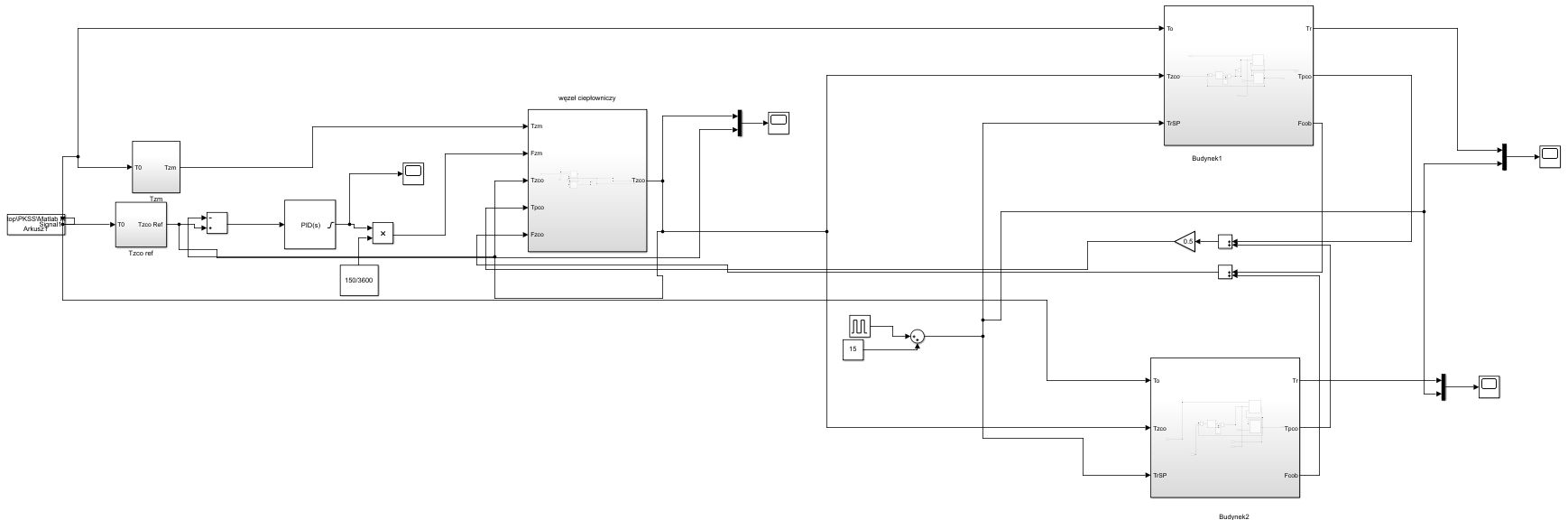
**Budowa rozproszonego systemu modelowania i sterowania instalacją CO na przykładzie dystrybucji ciepła w budynkach AGH**

1. **Model ogólny**

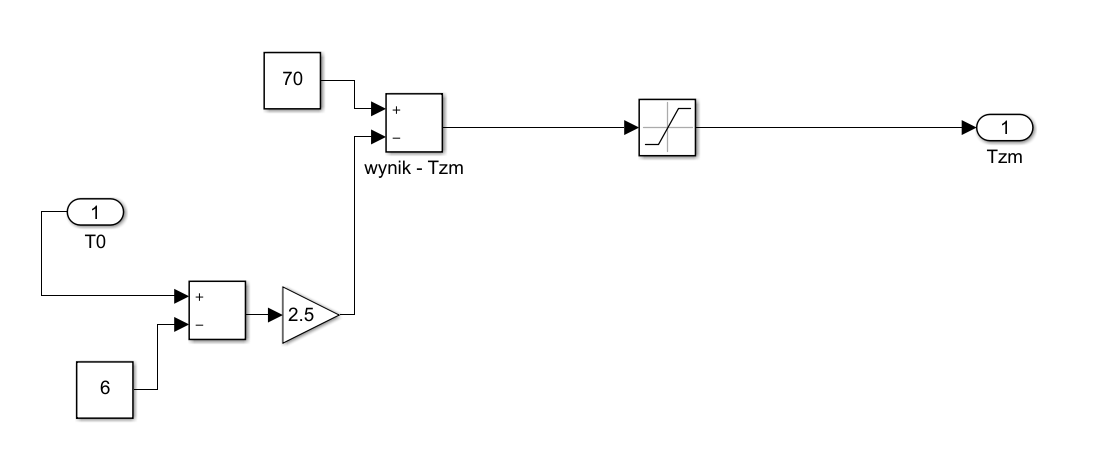
Model ogólny przedstawia subsystemy opisujące fizykę wymienników ciepła, budynków oraz kaloryferów. Odpowiednie wyjścia temperaturowe podłączono do bloczka Scope, aby móc obserwować ich przebiegi. Na modelu znajduje się także układ odpowiedzialny za regulację temperatur, który w zależności od uchybu temp. wody wypływającej z wymiennika, steruje dopływem wody do wymiennika. Wyjście PID’a jest w zakresie od 0 do 1 i jego wartość jest mnożona przez maksymalny przepływ. Na modelu zostały umieszczone także 2 subsystemy, które odpowiadają za zależność temperatury wody zasilającej oraz temperatury wody wypływającej z wymiennika.



***Rys 1****. Model ogólny*

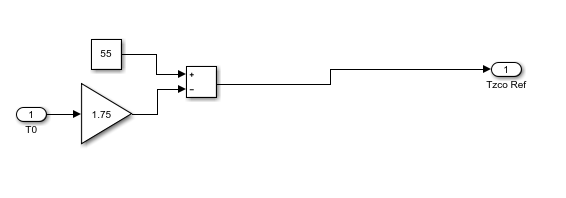
Poniżej przedstawiono modele opisujące zależność temperatury wody zasilającej oraz temperatury wody wypływającej z wymiennika, według poniższych wzorów (1) i (2)





***Rys 2****. Tzm – temperatura wody zasilającej*

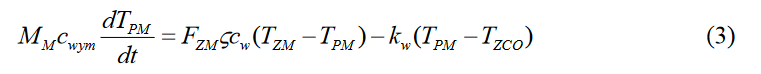


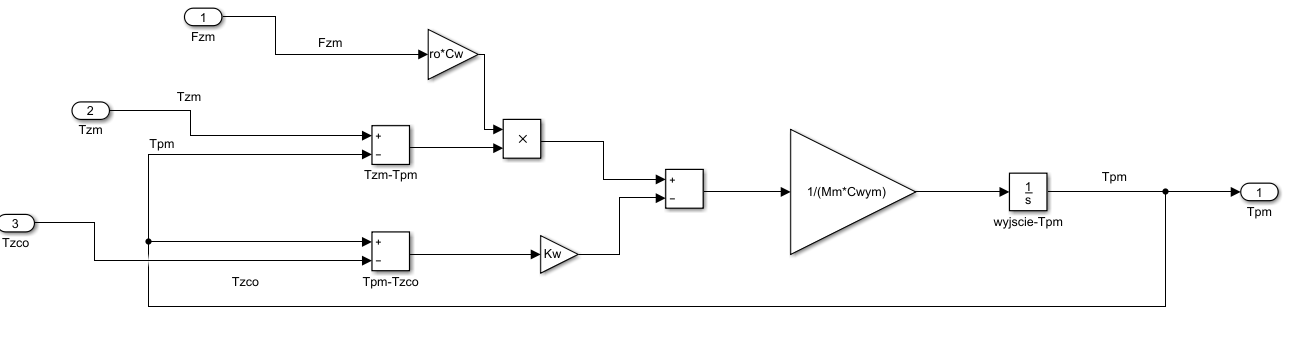


***Rys 3.*** *Tzco ref- referencyjna temperatury wypływającej z wymiennika*

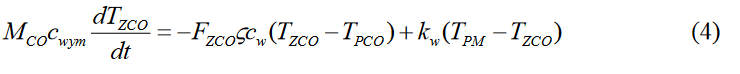
1. **Model wymiennika ciepła**

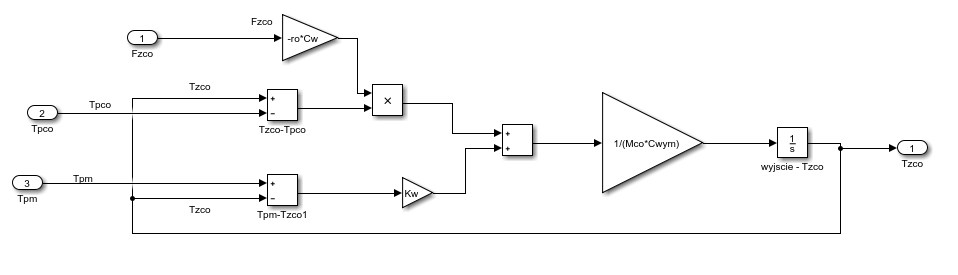
Model wymiennika ciepła został zrealizowany na podstawie równań (3) oraz (4), które opisują zależność aktualnej temperatury wypływającej z wymiennika oraz temperatury wody powracającej do wymiennika od innych temperatur i parametrów układu.





