

Zastosowanie czujnika głębi Kinect do interakcji z robotem mobilnym z wykorzystaniem platformy ROS*

Robert Muszyński[†]
Laboratorium Robotów Autonomicznych
Wydział Elektroniki
Politechnika Wrocławska

Spis treści

1 Cel ćwiczenia	1
2 Wymagania wstępne	2
3 Opis stanowiska	2
4 Wprowadzenie	2
5 Czujnik Kinect	2
5.1 Pakiety platformy ROS do obsługi czujnika Kinect	3
5.1.1 Pakiet <i>OpenNI_camera</i>	3
5.1.2 Pakiet <i>OpenNI_launch</i>	3
5.1.3 Pakiet <i>OpenNI_tracker</i>	4
5.2 Inne pakiety platformy ROS	6
6 Przykład integracji danych z czujnika Kinect	6
7 Zadania do wykonania	9
8 Sprawozdanie	9

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie ze sposobami wykorzystania danych pochodzących z czujnika głębi Kinect [15] z poziomu platformy programistycznej ROS [18, 4] i ich integracja w systemie sterowania robota mobilnego.

*Ćwiczenie laboratoryjne przeznaczone do realizacji w ramach kursu Robotyka (2) – data ostatniej modyfikacji: 16 kwietnia 2015.

[†]Katedra Cybernetyki i Robotyki

2 Wymagania wstępne

Przed przystąpieniem do realizacji ćwiczenia należy:

- zapoznać się z Instrukcją bezpieczeństwa i higieny pracy w Laboratorium Robotów Autonomicznych L1.5 [2],
- zapoznać się z Instrukcją bezpieczeństwa i higieny pracy przy obsłudze robotów Pioneer P3-DX [3],
- odbyć ćwiczenia zapoznające z podstawami środowiska ROS i sterowaniem robotów mobilnych z jego wykorzystaniem [19, 5],
- zapoznać się z niniejszą instrukcją.

3 Opis stanowiska

W skład stanowiska laboratoryjnego, umożliwiającego przeprowadzenie ćwiczenia, wchodzi jeden czujnik głębi Kinect, co najmniej jeden komputer stacjonarny działający pod nadzorem systemu Linux, oraz jeden robot mobilny Pioneer P3-DX wraz z komputerem pokładowym pełniącym rolę sterownika.

4 Wprowadzenie

O skutecznym funkcjonowaniu robotów w ich środowisku działania decydują informacje, jakie o tym środowisku są one w stanie uzyskać. W przypadku, gdy roboty operują w obecności ludzi, czy też nawet we współpracy z nimi, ważne jest, by były one w stanie w sposób właściwy odbierać przekazywane im informacje. Z punktu widzenia człowieka wygodnie jest, gdy informacje te przekazywane są w sposób dla niego naturalny: mową, gestem, dotykiem. Stąd potrzeba wyposażania robotów w sensory pozwalające na przekazywanie informacji tą drogą. Pozwalają na to takie urządzenia jak kamery, matryce mikrofonów, czy cieszące się ostatnio ogromną popularnością czujniki głębi, takie jak wprowadzony na rynek przez firmę Microsoft czujnik Kinect. Zaletą tych ostatnich jest to, że podczas gdy pierwsze z wymienionych sensorów wymagają od ich użytkowników implementowania skomplikowanych algorytmów przetwarzania pochodzących z nich danych, integralną częścią czujników głębi są takie algorytmy, co powoduje, że dostarczają one danych w postaci stosunkowo prostej do interpretacji.

5 Czujnik Kinect

Czujnik głębi Kinect jest urządzeniem wejściowym, pozwalającym użytkownikowi na interakcję z komputerem w sposób dla niego naturalny, za pomocą gestów i komend głosowych [15]. Czujnik ten dostarcza do komputera obraz z kolorowej kamery wraz z mapą głębi, reprezentowaną przez chmurę punktów, budowaną z wykorzystaniem oświetlenia strukturalnego, a także sygnał z macierzy 4 mikrofonów kierunkowych i akcelerometru. Informacje te dają możliwość programowego odczytania np. sylwetki człowieka i wykrycia jego gestykulacji, rozpoznawanie twarzy, lokalizację źródeł dźwięku, rozpoznawanie mowy.

