



Politechnika Wroclawska

Projekt Przejściowy Wirtualne Laboratorium L1.5

Prowadzący:
Janusz Jakubiak
Robert Muszyński

Artur Błażejowski
Kamil Bogus
Michał Burdka
Rafał Cymiński
Paweł Jachimowski
Krzysztof Kuczyński
Krzysztof Kwieciński
Witold Lipieta
Michał Prędkiewicz
Jędrzej Stolarz

Rok akademicki 2016/17

Wrocław, luty 2017

Spis treści

1. Wstęp	5
1.1. Motywacja	5
1.2. Cel projektu	5
2. Wykorzystane narzędzia	7
2.1. Docker	7
2.1.1. Podstawowe elementy	8
2.2. Gazebo	9
2.3. ROS	10
2.3.1. Struktura	10
2.4. Qt	12
2.5. Inventor	13
2.6. Blender	13
2.7. Zarządzanie projektem	14
2.7.1. Git	14
2.7.2. Trello	14
3. Opis systemu	17
3.1. Okienka	17
3.1.1. Konfiguracja świata	17
3.1.2. Zarządzanie robotami	18
3.1.3. Wyniki symulacji	19
3.2. Pluginy	21
3.2.1. GUI	22
3.2.2. World	22
3.3. Integracja Gazebo z ROS	23
3.3.1. Kompilacja pakietów Gazebo przy użyciu catkin	23
3.3.2. Uruchamianie Gazebo przy użyciu narzędzi z ROS	24
3.3.3. Udostępnione funkcjonalności do komunikacji ROS	24
3.4. Modele	24
4. Testy	27
5. Podsumowanie i wnioski	29
5.1. Serwer	29
5.2. Zastosowanie kontenerów	29
5.3. Zawartość projektu	30
5.4. Możliwości rozwoju	31
A. Instrukcja dla użytkownika	33
A.1. Wstęp	33
A.2. Przygotowanie systemu do uruchomienia	33
A.3. Obsługa systemu	35
A.3.1. Dodawanie modelu sali	35
A.3.2. Dodawanie modeli robotów	35
A.3.3. Resetowanie robotów i ustawianie ich w zadanych miejscach	36

A.3.4. Wyświetlanie i zapisywanie wyników	36
A.3.5. Wyświetlanie odczytów z czujników	37
A.3.6. Dodawanie i edycja modeli przeszkód na scenie	38
B. Instrukcja dla dewelopera	39
B.1. Wstęp	39
B.2. System operacyjny	39
B.3. Docker	39
B.4. Obraz	39
B.5. Repozytoria	40
B.6. Uruchamianie obrazu i konfiguracja	40
B.7. Dockerfile	41

1. Wstęp

1.1. Motywacja

Projekt wirtualnego laboratorium powstał jako pomoc dydaktyczna do zajęć laboratoryjnych wykonywanych na robotach Pioneer w środowisku ROS. Dzięki środowisku studenci mogą w ramach przygotowania do zajęć wykonać ćwiczenia na robotach Pioneer w systemie ROS dostępnych w wirtualnym laboratorium. Ponadto, środowisko pozwala zainteresowanym na zapoznanie się ze sposobem budowy środowiska jako odizolowanego kontenera dockera i jego obsługą.

1.2. Cel projektu

Celem projektu jest stworzenie wirtualnego laboratorium umożliwiającego obsługę robotów Pioneer w systemie ROS w środowisku odpowiadającym warunkom rzeczywistego laboratorium. Dostarczone środowisko umożliwia wybór laboratorium i robotów Pioneer oraz pozwala na dodawanie elementów sceny, takich jak przeszkody. Sterowanie robotami możliwe jest dzięki platformie ROS. Symulator pozwala na rejestrację i zapis pozycji robota w postaci graficznej wykresów oraz danych.

Środowisko dostarczone jest w postaci kontenera dockera, na którym znajduje się system operacyjny ubuntu 16.04 ROS Kintetic oraz gazebo 7.5. Osoba, której zadanie ograniczać się będzie do pracy z systemem (np. w celu wykonania ćwiczeń laboratoryjnych), powinna zapoznać się z instrukcją zawartą w dodatku A. Materiały zamieszczone w dodatku B mają ułatwić rozwój i modyfikacje projektu.

2. Wykorzystane narzędzia

2.1. Docker



Docker to narzędzie szturmem zdobywające popularność na serwerach, szczególnie w środowiskach chmurowych, gdzie z powodzeniem wspiera lub czasem nawet zastępuje klasyczną wirtualizację oferowaną przez rozwiązania typu VMware lub XEN.

