciskach termopar nie przekraczają kilkudziesięciu miliwoltów, a zmiana napięcia wraz ze zmianą temperatury jest rzędu mikrowoltów na stopień Celsjusza^[6].

Do budowy termopar wykorzystuje się różne rodzaje stopów metali, z czego wynika mnogość typów termopar. Każda kombinacja stopów jest przeznaczona do określonych zastosowań. Termopary dzieli się na trzy grupy: grupa I może mierzyć temperatury w zakresie od -200 do 1200 °C i nie stosuje się w nich metali szlachetnych, grupa II: termopary platynowo-rodowe o zakresie temperatur 0 – 1600 °C, grupa III: termopary wolframowo-renowe o zakresie od 0 do 2200 °C.

Najczęściej stosowane są termopary z grupy pierwszej, są one oznaczone literami J, K, N, E i T. Są stosunkowo dokładne i mają niską cenę w porównaniu do innych grup termopar.

Typ J (żelazo-konstantan, Fe-CuNi) – przeważnie używane w przemyśle tworzyw sztucznych, zakres temperatur od -40 do 600 °C. Nie powinny być wystawiane na temperatury powyżej 760 °C. Zmiana napięcia termoelektrycznego wraz ze wzrostem temperatury: 55 µV/°C.

Typ K (chromel-alumen, NiCr-NiAl) – stosowane w zakresie temperatur od -200 do 1200 °C, mogą ulec korozji w środowisku odtłuszczanym chemicznie. Zależność napięcia od temperatury: 41 µV/°C.

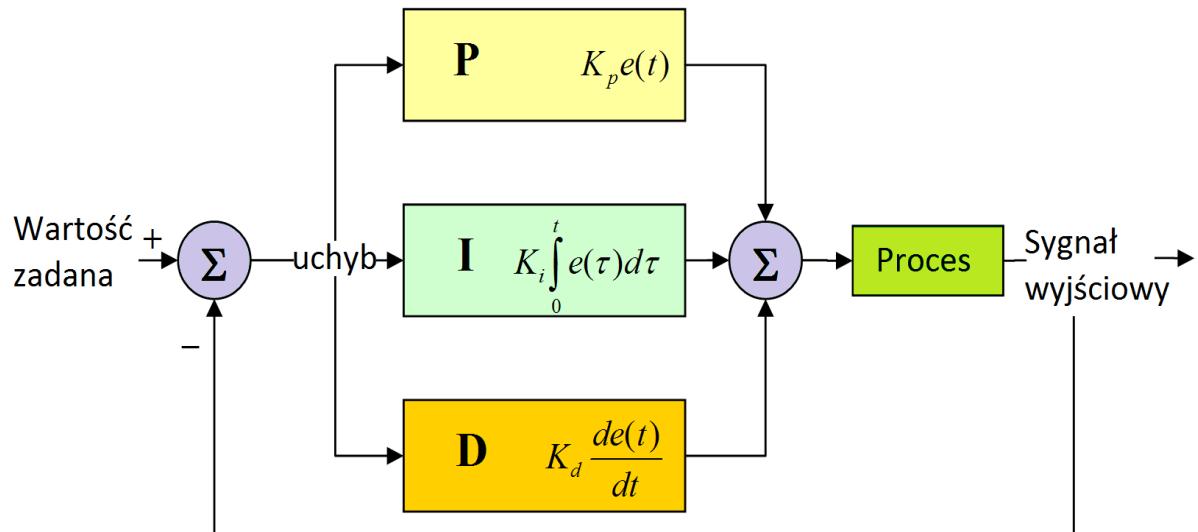
Typ N (nicrosil-nisil, NiCrSi-NiSi) – ma bardzo wysoką stabilność termiczną, jest odporna na utlenianie, dobrze nadaje się do dokładnych pomiarów w powietrzu do temperatur sięgających 1200 °C. Czułość termopar tego typu wynosi 39 µV/°C.

Typ E (chromel-konstantan, NiCr-CuNi) – stosowany w zakresie temperatur -200 do 900 °C. Ułatwia odczyt w niskich temperaturach ze względu na wysoką czułość – 68 µV/°C. Można ją stosować w atmosferze od próżni do łagodnie utleniającej.

Typ T (miedź-konstantan, Cu-CuNi) – powszechnie używane w przemyśle spożywczym. Zakres temperatur od -200 do 350 °C, czułość 30 µV/°C^[6].

3.4. Regulator PID

Regulator to element, którego zadaniem w układzie regulacji automatycznej jesttworzenie sygnału sterującego, który wpływa na wartość wielkości reguowanej^[7]. Gdy istotne jest utrzymanie reguowanej wartości na zadanym poziomie mimo występujących w otoczeniu zakłóceń w wielu procesach przemysłowych stosuje się regulatory PID. Nazwa wywodzi się od angielskich nazw poszczególnych członów wchodzących w skład regulatora: P – proporcjonalnego (ang. *proportional*), I – całkującego (ang. *integral*) i D – różniczkującego (ang. *Derivative*). Zadaniem regulatora PID jest takie dobieranie wartości sterującej, która będzie prowadzić do minimalizacji uchybu, który definiujemy jako różnicę pomiędzy pożądaną wartością zadaną a zmierzoną wartością sygnału wyjściowego. Regulator PID pracuje w pętli sprzężenia zwrotnego, co oznacza że sygnały wyjściowe mają wpływ na sygnały wejściowe układu (układ otrzymuje informacje o wynikach swojego działania) (*Ilustracja 3*).



Ilustracja 3: Schemat blokowy regulatora PID, źródło ilustracji commons.wikimedia.org/

Działanie poszczególnych członów PID w uproszczeniu można przedstawić następująco:

Człon P (proporcjonalny) redukuje uchyb bieżący.

Człon I (całkujący) kompensuje wszystkie minione wartości uchybu.

Człon D (różniczkujący) kompensuje przewidywane uchyby w przyszłości^[8].

Ważona suma tych trzech składników stanowi podstawę do obliczenia sygnału wyjściowego wpływającego na regulowany proces (np. zmiana prędkości obrotów silnika, zmiana mocy grzejnika). Współczynnikami proporcjonalności dla poszczególnych członów są K_p (wzmocnienie części proporcjonalnej), K_i (wzmocnienie części całkującej) oraz K_d (wzmocnienie części różniczkującej)^[8]. Odpowiedni dobór tych współczynników wpływa na jakość regulacji procesu. Celem regulacji jest otrzymanie stabilnego układu o jak najmniejszym uchybie ustalonym poprzez zminimalizowanie stopnia przeregulowania i ograniczenie oscylacji wokół wartości zadanej.

4. Projekt

4.1. Użyte podzespoły

Arduino UNO

W projekcie zostało wykorzystane Arduino w jednej z najpopularniejszych odmian Arduino UNO R3.



Ilustracja 4: Arduino UNO, źródło ilustracji commons.wikimedia.org/

Konwerter MAX 6675

Ma za zadanie odczytać wartość napięcia między stykami termopary i zamienić ją na sygnał cyfrowy. Sygnał składa się z 12 bitów wysyłanych po interfejsie SPI. Konwerter ma 7 pinów, może jedynie wysyłać dane tylko do odczytu (SO – *Slave Output*).