Sterowniki czujnika Kinect pozwalają nie tylko na dostęp do pochodzących z niego danych, ale zawierają też kompletny interfejs programistyczny aplikacji (API) o nazwie

OpenNI (Open Natural Interaction) [16], korzystający z oprogramowania pośredniczącego NiTE [17, 14]. Platforma OpenNI ROS [7] udostępnia wspomnianą funkcjonalność z poziomu systemu ROS, pozwalając na jego integrację z czujnikami głębi.

5.1 Pakiety platformy ROS do obsługi czujnika Kinect

Pierwotnie pakietem systemu ROS służącym do obsługi czujnika Kinect był pakiet *Openni_kinect*, zawierający sterowniki i skrypty, takie jak *openni_camera*, *openni_tracker*, *openni_launch*, zapewniające uzyskanie dostępu do funkcjonalności czujnika z poziomu ROSa. Po zmianie sposobu organizacji struktury i hierarchii pakietów w systemie ROS*, elementy składowe pakietu zostały z niego wydzielone, a sam pakiet został oznaczony jako nieaktualny. Poniżej opisano pokrótce funkcjonalności pakietów służących do wykorzystania czujnika Kinect w aplikacjach systemu ROS w wersji hydro, a także wybranych pakietów pomocniczych.

5.1.1 Pakiet *OpenNI_camera*

Pakiet *OpenNI_camera* [8] zapewnia systemowi ROS zgodny ze standardem OpenNI interfejs do czujników głębi. Obecnie dostarcza on sterowniki do następujących czujników:

- Microsoft Kinect,
- ASUS Xtion PRO,
- ASUS Xtion PRO Live,
- PrimeSense PSDK 5.0.

Wśród tematów[†], na publikowanie których pozwala pakiet *OpenNI_camera*, znajdują się:

- `rgb/image_raw` – zawierający pochodzący bezpośrednio z kamery RGB czujnika obraz zapisany w formacie Bayer GRBG for Kinect,
- `depth/image_raw` – zawierający mapę głębi w postaci dostarczanej przez czujnik,
- `ir/image_raw` – zawierający nieprzetworzony obraz z kamery na podczerwień czujnika.

W celu wykorzystania funkcjonalności dostarczanej przez pakiet najlepiej jest użyć skrypt uruchomieniowy *openni_launch* dostarczany przez pakiet *OpenNI_launch* (podrozdział 5.1.2), zestawiający w systemie węzły[‡] w sposób pozwalający na publikowanie wiadomości[§] wynikających z pracy obsługiwanego czujnika.

5.1.2 Pakiet *OpenNI_launch*

Pakiet *OpenNI_launch* [9] dostarcza skrypt uruchamiający w systemie ROS wszystkie węzły potrzebne do konwersji pochodzących z czujnika Kinect surowych obrazów na mapę głębi, mapę rozbieżności i chmurę punktów, reprezentujących obszar widziany przez czujnik. W celu uruchomienia skryptu należy wydać polecenie

```
roslaunch openni_launch openni.launch
```

*co miało miejsce w chwili wprowadzenia wersji groovy systemu

†ang. topics

‡ang. nodes

§ang. messages

5.1.3 Pakiet *OpenNI_tracker*

Pakiet *OpenNI_tracker* [10] pozwala na śledzenie sylwetek ludzi poprzez przypisanie im szkieletów. Informacje o śledzonych szkieletach publikowane są w postaci transformacji `tf` [12, 13, 1] układów współrzędnych związanych z charakterystycznymi punktami szkieletu. Lista publikowanych transformacji zawiera transformacje do układów współrzędnych o następujących nazwach:

- `/head_#`,
- `/neck_#`,
- `/torso_#`,
- `/left_shoulder_#`,
- `/left_elbow_#`,
- `/left_hand_#`,
- `/right_shoulder_#`,
- `/right_elbow_#`,
- `/right_hand_#`,
- `/left_hip_#`,
- `/left_knee_#`,
- `/left_foot_#`,
- `/right_hip_#`,
- `/right_knee_#`,
- `/right_foot_#`,

gdzie `#` oznacza numer śledzonego użytkownika. Wszystkie publikowane transformacje są typu `tf2_msgs/TFMessage` i domyślnie podawane są względem układu odniesienia `openni_depth_frame`[¶]. Typ `tf2_msgs/TFMessage` zawiera jedno pole `transforms`, które jest typu `geometry_msgs/TransformStamped[]`. Typ `geometry_msgs/TransformStamped` składa się z następujących pól:

```
std_msgs/Header header
string child_frame_id
geometry_msgs/Transform transform
```

które kolejno zawierają:

- `std_msgs/Header`
 - `uint32 seq`
 - `time stamp`
 - `string frame_id`

gdzie `seq` jest inkrementowanym identyfikatorem, `stamp` podaje chwilę zarejestrowania wiadomości (wyrażoną w sekundach (`stamp.sec`) i nanosekundach (`stamp.nsec`) od początku epoki Uniksa^{||}), zaś `frame_id` określa identyfikator nadrzędnego układu współrzędnych,

[¶]Układ odniesienia można zmienić przez zmianę wartości parametru `camera_frame_id` (np. `roscparam set /openni_tracker/camera_frame_id camera_link`).

^{||}czas POSIX

- `geometry_msgs/Transform`
`geometry_msgs/Vector3 translation`
`geometry_msgs/Quaternion rotation`

gdzie `translation` jest położeniem środka definiowanego układu w układzie nadrzędnym

```
geometry_msgs/Vector3 translation
float64 x
float64 y
float64 z
```

a `rotation` określa wzajemną orientację układów wyrażoną za pomocą kwaternionu

```
geometry_msgs/Quaternion rotation
float64 x
float64 y
float64 z
float64 w
```

- Pole `child_frame_id` określa nazwę układu współrzędnych, dla którego zdefiniowana jest dana transformacja.

Przykładowa wiadomość typu `tf2_msgs/TFMessage`, pochodząca z węzła `openni_tracker` a uzyskana poleceniem `rostopic echo tf` ma postać

`transforms:`

```
-
header:
  seq: 0
  stamp:
    secs: 1424420996
    nsecs: 997043513
  frame_id: oppenni_depth_frame
child_frame_id: right_foot_1
transform:
  translation:
    x: 0.334862131346
    y: -0.625047353971
    z: -0.705471994174
  rotation:
    x: 0.499999999997
    y: 0.500001836603
    z: 0.499999999997
    w: 0.499998163397
```

W celu uruchomienia węzła `openni_tracker` należy wydać polecenie

```
roslun oppenni_tracker oppenni_tracker
```

5.2 Inne pakiety platformy ROS

Pakiet *Image_view* Pakiet ten dostarcza prostych narzędzi do wyświetlania obrazów publikowanych w tematach systemu ROS, łącznie z obrazami stereoskopowymi i mapami rozbieżności. Wśród węzłów dostarczanych przez pakiet znajdują się:

- *image_view* – prosta przeglądarka obrazów publikowanych w tematach typu `sensor_msgs/Image`,
- *disparity_view* – prosta przeglądarka map rozbieżności publikowanych w tematach typu `stereo_msgs/DisparityImage`,
- *stereo_view* – przeglądarka pozwalająca na synchroniczne wyświetlanie obrazów stereoskopowych wraz z generowanymi z nich mapami rozbieżności.

Pakiet dostarcza także narzędzie do rejestracji strumienia wideo w pliku (*video_recorder*).

Pakiet *Rviz* Pakiet *Rviz* dostarcza rozbudowanego narzędzia do wizualizacji 3D w systemie ROS. Węzeł *rviz* uruchamia polecenie `roslaunch rviz rviz`.

Pakiet *Tf* Pakiet ten zawiera narzędzia do zarządzania wprowadzanymi w systemie układami współrzędnych. W szczególności jego komponenty pozwalają na monitorowanie aktualnego drzewa wszystkich występujących w systemie transformacji układów współrzędnych (`roslaunch tf tf_monitor`), czy wybranych (np. `roslaunch tf tf_monitor head_1 neck_1`), uzyskiwanie transformacji pomiędzy wybranymi układami współrzędnych (np. `roslaunch tf tf_echo head_1 neck_1`), uzyskiwanie w postaci graficznej drzewa transformacji (`roslaunch tf view_frames`, co powoduje utworzenie pliku `frames.pdf`).

