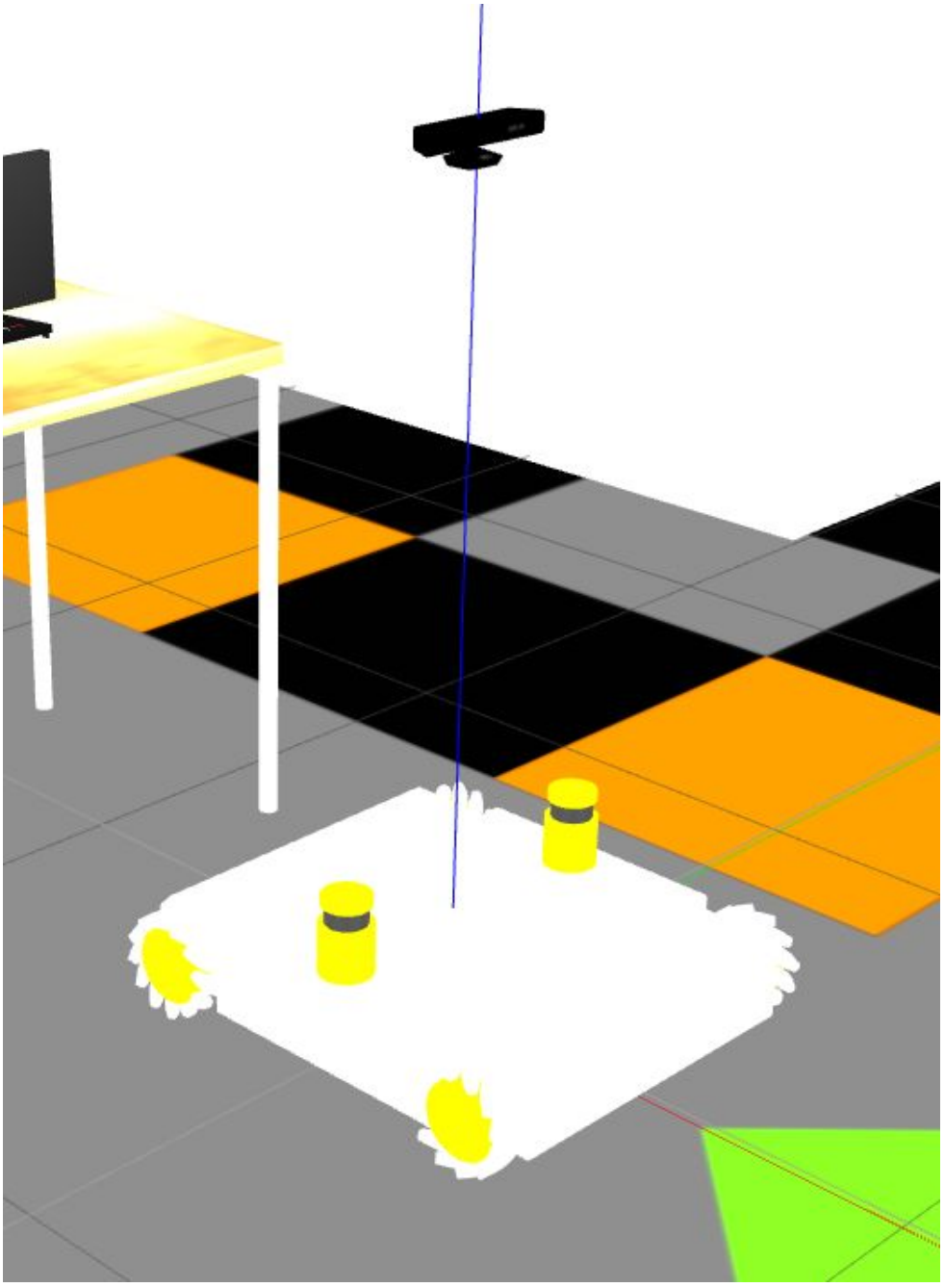
Dokumentacja symulatora Omnivelma



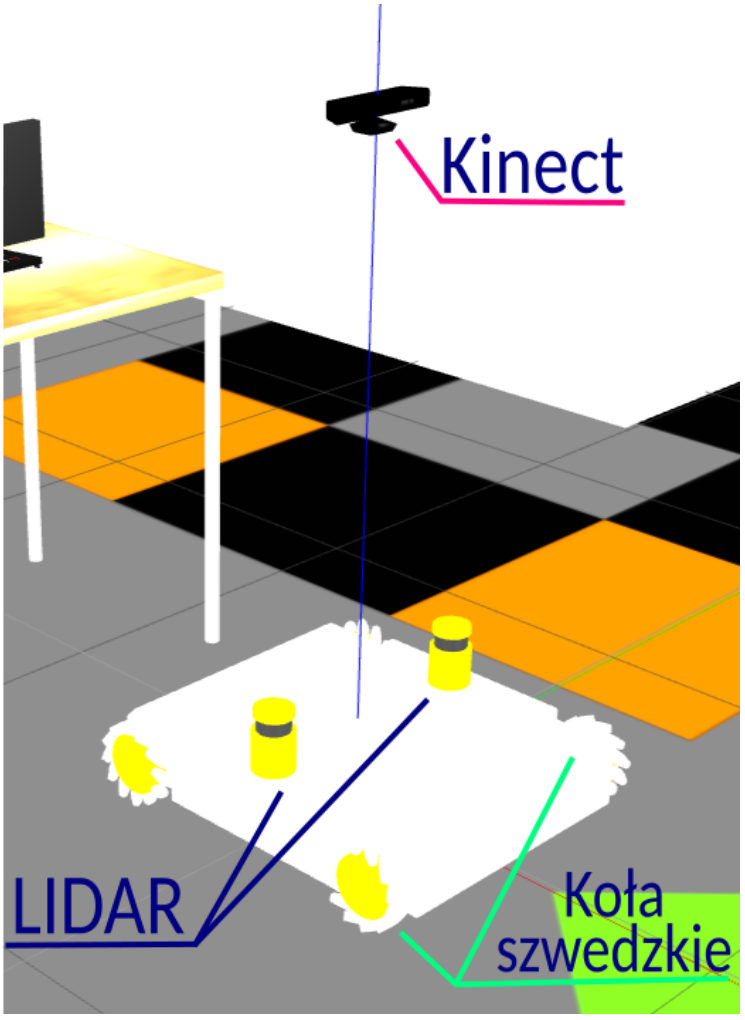
Rysunek 1: Widok bazy dookólnej Omnivelma w symulatorze Gazebo

Spis Treści

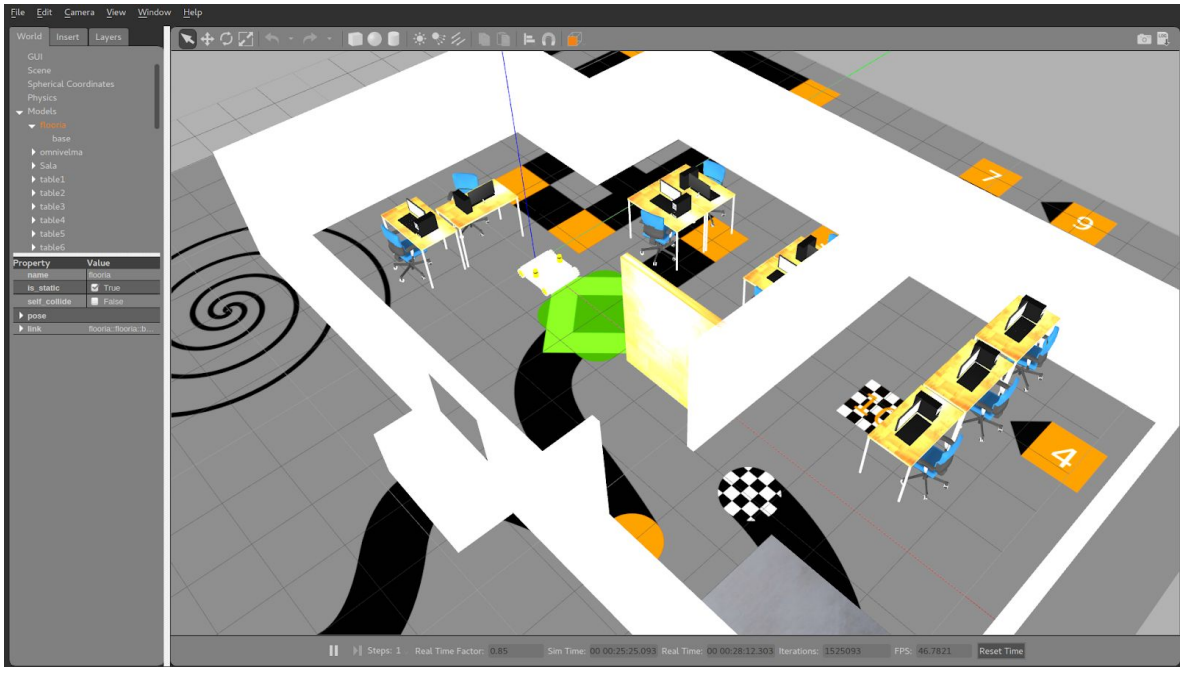
1. Opis systemu

1.1. Ogólnie o symulatorze

System zawiera symulator bazy dookólnej nazwanej tutaj Omnivelma. Jest on uruchamiany w programie Gazebo (wersja 7). Jego implementacja polegała na zdefiniowaniu plików sdf modelu oraz zaimplementowaniu jego logiki jako tzw. model plugin do Gazebo. Baza ta posiada możliwość jazdy w dowolnym kierunku (dzięki zastosowaniu kół szwedzkich) oraz rotacji w miejscu. Na bazie zamontowano dwa czujniki laserowe (LIDAR) oraz dodano obsługę symulowanego Kinecta, wraz z jego obrotem (w osi z).