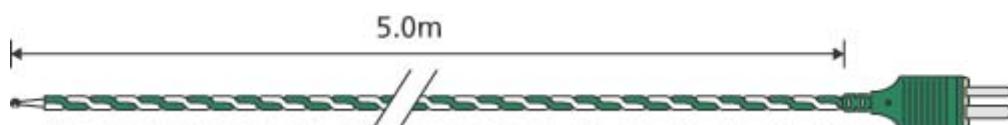


1. VSS (GND) – ujemny zacisk zasilania, masa.
2. T– wejście "-" złącza termopary.
3. T+ wejście złącza "+" termopary.
4. VDD – dodatni zacisk zasilania 5V.
5. SCK – *Serial Clock*, sygnał zegarowy.
6. CS – *Chip Select* (lub SS – *slave select*) – wykorzystywane w interfejsie SPI do wyboru urządzenia peryferyjnego, z którym przeprowadzana będzie komunikacja. Jeżeli jest na nim stan wysoki – urządzenie może przesyłać informacje, w przeciwnym przypadku jest przez mikrokontroler ignorowane.
7. MISO – *Master Input Slave Output* – wejście mikrokontrolera, wyjście danych z urządzenia peryferyjnego.^[10]

Ilustracja 5:
Schemat
konwertera MAX

6675, program
Fritzing

Termopara COMARK AK29M



Ilustracja 6: Termopara AK29M, źródło ilustracji
<https://www.comarkinstruments.net/product/ak29m-flexible-wire-air-probe/>

Termopara typu K. Długość przewodu: 5 m.

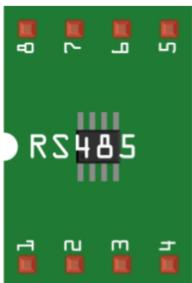
Zakres temperatur deklarowany przez producenta: -100 °C do 250 °C.

Przejściówka RS485

Ten moduł odpowiada za komunikację między mikrokontrolerem a zasilaczem. Standard transmisji RS-485

umożliwia komunikację między wieloma urządzeniami w sieci, która może odbywać się na stosunkowo dużych odległościach.

Za obsługę komunikacji z pomocą tego komponentu odpowiada biblioteka SoftSerial.h.

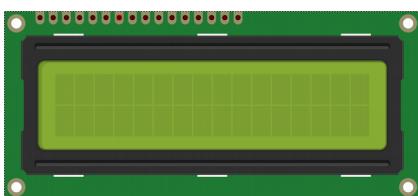


Ilustracja 7: Schemat przejściówki RS485,

program Fritzing

1. RO – *receive out* – dane odbierane przez mikrokontroler.
2. RE – *receive enable* – stan na pinach RE i DE odpowiadają za kierunek przesyłania danych. RE umożliwia odbiór danych.
3. DE – *data enable* – umożliwienie przyjmowania danych.
4. DI – *data in* – dane wysyłane przez mikrokontroler do urządzenia lub sieci urządzeń peryferyjnych..
5. GND – ujemny zacisk zasilania, masa.
6. i 7. - odpowiednio piny oznaczone literami A i B, po których wysyłane są dane.
8. VCC – dodatni zacisk zasilania 5V.

Wyświetlacz LCD



Ilustracja 8: Schemat ekranu LCD, program Fritzing

Wyświetlacz oparty na sterowniku HD44780, dzięki czemu może komunikować się z urządzeniami w trybie 4-bitowym lub 8-bitowym. W projekcie wykorzystano pierwszą opcję z uwagi na prostszą konstrukcję i możliwość użycia mniejszej liczby pinów na Arduino. Wyświetlacz może wyświetlić 2 rzędy znaków, po 16 w każdym. Za obsługę wyświetlacza odpowiada domyślna biblioteka Arduino LiquidCrystal.h.

Opis wyprowadzeń (numeracja szesnastu pinów w kolejności od lewej do prawej strony zgodnie z ilustracją 8):

1. GND – ujemny zacisk zasilania, masa.
2. Vcc – dodatni zacisk zasilania 5V.
3. V0 – regulacja kontrastu. W zależności od napięcia przyłożonego do pinu 3 można regulować kontrast z jakim są wyświetlane znaki. Dla wygodnej regulacji kontrastu można podłączyć potencjometr lub zastosować odpowiednio dopasowany dzielniczek napięcia.
4. RS – wybór rejestrów.
5. RW – wybór opcji odczyt/zapis. Ustawienie stanu niskiego na tym pinie (podłączenie go do masy) umożliwia zapis znaków na wyświetlaczu.
6. E – *enable*: zezwolenie na zapis do rejestrów.
7. do 10. - D0 do D3 – piny na dane wykorzystywane w komunikacji za pomocą 8-bitów. W projekcie nie używane.
11. do 14. - D4 do D7 – piny na dane wykorzystywane w komunikacji 4-bitowej.
15. Vpod – dodatni zacisk zasilania 5V diody podświetlającej ekran.
16. GND – ujemny zacisk (masa) zasilania diody podświetlającej ekran.



Ilustracja 9: Zasilacz laboratoryjny

Manson2405, źródło ilustracji
<http://www.manson.com.hk/>

Zasilacz laboratoryjny Manson 2405

Zasilacz laboratoryjny Manson 2405 jest programowalnym zasilaczem z funkcjami rejestrowania danych.

Posiada zintegrowane oprogramowanie na systemy operacyjne Windows rozszerzające funkcjonalność panelu przedniego zasilacza, m.in. pozwala na zapisywanie ustawień żądanego wartości wyjściowych, tworzenie programów czasowych, itp. Istnieje możliwość

połączenia naraz 31 zasilaczy. Możliwe jest także zaprojektowanie własnego układu sterowania zasilaczem dzięki dostarczonej do niego instrukcji opisującej działanie komend wysyłanych poprzez interfejs RS-232 lub RS-485^[11].

Napięcie wejściowe zasilacza: 100-240 VAC 50Hz/60Hz

Napięcie wyjściowe: 1-40 V DC

Natężenie wyjściowe: 0 – 5 A

Zabezpieczenia: OVP (ang. *Over Voltage Protection*), ograniczenie natężenia prądu, ochrona przed zbyt wysoką temperaturą

Komunikacja: interfejs RS-232 (pojedynczy zasilacz), RS-485 (do 31 zasilaczy).

4.2. Użyte biblioteki

W poniższym podrozdziale zostaną opisane użyte biblioteki oraz najważniejsze metody.

LiquidCrystal.h

Biblioteka pozwala Arduino na obsługę wyświetlaczy LCD opartych na chipie Hitachi HD44780, który jest obecny w większości wyświetlaczy tekstowych^[12].

Najważniejsze metody:

Konstruktor LiquidCrystal()

Tworzy obiekt klasy LiquidCrystal. Wyświetlacz może być kontrolowany przez 4 lub 8 złączy – w zależności od wybranej opcji do konstruktora należy dostarczyć inną liczbę argumentów.

```
LiquidCrystal(rs, enable, d4,d5,d6,d7)
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

Argumenty:

rs – numer pinu Arduino, który jest połączony z pinem RS ekranu LCD.

rw – numer pinu Arduino, do którego podłączony jest pin RW ekranu LCD. Opcjonalnie można przyłożyć pin RW do masy i pominąć ten argument w konstruktorze.

enable – numer pinu Arduino, który jest połączony z pinem enable ekranu LCD.

d0, d1, d2, d3, d4, d5, d6, d7 – numery pinów Arduino podłączonych do odpowiednich pinów ekranu LCD. Podłączenie pierwszych czterech pinów jest opcjonalne, należy wtedy również pominąć je w konstruktorze.

begin()

Inicjalizuje działanie ekranu LCD, określa wymiary wyświetlacza. Funkcja begin() musi zostać wywołana przed użyciem jakichkolwiek innych metod tej klasy.

```
lcd.begin(cols, rows)
```

Argumenty:

lcd - obiekt klasy LiquidCrystal.

cols – liczba kolumn znaków na wyświetlaczu.

rows – liczba wierszy znaków na wyświetlaczu.

clear()

Usuwa wszystkie znaki z ekranu i ustawia kurSOR na skrajne lewe góRne pole. Funkcja nie przyjmuje żadnych argumentów.