6 Przykład integracji danych z czujnika Kinect

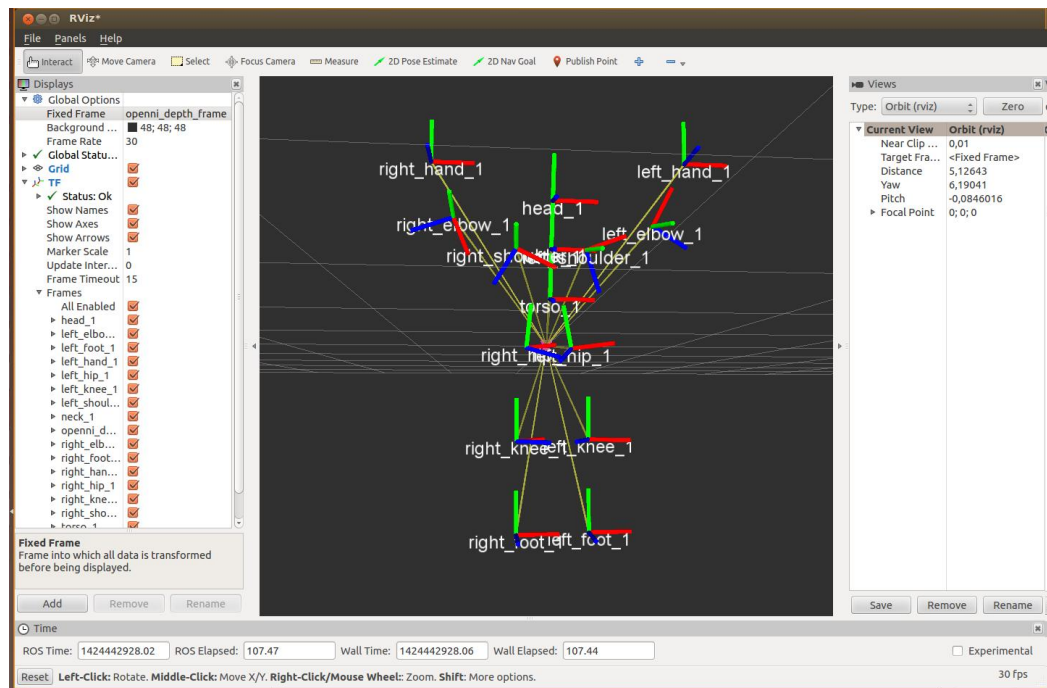
Poniżej przedstawiony został przykład wykorzystania czujnika Kinect z poziomu platformy programistycznej ROS za pośrednictwem pakietów *OpenNi_tracker* i *OpenNi_launch*. Przykład ilustruje przypadek podłączenia czujnika Kinect do lokalnego komputera, na którym uruchomione jest jądro systemu ROS (węzeł *roscore*). Opisane czynności wygodnie jest wykonywać z poziomu osobnych powłok użytkownika (terminali), co pozwoli na łatwe obserwowanie komunikatów o stanie procesów i ewentualnych błędach**. Należy zadbać, by w uruchamianych powłokach ustawiona była konfiguracja inicjowana skryptem `/opt/ros/hydro/setup.bash`^{††}.

1. Podłączyć do lokalnego komputera czujnik Kinect.
2. Uruchomić lokalny proces jądra ROS poleceniem `roscore`*

**By zapobiec pojawieniu się na pulpicie zbyt wielu okien terminala można skorzystać z aplikacji pozwalających na uruchomienie wielu terminali w jednym oknie, takich jak *terminator*.

^{††}ręcznie poleceniem `source /opt/ros/hydro/setup.bash`

*Lokalne ustawienia konfiguracji platformy ROS mogą spowodować pojawienie się po uruchomieniu jądra ostrzeżenia „WARNING: ROS_MASTER_URI [http://10.104.16.119:11311] host is not set to this machine auto-starting new master”, które należy zignorować. W takiej sytuacji, poniżej tego ostrzeżenia zostanie podany aktualny adres sieciowy uruchomionego jądra (np. `ROS_MASTER_URI=http://10.104.16.102:11311/`), który to należy każdorazowo ustawić przy uruchamianiu pozostałych komponentów systemu.