***Rys 4****. Wymiennik ciepła 1*

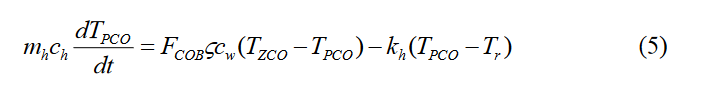


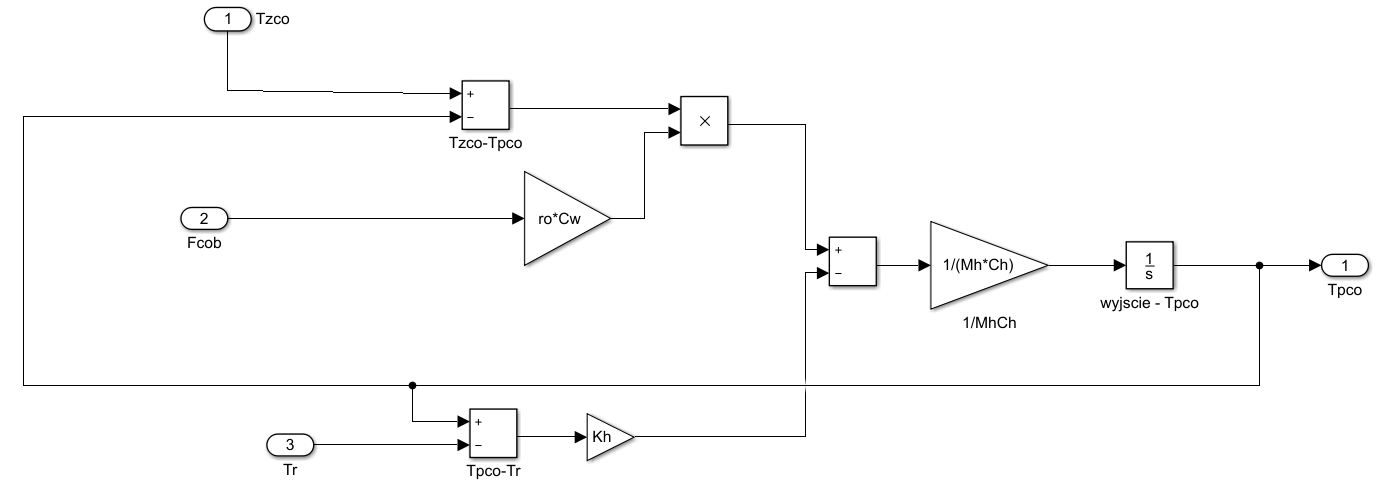


***Rys 5.*** *Wymiennik ciepła 2*

1. **Model kaloryferów**

Kaloryfery w danym budynku są rozpatrywane jako jeden zbiorczy kaloryfer. Model opisujący jego fizykę odnosi się do wzoru (5) i przedstawia zależność temperatury wody powrotnej od przepływu wody dopływającej do budynku, oczekiwanej temperatury w budynku, wody wypływającej z wymiennika, a także innych parametrów układu.

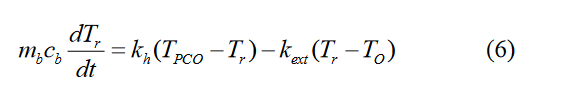


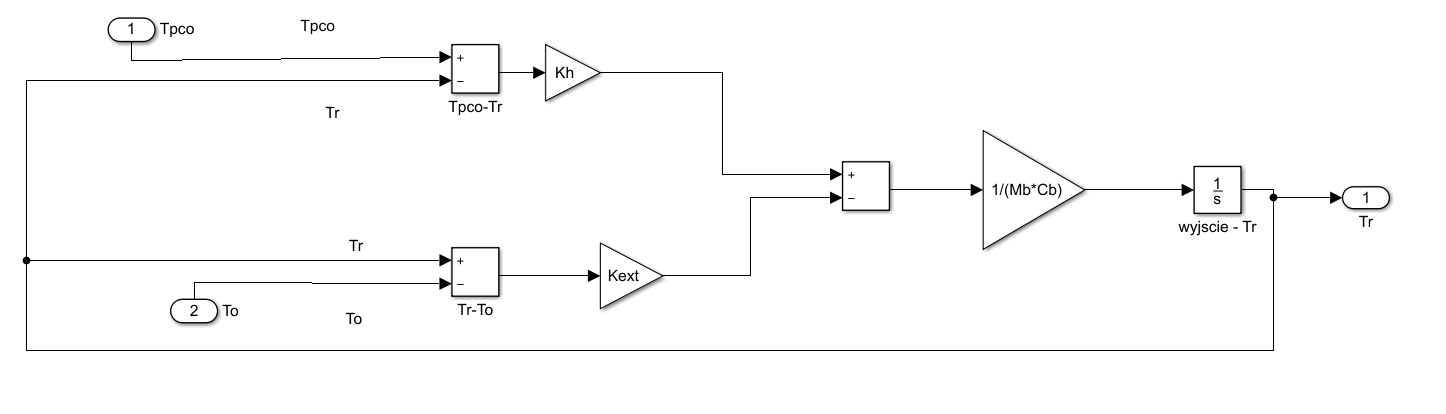


***Rys 6.*** *Model kaloryfera*

1. **Model budynku**

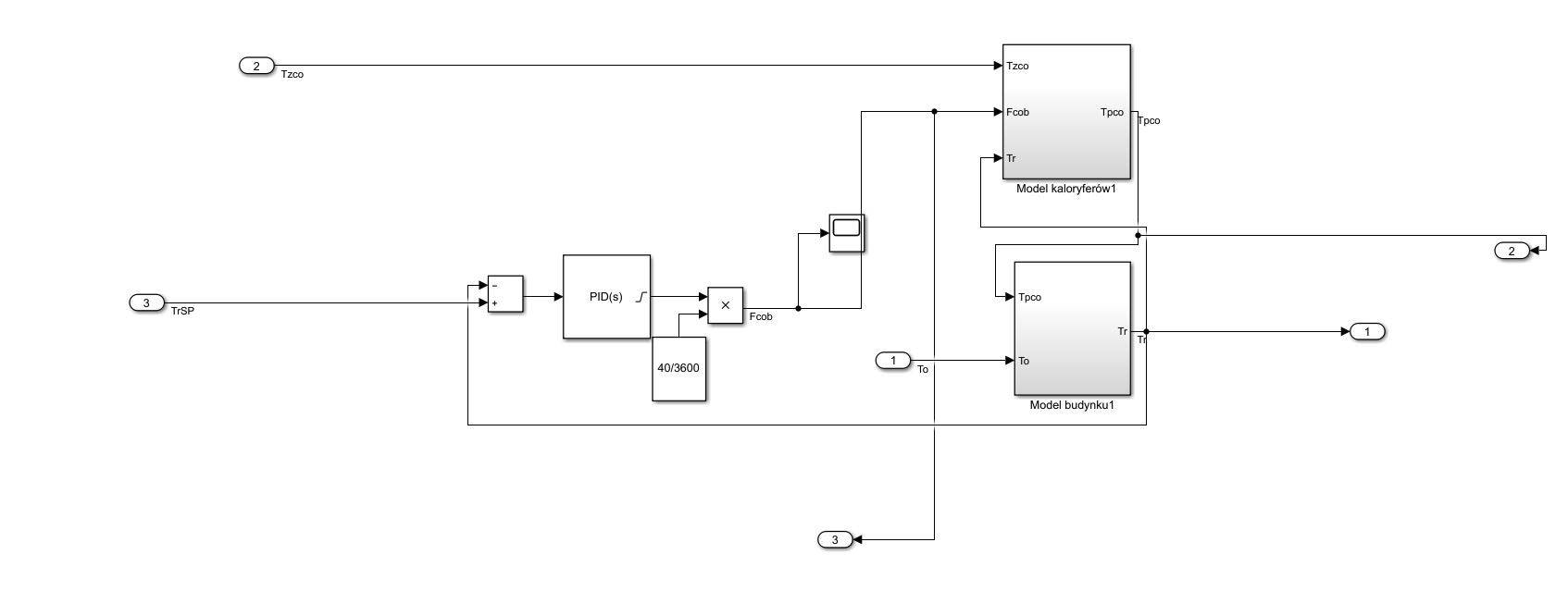
Model budynku uwzględnia jedynie powietrze znajdujące się w jego wnętrzu, został opisany równaniem (6), które przedstawia zależność temperatury panującej w budynku od temperatury otoczenia, temperatury wody powracającej oraz innych parametrów.



