Wydział Mechaniczny
Politechniki Białostockiej

SPRAWOZDANIE Z PRZEDMIOTU

Systemy sterowania robotów

Kod przedmiotu: **MYAR2S01006M**

Temat: Tworzenie własnych formatów wiadomości ROS

Imię i nazwisko: Janusz Chmaruk
Kierunek studiów: Automatyka i Robotyka
Specjalność: -
Semestr: I
Rok akademicki: 2022/2023
Data wykonania pracy: 19.04.2023

……
podpis studenta

Weryfikacja efektów kształcenia:	
EK1 ……..	EK5 ……..
EK2 ……..	EK6 ……..
EK3 ……..	EK7 ……..
EK4 ……..	EK8 ……..
Uwagi prowadzącego:	
Ocena sumaryczna: ……	……. *podpis prowadzącego*

Spis treści

1 Wstęp

Celem ćwiczenia jest zapoznanie z tworzeniem własnych formatów wiadomości w systemie ROS.

2 Zakres ćwiczenia

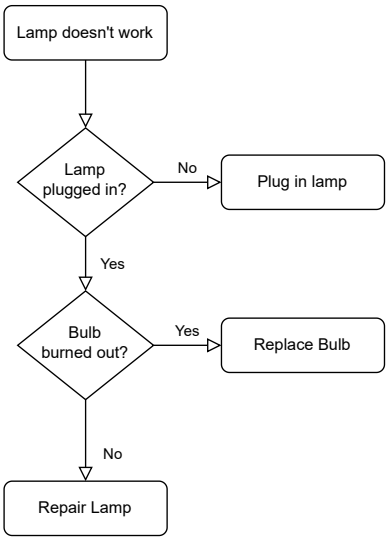
- Tworzenie własnych formatów wiadomości
- Obsługa czasu w systemie ROS
- Tworzenie wykresów przy pomocy narzędzia rqt_plot

3 Realizacja ćwiczenia

3.1 Tworzenie własnego typu wiadomości

W nowej paczce utwórz własny typ wiadomości Student o następujących polach: numer_albumu, imię, nazwisko, aktywny, oceny, kierunek. Napisz w dowolnym języku program nasłuchujący wiadomości na topicu /student i wypisujący ich zawartość do logu.

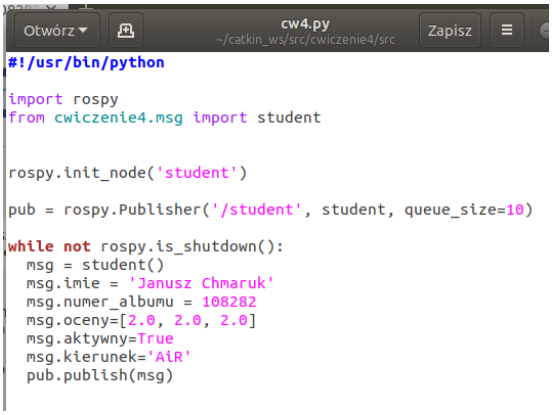
W podanym tekście opisano proces tworzenia, wysyłania i odbierania własnej wiadomości w systemie ROS (Robot Operating System). Omówiono elementy kodu publishera i subscribera oraz przedstawiono wyniki ich działania na zrzucie ekranu z terminala. Rys.1 przedstawia format utworzonej wiadomości. Kod publishera, który wysyła wiadomość o utworzonym formacie, pokazano na Rys.2. Wiadomość jest wysyłana na temacie /student z kolejką wynoszącą 10 wiadomości. W kodzie publishera linijka 6 uruchamia węzeł o nazwie student, a linijka 7 definiuje temat, format wiadomości i wielkość kolejki dla publishera. W linii 10-16 przypisywane są dane do poszczególnych składowych wiadomości, a ostatnia komenda wysyła utworzoną wiadomość. Kod subscribera, który nasłuchuje na temacie /student, znajduje się na Rys.3. W linii 6 kodu subscribera (Rys. 3) utworzono funkcję odbierającą wiadomość z tematu /student o nowym formacie i wyświetlającą odebrane dane. W linii 9 uruchamiany jest węzeł o nazwie student_listener, a w linii 11 zdefiniowano funkcję subscribera, określając temat do nasłuchiwania, oczekiwany typ wiadomości i funkcję do wykonania po otrzymaniu wiadomości. Funkcja rospy.spin() odpowiada za aktualizację zdarzeń systemu ROS. Na Rys. 4 przedstawiono zrzut ekranu z terminala, na którym widać działanie publishera wysyłającego dane oraz subscribera odbierającego wiadomość.