```
lcd.clear()
```

setCursor()

Pozwala ustawić kurSOR na dowolną pozycję na ekranie. Kolejne przesyłane na ekran znaki będą wyświetlane od tego miejsca.

```
lcd.setCursor(col, row)
```

Argumenty:

col – numer kolumny, na którą zostanie ustawiony kurSOR (0 jest numerem pierwszej kolumny)

row – numer wiersza, na który zostanie ustawiony kurSOR (0 jest numerem pierwszego wiersza)

print()

Wyświetla tekst na LCD. Funkcja zwraca ilość zapisanych bajtów.

```
lcd.print(data)
lcd.print(data, format)
```

Argumenty:

data – dana do wyświetlenia. Mogą to być zmienne typu char, int, long lub String.

format – funkcja print umożliwia wybranie systemu liczbowego wyświetlanych liczb: BIN (system dwójkowy), DEC (dziesiętny), OCT (ósemkowy) lub HEX (szesnastkowy).

SoftwareSerial.h

Płytki Arduino wspierają komunikację przez port szeregowy na pinach 0 i 1. Biblioteka SoftwareSerial może zostać użyta do przeniesienia funkcjonalności komunikacji na inne piny oraz zwiększenie ilości portów szeregowych (przydatne gdy potrzebna jest komunikacja z więcej niż jednym urządzeniem). Szybkość transmisji danych sięga 115200 bps (ang. *bits per second* – bity na sekundę). Należy zwrócić uwagę, że jeśli jest używana większa niż jeden liczba portów szeregowych to w danym momencie odbierać dane można tylko na jednym z nich^[13].

Najważniejsze metody:

Konstruktor **SoftwareSerial()**

Tworzy obiekt klasy SoftwareSerial. Można utworzyć wiele obiektów, ale tylko jeden może być aktywny w danym momencie.

```
SoftwareSerial(rxPin, txPin)
```

Argumenty:

rxPin – pin do odczytywania danych z portu szeregowego.

txPin – pin do wysyłania danych przez port szeregowy.

begin()

Ustawia szybkość transmisji danych przez port szeregowy (*baud rate*). Musi zostać wywołana aby umożliwić komunikację.

```
serial.begin(value)
```

Argumenty:

serial – obiekt klasy SoftwareSerial.

value – wartość szybkości przesyłu danych (baud rate). Może przyjmować następujące wartości: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600 i 115200.

print()

Wysyła dane przez port szeregowy w postaci znaków tablicy ASCII. Funkcja zwraca ilość wysłanych bajtów.

```
serial.print(data, format)
```

Argumenty:

data – dane jakie chcemy przesłać. Uwaga – liczby całkowite (int) zostają zmienione na odpowiednie znaki z tablicy ASCII, analogicznie typ liczb zmiennoprzecinkowych (float) z tym że przesyłane są tylko dwie cyfry po przecinku. Zmienne typu string i char (napisy i znaki) są przesyłane bez żadnych zmian.

format – niewymagany argument, pozwala na określenie systemu liczbowego wyświetlanych liczb: BIN (system dwójkowy), DEC (dziesiętny), OCT (ósemkowy) lub HEX (szestnastkowy).

write()

Wysyła przez port szeregowy dane w postaci binarnej, można wysłać pojedynczy bajt lub ich serię. Metoda zwraca ilość zapisanych bajtów.

```
serial.write(value)
serial.write(string)
serial.write(buffer, length)
```

Argumenty:

value – wartość do wysłania jako pojedynczy bajt.

string – ciąg znaków do wysłania jako seria bajtów.

buffer – tablica do wysłania jako seria bajtów.

length – rozmiar tablicy bufora.

read()

Zwraca znak który został odczytany na pinie RX portu szeregowego. Zwraca -1 jeżeli żaden znak nie jest dostępny.

```
serial.read()
```

MAX6675

Biblioteka do obsługi chipu MAX6675. Konwertuje sygnał analogowy z termopary na cyfrowy zapis liczbowy o długości 12 bitów. Posiada też funkcję zamieniającą te odczytane wartości na temperaturę w °C lub w °F^[14].

Najważniejsze metody:

Konstruktor **MAX6675()**

Tworzy obiekt klasy MAX6675 i pobiera numery pinów interfejsu SPI podłączonych do odpowiednich wyprowadzeń na płytce Arduino.

MAX6675 (SCLK, CS, MISO)

Argumenty:

SCLK, CS, MISO – numery pinów Arduino, do których zostały połączone odpowiednie wyjścia interfejsu SPI na konwerterze MAX6675.

spiread()

Odczytuje bity przesłane przez interfejs SPI i zwraca je w zmiennej typu byte. Jest to prywatna metoda używana później do przeliczenia na rzeczywistą wartość temperatury.

max.spiread()

Funkcja nie pobiera żadnych argumentów.

readCelsius()

Zamienia ciąg bitów odczytanych przez funkcję *spiread()* na temperaturę w stopniach Celsjusza. Zwracane przez tą metodę wartości mieszczą się w przedziale 0 – 1023.75 °C z podziałką 0.25 °C. Wynika to z faktu, że konwerter MAX6675 dostarcza 12 bitów danych (co daje zakres od 0 do 4095 w systemie dziesiętnym).

Jeżeli termopara nie jest podłączona do konwertera funkcja zwraca wartość NaN (ang. *Not a Number*).

max.readCelsius()

Funkcja nie pobiera żadnych argumentów

readFahrenheit()

Zwraca wartość temperatury w stopniach Fahrenheita. Funkcja przelicza tą temperaturę z wartości w stopniach Celsjusza wg wzoru $F = \frac{9}{5}C + 32$ (gdzie F – wartość temperatury w °F, C – wartość temperatury w °C).

max.readFahrenheit()

Funkcja nie pobiera żadnych argumentów.

Button.h

Biblioteka ułatwiająca wykorzystywanie fizycznych przycisków użytych do obsługi programu użytego w projekcie^[15].

Posiada prywatną zmienną 'state', w której na kolejnych bitach przechowywane są informacje o obecnym stanie pinu, do którego podłączony jest przycisk, jego poprzednim stanie oraz informacja czy ten stan się zmienił.

Najważniejsze metody:

Konstruktor Button()

Tworzy obiekt klasy Button, ustawia numer pinu, do którego został fizycznie podłączony przycisk.

Button(buttonPin, buttonMode)

Argumenty:

buttonPin – numer pinu na płytce Arduino, do którego podłączono przycisk.

buttonMode – może przyjąć wartość PULLUP (logiczna 1) lub PULLDOWN (logiczne 0), dzięki czemu można podłączyć do pinu rezystor podciągający.

isPressed(), wasPressed() i stateChanged()

Te funkcje odczytują wartości poszczególnych bitów prywatnej zmiennej *state*.

Funkcja isPressed() zwraca wartość true jeśli przycisk jest wciśnięty, w przeciwnym wypadku zwraca wartość false. Ponadto sprawdza czy stan przycisku się zmienił oraz zapisuje poprzedni stan.

Pozostałe dwie funkcje odczytują wartość odpowiedniego bitu zmiennej *state* i zwracają wartość true lub false.