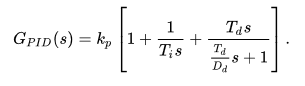


***Rys 7.*** *Model budynku*

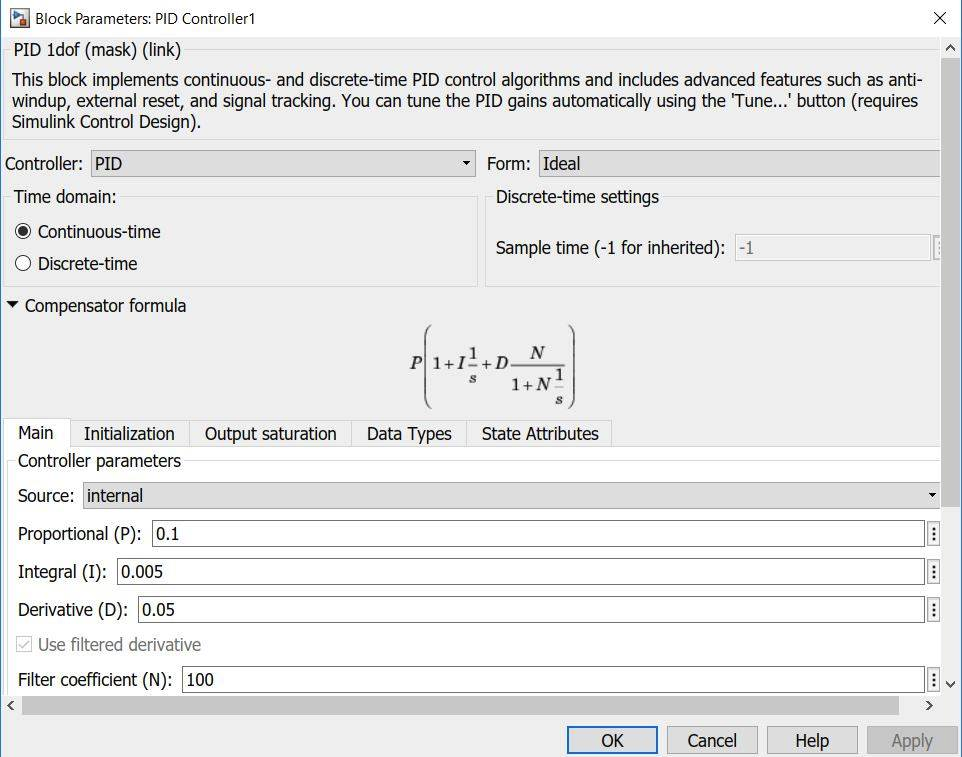
1. **Model regulacji przepływu budynku**

****

Regulator PID działa według wzoru:

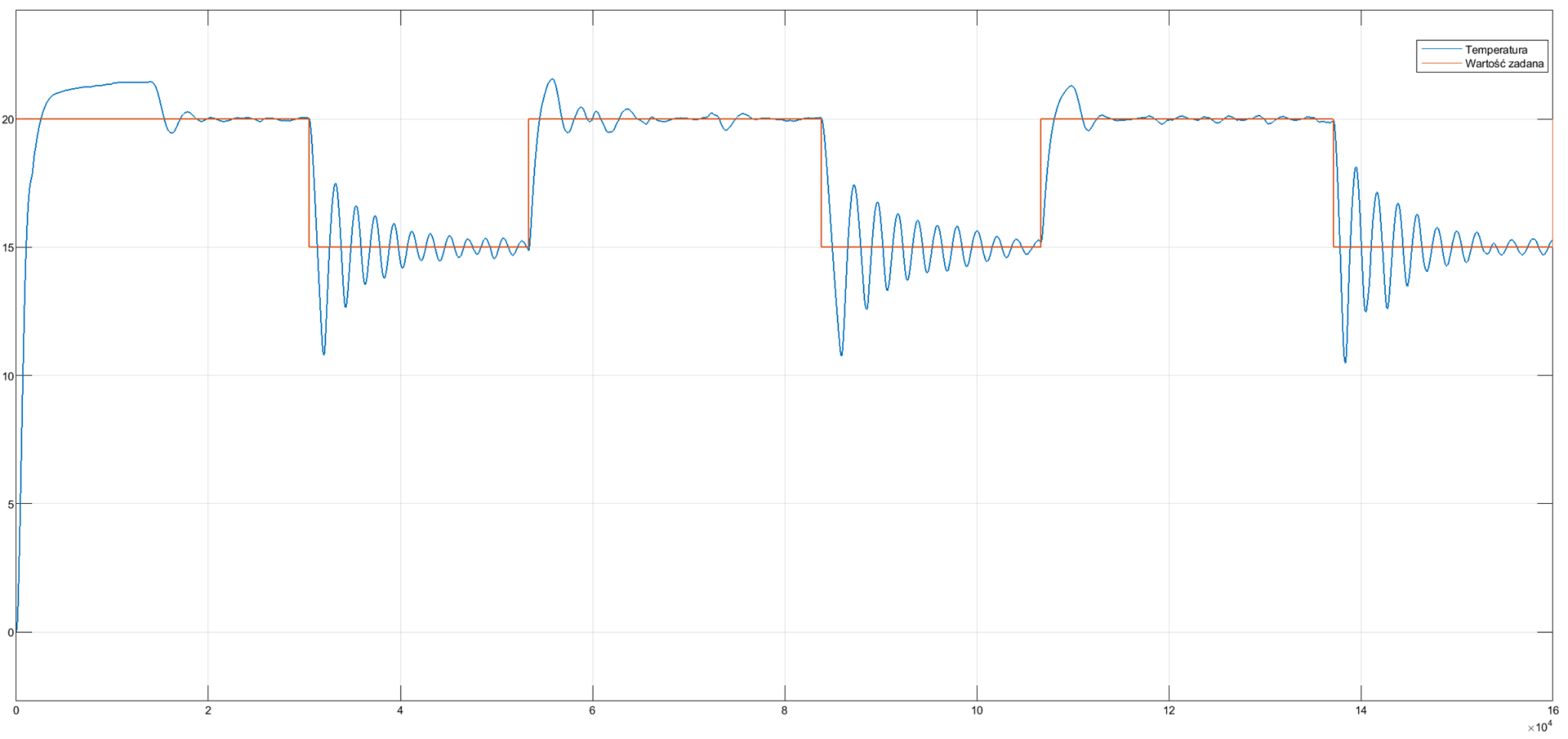
** (7)**

Parametry regulatora ustawiono następująco:



1. **Rezultaty**

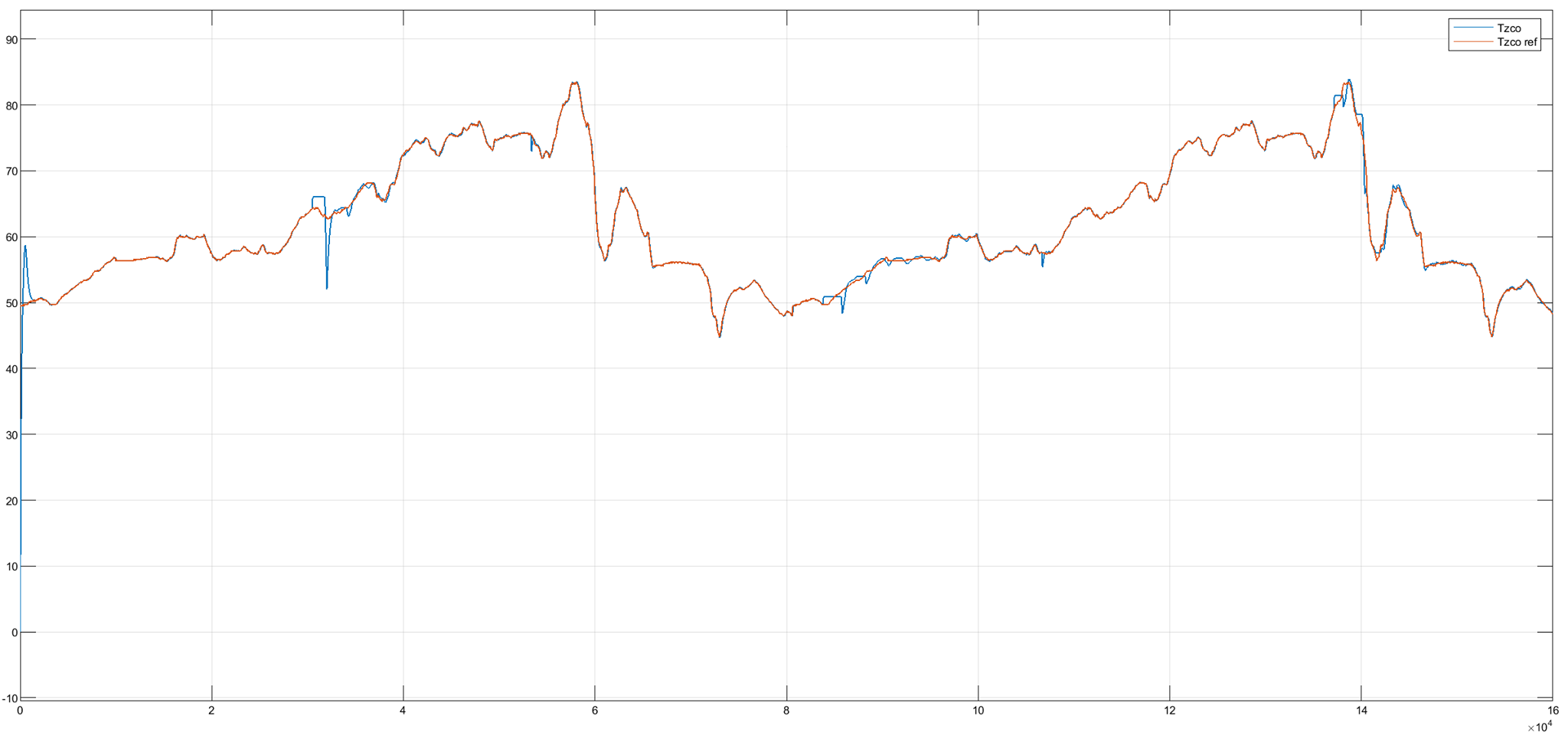
Efekty działania modelu przedstawiono poniżej