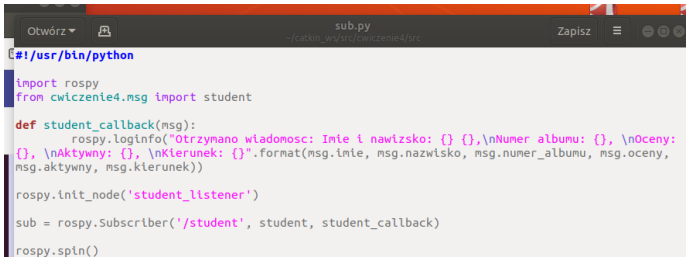
Rysunek 1: heading

```
uint32 numer_albumu
string imie
bool aktywny
float32[] oceny
string kierunek
```

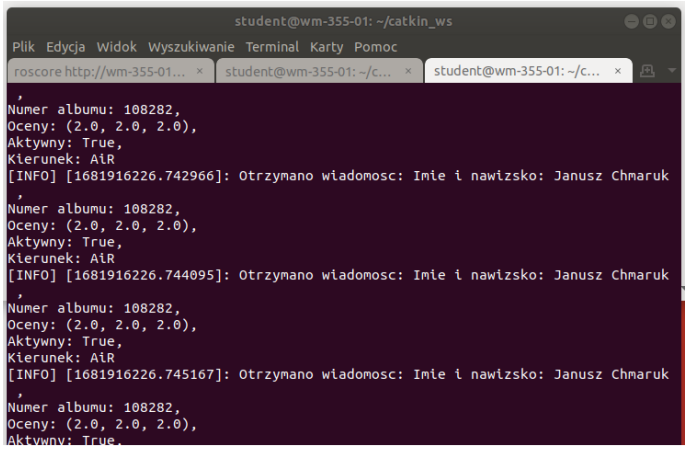
Rysunek 2: Utworzony typ wiadomości w pliku Student.msg



Rysunek 3: Kod publishera wykonany w języku python



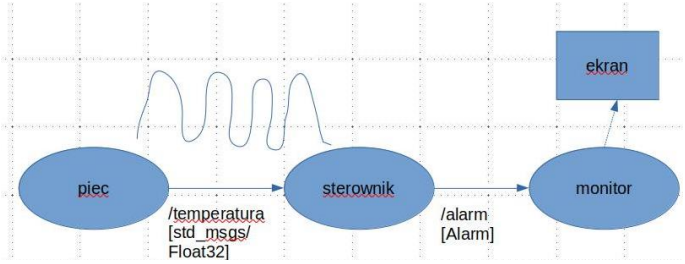
Rysunek 4: Kod subscribera wykonany w języku python



Rysunek 5: Efekt działania utworzonego publishera oraz subscribera

3.2 Węzeł alarmujący

Napisz program, który będzie alarmował o przekroczeniu dozwolonego stanu parametru procesu (np. temperatura w piecu). Program powinien nasłuchiwać na pewnym topicu aktualnej wartości stanu procesu, a w przypadku przekroczenia jej górnego lub dolnego limitu wysyłać wiadomość zawierającą: stan alarmu, który limit został przekroczony, o ile limit został przekroczony, oraz jak długo trwa stan alarmu. Sprawdź działanie programu. Wykreśl przy pomocy programu rqt_plot wykres obserwowanej wartości i określ, czy alarmy zostały poprawnie zgłoszone.