Docker działa tylko na jądrze Linux i pozwala uruchamiać tylko aplikacje przeznaczone dla Linuxa, ale dla wszystkich użytkowników Windows i Mac jest przygotowane narzędzie Docker Toolbox, które pozwala zainstalować Dockera w minimalnej maszynie wirtualnej pod kontrolą VirtualBox'a.

Zdecydowaną przewagą Dockera nad wirtualizacją jest możliwość uruchomienia aplikacji w wydzielonym kontenerze, ale bez konieczności emulowania całej warstwy sprzętowej i systemu operacyjnego. Docker uruchamia w kontenerze tylko i wyłącznie proces(y) aplikacji i nic więcej. Efektem jest większa efektywność wykorzystania zasobów sprzętowych, co przy rozproszonych aplikacjach instalowanych do tej pory na kilkunastu bądź kilkadziesiąciu wirtualnych maszynach przynosi konkretne oszczędności.

Docker zastępuje wirtualizację przez stosowanie konteneryzacji. Konteneryzacja polega na tym, że umożliwia uruchomienie wskazanych procesów aplikacji w wydzielonych kontenerach, które z punktu widzenia aplikacji są odrębnymi instancjami środowiska uruchomieniowego. Każdy kontener posiada wydzielony obszar pamięci, odrębny interfejs sieciowy z własnym prywatnym adresem IP oraz wydzielony obszar na dysku, na którym znajduje się zainstalowany obraz systemu operacyjnego i wszystkich zależności / bibliotek potrzebnych do działania aplikacji.

Kontenery Dockera działają niezależnie od siebie i do chwili, w której świadomie wskażemy zależność pomiędzy nimi, nic o sobie nie wiedzą.

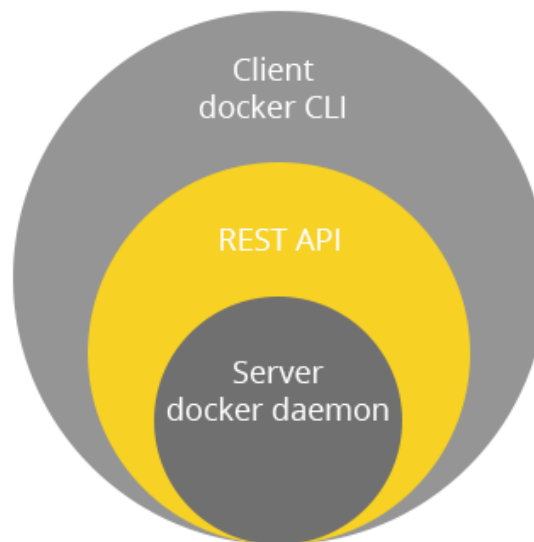
Dwie główne korzyści płynące z korzystania z Dockera to łatwość tworzenia środowisk deweloperskich oraz uproszczenie procesów dostarczania gotowych aplikacji na docelowe środowiska.

Zmora każdego programisty jest tworzenie środowiska deweloperskiego na potrzeby każdego kolejnego projektu. Wiadomo, że każdy projekt będzie działał na innej bazie danych, innym kontenerze aplikacji z inną listą dodatkowych usług, które do czasu pojawienia się narzędzi typu Vagrant instalowane były bezpośrednio na laptopie programisty. Utrzymanie kilku środowisk dla wielu projektów bywało niemożliwe. Vagrant w pewien sposób rozwiązuje ten problem przez tworzenie wirtualnego środowiska instalowanego za pomocą odpowiednich skryptów, ale to rozwiązanie nie rozwiązuje wszystkich problemów: nadal musimy napisać skrypty instalujące wszystkie zależności i potrzebny jest mocny sprzęt żeby udźwignąć pełne środowisko w trybie wirtualizacji. Kilka takich środowisk na laptopie może też skutecznie zająć całą przestrzeń dyskową.