Rysunek 2: Podstawowe elementy modelu



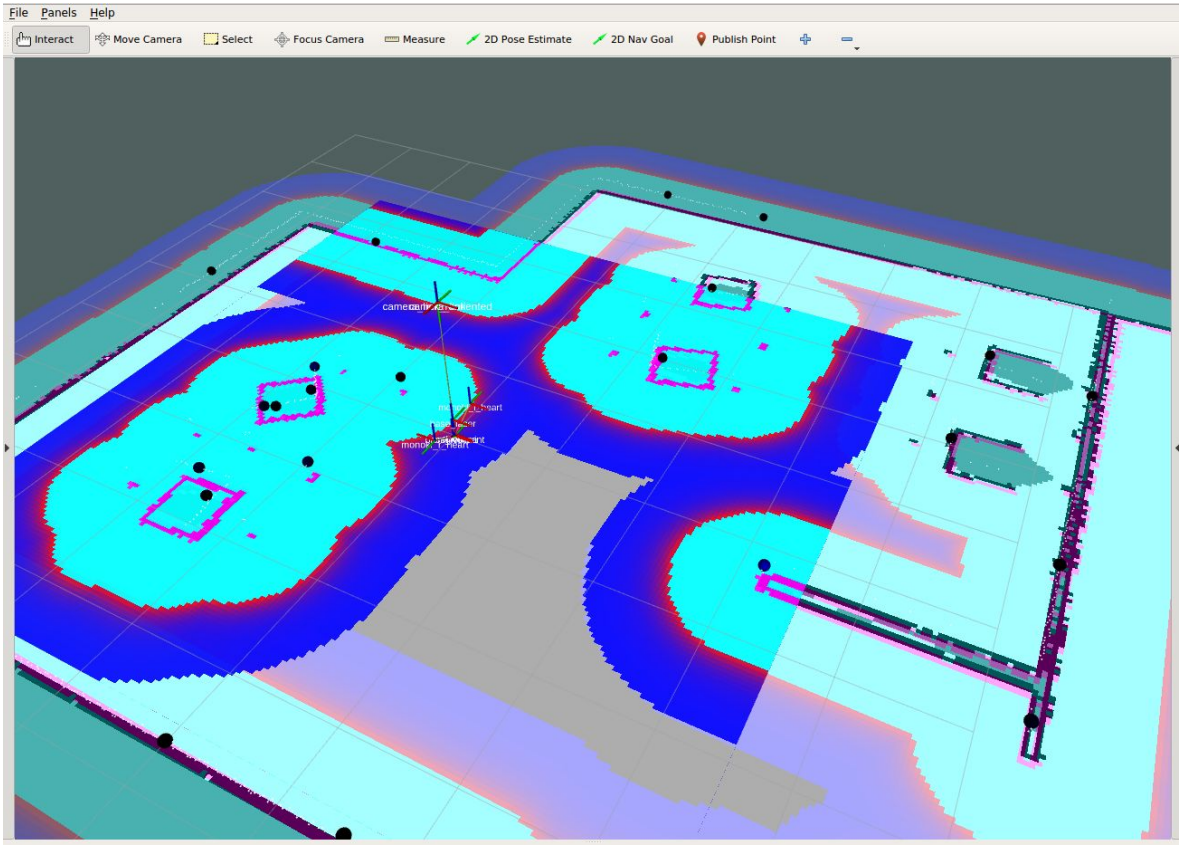
Rysunek 3: Widok symulacji w programie Gazebo

Zadaniem systemu jest pozyskanie danych ze środowiska (odczyty LIDARów i Kinecta), w którym znajduję się robot i obróbka tych danych w celu stworzenia mapy kosztów. Mapa kosztów tworzona jest również na podstawie statycznej, przygotowanej uprzednio mapy pomieszczenia. Robotowi (np. za pomocą programu RVIZ) można zadać punkt do którego powinien się udać. W tym celu planery globalny i lokalny, odwołując się do utworzonej mapy kosztów tworzą ścieżkę od położenia robota do zadanego celu (planer globalny, korzystając z algorytmu A*) bądź jej fragment (planer lokalny – Dynamic Window Approach). Następnie kontroler bazy zadaje odpowiednie prędkości liniowe i kątowe, tak by robot poruszał się po wytyczonej ścieżce. Działając na zasadzie sprzężenia zwrotnego odczytana zostaje informacja o aktualnej prędkości i położeniu robota w celu weryfikacji i dostosowania zadawanych prędkości.

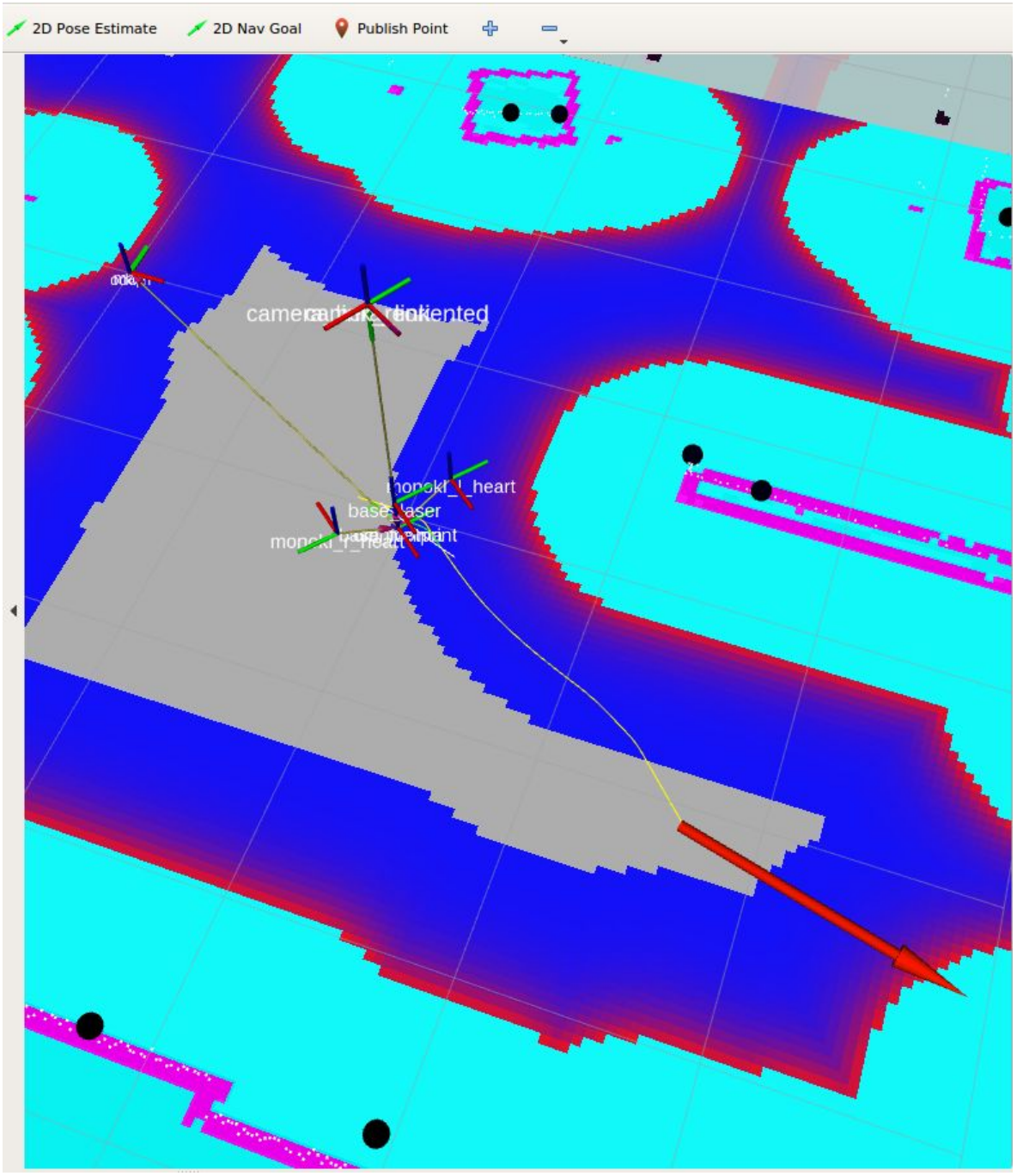
Symulator posiada możliwość ręcznego zadawania prędkości robota poprzez moduł o nazwie Lalkarz, którego autorem jest pierwotny autor symulatora. Subiektywnie najwygodniejszą opcją jest tryb sterowania znajdujący się pod klawiszem F11, pozwalający na sterowanie robota za pomocą myszki.

