

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	2
2 Zakres ćwiczenia	2
3 Realizacja ćwiczenia	2
3.1 Tworzenie własnego typu wiadomości	2
3.2 Węzeł alarmujący	2
4 Wnioski	4

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

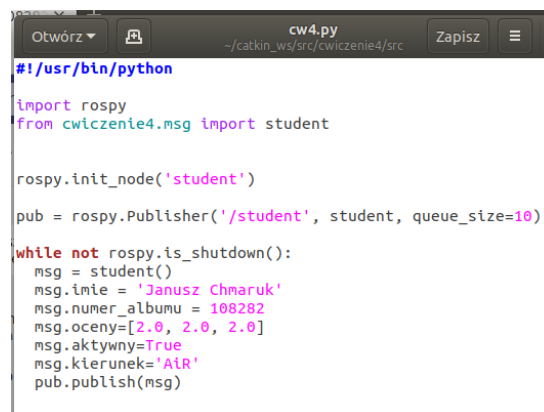
3.1 Tworzenie własnego typu wiadomości

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 nasłuchiwanie, 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.

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

Rysunek 1: Utworzony typ wiadomości w pliku Student.msg



```
#!/usr/bin/python

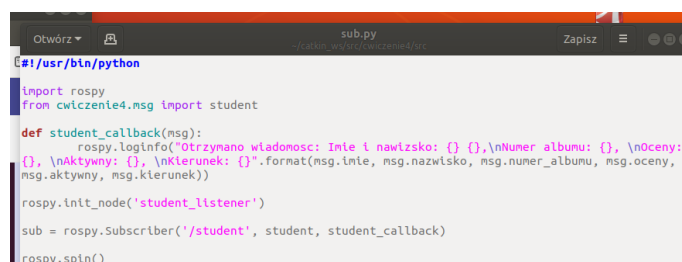
import rospy
from cwiczenie4.msg import student

rospy.init_node('student')

pub = rospy.Publisher('/student', student, queue_size=10)

while not rospy.is_shutdown():
    msg = student()
    msg.imie = 'Janusz Chmaruk'
    msg.numer_albumu = 108282
    msg.oceny=[2.0, 2.0, 2.0]
    msg.aktywny=True
    msg.kierunek='AIR'
    pub.publish(msg)
```

Rysunek 2: Kod publishera wykonany w języku python



```
#!/usr/bin/python

import rospy
from cwiczenie4.msg import student

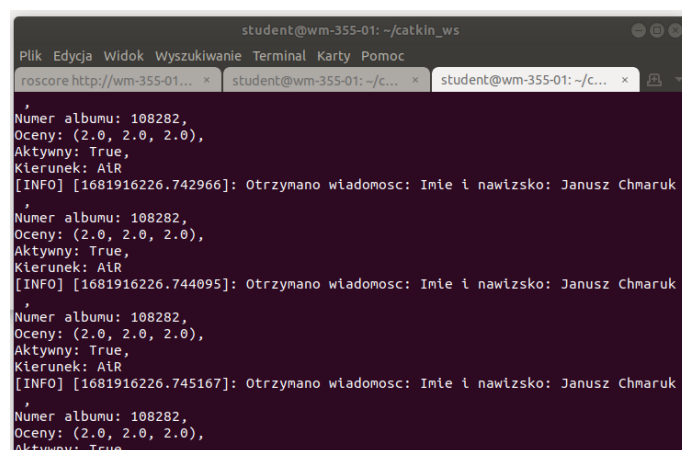
def student_callback(msg):
    rospy.logInfo("Otrzymano wiadomosc: Imie i nawizsko: {},\nNumer albumu: {}, \nOceny: {}, \nAktywny: {}, \nkierunek: {}".format(msg.imie, msg.nazwisko, msg.numer_albumu, msg.oceny, msg.aktywny, msg.kierunek))

rospy.init_node('student_listener')

sub = rospy.Subscriber('/student', student, student_callback)

rospy.spin()
```

Rysunek 3: Kod subscribera wykonany w języku python



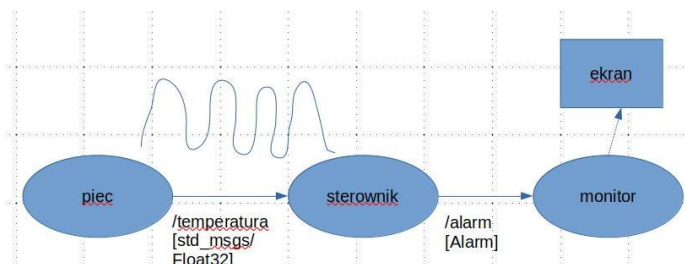
```
student@wm-355-01: ~/catkin_ws
Plik Edycja Widok Wyszukiwanie Terminal Karty Pomoc
roscore http://wm-355-01:~ student@wm-355-01:~/c... student@wm-355-01:~/c...
Numer albumu: 108282,
Oceny: (2.0, 2.0, 2.0),
Aktywny: True,
Kierunek: AIR
[INFO] [1681916226.742966]: Otrzymano wiadomosc: Imie i nawizsko: Janusz Chmaruk
Numer albumu: 108282,
Oceny: (2.0, 2.0, 2.0),
Aktywny: True,
Kierunek: AIR
[INFO] [1681916226.744095]: Otrzymano wiadomosc: Imie i nawizsko: Janusz Chmaruk
Numer albumu: 108282,
Oceny: (2.0, 2.0, 2.0),
Aktywny: True,
Kierunek: AIR
[INFO] [1681916226.745167]: Otrzymano wiadomosc: Imie i nawizsko: Janusz Chmaruk
Numer albumu: 108282,
Oceny: (2.0, 2.0, 2.0),
Aktywny: True,
```