Z pomocą przychodzi Docker, który pozwala zbudować środowisko deweloperskie bez wirtualizacji i bez większego wysiłku związanego z instalacją oprogramowania. Docker pozwala wykorzystywać gotowe obrazy zainstalowanych systemów, aplikacji i baz danych, które zostały wcześniej przygotowane i umieszczone w publicznym rejestrze. Rejestr jest dostępny za darmo i zawiera obrazy oficjalnie budowane przez opiekunów / twórców konkretnych rozwiązań: <https://hub.docker.com/explore/>. Dzięki temu, jeśli chcemy używać V-Rep, Gazebo czy ROS, jest duża szansa na to, że gotowy obraz będziemy mogli pobrać z repozytorium i nie tracić czasu na jego przygotowanie. Jeśli jednak nie znajdziemy tego czego szukamy, to zawsze możemy zbudować własny obraz bazując na jednym z bardziej generycznych zawierających tylko zainstalowany system operacyjny (Ubuntu, Fedora, itp.) lub zainstalowane środowisko uruchomieniowe (Java, Python czy ASP.NET).

2.1.1. Podstawowe elementy

Docker Engine to główna składowa i aplikacja typu klient-serwer, złożona z trzech elementów: klienta (CLI), serwera (daemon-a) oraz REST API.



Daemon Docker-a jest centralnym miejscem, z którego następuje zarządzanie kontenerami jak i obrazami. Dotyczy to zarówno ich pobierania, budowy czy uruchamiania. Polecenia do wykonywania tych czynności są przesyłane przez klienta z wykorzystaniem Docker REST API. Same obrazy są natomiast przechowywane w repozytorium obrazów, czy to publicznym czy prywatnym np. **Docker Hub**. W tym miejscu należy zaznaczyć, że poza wieloma oficjalnymi obrazami udostępniane są również nieoficjalne : budowane przez społeczność Docker-a.

W odróżnieniu od maszyn wirtualnych, kontenery wymagają dużo mniejszych zasobów do samego uruchomienia, a i sam czas ich uruchomienia jest znacząco niższy. Zostało to jednak uzyskane kosztem zmniejszenia izolacji pomiędzy kontenerem, a samym systemem operacyjnym : poprzez współdzielenie jądra systemu.

Obrazy są tak naprawdę szablonami w trybie read-only, z których kontenery są uruchamiane. Składają się one z wielu warstw (layer-ów), które, dzięki zastosowaniu ujednoczonego systemu plików (UFS), Docker łączy w jeden konkretny obraz. Podstawą każdego jest obraz bazowy, np. Ubuntu, na który nakładane są kolejne warstwy. Każda kolejna czynność (instrukcja) wykonywana na obrazie bazowym tworzy kolejną warstwę, np. wykonanie komendy czy utworzenie pliku/katalogu. Komplet instrukcji tworzących obraz

jest przechowywany w pliku Dockerfile. Podczas żądania pobrania obrazu przez klienta, plik ten jest przetwarzany, czego wynikiem jest finalny obraz.

Poprzez zastosowanie warstw uzyskano bardzo niskie zużycie przestrzeni dyskowej, ponieważ podczas zmiany obrazu czy jego aktualizacji, budowana jest nowa warstwa, która zastępuje poprzednią (aktualizowaną). Pozostałe warstwy pozostają nienaruszone. Oznacza to, iż możliwe jest współdzielenie warstw tylko do odczytu pomiędzy kontenerami, czego efektem jest dużo niższe zużycie przestrzeni dyskowej, w porównaniu do standardowych VM.

Registry, czyli repozytorium obrazów, jest faktycznym miejscem przechowywania obrazów. Może być publiczne bądź prywatne, lokalne bądź zdalne. Najpopularniejszym repozytorium jest Docker Hub, który oferuje wiele dodatkowych funkcjonalności, np. możliwość utworzenia repozytorium prywatnego. Oczywiście istnieją inne platformy, które pozwalają na przechowywanie swoich obrazów, jak np. Quay.io, ale możliwe jest także utworzenie własnej biblioteki : czy to lokalnie na komputerze, na którym został zainstalowany Docker czy zdalnie, na jednej z innych maszyn, którymi zarządzamy.

Wspomniane wielokrotnie **kontener** to efekt ujednoczenia warstw tylko do odczytu oraz pojedynczej warstwy do odczytu i zapisu, dzięki której możliwe jest funkcjonowanie wymaganych zadań. Kontener wykonuje określone wcześniej zadanie, najczęściej jedno. Zawiera system operacyjny, pliki użytkownika, a także tzw. metadane, dodawane automatycznie podczas tworzenia bądź startu kontenera.

Kontener określany jest również jako środowisko wykonywalne Dockera. Może przyjmować jeden z pięciu stanów:

- created : utworzony, gotowy do uruchomienia,
- up : działający, wykonujący zadanie,
- exited : wyłączony, w trybie bezczynności po zakończeniu zadania,
- paused : wstrzymany,
- restarting : w trakcie ponownego uruchamiania.