1.1. Wizualizacja w programie RVIZ

RVIZ jest programem służącym do wizualizacji, pozwalającym na wyświetlenie w czasie rzeczywistym tego co dzieje się w symulacji, wszelkich dodatkowych danych, które nie mogą być wyświetlone w Gazebo, takie jak np. mapa kosztów, chmura punktów, zaplanowana ścieżka ruchu



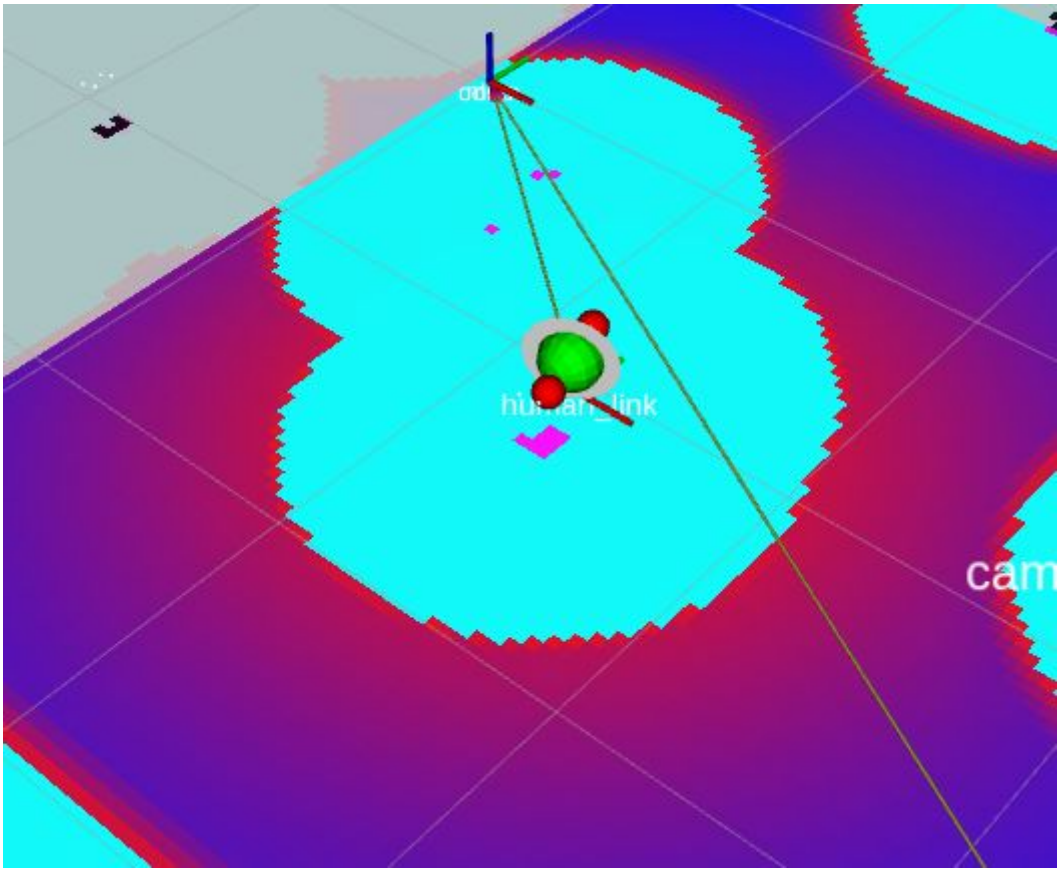
Rysunek 4: Widok symulacji w programie RVIZ. Na rysunku widać wygenerowaną mapę kosztów. Różne odcienie niebieskiego i czerwonego oznaczają koszt w danym miejscu. Lokalna mapa kosztów posiada duży kontrast, natomiast mapa globalna została ukazana jako ta z niskim kontrastem. Na różowo widzimy obszary w których wykryto przeszkody.



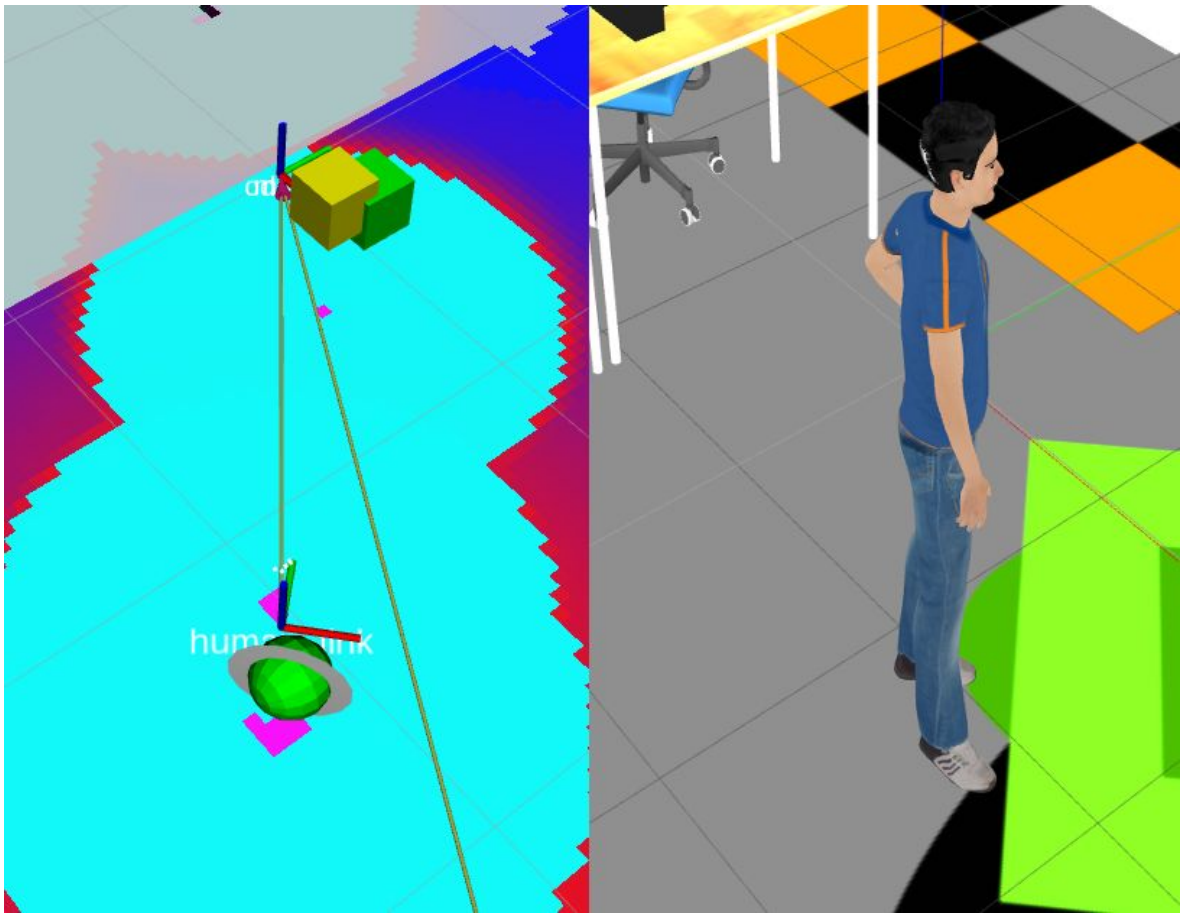
Rysunek 5: Widok poruszającego się robota. W centralnej części rysunku widać robota (ramki TF), pozycję docelową (czerwona strzałka) oraz ścieżkę poruszania się robota (żółta linia to ścieżka lokalna ruchu). Na rysunku znajduję również zielona linia będąca ścieżką globalna, lecz pokrywa się ona ze ścieżką lokalną, dlatego jest niewidoczna.