Rysunek 4: 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 5: 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 6: Utworzony typ wiadomości temperatura

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

Rysunek 7: Utworzony typ wiadomości alarm

```
1 #!/usr/bin/python3
2 #import bibliotek
3 import rospy
4 from c4.msg import regulacja
5
6 rospy.init_node('piec') #inicjalizacja wezla piec
7
8 pub = rospy.Publisher('/temperatura', regulacja, queue_size=10) #definicja publishera
9 #na temacie "/temperatura", oczekiwany typ
10 #wiadomosci "regulacja" z 10 elementami w kolejce
11
12 #definicja statycznych
13 aaa = 100
14 t = 1
15 rate = rospy.Rate(3) #czestotliwosc wykonywania programu 3Hz
16
17 while not rospy.is_shutdown(): #kiedy system ROS jest włączony
18     #definicja wysylanej danej
19     temp = regulacja()
20     temp.obecnatemperatura = aaa
21     pub.publish(temp) #wyslanie danej
22
23 #generacja sygnału sinusoidalnego
24
25 if t == 1:
26     aaa = aaa + 100
27     if aaa > 2000:
28         t = 0
29     else:
30         t = 1
31 else:
32     aaa = aaa - 100
33     if aaa < 200:
34         t = 1
35     else:
36         t = 0
37
38 rate.sleep() #odroczenie w celu zapewnienia wykonalnosci 3Hz
```

Rysunek 8: Utworzony program `piec.py`, w którym został umieszczony publisher wysyłający wiadomość temperaturę na temacie `/temperatura`

```
1 #!/usr/bin/python3
2 #import bibliotek
3 import rospy
4 from c4.msg import regulacja
5 from c4.msg import alarm
6
7 rospy.init_node('sterownik') #inicjalizacja wezla sterownik
8 rate = rospy.Rate(3) #czestotliwosc wykonywania programu 3Hz
9 #definicja statycznych
10 aaa = 0
11 altempmax = 0
12 altempmin = 0
13
14 def piec_callback(msg): #funkcja czytująca obecną temperaturę z tematu /temperatura
15     # / wyswietlenie informacji
16     rospy.loginfo("Otrzymano wiadomość: obecna temperatura: {} °C".format(msg.obecnatemperatura))
17 #definicja zmiennych
18 global altempmax
19 global altempmin
20 if aaa >= 1500: #temperatura za wysoka
21     altempmax = 1500
22     altempmin = 0
23 elif aaa <= 700: #temperatura za niska
24     altempmax = 0
25     altempmin = 700
26 else: #poprawna temperatura - brak alarmow
27     altempmax = 0
28     altempmin = 0
29 #definicja wysylanej danej
30 alarminfo = alarm()
31 alarminfo.tempmax = altempmax
32 alarminfo.tempmin = altempmin
33 pub.publish(alarminfo) #wyslanie informacji o alarmach na temacie /alarm
34
35 while not rospy.is_shutdown():
36     sub = rospy.Subscriber('/temperatura', regulacja, piec_callback) #definicja subscrybenta
37     #na temacie "/temperatura", oczekiwany typ
38     #wiadomosci "regulacja" oraz wywołanie funkcji
39     #piec_callback
40
41 pub = rospy.Publisher('/alarm', alarm, queue_size=10) #definicja publishera
42 #na temacie "/alarm", oczekiwany typ
43 #wiadomosci "alarm" z 10 elementami w kolejce
44
45 rate.sleep() #odroczenie w celu zapewnienia wykonalnosci 3Hz
46 rospy.spin() #aktualizacja zdarzen systemu ROS
```