***Rys 8****. Wykres przedstawiający przebieg temperatury w budynku oraz jej wartości zadanej*

*Legenda:*

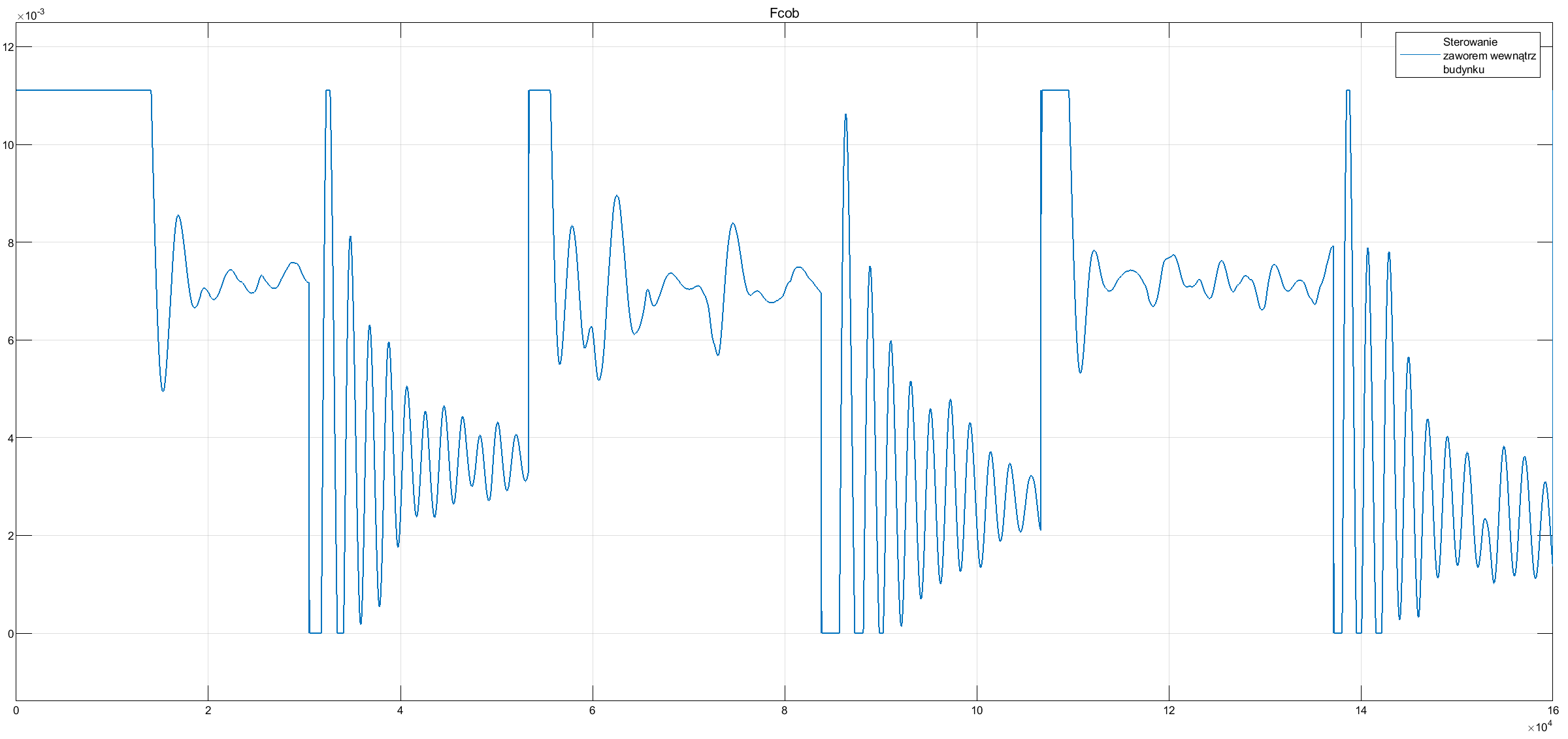
* ***niebieski*** *- current value*
* ***czerwony*** *- set point value*



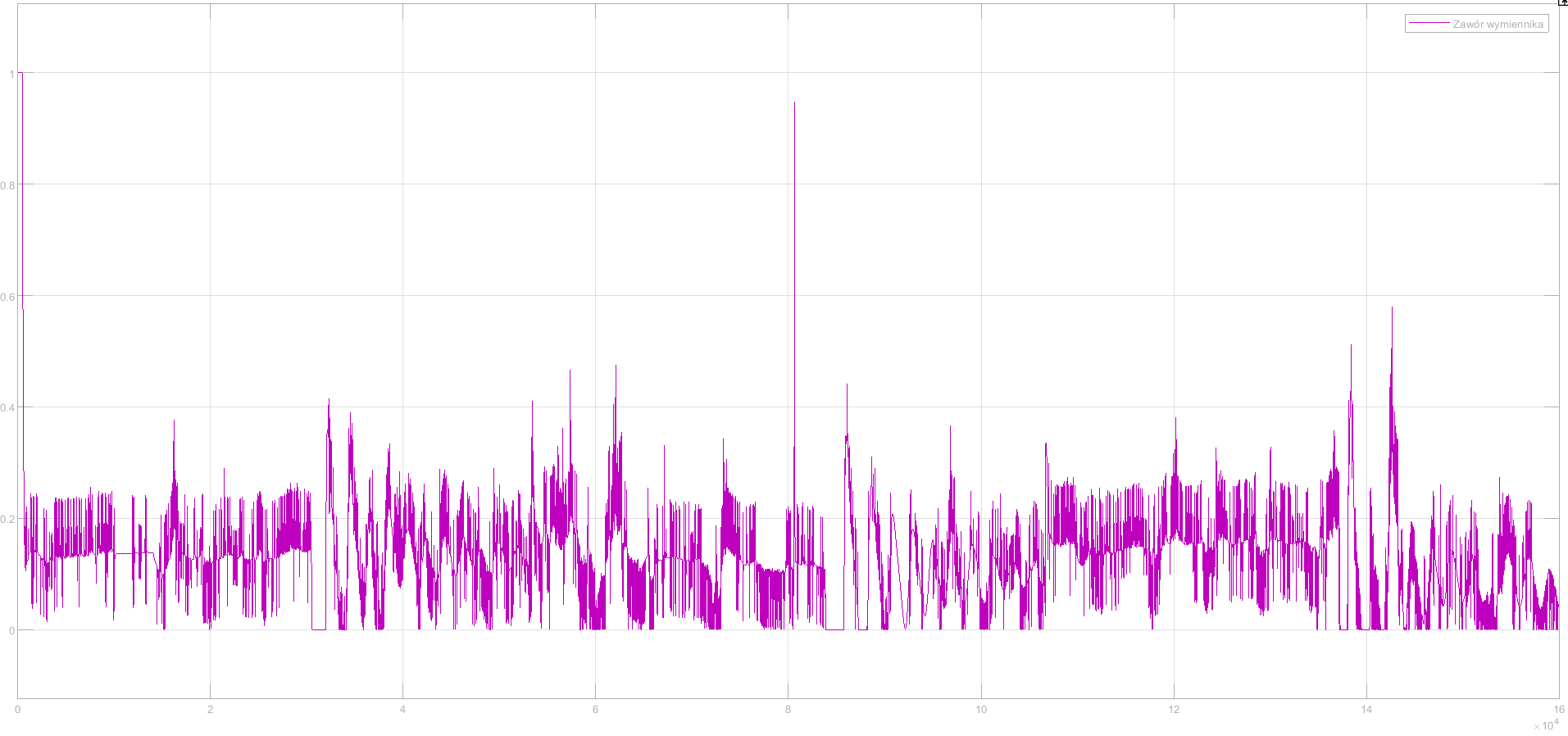
***Rys 9****. Wykres przedstawiający przebieg temperatury zasilania instalacji CO Tzco,oraz jej wartość zadaną*

*Legenda:*

* ***niebieski*** *- current value*
* ***czerwony*** *- set point value*

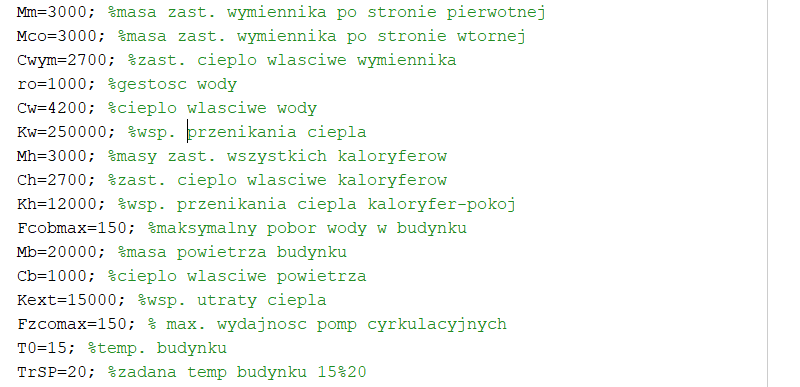


***Rys 9****. Wykres przedstawiający przebieg sterowania zaworem wewnątrz budynku*



***Rys 10****. Wykres przedstawiający przebieg sterowania zaworem wymiennika ciepła*

Stałe wprowadzone do modelu z workspace:



# 2. Implementacja w języku C

* Main

|  |
| --- |
| int\_T main(int\_T argc, const char \*argv[])  {  finale\_initialize();  while ((rtmGetErrorStatus(rtM) == (NULL)) && !rtmGetStopRequested(rtM))  {  rt\_OneStep();  }  return 0;  } |

* rt\_OneStep

|  |
| --- |
| void rt\_OneStep(void)  {  static boolean\_T OverrunFlag = false;  if (OverrunFlag) {  rtmSetErrorStatus(rtM, "Overrun");  return;  }  OverrunFlag = true;  finale\_step();  OverrunFlag = false;  } |

Funkcja *finale\_step()* zawiera wykonanie pojedynczego kroku i została opisana wyżej. Wywołuje ona również solver *ode3* - *Runge-Kutta*.