1.2. Wykrywanie ludzi

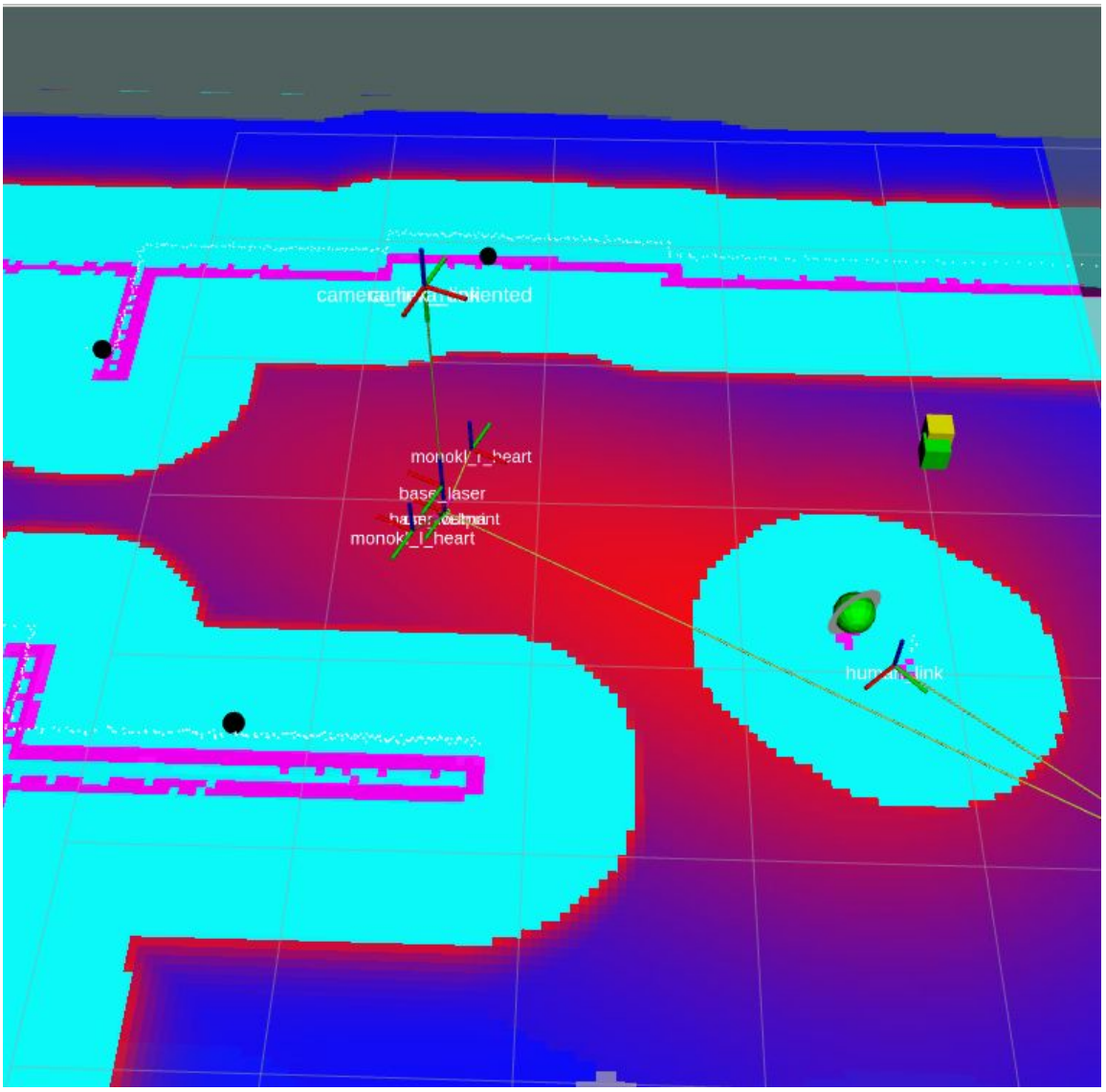
Kolejnym zadaniem robota jest wykrycie w środowisku pracy człowieka. Realizowane jest to za pomocą skanerów laserowych – wykrywane są nogi człowieka, a także Kinecta – wykrywane są fragmenty głowy modelu (front, profil nosa, bok głowy, tył), na podstawie których wyliczona zostaje pozycja i orientacja człowieka. Układ rozpoznaje również prędkości i kierunek poruszania się człowieka (poprzez odczyty LIDARów). Następnie informacje o pozycji, orientacji i prędkości człowieka interpretowane są przez odpowiednie moduły, które aktualizują mapę kosztów.



Rysunek 6: Marker ukazujący pozycję człowieka wykrytego przez LIDARy.



Rysunek 7: Z lewej strony, widać markery ukazujące wykryte części głowy (żółty - bok głowy, zielony - profil nosa). Z prawej korespondujący widok z Gazebo.



Rysunek 8: Rysunek przedstawia wykrytego człowieka oraz odpowiednio zaktualizowaną mapę kosztów. Kolor czerwony symbolizuję duży koszt ruchu.

Kinect posiada możliwość obrotu, tak by mógł śledzić wykrytą osobę, lub gdy takowa nie istnieje, obrać się zgodnie z kierunkiem wektora prędkości robota.

1.3. Model człowieka

Model człowieka został wykonany z użyciem programu Make Human (http://www.makehuman.org/) oraz Blender (https://www.blender.org/) w wersji 2.78.

Rysunek 9: Model człowieka w programie Gazebo

Pierwotnie model wygenerowany w programie Make Human, wymagał obróbki ze względu na nieprawidłowe ustawienie wartości kanałów alfa tekstur, oraz po to, by dodać z tyłu głowy coś, co przypomina kod QR. Powodem jest fakt, iż tył głowy jest elementem ciężko identyfikowalnym przez detektory, dlatego została podjęta decyzja o pewnym uproszczeniu (czy właściwie uszczegółowieniu) modelu.

Rysunek 10: Pseudokod QR pozwalający na wykrycie tyłu głowy.

W ten sposób otrzymujemy model 3D, tzw. mesha. Kolejnym krokiem było stworzenie modelu człowieka jako model z rozszerzeniem sdf.

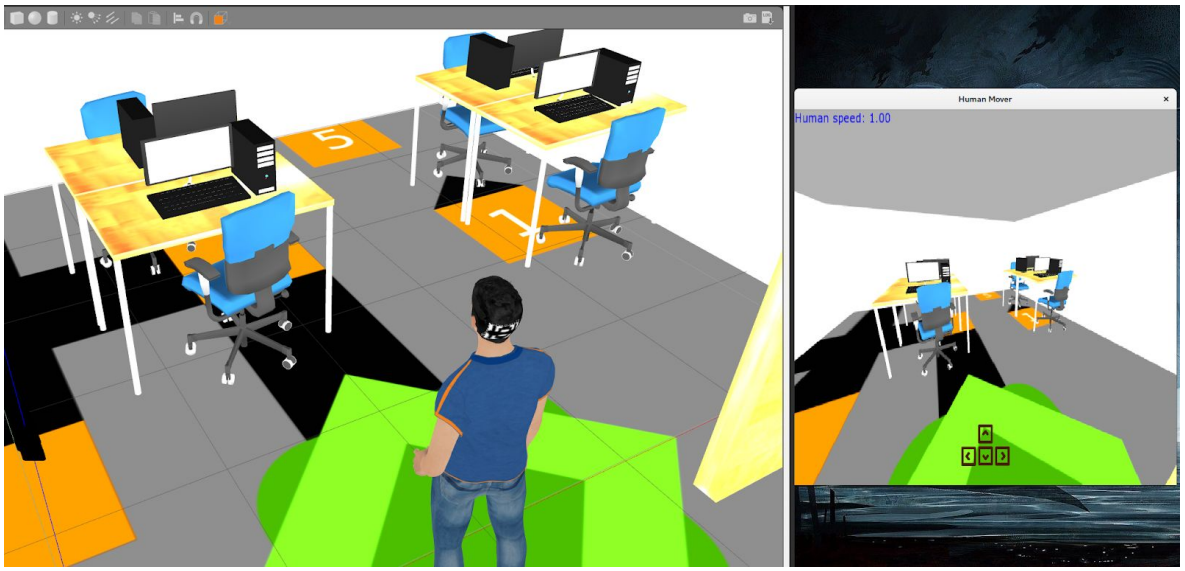
Bardzo dobrym poradnikiem jest ten film:

https://www.youtube.com/watch?v=7kEmT-NE75c&t=547s

Obrazuje on jak stworzyć człowieka, a następnie model w sdf.
Uwaga: Działa dla Blendera 2.78, w wyższym prawdopodobnie zmienił się sposób eksportu pliku collada, a dokładniej tekstur. Być może zmienił się jedynie interfejs użytkownika, ale nie udało mi się znaleźć odpowiednich opcji, o których mowa w poradniku.