Rysunek 6: Schemat układu do zadania 2

W drugim zadaniu stworzono dwa nowe typy wiadomości: regulacja (Rys. 6) oraz alarm (Rys.7). Aby zwiększyć czytelność sygnałów alarmowych w narzędziu rqt_plot, alarmy utworzono jako typ int32, przyjmujący wartość równą dozwolonym zakresom. Te wartości to 1500 dla tempmax i 700 dla tempmin. Na Rys. 8 przedstawiono kod programu piec.py, który tworzy sygnał sinusoidalny i przesyła go na temacie /temperatura. Kod programu sterownik.py, który nasłuchuje wiadomości na temacie /temperatura i sprawdza, czy przekroczone zostały dopuszczalne zakresy temperatur pieca, pokazano na Rys. 9. Następnie sterownik.py wysyła informacje o alarmach na temacie /alarm. Rys. 10 zawiera kod programu ekran.py, który odbiera i wyświetla informacje z tematów /temperatura oraz /alarm. Gdy wystąpi stan alarmowy, ekran.py oblicza czas trwania alarmu oraz wartość, o jaką został przekroczony zakres. Zrzut ekranu z terminala w trakcie działania tych programów można zobaczyć na Rys. 11.

```
1 float32 obecnatemperatura
```

Rysunek 7: Utworzony typ wiadomości temperatura

```
1 int32 tempmax
2 int32 tempmin
```

Rysunek 8: Utworzony typ wiadomości alarm

```
#!/usr/bin/python3

#import bibliotek
import rospy
from c4.msg import regulacja

rospy.init_node('piec')                    #inicjalizacja wezla piec

pub = rospy.Publisher('/temperatura', regulacja, queue_size=10)  #definicja publishera
                                           #na temacie "/temperatura", oczekiwany typ
                                           #wiadomosci "regulacja" z 10 elementami w kolejce
#definicje stalych
aaa = 100
i = 1
rate = rospy.Rate(3)   #czestotliwosc wykonywania rogramu 3Hz

while not rospy.is_shutdown():             #kiedy system ROS jest wlaczony

#definicja wysylanej danej
  temp = regulacja()
  temp.obecnatemperatura = aaa

  pub.publish(temp)       #wyslanie danej

#generacja sygnalu sinusoidalnego
  if i == 1:
    aaa = aaa + 100
    if aaa > 2000:
      i = 0
    else:
      i = 1
  else:
    aaa = aaa - 100
    if aaa < 200:
      i = 1
    else:
      i = 0

  rate.sleep()     #odczekanie w celu zapewnienia wykonywalnosci 3Hz
```

Rysunek 9: Utworzony program piec.py, w którym został umieszczony publisher wysyłający wiadomość temperatura na temacie /temperatura

```
#!/usr/bin/python3
#import bibliotek
import rospy
from c4.msg import regulacja
from c4.msg import alarm

rospy.init_node('sterownik')  #inicjalizacja wezla sterownik
rate = rospy.Rate(3)          #czestotliwosc wykonywania programu 3Hz
#definicja stalych
aaa = 0
altempmax = 0
altempmin = 0

def piec_callback(msg) : #funkcja zczytujaca obecna temperature z tematu /temperatura
                         # \/ wyswietlenie informacji
  rospy.loginfo("Otrzymano wiadomo: obecna temperatura: {} °C".format(msg.obecnatemperatura))
#definicje zmiennych
  aaa = msg.obecnatemperatura
  global altempmax
  global altempmin
  if aaa >=1500:     #temperatura za wysoka
    altempmax = 1500
    altempmin = 0
  elif aaa <=700:    #temperatura za niska
    altempmax = 0
    altempmin = 700
  else:              #poprawna temperatura - brak alarmow
    altempmax = 0
    altempmin = 0
#definicja wysylanej danej
  alarminfo = alarm()
  alarminfo.tempmax = altempmax
  alarminfo.tempmin = altempmin
  pub.publish(alarminfo)   #wyslanie informacji o alarmach na temacie /alarm
while not rospy.is_shutdown():
  sub = rospy.Subscriber('/temperatura', regulacja, piec_callback) #definicja subscribera
                                           #na temacie "/temperatura", oczekiwany typ
                                           #wiadomosci "regulacja" oraz wywolanie funkcji
                                           #piec_callback

  pub = rospy.Publisher('/alarm', alarm, queue_size=10)#definicja publishera
                                           #na temacie "/alarm", oczekiwany typ
                                           #wiadomosci "alarm" z 10 elementami w kolejce

  rate.sleep()    #odczekanie w celu zapewnienia wykonywalnosci 3Hz
  rospy.spin()    #aktualizacja zdarzen systemu ROS
```