* Temperatura wody zasilającej



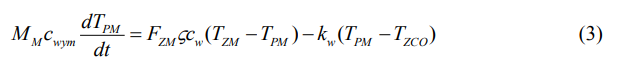
|  |
| --- |
| Tzm= 70.0 - (T0[T0counter] - 6.0) \* 2.5; |

* *Temperatura wody wypływającej z wymiennika*

**

|  |
| --- |
| Tzco = 55 - 1.75 \* T0[T0counter]; |

* Wymiennik ciepła 1 - Temperatura wody powracającej do wodociągów



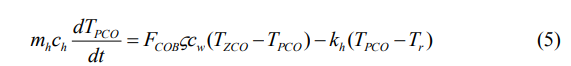
|  |
| --- |
| Gain2\_tmp = (rtX.Tpm - rtX.Tzco) \* 250000.0;  Tpm= rtDW.Gain2 = (Fcob\_i \* 0.0416 \* 4.2E+6 \* (rtDW.Saturation - rtX.Tpm) - Gain2\_tmp) \* 1.234E-7;  rtDW.Saturation=Tzm (po saturacji) |

* Wymiennik ciepła 2 - Temperatura wody wypływającej z wymiennika



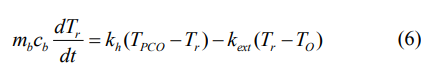
|  |
| --- |
| Tzco= rtDW.u = ((rtX.Tzco - (rtX.Tpco1 +rtX.Tpco2) \* 0.5) \* ((rtb\_Fcob + Fcob\_i) \* -4.2E+6) + Gain2\_tmp) \* 1.234E-7; |

* model kaloryfera - Temperatura wody powracającej do wymiennika

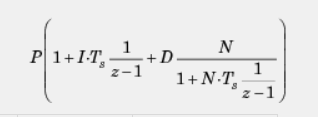


|  |
| --- |
| uMhCh\_tmp = (rtX.Tpco1 - rtX.Tr1) \* 12000.0;  Tpco1= rtDW.uMhCh = ((rtX.Tzco - rtX.Tpco1) \* (4.2E+6 \* rtb\_Fcob) - uMhCh\_tmp) \* 1.23E-7;  Gain2\_tmp = (rtX.Tpco2 - rtX.Tr2) \* 12000.0;  Tpco2= ((rtX.Tzco - rtX.Tpco2) \* (4.2E+6 \* Fcob\_i) - Gain2\_tmp) \* 1.23E-7; |

* model budynku - Temperatura uśredniona pomieszczeń



|  |
| --- |
| uMhCh\_tmp = (rtX.Tpco1 - rtX.Tr1) \* 12000.0;  Tr1=rtDW.Gain7\_c = (uMhCh\_tmp - (rtX.Tr1 - T0[T0counter] ) \* 1500.0) \* 5.0E-8;  Gain2\_tmp = (rtX.Tpco2 - rtX.Tr2) \* 12000.0;  Tr2=rtDW.Gain7 = (Gain2\_tmp - (rtX.Tr2 - T0[T0counter]) \* 15000.0) \* 5.0E-8; |



* PID sterujący przepływem z wymiennika

Ub

|  |
| --- |
| Ub = (((Tzco - rtX.Tzco) + rtX.Integrator\_um) + rtDW.Derivative\_um) \* P;  Integrator\_um=rtDW.IntegralGain = (Tzco - rtX.Tzco) \* I;  rtDW.Derivative\_um = ((Tzco - rtX.Tzco) \* D - rtX.Filter\_CSTATE) \* 100.0;  rtX.Filter\_CSTATE=0 |

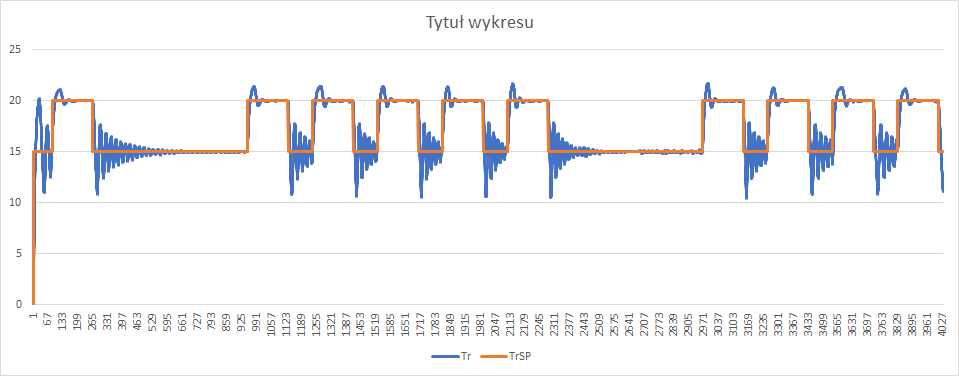
* PID sterujący przepływem wody do budynku

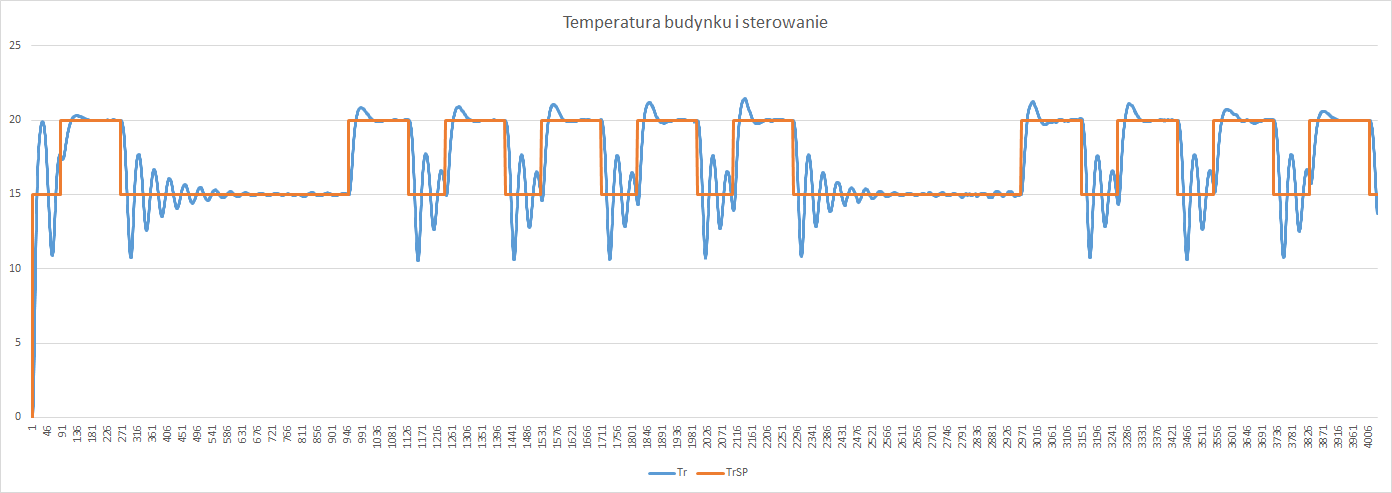
|  |
| --- |
| Ub1 = (((TrSp[T0counter] - rtX.Tr1) + rtX.Integrator\_ub1) + rtDW.Derivative\_ub1) \* P;  rtX.Integrator\_ub1=rtDW.IntegralGain\_j = (TrSp[T0counter] - rtX.Tr1) \* I;  rtDW.Derivative\_ub1 = ((TrSp[T0counter] - rtX.Tr1) \* D - rtX.Filter\_CSTATE\_n) \* 100.0;  rtX.Filter\_CSTATE\_n=0  ------------------------------------------------------------------------------------------------------------------------  Ub2= (((TrSp[T0counter] - rtX.Tr2) + rtX.Integrator\_ub2) + rtDW.Derivative\_ub2) \* P; |

* Funkcja wczytująca z pliku dane temperaturowe

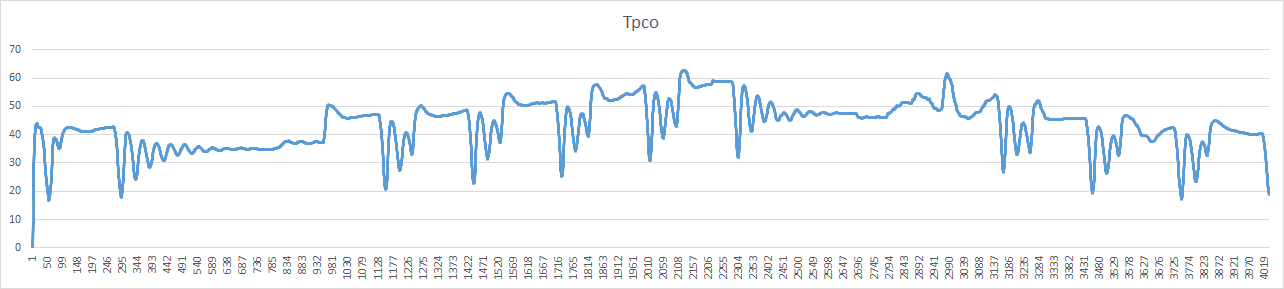
|  |
| --- |
| while((line=fgets(buffer,sizeof(buffer),pFile))!=NULL)  {  record=strtok(line,";");  while(record!= NULL)  {  cnt++;  if(cnt%2==0)  {  if(record[1]==',')  {  record[1]='.';  }  if(record[2]==',')  {  record[2]='.';  }  T0[cnt/2]=atof(record);  printf("\tT0 %f \t",T0[cnt/2]);  }  record= strtok(NULL,";");  }  }  //odczyt TrSP  cnt=0;  pFile=fopen("Temperatura\_w\_budynku.csv","r");  if(pFile==NULL)  {  printf("error");  }  while((line=fgets(buffer,sizeof(buffer),pFile))!=NULL)  {  record=strtok(line,";");  while(record!= NULL)  {  cnt++;  if(cnt%2==0)  {  if(record[1]==',')  {  record[1]='.';  }  if(record[2]==',')  {  record[2]='.';  }  TrSp[cnt/2]=atof(record);  printf("\tTrSp %f \t",TrSp[cnt/2]);  }  record= strtok(NULL,";");  }  }  fp=fopen("Wyniki.csv","w+");  fprintf(fp,"T0;Tr;Tpco;Tpm;Tzco;Tzm;Fcob\_i;rtb\_Fcob;TrSP"); |