1.4. Poruszanie się człowieka

Modelem człowieka, po dodaniu go do symulacji w Gazebo, można sterować za pomocą programu human_mover (rosrun human_mover human_mover). Program zapewnia widok z perspektywy pierwszoosobowej. Sterowanie odbywa się z pomocą strzałek na klawiaturze. Prędkością można sterować za pomocą klawiszy +/-. Wykorzystano bibliotekę graficzną SFML.



Rysunek 11: Widok programu human_mover umożliwiającego sterowanie człowiekiem oraz ten sam widok z perspektywy Gazebo

2. Proces instalacji

- Tworzymy katalog w którym zamierzamy przechowywać system Omnivelmy. Na przykład ~/catkin_pw

http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

- Pobieramy repozytorium DODAJ LINK
- Pobieramy bibliotekę dlib i ją instalujemy https://github.com/davisking/dlib
- Zmieniamy ścieżkę do dliba w pakiecie object_detector (CMAKELIST)
- W katalogu src znajduje się skrypt install_omnivelma_with_dependencies.sh, który zainstaluje nam system ROS, a także pakiety niezbędne do prawidłowej kompilacji i działania systemu.

- Proces instalacji jest nieco skomplikowany ze względu na pewne zależności w pakiecie people. Pierwotnie został o przygotowany na którąś ze starszych dystrybucji ROSa, zawierającą OpenCV2. Od dystrybucji Kinetic został on zastąpiony OpenCV3, z czym nie poradzili sobie twórcy pakietu people. Dokładniej mówiąc leg_detector musi zostać skompilowany z użyciem OpenCV2, a face_detector z OpenCV3.

  Dlatego w celu kompilacji repozytorium zawiera pakiety opencv2_catkin oraz opencv3_catkin, będące wrapperami OpenCV, pozwalającymi na kompilacji z użyciem catkina. Niestety OpenCV2 i OpenCV3 nie mogą być skompilowane i używane jednocześnie, ze względu na współdzielenie pewnego katalogu. Oznacza to, że w danym momencie możemy używać tylko jednej z wersji. face_detector, o czym można przeczytać w opisie pakietów, jest obecnie nieużywany, co rozwiązuje niejako problem, lecz gdyby zaistniała konieczność jego kompilacji wraz z leg_detectorem należy:
  - skompilować opencv2_catkin (wyłączając z niej opencv3_catkin poprzez zmianę nazwy pliku package.xml na package.xmlNOBUILD) oraz wszystkie programy pakietu

people za wyjątkiem face_detector (aby odznaczyć go z kompilacji należy zmienić nazwę pliku package.xml na package.xmlNOBUILD)

    - po udanej kompilacji należy dodać do kompilacji opencv3_catkin i face_detector (nie wyłączając z niej poprzednio skompilowanych pakietów!). Jeżeli nie dokonamy zmian w plikach i nie zajdzie potrzeba rekompilacji, wszystkie programu pakietu people powinny działać prawidłowo. W przeciwnym wypadku należy dokonać kompilacji odpowiedniej wersji opencvX_catkin

- Przechodzimy do katalogu ~/catkin_pw

- source devel/setup.bash
- catkin_make

- Poszczególne pakiety mogą być kompilowane pojedynczo z wykorzystaniem catkin_make -DCATKIN_WHITELIST_PACKAGES=”nazwa_pakietu”. Aby ponownie skompilować całość należy użyć komendy catkin_make -DCATKIN_WHITELIST_PACKAGES=””

3. Przykładowy opis użycia

Po udanej kompilacji należy przejść do katalogu src/omnivelma_navigation i wykonać source init.sh. Spowoduje to załadowanie odpowiednich zmiennych środowiskowych. Znajduję się tam również komenda, która pozwala na zmianę ścieżek w których python poszukuje modułów, domyślnie wyłączona.

Przed uruchomieniem symulatora należy:

- przejść do katalogu src/omnivelma_navigation/
- w pliku move_base zmienić ścieżkę do pliku mapy:

<arg name="map_file" default="/home/pwalas1/catkin_pw/src/omnivelma_navigation/maps/lab_map.yaml"/>

- w pliku detectors.launch zmienić ścieżkę do modeli części twarzy

np. z
/home/pwalas1/catkin_pw/src/object_detector/params/nose_side_detector.svm
na
twój_folder_catkin/src/object_detector/params/nose_side_detector.svm

Następnie należy wywołać ./launch.sh. Czekamy aż uruchomiony zostanie Gazebo. Pierwsze uruchomienie może potrwać dłużej, ponieważ Gazebo pobiera pewne elementy z sieci. Jeżeli jednak dostajemy komunikat o błędzie, bądź czas oczekiwania jest wyjątkowo długi (więcej niż 5min przy pierwszym uruchomieniu i 1min przy kolejnych, gdzie jednak Gazebo udało się wcześniej uruchomić) może to oznaczać, że nieprawidłowo ustawione są ścieżki do modeli/pluginów, bądź Gazebo nie potrafi wczytać któregoś z modeli. W takiej sytuacji należy poczekać na komunikat o błędzie lub przejść do katalogu src/omnivelma/src/velmaverse/worlds/omnivelma.world i dokonać prób usunięcia części modeli w celu znalezienia przyczyny.

Gdy Gazebo zostanie uruchomione (DODAJ ODPOWIEDNI KOMUNIKAT!), należy w przejść do konsoli i wcisnąć dowolny klawisz. Spowoduje to ładowanie pakietów nawigacyjnych.

Po uruchomieniu symulacji oraz pakietów symulacyjnych (powinien pojawić się komunikat „odom received"). Symulator jest gotowy do pracy.

Uwaga: mogą pojawić się komunikaty o błędach – jeden pochodzi z ira_laser_merger, jeżeli mówi on o tym, nie udało się pobrać pomiarów z przeszłości to nie należy się nim przejmować. W trakcie działania programu będą się pojawiać wyjątki w działaniu leg_detector mówiący o niemożności dzielenia przez zero. Nie stwierdzono jednak negatywnego działania tych błędów na ogólne działanie detektora (powinny być one jednak obsłużone w samym detektorze, co jednak należało by do jego twórców).
Kolejnym działaniem powinno być uruchomienie programu RVIZ (rosrun rviz rviz) i zwizualizowanie takich elementów jak mapy kosztów, ścieżki poruszania się, ramki tf.

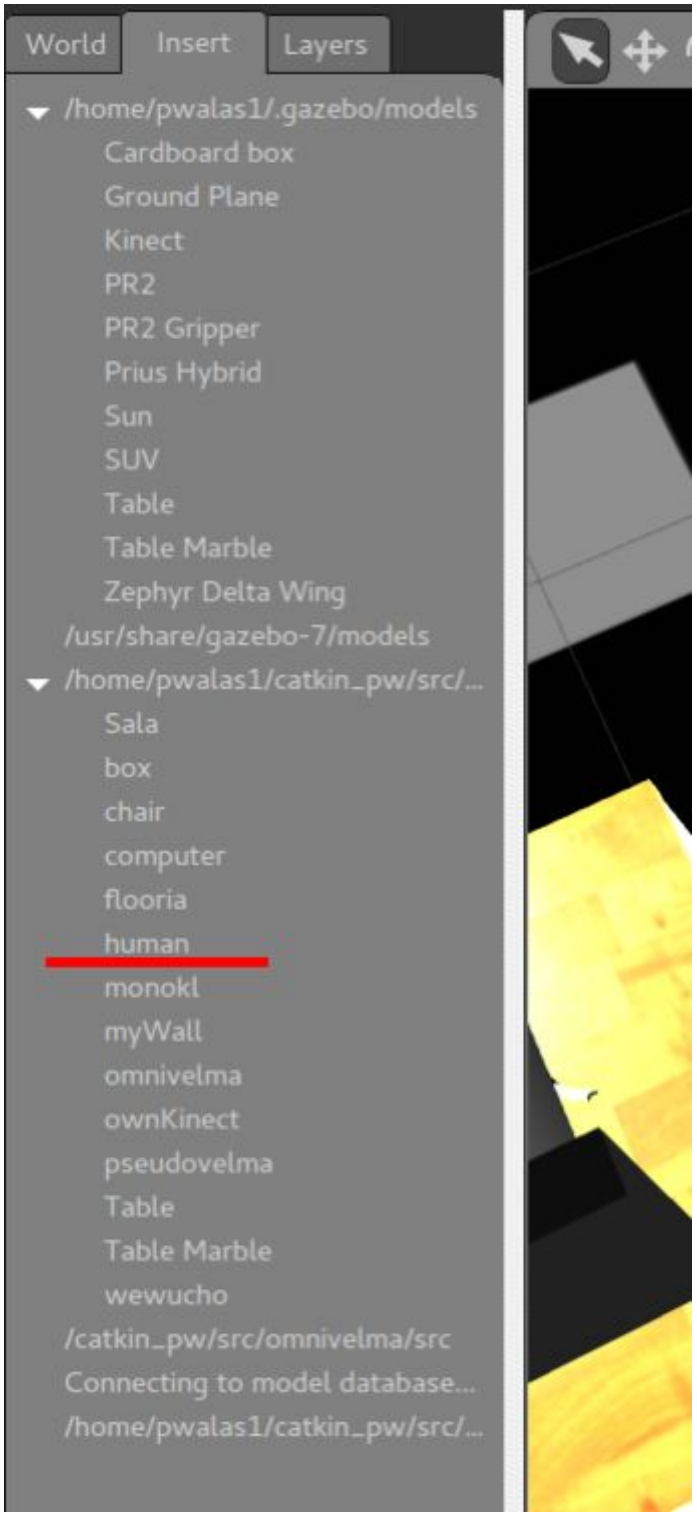
Więcej można przeczytać pod tym adresem:

http://wiki.ros.org/navigation/Tutorials/Using%20rviz%20with%20the%20Navigation%20Stack