Czas działania kontenera jest zależny od zadania, które wykonuje. Samo uruchomienie składa się z siedmiu kroków:

1. Pobrania wybranego obrazu, pod warunkiem, że nie został już pobrany wcześniej.
2. Utworzenia kontenera.
3. Załadowania systemu plików i utworzenia warstwy do odczytu i zapisu.
4. Zainicjowania sieci bądź mostka sieciowego.
5. Konfiguracji sieci (adresu IP).
6. Uruchomienia zadania.
7. Przechwytywania wyjścia i prowadzenia dziennika zdarzeń.

Możliwe jest łączenie (linkowanie) kontenerów, przez co zyskują one bezpośrednio połączenie ze sobą. Same zadania wykonywane przez kontener są tak samo wydajne, jakby były uruchamianie bezpośrednio w systemie gospodarza.

2.2. Gazebo

Symulator robotów jest doskonałym narzędziem dla każdej osoby zajmującej się robotyką. Pozwala szybko przetestować różne algorytmy i konstrukcje oraz skomplikowane systemy realizujące niecodzienne scenariusze. Jednym z takich narzędzi jest darmowy program Gazebo, przeznaczony do tworzenia dokładnych i efektywnych symulacji 3D robotów działających w złożonych środowiskach. Posiada zaawansowany silnik fizyki, wyso-

kiej jakości grafikę oraz wygodne i programowalne interfejsy. Symulator dostępny jest dla systemu Linux na licencji Apache 2.0.

Program Gazebo powstał na Uniwersytecie Południowej Kalifornii jako część systemu The Player Project, w którego skład wchodził również symulator 2D - Stage.

W 2011 roku symulator ten został zintegrowany z bibliotekami Robot Operating System, co przyczyniło się do znacznego zwiększenia jego popularności. Rozwojem Gazebo zajmuje się teraz Open Source Robotic Foundation.

Silnik graficzny wykorzystany w Gazebo to OGRE (Object Oriented Graphics Rendering Engine). Odpowiada on za wygląd użytych modeli, oświetlenie oraz cienie. Poprawnie odwzorowaną fizykę (między innymi model zderzeń i kolizji) zapewnia silnik ODE (Open Dynamics Engine).

Gazebo daje możliwość symulowania takich sensorów jak: czujniki odległości, kamery, skanery 3D czy moduły GPS. Program pozwala na dodanie do każdego modelu kontrolera, sterującego jego pracą.

Możliwe jest tworzenie prostych modeli bezpośrednio w Gazebo oraz importowanie bardziej złożonych, zaprojektowanych w zewnętrznych programach do grafiki 3D.

Podczas wyboru symulatora najistotniejsza była możliwość integracji z ROsem, a pod tym względem Gazebo jest bezkonkurencyjne. Nie bez znaczenia jest również dobra dokumentacja dostępna na stronie <http://gazebosim.org/>.