Rysunek 1: Przykład wizualizacji uzyskanej za pomocą węzła *rviz*

3. Uruchomić węzeł *openni_tracker* (`roslaunch openni_tracker openni_tracker†`) służący do śledzenia szkieletu człowieka‡ [10].
4. Uruchomić węzeł *rviz* (`roslaunch rviz rviz†`) pozwalający na wizualizację pojawiających się w systemie obiektów [11, 6]. Uzyskane w ten sposób okno zawiera trzy główne obszary (zobacz rysunek 1) – wielkie czarne coś po środku jest oknem wizualizacji sceny, po którego lewej znajduje się lista wizualizowanych elementów (*Displays*), zaś po prawej panel pozwalający na konfigurację systemu. Do listy *Displays* należy dodać wizualizację transformacji układów współrzędnych *TF* (klawisz *Add* znajdujący się poniżej listy) i upewnić się, że w uzyskanym węźle zaznaczona jest opcja *Frames/All Enabled*.
5. Stanąć w polu widzenia czujnika Kinect, czemu powinien towarzyszyć komunikat „New User 1”§ pojawiający się w terminalu, w którym został uruchomiony węzeł *openni_tracker*. Dokonać kalibracji użytkownika, przez ustawienie się przodem w odległości ok. 2 metrów od czujnika, z rękami uniesionymi do góry (na kształt litery Ψ). W przypadku prawidłowo przeprowadzonej kalibracji w terminalu z uruchomionym węzłem *openni_tracker* będą kolejno pojawiać się komunikaty typu „Pose Psi detected for user 1”, „Calibration started for user 1”, „Calibration

† Jeśli przy uruchomieniu lokalnego procesu jądra ROS pojawiło się wspomniane ostrzeżenie o adresie jądra, przed wydaniem polecenia należy nie zapomnieć o właściwym ustawieniu wartości zmiennej `ROS_MASTER_URI` (np. `export ROS_MASTER_URI=http://10.104.16.102:11311/`).

‡ W przypadku potrzeby uruchomienia węzłów do obsługi kilku czujników Kinect w ramach jednego jądra systemu ROS należy dla każdego z nich ustawić inną wartość zmiennej środowiskowej `ROS_NAMESPACE` (np. `export ROS_NAMESPACE="STUDENTx"`) i w miejsce węzła *openni_tracker* uruchomić węzeł *openni_tracker_multi* z pakietu *OpenNI_tracker_multi*.

§ Występująca na końcu komunikatu liczba jest numerem zidentyfikowanego użytkownika; liczbą tą będą opatrywane wszystkie definiowane w systemie obiekty związane z danym użytkownikiem. Należy zwrócić uwagę, że jeżeli w polu widzenia kinecta pojawi się ponownie „widziany” wcześniej użytkownik, system z dużym prawdopodobieństwem oznaczy go wcześniej przydzieloną mu liczbą.

complete, start tracking user 1". W chwili pojawienia się ostatniego z wymienionych komunikatów w oknie wizualizatora *rviz* na liście *Displays/TF/Frames* pojawiają się nazwy układów współrzędnych przypisane zidentyfikowanemu użytkownikowi. Jeśli równocześnie w oknie wizualizacji 3D nie uwidoczni się przypisany do użytkownika szkielet należy na liście *Displays* zmienić ustawienia pola *Global Options/Fixed Frame* (np. na *openni_depth_frame*). Przykładowy, docelowy widok uzyskany za pomocą węzła *rviz* pokazano na rysunku 1. W przypadku utraty użytkownika z pola widzenia czujnika węzeł *openni_tracker* zakomunikuje ten fakt.

6. Dokonać kalibracji kolejnych użytkowników czujnika.
7. Wykorzystać polecenia pakietu *Tf* (podrozdział 5.2) do monitorowania układów współrzędnych związanych ze śledzonymi użytkownikami.

W trakcie realizacji powyżej opisanych czynności można wspomóc działania poprzez podgląd obrazu widzianego przez kamery czujnika Kinect. W tym celu należy:

1. Uruchomić skrypt *openni.launch* (`roslaunch openni_launch openni.launch`[¶]) pozwalający na utworzenie odpowiednich węzłów pakietu *OpenNI*.
2. Uruchomić węzeł *image_view* z przypisaniem wyjścia z kamery RGB czujnika Kinect (`roslaunch image_view image_view image:=/camera/rgb/image_color`[¶]). Dodatkowo uruchomić węzeł *image_view*, który pozwoli na dokonanie wizualizacji mapy głębi (`roslaunch image_view image_view image:=/camera/depth/image`), czy mapy rozbieżności (`roslaunch image_view disparity_view image:=/camera/depth/disparity`).