Kolejną rzeczą, którą może zrobić użytkownik to wydanie robotowi rozkazu poruszania się do określonego punktu. W tym celu w programie RVIZ należy skorzystać z opcji 2D Nav Goal

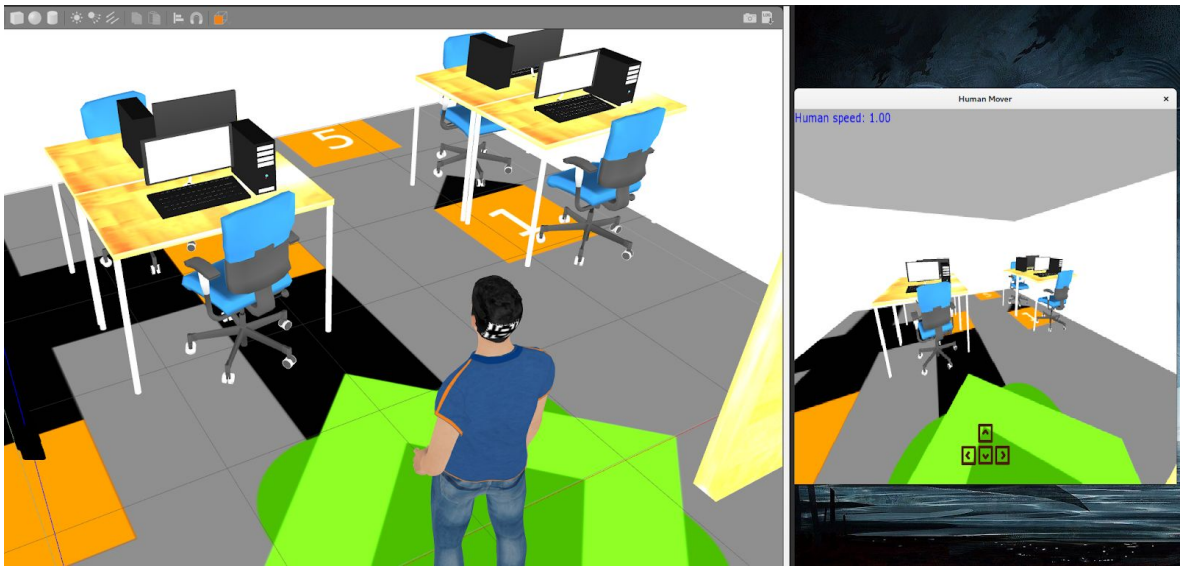
Rysunek 12: Widok programu RVIZ wraz z zaznaczoną opcją 2D Nav Goal

Do symulacji można wprowadzić człowieka, jeżeli wszystkie zmienne środowiskowe zostały zapisane prawidłowo, to można to zrobić z panelu bocznego Gazebo



Rysunek 13: Opcja dodania modelu człowieka poprzez panel Insert Gazebo

Ostatecznie można, za pośrednictwem programu human_mover, sterować danym modelem człowieka.



Rysunek 14: Widok Gazebo (lewa strona) oraz programu human_mover

4. Opis użytych pakietów

4.1. Omnivelma

Pakiet zawierająca zmodyfikowany model bazy jezdnej velmy nazwany "omnivelma". Pierwotnym autorem jest Radosław Świątkiewicz -

https://github.com/Antyradek/omnivelma

Względem pierwowzoru zmieniono:

- Obrócono model o 90 stopni w osi z - tak by jego (umowna, ze względu na fakt dookólności) oś x pokrywała się z osią x układu wykorzystywanego przez pakiet nawigacyjny ROSa.
- Dodano rozgłaszanie odometrii przez model dynamiczny.
- Usunięto parametry solvera ode ustawione przez autora - uniemożliwiały uruchomienie symulacji.
- Dodano obsługe kinecta, wraz z jego prostym obracaniem poprzez rozgłaszanie kąta obrotu na odpowiednim temacie.
- Dodano model człowieka.
- Zmieniono strukturę wysyłanych ramek
- Dodano narzędzie opisane jako human_mover - umożliwia poruszanie się człowiekiem w sumulacji patrząc z jego perspektywy. Sterowanie odbywa się za pomocą strzałek na klawiaturze. Prędkość człowieka kontrolowana jest poprzez klawisze +/-.
- Dodano model komputera, stolika, krzesła oraz wymodelowano pomieszczenie laboratoryjne (w sposób uproszczony).

4.2. omnivelma_navigation

Główny pakiet łączący funkcjonalność poszczególnych pakietów. Istotą pakietu jest uruchomienia symulatora, costmap, planerów, amcla, serwera map, kontrolera prędkości, detektorów oraz niezbędnych pakietów dostarczających transformacje ramek TF. Definiuje również niezbędne parametry za pomocą plików z rozszerzeniem yaml.

Skrypty uruchomieniowe:

init.sh (użycie: source init.sh) – exportuje zmienne środowiskowe niezbędne do uruchomienia symulatora

launch.sh (użycie: ./launch.sh) – uruchomienie symulatora oraz pakietów zależnych (nawigacji, detektorów). Ze względu na fakt, iż ROS za pomocą polecenia roslaunch uruchamia wszystkie pakiety symultanicznie, a pakiety nawigacyjne oraz detektory wymagają w pełni uruchomionego symulatora zanim one zostaną uruchomione, należy uruchomić program launch.sh (./launch.sh), zaczekać na uruchomienie gazebo, a następnie wcisnąć dowolny klawisz w terminalu by kontynuować ładowanie modułów.

detectors.sh – (użycie ./detectors.sh) – uruchomienie detectora punktów głowy – object_detectora – umożliwia modyfikacje użytych modeli.

haar_detectors.sh – uruchomienie detektorów opartych na kaskadowym detektorze haara – mocno obciążające procesor, dające dużą liczbę fałszywych wykryć. W praktyce uruchamiający program face_detector pakietu people z różnymi modelami i ze wskazaniem na publikowany temat. Ze względu na słabą wydajność praca nad tym rozwiązaniem została porzucona, dlatego też wykryte cechy nie są w żaden sposób interpretowane przez pluginy costmapy (w szczególności przez plugin proxemic_layer odpowiadający za umieszczenie strefy komfortu człowieka na costmapie). Możliwe jest jedynie zwizualizowanie wyników detektorów w rvizie. Możliwe jest jednak obrobienie

danych z tych detektorów w podobny sposób jak zostało to uczynione w pakiecie object_detector.

Pliki .launch:

omnivelma.launch – uruchomienie symulatora (samego symulatora, bez części nawigacyjnej i detektorów).

move_base.launch – uruchomienie pakietu nawigacyjnego (costmap, planerów, amcla, serwera map)

move_base_gmapping.launch – uruchomienie alternatywnej wersji move_base jedynie z podstawowymi pakietami i z slam_gmapping zamiast amcla – wykorzystywany do tworzenia statycznej mapy na podstawie odczytów skanerów laserowych.

detectors.launch – uruchomienie object_detector (patrz detectors.sh).

haar_detectors.launch – uruchomienie detektorów haara (patrz haar_detectors.sh)

Pliki parametrów .yaml:

base_local_planner_params.yaml – definiuje parametry planera globalnego (global_planner) i planera lokalnego (dwa_local_planner)

costmap_common_params.yaml – parametry używane przez globalną i lokalną mapę kosztów

global_costmap_params.yaml – parametry używane przez globalną mapę kosztów.

local_costmap_params.yaml – parametry używane przez lokalną mapę kosztów.

move_base_params.yaml – definiuje parametry planera i kontrolera (głównie częstotliwość ich pracy).