```
button.isPressed()
button.wasPressed()
button.stateChanged()
```

Metody te nie pobierają żadnych argumentów.

uniquePress()

Funkcja ta zwraca wartość true tylko raz, w momencie naciśnięcia przycisku (w przeciwieństwie do metody isPressed(), która zwraca wartość true przez cały czas kiedy przycisk jest wciśnięty).

timePressed()

Do biblioteki Button.h została dodana metoda timePressed() aby rozszerzyć jej funkcjonalność i umożliwić pomiar czasu, przez jaki przycisk został przytrzymany. Metoda zwraca ten czas jako wynik w milisekundach.

Kod metody:

```
unsigned long Button::timePressed(void) {
    static unsigned long start = 0;
    static unsigned long stop = 0; // 1
    if (uniquePress())
        start = millis(); // 2
    if (!isPressed()) start = millis();

    stop = millis();
    if (!isPressed()) return 0;
    else return (stop - start); // 3
}
```

1. Inicjalizacja zmiennych statycznych (pamiętających swoją wartość pomiędzy wywołaniami funkcji). Dzięki nim możliwe jest odczytanie czasu wciśnięcia przycisku w każdym wykonaniu głównej funkcji *loop()*^[16].

2. W momencie wciśnięcia przycisku do zmiennej start zostaje zapisana ilość milisekund jako minęła od momentu uruchomienia mikrokontrolera (funkcja *millis()*)

3. Jeżeli przycisk nie jest naciśnięty funkcja zawsze zwraca wartość 0. W przeciwnym wypadku zwraca różnicę aktualnego czasu pozyskanego z funkcji *millis()* i czasu w momencie wciśnięcia przycisku (wynikiem tego działania jest łączny czas wciśnięcia przycisku).

Pełny kod źródłowy bibliotek *MAX6675.h* i *Button.h* znajduje się w *Załączniku nr 2*.

Oprócz zaimportowanych bibliotek zostały ponadto utworzone dodatkowe klasy, znajdują się one w plikach *proj.h* i *proj.cpp*.

4.3. Utworzzone klasy

Temperature

Deklaracja klasy zawarta w pliku nagłówkowym *proj.h*

```
class Temperature
{
    private:
        float val; // 1

    public:
        Temperature(float=0);
        float getTempValue(MAX6675);
        float value();
        float set(float);
};
```

Klasa zawiera zmienną prywatną *val*, w której przechowywana jest wartość temperatury.

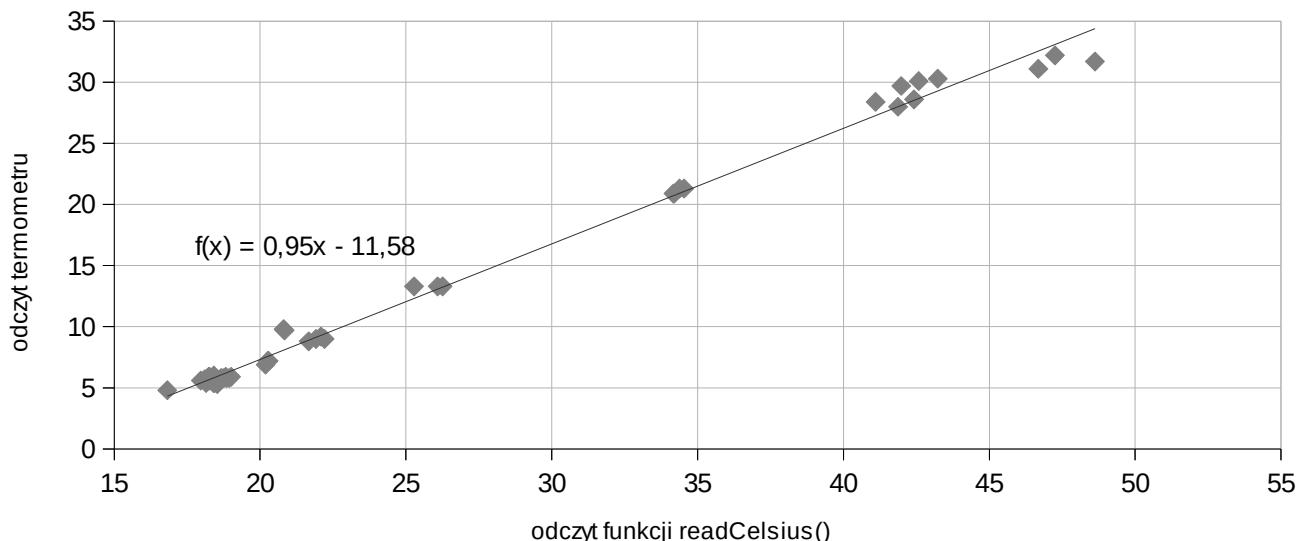
W zależności od potrzeb można ustawić ją za pomocą funkcji *set()*, która jako parametr przyjmuje liczbę zmiennoprzecinkową i przypisuje wartość do składowej zmiennej *val* (przydatne gdy chcemy ustawić wartość pożądanej temperatury).

Do odczytania wartości zapisanej w składowej *val* należy posłużyć się metodą *value()*, która zwraca wartość tej składowej.

Aby odczytać aktualną wartość temperatury sczytaną z termopary należy użyć funkcji *getTempValue()* i jako parametr podać obiekt klasy MAX6675. Kod źródłowy funkcji:

```
float Temperature::getTempValue(MAX6675 thermocouple) {
    val = 0.95*(thermocouple.readCelsius()) - 11.58;
    return val;
}
```

Wątpliwości może budzić fakt przeliczania wartości odczytanej przez funkcję *readCelsius()*. Odczyty uzyskiwane z pomocą tej funkcji przy zastosowaniu niededykowanej dla chipu termopary firmy Comark znacząco odbiegały od spodziewanych wartości. Dlatego wykonano kalibrację z pomocą termometru. Zostały porównane wartości odczytów za pomocą termometru oraz termopary korzystającej z funkcji *readCelsius()*. Wyniki pomiarów przedstawia poniższy wykres.



Wykres 1: Pomiary do kalibracji termopary

Z pozyskanych punktów, za pomocą arkusza kalkulacyjnego wyznaczono krzywą regresji. Otrzymana zależność liniowa $T=0,95 T_{rc} - 11,56$ pozwala przeliczyć wartość temperatury T_{rc} odczytanej przez funkcję `readCelsius()` na wartość rzeczywistą T . Rozdzielcość podziałki odczytu temperatury z pomocą tej funkcji to 0.23 °C.

Timer

Biblioteka powstała aby ułatwić obsługę zdarzeń, które muszą wystąpić co jakiś określony czas. Funkcja `delay()` nie jest idealna, ponieważ "blokuje" na pewien czas wykonanie programu stąd wystąpiła konieczność znalezienia lepszego rozwiązania. Przykładem zastosowania tej klasy jest możliwość odczytywania temperatury co pewien określony czas przy jednoczesnym zachowaniu responsywności przycisków. Przy zastosowaniu ww. funkcji `delay()` układ jest "uśpiony" i nie reaguje na działania użytkownika.

Deklaracja klasy:

```
class Timer
{
    private:
        unsigned long    millisNow;
        unsigned long    millisStart;
    public:
        Timer();
        long int        threshold;
        boolean         stepTimer(unsigned long);
};
```

Składowe zmienne prywatne `millisNow` i `millisStart` przechowują aktualny czas oraz czas w momencie rozpoczęcia odliczania.

W konstruktorze `Timer()` obie zmienne są ustawiane na aktualny czas za pomocą funkcji `millis()`.

Metoda `stepTimer()`

```
boolean Timer::stepTimer(unsigned long milliseconds) {
    millisNow = millis();
    if((millisNow - millisStart > milliseconds) &&
       (millisNow - millisStart) % milliseconds < threshold ) {
        millisStart = millis();
        return true;
    }
    else return false;
}
```

Metoda zwraca wartość **true** za każdym razem gdy upłynie ilość czasu podana w milisekundach jako argument tej funkcji. W każdym innym momencie zwraca wartość **false**. Poprzez zmianę składowej `threshold` można zmienić czas przez jaki utrzymuje się wartość **true** zwracana przez funkcję.

Przykład zastosowania metody `stepTimer()`:

```
if( timer.stepTimer(1000) ) {
    // instrukcja, która ma być wykonywana co 1 s, np. Odczytanie
    // temperatury
}
```

Podczas działania programu wykorzystującego tę metodę nie ma przerw na "uśpienie" układu, w których mogłyby np. zostać pominięte pewne sygnały wejściowe – takie rozwiązanie wprowadza przewagę wobec użycia wbudowanej funkcji `delay()`.

Manson2405

Klasa zawiera metody do komunikacji z zasilaczem Manson 2405.