W celu wykorzystania z poziomu opracowywanego oprogramowania informacji dostarczanych przez węzeł *openni_tracker* należy użyć standardowych bibliotek dostarczanych przez pakiet *Tf*. Przykładowo, by uruchomić węzeł, który będzie cyklicznie informował o transformacjach pomiędzy wybranymi układami współrzędnych należy:

1. Wejść do katalogu użytkownika, przeznaczonego do przechowywania plików źródłowych pakietów systemu ROS (`cd ~/catkin_ws/src`).
2. Zainicjować nowy projekt ROSa (np. `catkin_create_pkg kinect_example`).
3. W katalogu zainicjowanego projektu utworzyć katalog `bin` i umieścić w nim plik z przykładowym programem pokazanym na rysunku 2 (np. w pliku `kinect.py` – znajdziesz go w katalogu `/opt/ROS_Lab1_5/python`)^{||}. Ustawić prawo do wykonywania umieszczonego pliku (`chmod +x kinect.py`).
4. Wejść do katalogu projektów systemu ROS (`cd ~/catkin_ws`), skompilować projekt (`catkin_make`)^{**} i zaktualizować zmienne systemowe (`source devel/setup.bash`).

[¶]Jeśli przy uruchomieniu lokalnego procesu jądra ROS pojawiło się wspomiane ostrzeżenie o adresie jądra, przed wydaniem polecenia należy nie zapomnieć o właściwym ustawieniu wartości zmiennej `ROS_MASTER_URI` (np. `export ROS_MASTER_URI=http://10.104.16.102:11311/`).

^{||}Ponieważ przykładowy program jest przygotowany w języku skryptowym, jego źródło zostaje umieszczone od razu w katalogu przeznaczonym do przechowywania plików wykonywalnych. W takim przypadku nie jest potrzebny proces kompilacji programu i związana z nim modyfikacja pliku `CMakeLists.txt`. W przykładzie tym pomijamy kwestie związane z właściwym dokumentowaniem pakietów, wymagającym modyfikacji pliku `package.xml`.

^{**}Wprawdzie przygotowywany program jest napisany w niewymagającym kompilacji języku skryptowym, jednakże proces kompilacji w systemie ROS jest potrzebny do uaktualnienia informacji o dostępnych w systemie pakietach – stąd w przypadkach takich jak tu czynność tę musimy wykonać, ale jedynie jednorazowo, po dodaniu do systemu nowego projektu.


```
#!/usr/bin/env python
import roslib
import rospy
import tf
import geometry_msgs.msg

def runme():
    rospy.init_node('kinect')
    listener = tf.TransformListener()           #inicjacja subskrypcji wiadomosci tf
    rate = rospy.Rate(2.0)                     #ustalenie czestotliwosci nasluchiwania
    while not rospy.is_shutdown():
        try:                                    #odczyt transformacji
            (trans,rot) = \
                listener.lookupTransform('/right_hand_1', '/left_hand_1', rospy.Time(0))
        except(tf.LookupException,tf.ConnectivityException,tf.ExtrapolationException):
            continue
        rospy.loginfo("odleglosc: %f",trans[0]); #wyswietlenie wspolrzednej x
        rate.sleep()

if __name__ == '__main__':
    runme()
```

Rysunek 2: Przykład programu w pythonie do nasłuchiwanie informacji dostarczanych przez węzeł *openni_tracker*

5. Utworzyć węzeł, który będzie realizował zaprogramowaną w powyższy sposób funkcjonalność (rosrun kinect_example kinect.py¹). W momencie, gdy uruchomiony wcześniej węzeł *openni_tracker* zidentyfikuje użytkownika nr 1 utworzony węzeł powinien wypisać współrzędną x transformacji pomiędzy wskazanymi w programie układami współrzędnych.

7 Zadania do wykonania

W trakcie ćwiczenia należy:

1. Uruchomić przykładową sesję systemu ROS wykorzystującą czujnik Kinect (np. opisaną w rozdziale 6).
2. Zidentyfikować układy współrzędnych przypisane przez pakiet *OpenNI_tracker* do szkieletu wraz ze strukturą opisujących je transformacji.
3. Zapropionować sposób interpretacji danych pochodzących z węzła *openni_tracker*, pozwalający na sterowanie robotem.
4. Uruchomić węzeł obsługujący czujnik Kinect w sposób wskazany przez prowadzącego, pozwalający na wykorzystanie generowanych przez nie komunikatów do sterowania przydzielonym do realizacji ćwiczenia robotem Pioneer P3-DX.
5. Oprogramować i przetestować zaproponowany sposób interpretacji danych na symulatorze robota.
6. Przygotować robota Pioneer P3-DX do pracy [3].
7. Przetestować działanie opracowanego algorytmu sterowania na robocie.