maps:

zawiera dwie przygotowane mapy – initial_map – prosta mapa zawierająca jedynie 3 przeszkody (pudła) oraz lab_map – mapa laboratorium zdefiniowana w w pakiecie omnivelma/velmaverse/worlds/omnivelma.world – uruchamiana standardowo.
Aby użyć mapy initial_map należy w pliku usunąć/zakomentować wszystkie zdefiniowane przeszkody (również ściany laboratorium – model „Sala") i odkomentować końcową część („box1"-"box4") pliku omnivelma/velmaverse/worlds/omnivelma.world

Programy:

camera_tf_broadcaster – wysyła transformację kinecta - dokładniej rzecz ujmując wysyła transformacje camera_link_reoriented → camera_link. Ramka camera_link_reoriented rozysłana jest przez gazebo (plugin do modelu omnivelma). Powodem tego jest, iż obraz (chmura punktów kinecta) ma inny układ współrzędnych niż ten przyjęty przez pakiety ROSa oraz RVIZa. camera_link_reoriented pokazuję fizyczną orientacje kinecta, natomiast camera_link odnosi się do pobieranego obrazu.

omnivelma_tf_broadcaster – wysyła transformacje omnivelma → base_footprint (realnie nie zostaje ona wykorzystana, choć może się zdarzyć, iż pakiet może wymagać transformacji odom->base_footprint->base_link

laser_tf_broadcaster – rozsyła położenie lidarów względem omnivelmy, a także punktu arbitralnie uznanego za środek odcinka pomiędzy nimi – wykorzystywany jest on przez ira_laser_merger

main_controller – program, który zadaje kolejne punkty do których ma udać się omnivelma, po dotarciu do celu wybierany jest kolejny punkt
Temat rozsyłany: *move_base/goal*

omnivelma_speed_broadcaster – program odpowiedzialny za odbiór prędkości z tematu cmd_vel i wysłanie

odpowiednich prędkości do silników omnivelmy (omnivelma/vels).

kinect_follower – program pobiera pozycje ludzi z dwóch źródeł:

people_merger - /people_positions_kinect
obstacle_with_human_layer - /people_positions_laser

Mając dane o wykrytych osobach program wylicza kąt pomiędzy osią x układu współrzędnych i najbliższym człowiekiem – kąt ten jest następnie wysłany w temacie omnivelma/kinect_rotation. Odczytywany jest w pluginie gazebo do modelu omnivelmy, a sam kinect jest obracany do odpowiedniego położenia.

Wykrycia osób pochodzące z kinecta są premiowane ponad te z użyciem LIDARów - jeżeli kinect widzi osobę, a LIDAR wykryję drugą bliżej robota, to i tak śledzona będzie osoba wykryta poprzez Kinecta – związana jest to z faktem, że wskazania kinecta są dokładniejsze, w znaczeniu bardzo małej liczby fałszywych wykryć.

Jeżeli przez ustalony czas nigdzie nie zostanie wykryty człowiek, lub dane o wykrytym przestaną być aktualne Kinect będzie obracany zgodnie z aktualnym wektorem prędkości robota – tak by robot „widział" to co znajduję się w kierunku, w którym się porusza.

Tematy nasłuchiwane: /people_positions_kinect
/people_positions_lidar
/omnivelma/twist

Tematy rozsyłanie: /omnivelma/kinect_rotation

4.3. human_obstacle_msgs

Zawiera prostą wiadomość zawierającą jedynie

geometry_msgs/Point[]

4.4. human_obstacle

Pakiet zawiera pluginy dodające do costmap informacje o wykrytych ludziach:

obstacle_with_human_layer – jest to nieco zmodyfikowana wersja obstacle_layer z pakietu navigation ROSa. Względem pierwowzoru usuwa ona ludzi z costmapy gdy odległość robota od wykrytego człowieka jest większa od ustalonej.

Powodem implementacji był fakt, iż człowiek po tym, gdy został wykryty, a następnie robot oddalił się na pewną odległość nie był usuwany z costmapy o ile robot ponownie nie znalazł się w okolicy miejsca, gdzie wcześniej wykryto człowieka i sensory nie przekazały informacji o tym, że obecnie nie ma już w tym miejscu żadnej przeszkody. Jest to rozwiązanie niepożądane, gdyż człowiek z dużym prawdopodobieństwem się przemieszcza.

Tematy nasłuchiwane: tożsame z pluginem ObstaclePlugin + /people

Tematy rozsyłane: tożsame z pluginem ObstaclePluing + /people_positions_laser

human_layer – OBECNIE NIE WYKORZYSTYWANY **-** plugin odpowiedzialny za dodawanie człowieka do costmapy, jednakże uwzględnia on zarówno położenie człowieka jak i robota.
Początkowym założeniem było, że gdy kinect wykryje położenie człowieka, to costmapa zostanie zaktualizowana o „łezkę” odpowiadającą w jakimś sensie strefie komfortu człowieka, mającą swój początek w pozycji człowieka i skierowana w stronę robota.
W toku dalszych prac zrezygnowano z tego podejścia, na rzecz użycia pluginu proxemic_layer

Tematy nasłuchiwane: /face_detector/faces_cloud_upperbody

Tematy rozsyłane: */people_position_kinect*
/people_facing

human_merger – program umożliwiający zbieranie, łączenie i wysyłanie informacji o wykrytych osobach z dwóch źródeł – LIDARów oraz Kinecta. Informacje te są następnie wysyłane w temacie /people, na którym nasłuchuje proxemic_layer – plugin mapy kosztów umożliwiający umieszczenie osoby oraz jej strefy komfortu na mapie kosztów. Powodem powstania modułu jest fakt, iż proxemic_layer nadpisuje informacje o wykrytych osobach po każdym odebraniu informacji na temacie /people – wysyłając dane z LIDARów, tracimy nie tylko poprzednie dane z LIDARów, ale również wszelkie dane o osobach wykrytych za pomocą Kinecta. Analogiczna sytuacja występuję dla osób wykrytych Kinectem – przy aktualizacji danych tracimy informacje o osobach wykrytych za pomocą LIDARów. Warstwa proxemic posiada wprawdzie parametr czasowy odpowiadający za „bezwładność" pomiarów, czyli utrzymywaniu informacji o poprzednich pomiarach, lecz jego wykorzystanie okazało się niewystarczające.

Tematy nasłuchiwane: /people_detected

Tematy rozsyłane: */people*
/people_position_kinect

4.5. people

Pakiet zawiera detektory nóg i twarzy.

leg detector – na podstawie odczytów z LIDARów pakiet wykrywa pozycje nóg, a na podstawie detekcji nóg wykrywana jest pozycja człowieka. Po zwizualizowaniu wykrytych nóg oraz ludzi wysuwa się wniosek iż pogram generuję dość dużą liczbę fałszywych wykryć nów. Wprawdzie nie oznacza to dużej liczby fałszywych wykryć ludzi, gdyż wykryte nogi muszą spełnić dotyczące chociażby maksymalnej odległości, jednakże istotnie powoduję, że ludzie wykrywani są w miejscu krańców ścian (np. dla ścian

działowych), czy też mówiąc ogólniej, w miejscach ostrych kantów (krawędzie pudeł, szafek). Pakiet zawiera dość pokaźną liczbę parametrów które można skonfigurować, lecz nie udało się znaleźć w pełni zadowalającego zestawu parametrów.

Temat rozsyłany: uwaga: zmiana z /people na /people_lidar

people_velocity_tracker – pakiet odbiera informacje od leg_trackera a następnie analizuję dostarczone dane wyliczając prędkość poruszania się człowieka. Dokonuje konkatenacji danych o położeniu i prędkości a następnie publikuje je w temacie /people_lidar

UWAGA: Standardowo people_velocity_tracker publikuje listę wykrytych ludzi na temacie **/people**, co powodowało kolizję w innym miejscu systemu. Temat ten jest ustawiony na sztywno w kodzie. Dokonano podmiany publikowanego tematu z **/people** na **/people_lidar,** co ostatecznie pozwala na zachowanie czytelności i rozwiązuje problem kolizji nazw tematów.

face-detector – pakiet wykrywa twarz na podstawie uprzednio przygotowanego modelu. Wykorzystuje kaskadowy detektor haara z biblioteki OpenCV. Zasadniczo, podmieniając przygotowany model można wykrywać różne obiekty. W OpenCV dostępne są modele dla twarzy frontalnych, profilów twarzy, całości sylwetki, górnej i dolnej części ciała, a także oczu i uszu. Problemem tego detektora jest duży narzut wydajnościowy oraz duża liczba fałszywych wykryć (szczególnia widoczna w przypadku mniejszych obiektów). W toku prac został porzucony na rzecz przygotowanego pakietu object_detector.

http://wiki.ros.org/people

4.6. inflated_merger

OBECNIE NIE WYKORZYSTYWANY – plugin łączący blisko położone strefy o koście inflated (domyślnie 253). Zapobiega to powstawaniu bardzo wąskich przejść między takimi strefami.