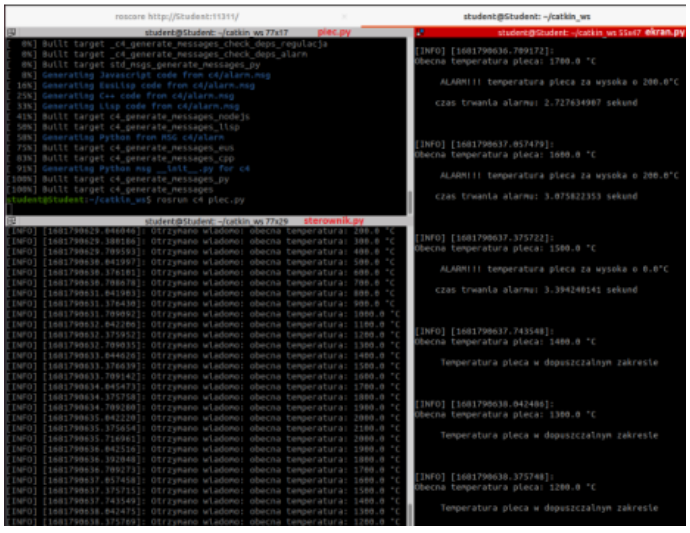
Rysunek 10: Utworzony program sterownik.py zawierający subscribera odbierającego wiadomości na temacie /temperatura oraz publishera wysyłającego wiadomość typu alarm na temacie /alarm

```python
#!/usr/bin/python3

#dolaczenie bibliotek
import rospy
from c4.msg import regulacja
from c4.msg import alarm
rospy.init_node('ekran') #inicjalizacja wezla o nazwie ekran
rate = rospy.Rate(3)     #czestotliwosc wykonania programu 3Hz

def piec_callback(msg) :  #funkcja zczytujaca obecna temperature z tematu /temperatura
  rospy.loginfo("\nObecna temperatura pieca: {} °C".format(msg.obecnatemperatura))
  global aaa
  aaa = msg.obecnatemperatura   #przypisanie obecnej temperatury pieca do zmiennej aaa
def sterownik_callback(ms) :  #funkcja zczytujaca stan alarmow z tematu /alarm
  if ms.tempmin == 700 and ms.tempmax == 0: #jesli przekroczony dolny limit to:
    bbb = 700 - aaa                         #obliczenie o ile limit zostal przekroczony
                                            # i wyswietlenie informacji
    print('\n' + '     ALARM!!! temperatura pieca za niska o ' + str(bbb) + '°C')
    ta = rospy.Time.now() - t               #obliczenie czasu trwania alarmu
    ts = ta.to_sec()                        #zamiana na sekundy
                                            # i wyswietlenie informacji
    print('\n' + '    czas trwania alarmu:' + str(ts) + '\n\n\n')
  elif ms.tempmin == 0 and ms.tempmax == 1500:  #jesli przekroczony gorny limit to:
    bbb = aaa - 1500                            #obliczenie o ile limit zostal przekroczony
                                                # i wyswietlenie informacji
    print('\n' + '     ALARM!!! temperatura pieca za wysoka o ' + str(bbb) + '°C')
    ta = rospy.Time.now() - t               #obliczenie czasu trwania alarmu
    ts = ta.to_sec()                        #zamiana na sekundy
                                            # i wyswietlenie informacji
    print('\n' + '    czas trwania alarmu: ' + str(ts) +' sekund\n\n\n')
  else:                                     #jesli brak alarmow to:
                                            # i wyswietlenie informacji
    print('\n' + '     Temperatura pieca w dopuszczalnym zakresie' + '\n\n\n')
while not rospy.is_shutdown():   #kiedy system ROS jest wlaczony to:
  t = rospy.Time.now()           #pobranie aktualnej godziny
  sub = rospy.Subscriber('/temperatura', regulacja, piec_callback) #definicja subscribera
                                          #na temacie "/temperatura", oczekiwany typ
                                          #wiadomosci "regulacja" oraz wywolanie funkcji
                                          #piec_callback
  sub = rospy.Subscriber('/alarm', alarm, sterownik_callback)      #definicja subscribera
                                          #na temacie "/alarm", oczekiwany typ
                                          #wiadomosci "alarm" oraz wywolanie funkcji
                                          #sterownik_callback
  rate.sleep()                    #odczekanie w celu zapewnienia wykonywalnosci 3Hz
  rospy.spin()                    #aktualizacja zdarzen systemu ROS
```