```
String sendCommand(SoftwareSerial serial, String caption);
String startSession(SoftwareSerial serial, int adress); // wyłącza przedni
panel
String endSession(SoftwareSerial serial, int adress); // włącza przedni panel
i kończy sesję
sendCommand()
```

Wysyła ciąg znaków *caption* do zasilacza przez połączenie szeregowe *serial*. Na koniec każdej komendy dodaje znak ASCII – CR (ang. *carriage return*), który oprogramowanie zasilacza odbiera jako zakończenie komendy.

startSession()

Wysyła komendę odpowiedzialną za zablokowanie panelu frontowego zasilacza i przejście w tryb zdalnego sterowania. Należy podać *adres* zasilacza.

endSession()

Kończy sesję zdalnego sterowania, odblokowuje panel frontowy.

RegPID

W klasie RegPID podjęto próbę implementacji algorytmu regulatora PID.

Deklaracja klasy:

```
class RegPID
{
    private:
        float k;
        float k_i;
        float k_d;

    public:
        void setK(float);
        void setKI(float);
        void setKD(float);
        RegPID(float = 100, float= 0.002, float = 20);
        float regulator(float, float);
};
```

Prywatne składowe klasy *k*, *k_i* oraz *k_d* to nastawy regulatora (kolejno: współczynnik części proporcjonalnej, współczynnik części całkującej, współczynnik części różniczkującej).

Metody *setK()*, *setKI()*, *setKD()* ustawiają zmienną podaną w argumencie jako wartość nastawy.

Konstruktor *RegPID()* wykorzystuje powyższe funkcje aby przypisać domyślne wartości nastaw przy inicjalizacji obiektu. Jako argumenty przyjmuje trzy liczby, każda zostaje przypisana do odpowiedniej nastawy.

```
RegPID::RegPID(float k_, float k_i_, float k_d_){
    setK(k_);
    setKI(k_i_);
    setKD(k_d_);
}
```

Metoda **regulator()**

Jako argumenty przyjmuje wartość pożądaną oraz wartość wyjścia układu (czyli aktualną wartość temperatury). Jej zadaniem jest wyznaczenie sygnału wyjściowego regulatora, który następnie zostanie wykorzystany do przesłania odpowiedniej komendy do zasilacza.

Kod źródłowy:

```
float RegPID::regulator(float desired, float tval) {
    float p, i, d, r; // 1
    float e; // 2
    static float e_p = 0; // 3
    static float s_e = 0; // 4
    e = desired - tval; // 5

    // wyznaczenie składnika proporcjonalnego
    p = k * e;

    // wyznaczenie składnika całkowego
    s_e += e; // najpierw trzeba wyliczyć sumę wszystkich uchybów;
    i = k_i * s_e;

    // wyznaczenie składnika D
    d = k_d * (e - e_p);
    e_p = e; // zapisanie chwilowej wartości uchybu