**Wyniki**

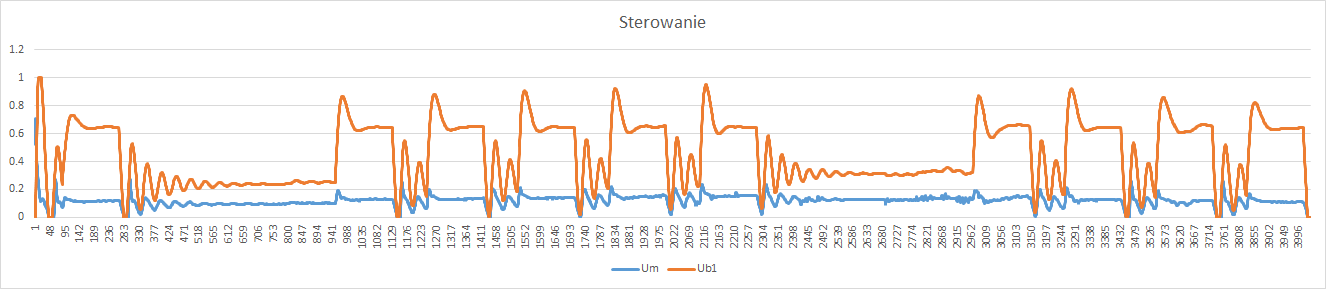
**



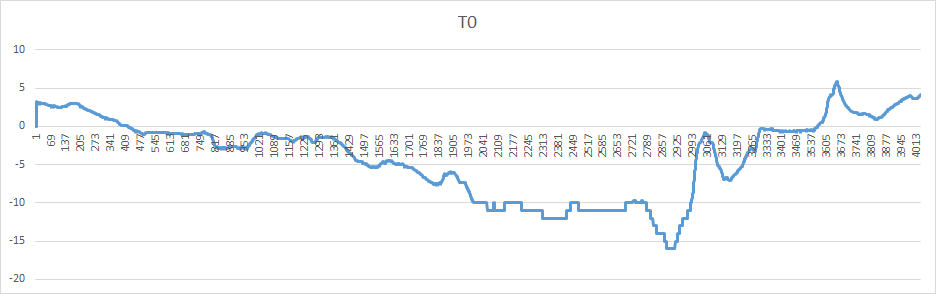
***Rys 2.1****. Wykres przedstawiający przebieg temperatury Tr,oraz jej wartość zadaną z modelu i z kodu w C.*



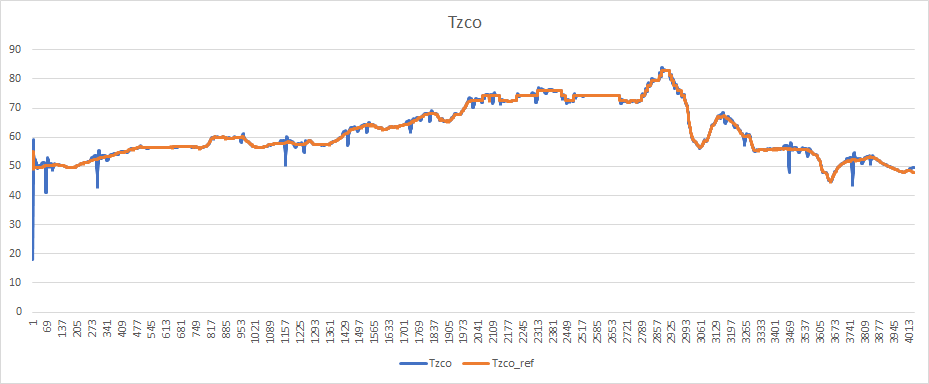
***Rys 2.2****. Wykres przedstawiający przebieg temperatury powracającej do wymiennika*



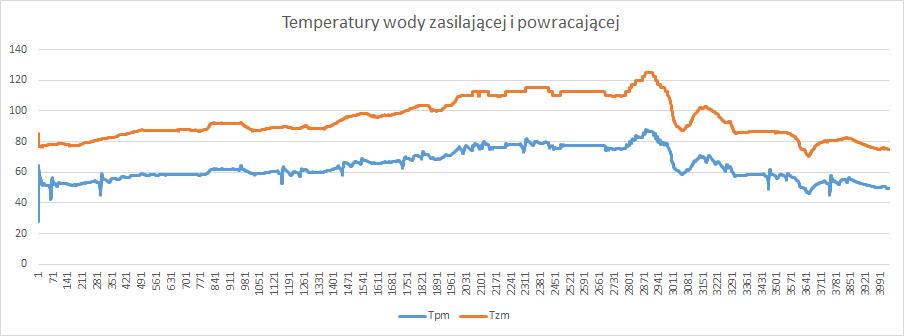
***Rys 2.3****. Wykres przedstawiający przebieg sterowania zaworem wymiennika ciepła Um oraz zaworem wewnątrz budynku Ub1*



***Rys 2.4****. Wykres przedstawiający przebieg temperatury zewnętrznej T0*

**

***Rys 2.5****. Wykres przedstawiający przebieg temperatury wypływającej z wymiennika - Tzco, oraz jej wartość zadaną*

**

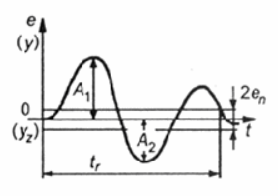
***Rys 2.6****. Wykres przedstawiający przebieg temperatury zasilającej i powracającej do wymiennika*

**Wskaźniki jakości**

* **Przeregulowanie**

Przeregulowaniewyznaczono jako stosunek kolejnych maksymalnych amplitud uchybu na podstawie poniższego wzoru (7):

(7)



***Rys.2.7****. Interpretacja graficzna przeregulowania i czasu regulacji*

Dla idealnego modelu wartość przeregulowania wynosi

X = 31,25 %

Dla symulacji zaimplementowanej w C:

X = 35,71 %

Obserwujemy że wartość wskaźnika jakości w przypadku kodu w C a więc nieidealnego modułu, jest nieznacznie gorsza o około 4%, co odpowiada naszym przypuszczeniom.

* **Czas przeregulowania**

**Czas przeregulowania,** jest to czas po jakim uchyb regulacji jest w sposób trwały mniejszy od założonej wartości Δ. Dla tego przypadku przyjęliśmy 10%.

**Dla symulacji w C:**

Różnica czasów między skokiem wartości zadanej a punktem gdy błąd względny jest mniejszy od 10% to 311-267 = 44 próbek. Każda próbka ma 5 minut, więc czas przeregulowania to około 3,5 h.

**Dla idealnego modelu:**

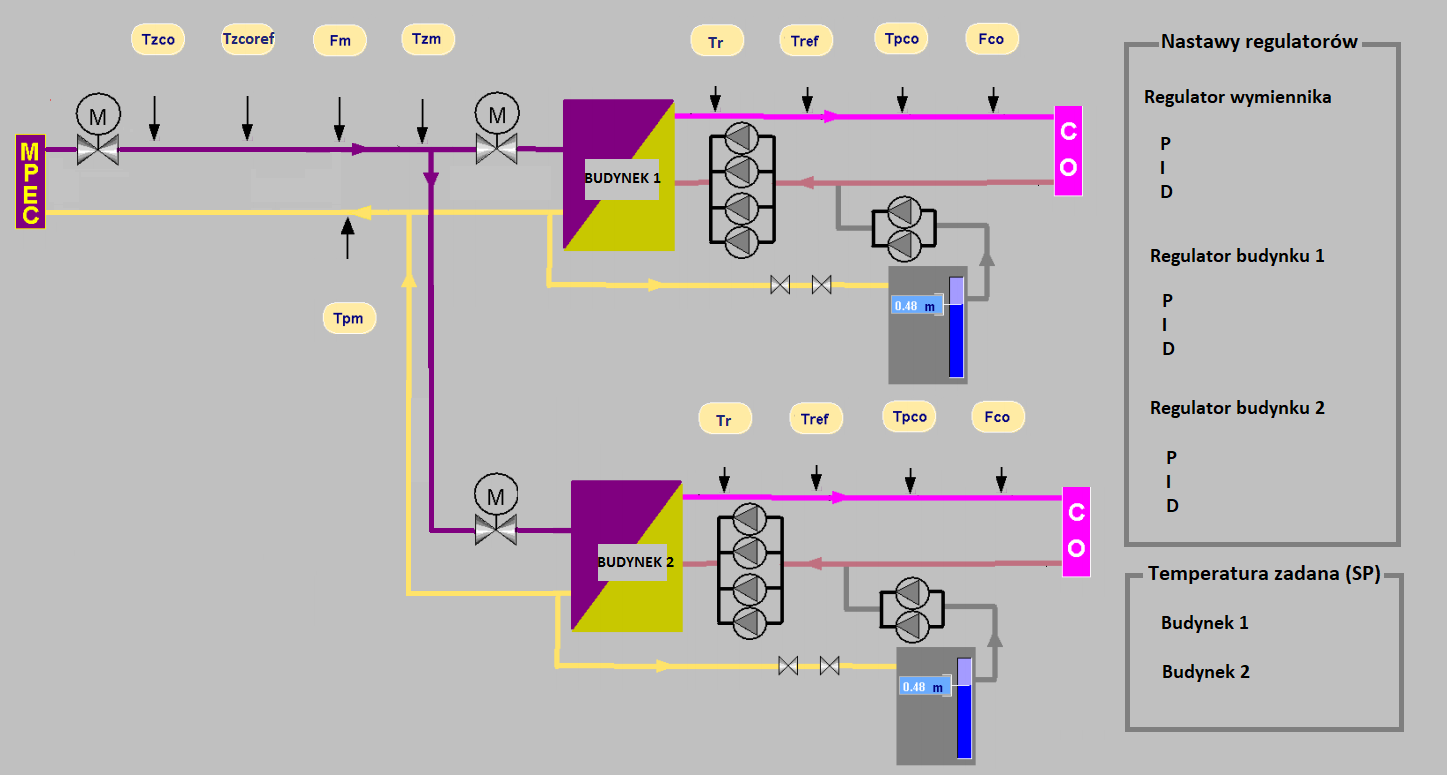
Do tego celu wykorzystaliśmy wykres z pierwszej części. Analizując go przyjęliśmy, że ta wartość wynosi około 3 godziny.