8 Sprawozdanie

Sprawozdanie z przebiegu ćwiczenia powinno zawierać:

- Imię i nazwisko autora, numer i termin grupy, skład grupy, temat ćwiczenia, datę wykonania ćwiczenia.
- Cel ćwiczenia.
- Opis przebiegu identyfikacji transformacji układów współrzędnych publikowanych przez węzeł *openni_tracker*.
- Charakterystykę zaproponowanego sposobu interpretacji danych pochodzących z węzła *openni_tracker*, pozwalającego na sterowanie robotem.
- Opis sposobu uruchomienia węzłów obsługujących czujnik Kinect, pozwalającego na wykorzystanie czujnika w zadaniu sterowania robotem.
- Podsumowanie przebiegu testu działania zaproponowanego rozwiązania na symulatorze robota.
- Opis przebiegu testów na rzeczywistym robocie.
- Wnioski końcowe.

Literatura

- [1] Tully Foote. tf: The transform library. *Technologies for Practical Robot Applications (TePRA)*, 2013 IEEE International Conference on, Open-Source Software Workshop, strony 1–6, 2013.
- [2] Mariusz Janiak. *Instrukcja bezpieczeństwa i higieny pracy w Laboratorium Robotów Autonomicznych L1.5*. Katedra Cybernetyki i Robotyki, Politechnika Wrocławska, Instrukcja Laboratorium Robotów Autonomicznych L1.5, 2015.
- [3] Mariusz Janiak, Aleksandra Grzelak. *Instrukcja bezpieczeństwa i higieny pracy przy obsłudze robotów Pioneer 3-DX*. Katedra Cybernetyki i Robotyki, Politechnika Wrocławska, Instrukcja Laboratorium Robotów Autonomicznych L1.5, 2015.
- [4] Jason M. O’Kane. *A Gentle Introduction to ROS*. Independently published, 2013. Available at <http://www.cse.sc.edu/~jokane/agitr/>.
- [5] Joanna Ratajczak. *Weryfikacja własności ruchowych układów nieholonomicznych na przykładzie robota mobilnego Pioneer 3DX*. Katedra Cybernetyki i Robotyki, Politechnika Wrocławska, Instrukcja Laboratorium Robotów Autonomicznych L1.5, 2015.
- [6] Rethink Robotics. Baxter Research Robot Wiki: Rviz. <http://sdk.rethinkrobotics.com/wiki/Rviz>, 2015.
- [7] ROS.org. Openni – Package Summary. <http://wiki.ros.org/openni>, 2015.
- [8] ROS.org. Openni_camera – Package Summary. http://wiki.ros.org/openni_camera, 2015.
- [9] ROS.org. Openni_launch – Package Summary. http://wiki.ros.org/openni_launch, 2015.

-
- [10] ROS.org. Openni_tracker – Package Summary. http://wiki.ros.org/openni_tracker, 2015.
 - [11] ROS.org. Rviz – User Guide. <http://wiki.ros.org/rviz/UserGuide>, 2015.
 - [12] ROS.org. Tf – Package Summary. <http://wiki.ros.org/tf>, 2015.
 - [13] ROS.org. Tutorials: Introduction to tf. <http://wiki.ros.org/tf/Tutorials/Introduction%20to%20tf>, 2015.
 - [14] Debian Wiki. PrimeSenseNite. <https://wiki.debian.org/PrimeSenseNite>, 2015.
 - [15] Wikipedia. Kinect. <http://pl.wikipedia.org/wiki/Kinect>, 2015.
 - [16] Wikipedia. OpenNI. <http://pl.wikipedia.org/wiki/OpenNI>, 2015.
 - [17] Wikipedia. PrimeSense: NiTE Middleware. <http://en.wikipedia.org/wiki/PrimeSense>, 2015.
 - [18] Wikipedia. ROS (Robot Operating System). http://pl.wikipedia.org/wiki/ROS_%28Robot_Operating_System%29, 2015.
 - [19] Katarzyna Zadarnowska. *Wprowadzenie do Systemu Operacyjnego Robotów ROS: symulator turtlesim*. Katedra Cybernetyki i Robotyki, Politechnika Wrocławska, Instrukcja Laboratorium Robotów Autonomicznych L1.5, 2015.