4.7. ira-laser-tools

Pakiet odpowiedzialny za połączenie wyników pomiarów z obu lidarów – połączony pomiar wysyłany jest na temat /scan

Tematy nasłuchiwane: /monokl_l/scan
/monokl_r/scan

Tematy rozsyłane: /scan

https://github.com/iralabdisco/ira_laser_tools

4.8. kinect

Pakiet zawierający sterownik kinecta oraz bibliotekę OpenNI implementującą np. wkrywanie człowieka i jego szkieletyzacje.

4.9. navigation

Pakiet dostarczający costmapy, podstawowe pluginy, planery globalne oraz planery lokalne – jest to standardowy pakiet ROSa.

Obecnie wykorzystane zostały:
Planer globalny: global_planner
Planer lokalny: dwa_local_planner
Kontroler: move_base

http://wiki.ros.org/navigation

4.10. navigation_layers

Pakiet dostarczający plugin pozwalający na uwzględnienie człowieka w środowisku – zarówno jego pozycje jak i rotacje (obecnie rotacje głowy)

http://wiki.ros.org/social_navigation_layers

4.11. object_detector

Pakiet pozwalający na wykrywanie, z założenia dowolnych, obiektów. Został on jednak uszczegółowiony do wykrywania frontu, profilu nosa, profilu głowy oraz tyłu głowy* człowieka znajdującego się w symulacji. Na podstawie wykrytych cech zostaje pobrana informacja o ich położeniu w przestrzeni 3D, a następnie obliczany jest odpowiedni wektor. Wektor ten reprezentuje aktualną rotację głowy. Znając ten wektor, pozycje człowieka oraz pozycje robota można zaktualizować costmapę o umowną strefę komfortu człowieka. W praktyce zrealizowano to poprzez przekazanie informacji o wykrytym człowieku wraz z wektorem rotacji głowy do pluginu proxemic pakietu navigation_layers**.

* Model człowieka, z tyłu głowy, posiada coś na kształt kodu QR. Podczas wykrywania tyłu głowy tak naprawdę to on jest wykrywany. Uproszczenie to wynika z faktu, iż tył głowy jest obiektem mało charakterystycznym i nie ma prostego sposobu na jego wykrycie.

** Warstwa proxemic przyjmuję położenie człowieka i jego prędkość, a nie orientacje i to prędkość jest brana pod uwagę przy wyliczaniu kształtu, który będzie potem naniesiony na mapę kosztu jako umowna strefa komfortu człowieka. Dlatego też wektor rotacji człowieka podawany jest jako jego prędkość.

Tematy nasłuchiwane: /camera/rgb/image_raw
/camera/depth/imager_raw
/omnivelma/pose

Tematy rozsyłane: /people_detected
/face_feature_array

4.12. openni_camera + openni_launch

Pakiety odpowiedzialne za połączenie się do sterownika (fizycznie połączonego do portu USB) kinecta i rozesłanie odpowiednich tematów ROSa z obrazem rgb, obrazem głębi oraz chmurą punktów.

http://wiki.ros.org/openni_camera

4.13. openni_tracker

Pakiet wykrywający ludzi w chmurze punktów dostarczonej z kinecta. Wymaga połączenia ze sterownikiem kinecta, dlatego nie może być on wykorzystany z kinectem symulowanym w gazebo. Informacja o wykrytym człowieku jest publikowana jako zbiór ramek tf.

http://wiki.ros.org/openni_tracker

4.14. catkin_simple

Pakiet użyty do kompilacji opencv2_catkin oraz opencv3_catkin

4.15. cmake_modules

Pakiet użyty do kompilacji opencv2_catkin oraz opencv3_catkin

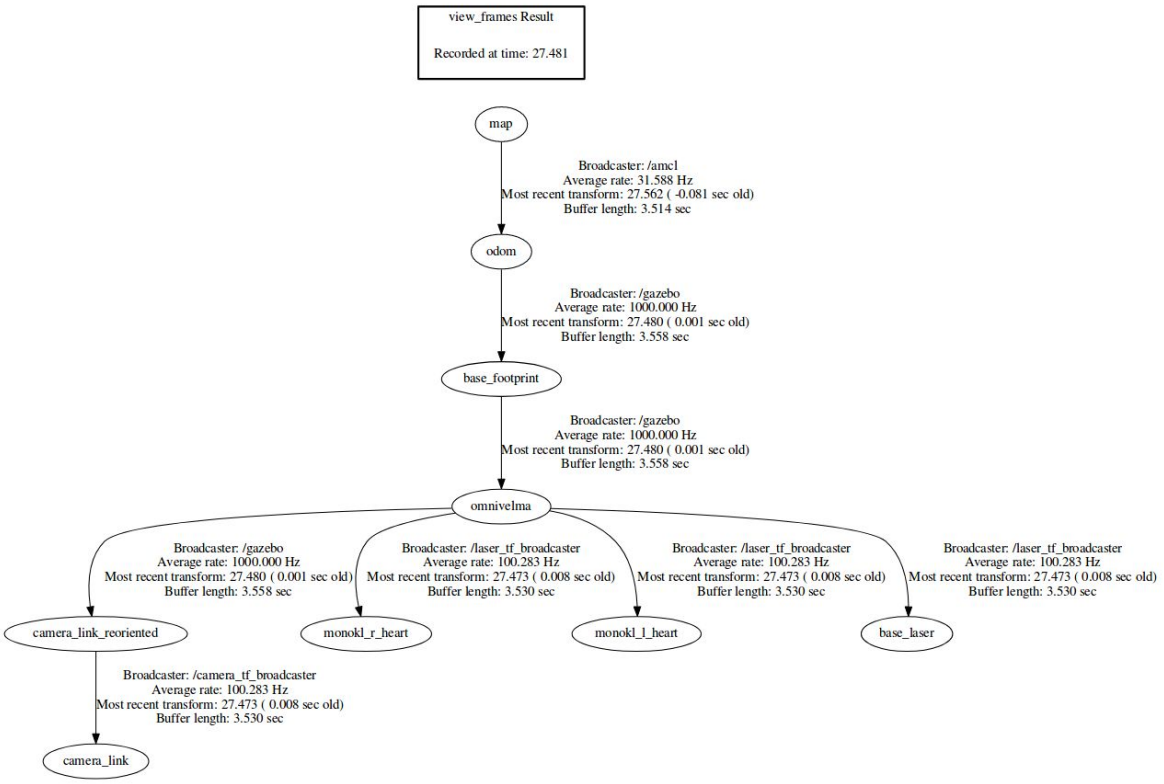
4.16. gazr

Pakiet obecnie nieużywany!

Zawiera implementacje wykrywania twarzy człowieka (jedynie gdy twarz jest skierowana frontalnie +/- 30 stopni. Wykrywanie oparte jest na detektorze z biblioteki dlib oraz modelu o nazwie shape_predictor_68_face_landmarks na którym wyszczególniono 68 punktów twarzy. Na tej podstawie pakiet jest w stanie wykryć twarz, wyznaczyć jej położenie oraz położenie punktów charakterystycznych i na tej podstawie wyliczyć kąt obrotu twarzy. Cała operacja jest jednak ograniczona jedynie do twarzy frontalnej, dlatego pakiet ostatecznie nie został użyty.

https://github.com/severin-lemaignan/gazr

5. Struktura TF


view_frames Result
Recorded at time: 27.481
map
Broadcaster: /amcl
Average rate: 31.588 Hz
Most recent transform: 27.562 (-0.081 sec old)
Buffer length: 3.514 sec
odom
Broadcaster: /gazebo
Average rate: 1000.000 Hz
Most recent transform: 27.480 (0.001 sec old)
Buffer length: 3.558 sec
base_footprint
Broadcaster: /gazebo
Average rate: 1000.000 Hz
Most recent transform: 27.480 (0.001 sec old)
Buffer length: 3.558 sec
omnivelma
Broadcaster: /gazebo
Average rate: 1000.000 Hz
Most recent transform: 27.480 (0.001 sec old)
Buffer length: 3.558 sec
Broadcaster: /laser_tf_broadcaster
Average rate: 100.283 Hz
Most recent transform: 27.473 (0.008 sec old)
Buffer length: 3.530 sec
Broadcaster: /laser_tf_broadcaster
Average rate: 100.283 Hz
Most recent transform: 27.473 (0.008 sec old)
Buffer length: 3.530 sec
Broadcaster: /laser_tf_broadcaster
Average rate: 100.283 Hz
Most recent transform: 27.473 (0.008 sec old)
Buffer length: 3.530 sec
camera_link_reoriented
monokl_r_heart
monokl_l_heart
base_laser
Broadcaster: /camera_tf_broadcaster
Average rate: 100.283 Hz
Most recent transform: 27.473 (0.008 sec old)
Buffer length: 3.530 sec
camera_link