# 3. Graphical User Interface

Do stworzenia GUI wykorzystano język Python oraz bibliotekę Tkinter, która daje możliwości łatwej obsługi panelu graficznego.

### 3.1. Wizualizacja

Aby wizualizacja danych odpowiadała stworzonemu modelowi, zmieniono obraz tworzący tło aplikacji, dostosowując go do analizowanych danych i wprowadzanych przez operatora wartości. Jego wygląd przedstawiono poniżej.



***Rys 3.1****. Panel wczytujący wartości podane przez użytkownika*

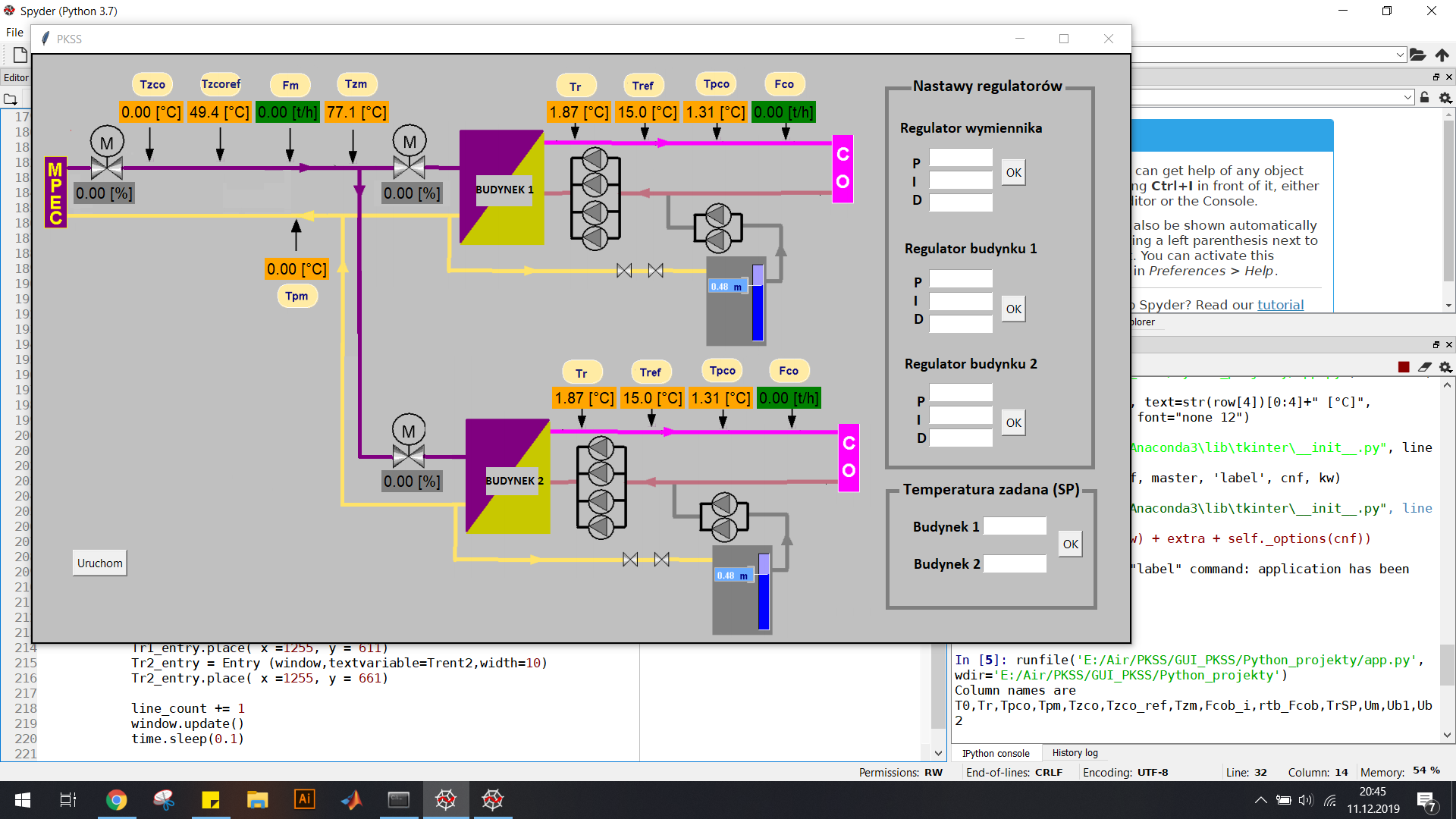
### 3.2. Wczytywanie nastaw regulatorów i wartości zadanej temperatury

Pierwszym elementem programu było wczytanie nastaw regulatorów PID. W celu wczytywania wartości nastaw PID-ów oraz zadanej temperatury w budynkach, użyto elementów *entryBox*. Usytuowano je w odpowiednich miejscach za pomocą funkcji place(x,y), której parametrami są współrzędne lewego górnego piksela.

|  |
| --- |
| #Nastawy PID - entryBox  P1 = Entry (window,textvariable=Pent1,width=10)  P1.place( x =1184, y = 125)  I1 = Entry (window,textvariable=Ient1,width=10)  I1.place( x =1184, y = 155)  D1 = Entry (window,textvariable=Dent1,width=10)  D1.place( x =1184, y = 185)  #Wartosc zadania tempertury budynku - entryBox  Tr1\_entry = Entry (window,textvariable=Trent1,width=10)  Tr1\_entry.place( x =1255, y = 611)  Tr2\_entry = Entry (window,textvariable=Trent2,width=10)  Tr2\_entry.place( x =1255, y = 661) |

Dodano przyciski ‘OK’ sygnalizujące zatwierdzenie wpisanych wartości, które wywołują konkretne funkcje *pidum\_ok()*/.../*Tr\_ok()* (tj. zatwierdzenie wpisania nastaw regulatora wymiennika, budynku 1, budynku 2 oraz zadanych wartości temperatury pomieszczeń w budynkach 1 i 2.

|  |
| --- |
| button1=Button(window,text='OK',command=pidum\_ok)  button1.place( x =1280, y = 140)  button2=Button(window,text='OK',command=pidub1\_ok)  button2.place( x =1280, y = 320)  button3=Button(window,text='OK',command=pidub2\_ok)  button3.place( x =1280, y = 470)  button4=Button(window,text='OK',command=Tr\_ok)  button4.place( x =1355, y = 630)  button5=Button(window,text='Uruchom',command=uruchom\_program)  button5.place( x =55, y = 655) |

  
***Rys 3.2****. Panel wczytujący wartości podane przez użytkownika*

### 3.3. Zapis wartości nastaw do pliku

Wspomniane wyżej funkcje miały za zadanie pobierać wartość z entryBox’a i zapisywać je do odpowiednich zmiennych tudzież plików.