    r = p + i + d; // 6

    if (r < 0) r = 0;
    if (r > 100) r = 100; // 7

    return r;
}
```

Rozwinięcie komentarzy.

1. Utworzenie zmiennych pomocniczych, w których będą przechowywane wartości poszczególnych członów.
2. e – uchyb regulacji.
3. e_p – wartość uchybu w poprzednim wywołaniu metody, przechowywana w statycznej zmiennej float. Potrzebna do obliczenia składnika i .
4. s_e – suma wszystkich dotychczasowych uchybów regulacji. Potrzebna do obliczenia składnika d .
5. Obliczenie uchybu jako różnicy między wartością zadaną a wartością zmierzona.
6. Składnik p obliczamy jako iloczyn współczynnika części proporcjonalnej i aktualnego uchybu.
- Składnik i to iloczyn współczynnika części całkującej i sumy wszystkich uchybów.

Składnik d jest iloczynem współczynnika części różniczkującej i wartości uchybu w poprzednim wywołaniu funkcji.

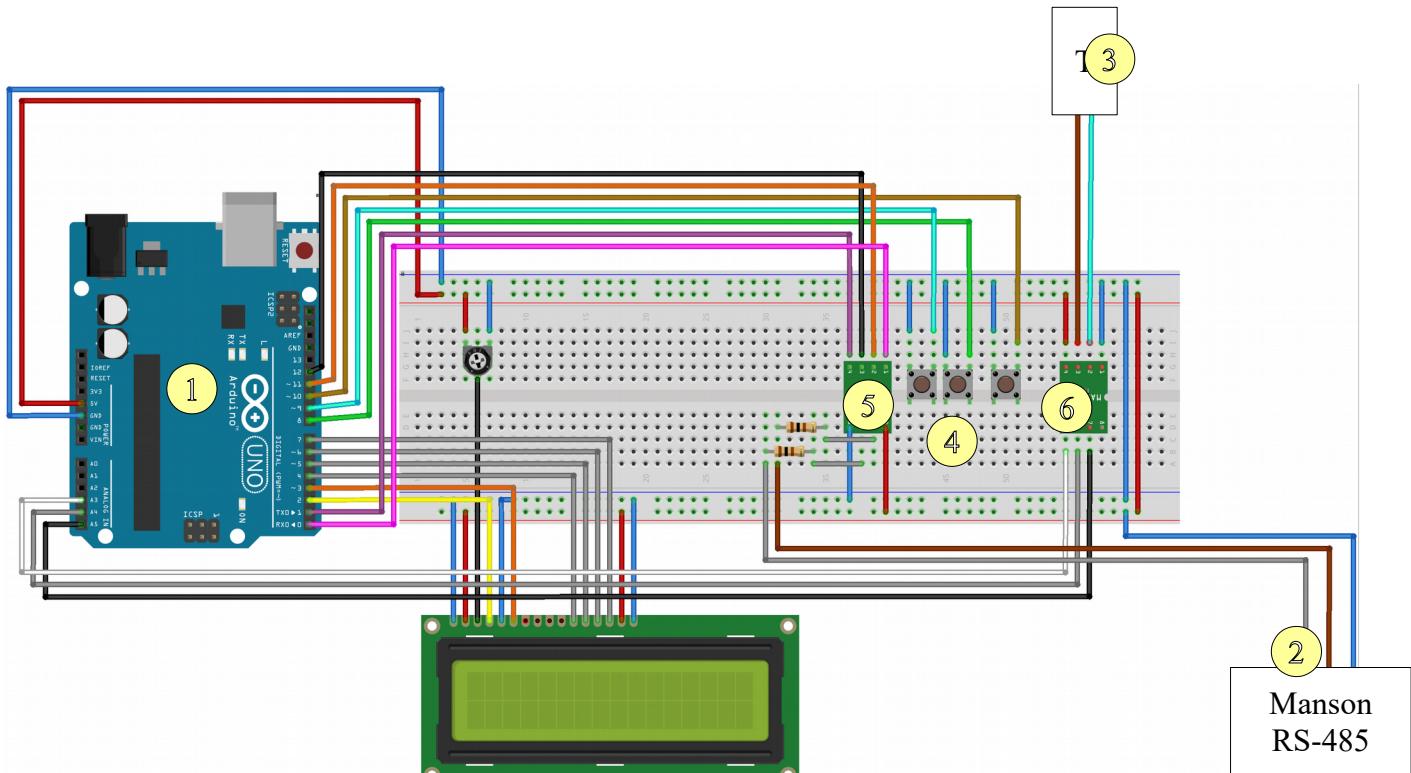
Po zsumowaniu składników p, i, d regulatora otrzymujemy sygnał wyjściowy r .

7. Wartość r jest ograniczona do przedziału od 0 do 100. Ułatwia to późniejszą implementację do funkcji przeliczającej potrzebne napięcie wyjściowe zasilacza, wtedy wartość zwrócona przez funkcję jest wartością procentową i jest mnożona przez wcześniej ustaloną wartość napięcia maksymalnego.

Pełny kod źródłowy z plików *proj.h* i *proj.cpp*, z których pochodzą powyższe fragmenty kodu znajduje się w Załączniku nr 3.

5. Implementacja

5.1. Schemat połączeń układu



Ilustracja 10: Schemat połączeń układu stworzony z pomocą programu Fritzing

1. Płytką Arduino Uno podłączona do źródła zasilania (pominięte na schemacie dla zachowania czytelności).
2. Połączenie z wejściem RS-485 na tylnym panelu zasilacza Manson 2045. Odpowiednio połączone piny A, B oraz GND (masa).
3. Termopara typu K firmy Comark. Odpowiednie biegunki podpięte do pinów konwertera MAX6675.
4. Przyciski do sterowania programem. Od lewej są to kolejno przyciski opisane jako: "minus" (pin z numerem 9 na płytce Arduino), "plus" (pin 8) i "accept" (pin 10).
5. Przejściówka RS-485, podłączenie pinów: RO do pinu 0 na Arduino, DI: pin 1, RE: pin 11, DE: pin 12. Piny A i B połączone z odpowiadającymi wejściami w zasilaczu.
6. Konwerter MAX6675 – łączy interfejsu SPI połączone z pinami analogowymi na Arduino. SCK: pin A3, CS: pin A4, SO: pin A5 (piny A3-A5 zostały ustawione jako piny cyfrowe).

5.2. Algorytm działania programu



Ilustracja 11: Schemat blokowy działania programu

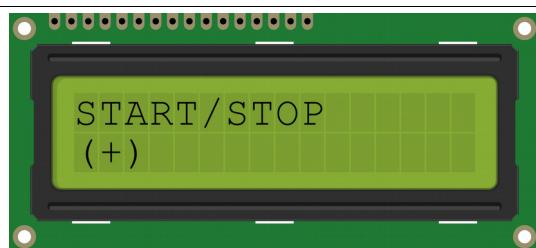
Na ilustracji 11 został przedstawiony blokowy schemat działania algorytmu programu.

Widoki opcji menu zostały przedstawione w poniższej tabeli. Numery danej opcji menu na schemacie (umieszczone w nawiasach) pokrywają się z numerami z tabeli.

Tabela Opcje menu:

Opis	Widok na wyświetlaczu
1. Ustawienie pożądanej temperatury – za pomocą przycisków "plus" i "minus" należy ustawić pożądaną temperaturę. Przytrzymanie przycisku pozwala na szybsze zmiany większych wartości. W lewym dolnym rogu wyświetlana jest aktualna temperatura (1), w prawym dolnym rogu – ustawiona temperatura (2). Wartości podane w $^{\circ}\text{C}$.	
2. Opcje zasilacza – posiada dwa podmenu, do których można dostać się naciśkając przycisk "plus". Pomiędzy opcjami podmenu można przełączać się znów za pomocą przycisku "accept".	
Podmenu wyboru opcji zasilacza	
2.1. Start sesji – po naciśnięciu przycisku "plus" należy wybrać adres zasilacza (1) za pomocą przycisków "plus" i "minus". Adres ten należy ustawić lub odczytać na zasilaczu. Ustawienia komunikacji na zasilaczu Manson 2405 można dokonać naciśkając odpowiednie przyciski panelu frontowego: SHIFT a następnie RS-232/RS-485. Zakres możliwych do wyboru adresów 0 – 31. Po naciśnięciu przycisku "accept" w pamięci zostaje zapisany adres, a do zasilacza jest wysyłana komenda odpowiedzialna za zablokowanie panelu przedniego i przejście w tryb zdalnego sterowania (sygnalizuje to ikona kłódki w dolnej części ekranu zasilacza). Domyślana wartość adresu zasilacza: 0.	
2.2. Koniec sesji – po naciśnięciu przycisku "plus" wysyłana jest komenda odpowiedzialna za zamknięcie połączenia między zasilaczem a kontrolerem oraz odblokowanie panelu przedniego zasilacza.	
Aby wyjść z podmenu należy przytrzymać "accept" przez sekundę.	

3. Rozpoczęcie pracy – aby rozpocząć pracę należy wcisnąć przycisk "plus". Aby zatrzymać proces ogrzewania należy przycisnąć przycisk "accept".

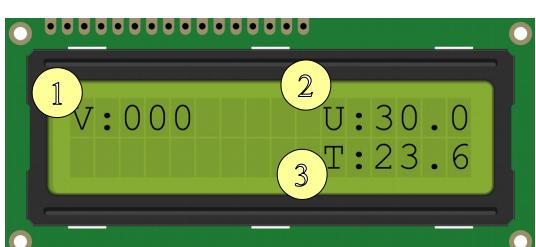


3.1. W czasie pracy urządzenia na ekranie wyświetlane są informacje:

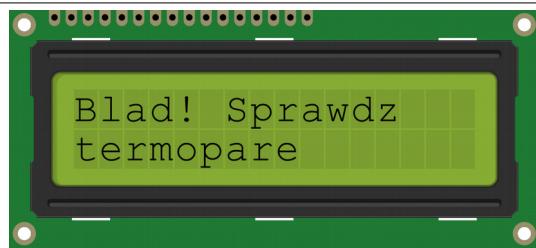
(1) Informacja o napięciu. Jest to informacja o tym jaką komendą zostanie wysłana do zasilacza. Komenda składa się z trzech cyfr, wartość pożądanego napięcia jest dziesięciokrotnie mniejsza od liczby powstałej z tych cyfr. Wartość rzeczywistą obecnego napięcia można odczytać na ekranie zasilacza.

(2) Wartość ustawionej temperatury w °C.

(3) Obecna wartość temperatury odczytana z pomocą termopary w °C.



4. Komunikat wyświetlany za każdym razem gdy odczyt temperatury jest nieprawidłowy – wartość zwracana przez funkcję `readCelsius()` jest określona jako `Nan` (*not a number*). Zgodnie z komunikatem należy sprawdzić poprawność podłączenia termopary.



Zabezpieczenia

Każde wykonanie głównej pętli `loop()` rozpoczyna się od pomiaru temperatury, następnie jest sprawdzana poprawność pomiaru. Oprogramowanie nie pozwala na podjęcie jakichkolwiek akcji i wyświetla komunikat o błędzie dopóki odczyty termopary nie będą prawidłowe.

Temperatura przekazywana jako argument do obliczeń to tak naprawdę uśredniona temperatura z ostatnich pięciu sekund. Wyniki poprzednich pomiarów przechowywane są w tablicy dziesięcioelementowej (bo pomiar temperatury odbywa się co 0.5 s).

Należy również wspomnieć o tym, że podczas pracy regulatora w pętli sprawdzany jest warunek (na schemacie z ilustracji 11 opisany jako warunek bezpieczeństwa), który ma zapobiegać niekontrolowanemu podgrzewaniu. Jeżeli jest spełniony w pętli zagnieżdzonej wyświetlany jest komunikat o błędzie dopóki nieprawidłowość nie zostanie usunięta.