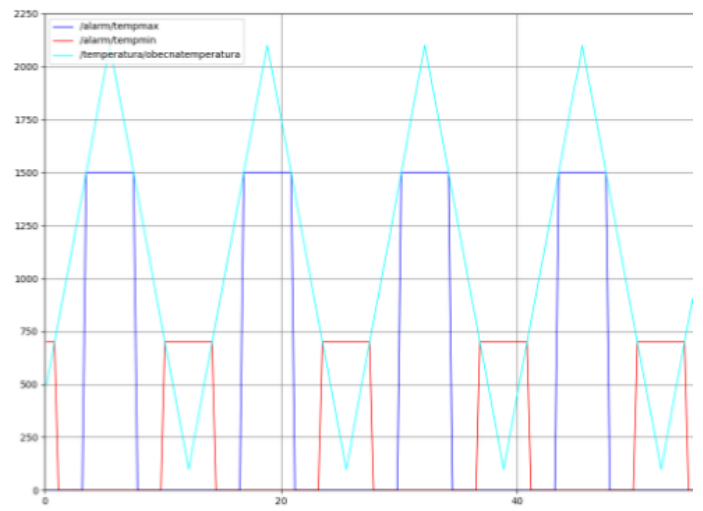
Rysunek 9: Utworzony program `sterownik.py` zawierający subscrybenta odbierającego wiadomości na temacie `/temperatura` oraz publisher'a wysyłającego wiadomość typu alarm na temacie `/alarm`

```

1#!/usr/bin/python3
2
3#doładowanie bibliotek
4import rospy
5from c4.msg import regulacja
6from c4.msg import alarm
7rospy.init_node('ekran') #inicjalizacja węzła o nazwie ekran
8rate = rospy.Rate(1) #częstotliwość wykonywania programu 1Hz
9
10def piec_callback(msg): #funkcja czytująca obecną temperaturę z tematu /temperatura
11    rospy.loginfo('Obecna temperatura pieca: [%f] °C'.format(msg.obecnatemperatura))
12
13global aaa
14aaa = msg.obecnatemperatura #przypisanie obecnej temperatury pieca do zmiennej aaa
15def sterownik_callback(msg): #funkcja czytująca stan alarmów z tematu /alarm
16    if ms.templein == 700 and ms.temphax == 0: #jeśli przekroczony dolny limit to:
17        #liczenie o ile limit został przekroczony
18        # /wyswietlenie informacji
19        print('\n' + 'ALARM!!! temperatura pieca za niska o ' + str(bbb) + ' °C')
20        ta = rospy.Time.now() - t #liczenie czasu trwania alarmu
21        ts = ta.to_sec() #zamiana na sekundy
22        # /wyswietlenie informacji
23        print('\n' + 'Czas trwania alarmu: ' + str(ts) + '\n\n')
24    elif ms.templein == 0 and ms.temphax == 1500: #jeśli przekroczony górny limit to:
25        #liczenie o ile limit został przekroczony
26        # /wyswietlenie informacji
27        print('\n' + 'ALARM!!! temperatura pieca za wysoka o ' + str(bbb) + ' °C')
28        ta = rospy.Time.now() - t #liczenie czasu trwania alarmu
29        ts = ta.to_sec() #zamiana na sekundy
30        # /wyswietlenie informacji
31        print('\n' + 'Czas trwania alarmu: ' + str(ts) + ' sekund\n\n')
32    else: #jeśli brak alarmów to:
33        # /wyswietlenie informacji
34        print('\n' + 'Temperatura pieca w dopuszczalnym zakresie' + '\n\n')
35
36while not rospy.is_shutdown(): #kiedy system ROS jest włączony to:
37    t = rospy.Time.now() #pobranie aktualnej godziny
38    sub = rospy.Subscriber('/temperatura', regulacja, piec_callback) #definicja subskrybenta
39    #nazwa tematu "regulacja", oczekiwany typ
40    #wiadomości "regulacja" oraz wywołanie funkcji
41    #piec_callback
42    sub = rospy.Subscriber('/alarm', alarm, sterownik_callback) #definicja subskrybenta
43    #nazwa tematu "alarm", oczekiwany typ
44    #wiadomości "alarm" oraz wywołanie funkcji
45    #sterownik_callback
46    rate.sleep() #odczekiwanie w celu zapewnienia wykonywalności 1Hz
47    rospy.spin() #kontynuacja działania systemu ROS

```

Rysunek 10: Utworzony program ekran.py, który posiada dwa subskrybenty nasłuchującego wiadomości na temacie /temperatura oraz /alarm

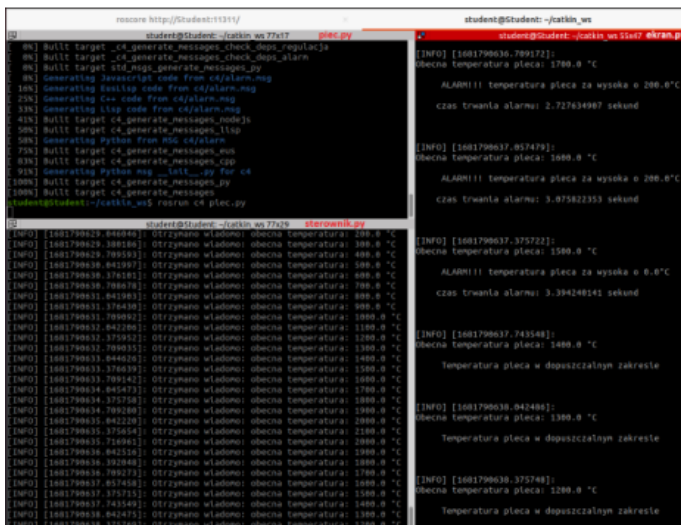


Rysunek 12: Zarejestrowane przebiegi sygnałów z wykorzystaniem narzędzia rqt_plot

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 cmake.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.



Rysunek 11: Efekt działania utworzonych programów piec.py, sterownik.py oraz ekran.py

Po przetestowaniu działania programu, przy pomocy 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.