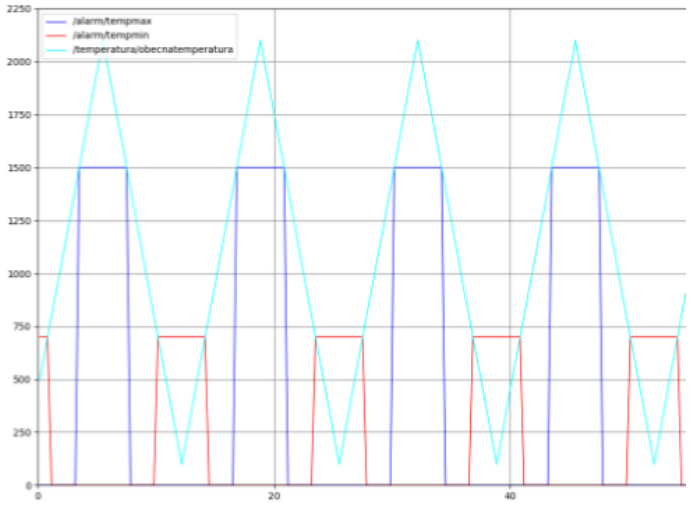
Rysunek 11: Utworzony program ekran.py, który posiada dwa subscribery nasłuchującego wiadomości na temacie /temperatura oraz /alarm



Rysunek 12: Efekt działania utworzonych programów piec.py, sterownik.py oraz ekran.py

Po przetestowaniu działania programu, przy pomocny węzła rqt_plot zostały wyznaczone przebiegi czasowe zawierające temperaturę pieca oraz sygnały alarmowe

- tempmax - informujący o przekroczeniu górnego zakresu temperatury,
- tempmin - informujący o przekroczeniu dolnego zakresu temperatury.



Rysunek 13: Zarejestrowane przebiegi sygnałów z wykorzystaniem narzędzia rqt_plot

Program działa zgodnie z początkowymi założeniami. Zarejestrowane przebiegi są prawidłowe tzn. alarmy pojawiają się oraz znikają w chwilach wyjścia/wejścia z zakresu wielkości procesowej (temperatury).

4 Wnioski

Realizacja ćwiczenia pozwoliła na zapoznanie się z procesem tworzenia własnych typów wiadomości w systemie ROS. Ważne jest, aby pamiętać o modyfikacji plików cmakelist.txt oraz package.xml, które znajdują się w katalogu tworzonej paczki w przestrzeni roboczej, np. ~/catkin_ws/src/c4. Narzędzie rqt_plot umożliwia śledzenie przebiegów wybranych zmiennych. Przy korzystaniu z tego narzędzia warto ustawić odpowiednie zakresy osi X i Y, gdyż domyślne wartości mogą być niewystarczające. Podczas pracy z systemem ROS niezbędne jest uruchomienie węzła centralnego za pomocą polecenia roscore. Nieuruchomienie lub wyłączenie węzła centralnego uniemożliwi nawiązanie połączenia pomiędzy nowo otwartymi programami. Warto jednak zauważyć, że wyłączenie węzła centralnego nie wpływa na działanie węzłów, które już komunikują się ze sobą.