```
while( isnan(thermocouple.readCelsius()) || (temperature > tDesired.value() + 10) ) {
    // wyświetlanie komunikatu o błędzie, wyłączenie zasilacza
}
```

Powyższa pętla zostanie wykonana, gdy wartość odczytywanej temperatury będzie nieprawidłowa. Funkcja `isnan()` sprawdza czy parametr do niej podany przyjmuje wartość `NaN` (*not a number*) i jeżeli tak jest zwraca wartość `true`. Odczyt temperatury przyjmie taką wartość gdy termopara zostanie fizycznie odłączona – nie ma wtedy możliwości śledzenia zachowania układu. Jeżeli w takiej sytuacji ogrzewanie nie zostało przerwane mogłoby to prowadzić do niekontrolowanego wzrostu temperatury i w efekcie spowodowania uszkodzeń. Dlatego natychmiast po spełnieniu pierwszego warunku do zasilacza zostaje wysłana komenda

odpowiedzialna za odłączenie zasilania (i jest ona powtarzana do momentu uzyskania poprawnych odczytów z termopary).

Drugi warunek pętli (`temperature > tDesired.value() + 10`) jest dodany aby upewnić się czy w układzie nie panuje zbyt wysoka temperatura – w tym przypadku o 10 stopni większa od temperatury pożąданiej `tDesired`. Warunek może zostać spełniony np. gdy na układ zadziała otoczenie lub nastawy regulatora PID zostaną dobrane niepoprawnie i oscylacje wokół wartości zadanej będą zbyt duże. W takiej sytuacji również wyłączany jest zasilacz aby nie ogrzewać dodatkowo układu, którego temperatura już osiągnęła zbyt wysoką wartość.

Komunikacja z zasilaczem

Aby zasilacz działał zgodnie z zamysłem użytkownika pragnącego sterować nim za pomocą interfejsu RS-485 należy wysyłać do niego odpowiednie komendy. Wypis wszystkich komend i ich działania znajduje się w instrukcji zasilacza Manson 2405.

Pojedyncza komenda wysyłana do zasilacza składa się z ciągu bajtów zakończonych bajtem CR – *Carriage return*. Jest to specjalny znak z tablicy ASCII o wartości 0x0D (w zapisie szesnastkowym). Budowa pojedynczej komendy przedstawia się następująco:

<komenda><adres><wartości><CR>

Komenda – 4 pierwsze znaki określają jakie działanie ma zostać wykonane, np. "SESS" – rozpoczęcie sesji zdalnego sterowania zasilaczem.

Adres – adres przypisany do zasilacza (można go samodzielnie ustawić korzystając z panelu przedniego zasilacza).

Wartości – dodatkowe bajty polecenia, jeżeli są wymagane, np. do komendy ustawiania napięcia musimy podać jego wartość.

Komendy zastosowane w projekcie (w poniższych przykładach przyjęto że adres zasilacza ustawiony jest na "01"):

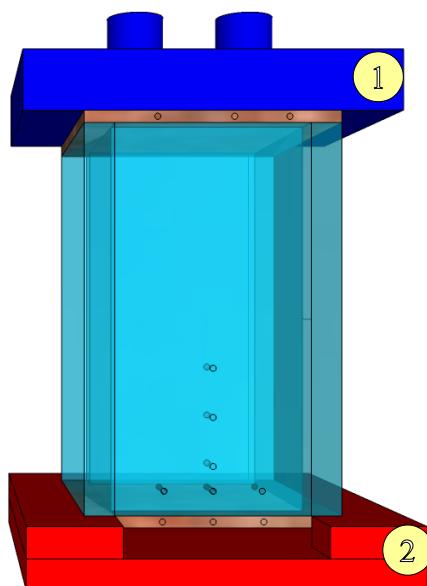
- "SESS01"<CR> - rozpoczyna sesję zdalnego sterowania i wyłącza możliwość sterowania zasilaczem za pomocą panelu przedniego.
- "ENDS01"<CR> - kończy sesję zdalnego sterowania.
- "CURR01040"<CR> - ustawia natężenie prądu, wartość tego natężenia podaje się na trzech ostatnich bajtach polecenia – "040" oznacza wartość 0.40 A.
- "VOLT01060"<CR> - analogicznie ustawia napięcie. Różnica polega na tym, że pierwszy bajt wartości napięcia oznacza dziesiątki woltów więc "060" odpowiada napięciu 6.0 V.

Opisane komendy są wysyłane za pomocą wcześniej opisanej metody `sendCommand()` z klasy `Manson2405`.

Pełny kod głównego szkicu `thermostat.ino` znajduje się w Załączniku nr 1.

5.3. Przeprowadzone testy, wyniki

Testy poprawności działania skonstruowanego urządzenia przeprowadzono na układzie przedstawionym na poniższej ilustracji (*Ilustracja 12*):



Ilustracja 12: Schemat obiektu testowego

Układ ten zbudowany jest z dwóch płyt miedzianych położonych na dolnej i górnej ścianie prostopadłościennego zbiornika wypełnionego wodą. Na górnej płycie chłodnica (1) utrzymywała stałą temperaturę 18 °C. Biegunki zasilacza laboratoryjnego Manson 2405 zostały podłączone do grzałki (2) ogrzewającej dolną płytę (oznaczenie kolorem czerwonym).

Zadaniem projektowanego termostatu dla powyższego układu jest ogrzanie dolnej płytki do określonego poziomu i utrzymanie stałej różnicy temperatur pomiędzy ścianką górną a dolną.

Aby otrzymywać odczyty temperatur dolnej płytki umieszczono końcówkę termopary wewnętrz otworu znajdującego się w tej płytce.

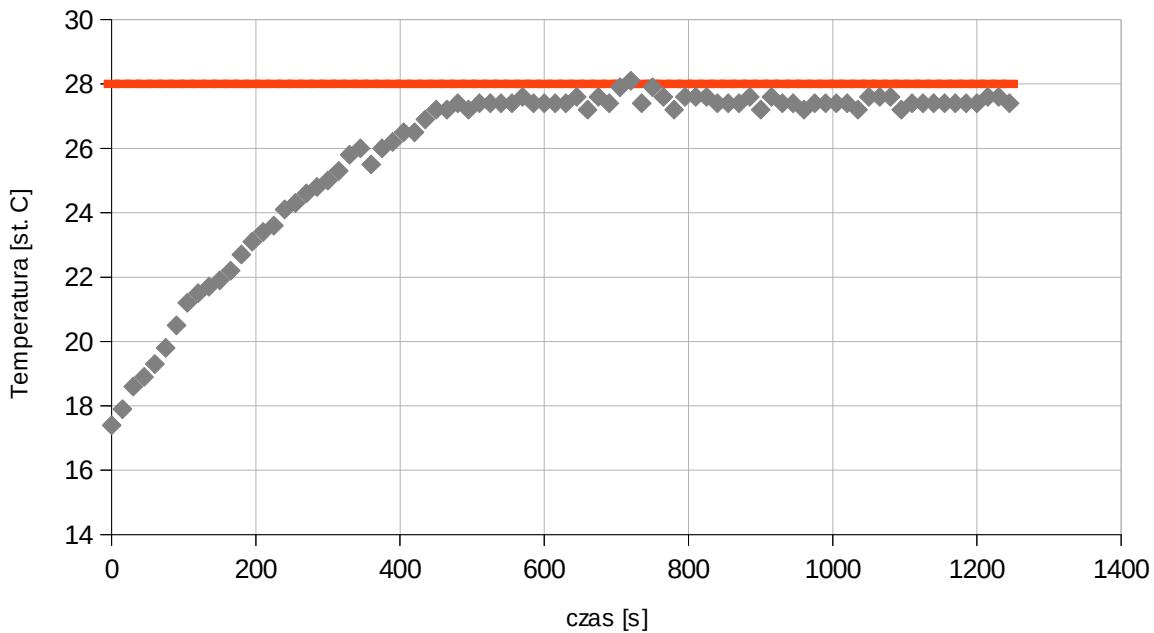
W momencie startu pracy urządzenia odczyt temperatury dla dolnej płytki wynosił 17.4 °C.

Do celów testu oprogramowanie termostatu zostało skonfigurowane w taki sposób, że maksymalne napięcie jakie mogło zostać ustawione przez zasilacz $V_{max} = 6$ V, a maksymalne natężenie prądu $I_{max} = 0.4$ A.

Moc maksymalna dostarczona przez zasilacz $P_{max} = 2.4$ W.

Temperaturę zadaną dolnej płytki wybrano za pomocą programu, została ona ustawiona na 28 °C.

Po odpowiedniej konfiguracji ustawień programu, wystartowano pracę termostatu. Wyniki jego pracy (zmiana temperatury dolnej wygrzewanej płytki w czasie) przedstawia poniższy wykres 2.



Wykres 2: Wyniki działania termostatu – zmiana temperatury wygrzewanej płytki miedzianej w czasie