P*idum\_ok()/Pidub1\_ok()/Pidub2\_ok()* wykorzystują funkcję *writePID()* do zapisu danych oraz dodatkowo wyświetlają obecne wartości nastaw wpisane przez użytkownika.

|  |
| --- |
| def pidum\_ok():  P\_Um = Pent1.get()  I\_Um = Ient1.get()  D\_Um = Dent1.get()  P1label = Label(window, text=P\_Um,font="none 12")  P1label.place( x =1344, y = 125)  P2label = Label(window, text=I\_Um,font="none 12")  P2label.place( x =1344, y = 155)  P3label = Label(window, text=D\_Um,font="none 12")  P3label.place( x =1344, y = 185)  writePID(P\_Um,I\_Um,D\_Um,0) |

Kolejnym etapem jest zapisanie wartości do pliku, z którego następnie korzystał plik .exe wykonujący symulację. Zapis został zaimplementowany w funkcji *writePID(P,I,D,d)*, której parametry to wartości odpowiednich nastaw oraz indeks odpowiadający regulatorowi, którego dotyczą (0,1,2 - *Um*, *Ub1*, *Ub2*).  
Początkowe wartości nastaw są zainicjalizowane, zostają one zmienione tylko w przypadku podania i zatwierdzenia z panelu.

|  |
| --- |
| #Zapis nastaw PID do pliku  nastawy=[0.1,0.005,0.05,0.1,0.005,0.05,0.1,0.005,0.05]  def writePID(P,I,D,d):  f = open("PID.txt", "w")  f.truncate()  nastawy[d\*3]=float(P)  nastawy[d\*3+1]=float(I)  nastawy[d\*3+2]=float(D)  os.chdir('E:\Air\PKSS\GUI\_PKSS\Python\_projekty')  with open("E:\Air\PKSS\GUI\_PKSS\Python\_projekty\PID.txt", 'a', encoding='utf-8', newline='') as csvfile:  csvwriter=csv.writer(csvfile,delimiter=";")  for i in range(0,9):  csvwriter.writerow([nastawy[i]])  f.close() |

### 

### 3.4. Zapis zadanych temperatur do pliku

Funkcja *Tr\_ok()* zapisuje wpisane wartości do pliku oraz ustawia flagę na 0 lub 1 w zależności od tego czy użytkownik podał wartość zadaną *Tr* czy nie. Na tej podstawie symulacja decyduje czy wczytać wartości podane w pliku czy wczytane z GUI.

|  |
| --- |
| def Tr\_ok():  f = open("TrSP.txt", "w")  f.truncate()  Tr1 = Trent1.get()  Tr2 = Trent2.get()  flaga=1 #flaga ustawiana na 1 jesli uzytkownik poda wartosci zadane  Tr=[flaga,float(Tr1),float(Tr2)]  with open("E:\Air\PKSS\GUI\_PKSS\Python\_projekty\TrSP.txt", 'a', encoding='utf-8', newline='') as csvfile:  csvwriter=csv.writer(csvfile,delimiter=";")  for i in range (0,3):  csvwriter.writerow([Tr[i]]) |

### 3.5. Obsługa jednoczesnego odczytu i zapisu do pliku

Uruchomienie pliku wykonywalnego exe odbywa się w funkcji *uruchom\_program()*. Funkcja ta używa struktury języka Python *Try/Except* w celu zapobiegnięcia potencjalnemu problemowi związanemu z równoczesnym odczytywaniem zawartości pliku CSV przez GUI oraz zapisywaniu danych przez plik wykonywalny do tego samego miejsca. *Except* wygląda w sposób następujący:

|  |
| --- |
| except EOFError:  print("Unexpected EOF")  except OSError:  print("File not found")  except:  print("Unexpected Error but have no idea which kind - sorry ") |

Pokrywa on najważniejsze dwa błędy mogące potencjalnie wyniknąć w toku działania programu, a więc próbę odczytania zawartości gdy danych w pliku już nie ma oraz brak istnienia danego pliku. Wszystkie pozostałe błędy pokrywane są przez ostatni ogólny wyjątek.

### 

### 3.6. Wyświetlanie aktualnych wartości zmiennych

W każdej iteracji odczytu z pliku csv na główny obraz nanosiliśmy wartości takie jak *Tzco*, *Tzcoref*, *Fzm*, Tzm, Tpm oraz oddzielne wartości dla budynku 1 oraz 2: *Tr*, *Tref*, *Tpco*, *Fcob*.

Polegało to na wybraniu odpowiedniej wartości z pliku csv. Następnie sparsowaniu wyświetlanego tekstu i ustawiania takich parametrów Labela jak czcionka, kolory tekstu, tła i pozycja na wyświetlanym obrazie.

* **Wymiennik ciepła**

|  |
| --- |
| with open(name, mode='r') as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=';')  for row in csv\_reader:  # ?MPEC?  # Tzco  Tzco = Label(window, text=str(row[4])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  Tzco.place(x=116, y=63)  # Tzcoref  Tzcoref = Label(window, text=str(row[5])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  Tzcoref.place(x=206, y=63)  # Fzm  fzm = str("{0:.2f}".format(float(row[10][0:6]) \* 80))  Fzm = Label(window, text=str(fzm) + " [t/h]", fg="black", bg="green", font="none 12")  Fzm.place(x=296, y=63)  # Tzm  Tzm = Label(window, text=str(row[6])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  Tzm.place(x=396, y=63)  # Tpm  Tpm = Label(window, text=str(row[3])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  Tpm.place(x=308, y=270) |

* **Budynek 1**

|  |
| --- |
| # Tr  Tr1 = Label(window, text=str(row[1])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  Tr1.place(x=680, y=63)  # TrSP  TrSP1 = Label(window, text=str(row[9])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  TrSP1.place(x=770, y=63)  # Tpco  Tpco1 = Label(window, text=str(row[2])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  Tpco1.place(x=860, y=63)  # Fco1  fco1 = str("{0:.2f}".format(float(row[11][0:6]) \* 40))  Fco1 = Label(window, text=str(fzm) + " [t/h]", fg="black", bg="green", font="none 12")  Fco1.place(x=950, y=63) |

* **Budenek 2**

|  |
| --- |
| # Budynek 2  # Tr  Tr2 = Label(window, text=str(row[13])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  Tr2.place(x=687, y=450)  # TrSP  TrSP2 = Label(window, text=str(row[14])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  TrSP2.place(x=777, y=450)  # Tpco  Tpco1 = Label(window, text=str(row[2])[0:4] + " [°C]", fg="black", bg="orange", font="none 12")  Tpco1.place(x=867, y=450)  # Fco1  fco1 = str("{0:.2f}".format(float(row[11][0:6]) \* 40))  Fco1 = Label(window, text=str(fzm) + " [t/h]", fg="black", bg="green", font="none 12")  Fco1.place(x=957, y=450)  # Regulatory[%]  # Um  um = str("{0:.2f}".format(float(row[10][0:6]) \* 100))  Um = Label(window, text=um + " [%]", fg="black", bg="grey", font="none 12")  Um.place(x=56, y=170)  # Ub1  um = str("{0:.2f}".format(float(row[11][0:6]) \* 100))  Um = Label(window, text=um + " [%]", fg="black", bg="grey", font="none 12")  Um.place(x=462, y=170)  # Ub2  um = str("{0:.2f}".format(float(row[12][0:6]) \* 100))  Um = Label(window, text=um + " [%]", fg="black", bg="grey", font="none 12")  Um.place(x=462, y=550)  #print(f'{i} \n ')  window.update()  #time.sleep(0.1) |

# 

# 4. Podział zadań

* DW - Dorota Wejdman
* SW - Sebastian Wach
* KW - Konrad Wajda

**4.1. Model**

* Stworzenie pierwszej wersji modelu z obsługą 1 budynku (DW, KW)
* Weryfikacja działania oraz szukanie błędów (DW, KW)
* Dobór nastaw PID (KW)
* Dodanie obsługi 2 budynku (KW)
* Stworzenie m-skryptu oraz pliku z TrSP (DW)

**4.2. Implementacja w C**

* Generacja kodu z matlaba, ustawienie odpowiednich parametrów (SW)
* Dodanie zapisu i odczytu z pliku (SW)
* Analiza i opis kodu (SW, KW)
* Ujednolicenie zmiennych z konspektem (DW)
* Dodanie odczytu wartości zadanej Tr oraz nastaw z pliku z GUI (SW)
* Dobór nastaw PID (DW, SW)

**4.3. GUI**

* Stworzenie podstawowego interfejsu użytkownika (SW)
* Wyświetlanie aktualnych wartości (SW)
* Dostosowanie obrazu tła do potrzeb aplikacji i danych (DW)
* Wczytywanie z panelu wartości nastaw i zadanych Tr (DW)
* Zapis do pliku wczytanych danych (DW)
* Równoległa obsługa odczytu i zapisu do pliku (KW)

**4.4. Dokumentacja**

* Opis elementów modelu (DW)
* Opis funkcji w C (KW)
* Opis funkcji GUI (DW)
* Generacja wykresów z modelu (KW)
* Generacja wykresów z C (SW)
* Estetyka dokumentacji (DW)

# 5. Wnioski

* Współodpowiedzialność za każdy etap pracy pozwoliła zaznajomić z różnymi aspektami tworzenia projektu każdemu z członków zespołu.
* Pierwszy raz wykorzystaliśmy generację kodu z Matlaba, mogliśmy dzięki temu zapoznać się z tym procesem, poznać jego wady i zalety. Miało to duży wpływ na przyspieszenie postępów pracy, pozwoliło skupić się na pozostałych elementach   
  oraz je dopracować.
* Na etapie tworzenia modelu w środowisku Matlab bardzo ważne było dbanie o przejrzystość oraz opis dodawanych subsystemów i schematów, co umożliwiało dalszą weryfikację i szukanie błędów.
* Odpowiednie nazewnictwo wejść/wyjść modelu ma duży wpływ na nazwy zmiennych wygenerowanych poprzez *Simulink Coder*, odpowiednia wiedza o tym pozwala na uniknięcie wygenerowania kodu z nieintuicyjnymi nazwami zmiennych
* W tworzeniu GUI największą trudnością była równoległa obsługa zapisu i odczytu z pliku, która było możliwa po zapoznaniu się ze strukturą Try/Except.
* Przy projektowaniu systemu wizualizacyjnego należy kierować się przede wszystkim prostotą oraz łatwością odczytywania danych przez przyszłego operatora, bez zbędnych elementów graficznych