Podczas testu zastosowano w oprogramowaniu następujące wartości nastaw regulatora PID:

- współczynnik proporcjonalny $K = 50$,
- współczynnik całkujący $K_i = 0.5$,
- współczynnik różniczkujący $K_D = 20$.

Zgodnie z tym co jest ukazane na wykresie, w początkowej fazie odbywało się z maksymalną możliwą mocą (wcześniej ustaloną w oprogramowaniu) 2.4 W. Gdy temperatura mierzona na miedzianej płytce zbliża się do temperatury zadanej (czerwona linia na wykresie), moc grzalki stanowi tylko ułamek mocy maksymalnej wystarczający aby utrzymać tę temperaturę na jednym poziomie. Wartość mocy jest aktualizowana w każdej sekundzie, wraz z każdym pomiarem temperatury, regulator powinien też reagować na ewentualne zakłócenia pojawiające się w układzie.

Jak widać na wykresie, układ nie powoduje przekraczenia zadanej wartości, podczas zbliżania się do ustawionej temperatury stopniowo zmniejsza moc grzania.

Na wykresie nie obserwuje się też oscylacji temperatury wokół wartości zadanej co można uznać za wynik zadowalający biorąc pod uwagę fakt, że układ miał utrzymywać stałą różnicę temperatur między górną a dolną płytka miedzianą.

Wahania temperatury w stanie ustalonym nie przekraczają +/- 1 stopnia Celsjusza.

Natomiast temperatura utrzymywana jest około 0.5 °C poniżej wartości zadanej. Dla wielu zastosowań będzie to wartość bez dużego znaczenia, jednak dla procesów wymagających większych precyzji można będzie zastosować inne wartości nastaw regulatora PID. Należy również pamiętać, że stosowana w projekcie termopara wraz z konwerterem MAX6675 daje podziałkę 0.23 °C więc dla bardziej precyzyjnych procesów należałoby rozważyć zmianę sposobu odczytu temperatury.

6. Podsumowanie

Projekt miał na celu stworzenie programowalnego termostatu utrzymującego stałą zadaną temperaturę. Zastosowanie algorytmu regulatora PID pozwoliło na uzyskanie stabilnej regulacji ograniczając ingerencję użytkownika jedynie do wprowadzenia żądanej wartości. Został zaprojektowany prosty w obsłudze interfejs wyświetlany na ekranie LCD ułatwiający obsługę układu oraz pozwalający śledzić jego pracę.

Skonstruowany w tej pracy projekt na bazie Arduino może pracować tylko z konkretną serią modeli zasilaczy jako źródła mocy do ogrzewania oraz z termoparami jednego typu jednak dzięki próbom wdrożenia paradygmatu programowania obiektowego otwarta jest możliwość zamiany podzespołów i zastosowania projektu w innym procesie. Podmiana czujnika sprowadzałaby się jedynie do zaimplementowania sposobu odczytu wartości temperatury z pomocą nowego miernika. Bardziej skomplikowana byłaby zmiana sposobu ogrzewania jednak poza tym sposób działania algorytmu programu pozostaje bez zmian.

Sprawdzenie efektywności działania algorytmu regulatora PID podczas testów wykazało osiągnięcie pożądanych rezultatów w zadowalającym stopniu. Ponadto dokładność regulacji można zwiększyć jeszcze bardziej gdy jest to wymagane w danym procesie inaczej dobierając nastawy regulatora.

W oprogramowaniu wprowadzono ograniczenia zabezpieczające przed niekontrolowanym wzrostem temperatury – w przypadku wykrycia nieprawidłowości układ ogrzewania jest wyłączany a użytkownik natychmiast jest powiadamiany o błędzie.

System ma perspektywy zastosowania i dalszego rozwijania zarówno w procesach wymagających dużej dokładności, jak i tych gdzie potrzeba precyzji regulacji jest mniejsza.

Bibliografia

- [1] *Arduino – Introduction*, www.arduino.cc/en/Guide/Introduction
- [2] *Atmel Atmega328P Datasheet* www.atmel.com/devices/ATMEGA328P.aspx
- [3] S. Monk: *Arduino dla początkujących. Podstawy i szkice*, wy.: Helion, 2014, ISBN 978-83-246-8737-4
- [4] *Arduino*, en.wikipedia.org/wiki/Arduino
- [5] A. Chłopik *Pomiary z wykorzystaniem termopar bez tajemnic*, 2007, automatykab2b.pl/technika/139-pomiary-z-wykorzystaniem-termopar-bez-tajemnic
- [6] *Termopara*, pl.wikipedia.org/wiki/Termopara
- [7] J. Kowal *Podstawy automatyki*, t. 1, wyd.: Wydawnictwa AGH, 2006, ISBN 83-7464-108-8
- [8] *Regulator PID*, pl.wikipedia.org/wiki/Regulator_PID
- [9] *Termostat*, pl.wikipedia.org/wiki/Termostat
- [10] *MAX6675 Cold-Junction-Compensated K-Thermocouple to Digital Converter Datasheet*, cdn-shop.adafruit.com/datasheets/MAX6675.pdf
- [11] *LABORATORY GRADE REMOTE PROGRAMMING SWITCHING MODE DC regulated Power Supplies SDP Series SDP – 2210 / 2405 / 2603 User Manual*,
www.manson.com.hk/getimage/index/action/images/name/51498ed4ec81f.pdf
- [12] *LiquidCrystal Library Reference*, www.arduino.cc/en/Reference/LiquidCrystal
- [13] *SoftwareSerial Library Reference*, www.arduino.cc/en/Reference/softwareSerial
- [14] *Arduino library for interfacing with MAX6675 thermocouple amplifier*,
[github.com/adafruit\(MAX6675-library](https://github.com/adafruit(MAX6675-library)
- [15] *Button Library*, playground.arduino.cc/Code/Button
- [16] J. Grębosz: *Symnfonie C++* t. 2, wyd.: Oficyna Kallimach, 1994, ISBN 83-901689-0-1