2.3. ROS

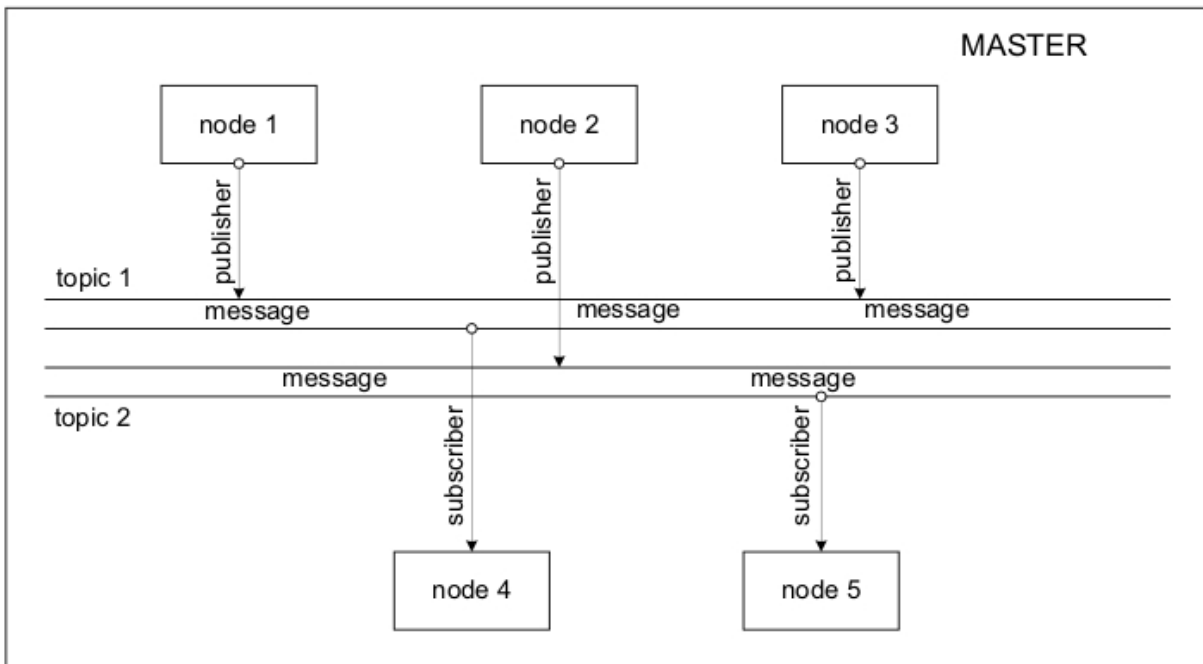
ROS jest meta-systemem operacyjnym rozwijanym w ramach ruchu wolnego oprogramowania. Zawiera: podstawowe procesy systemowe obsługujące urządzenia sprzętowe robota, sterowanie niskopoziomowe, implementacje wykonywania typowych funkcji, komunikację międzywątkową oraz zarządzanie pakietami. Oprócz tego dostarcza użytkownikowi narzędzia i biblioteki pozwalające na tworzenie i uruchomienie programu równocześnie na wielu komputerach. Jest platformą programistyczną przeznaczoną do tworzenia oprogramowania sterującego robotami. ROS jest szczegółowo opisany na stronie internetowej projektu [?].

2.3.1. Struktura

Dzięki swojej strukturze (zbiór narzędzi, bibliotek i konwencji) ROS znacząco upraszcza proces modelowania złożonych zachowań robota jednocześnie zapewniając łatwość przenośności kodu pomiędzy różnymi platformami robotycznymi. Zasada działania projektu w ROS polega na komunikacji *peer-to-peer* luźno połączonych ze sobą procesów (mogą być uruchomione na różnych maszynach) za pomocą infrastruktury komunikacyjnej zapewnianej przez platformę. Idea struktury środowiska przedstawiona jest na rysunku 2.1.

Pakiety (*packages*)

Pakiety są główną jednostką służącą do organizacji oprogramowania. Mogą zawierać: węzły, biblioteki zależne od ROSa, zbiory danych, pliki konfiguracyjne lub inne elementy użytecznie powiązane ze sobą. Pakiety są najmniejszą jednostką budulcową systemu ROS, czyli najmniejszą rzeczą, którą można zbudować i udostępnić. Każdy z nich zawiera dokumentację (*manifest*) opisaną w pliku `package.xml`.



Rysunek 2.1. Schemat projektu na platformie ROS [?]

Węzły (*nodes*)

Węzły są wykonywalnymi instancjami programów środowiska ROS. System sterowania robotem zazwyczaj składa się z wielu węzłów. Przykładowo dla jednego robota mobilnego można wyróżnić kilka węzłów odpowiadających za różne funkcjonalności: skaner laserowy, silniki, lokalizację, planowanie ruchu itd. Wyróżnia się również węzeł nadrzędny *master*, który odpowiada za poprawność komunikacji między wszystkimi węzłami systemu.

Tematy (*topics*) i wiadomości (*messages*)

Węzły porozumiewają się ze sobą poprzez przesyłanie wiadomości. Wiadomość jest prostą strukturą danych, w której znajdować się mogą pola różnych typów prostych oraz tablic typów prostych. Co istotne, struktura może mieć zagnieżdżoną budowę, czyli składać się z dowolnej liczby zagnieżdżonych struktur i tablic. Wiadomości są przesyłane poprzez system transportowy zorganizowany na zasadzie nadawca–odbiorca (*publisher–subscriber*).

Węzeł wysyła wiadomość poprzez opublikowanie jej w danym temacie. Temat jest nazwą służącą do identyfikacji zawartości wiadomości. Węzeł odbierający śledzi wybrane tematy i otrzymuje wiadomości wysyłane do nich. Może być wiele równoczesnych nadawców i odbiorców dla jednego tematu oraz jeden węzeł może publikować i subskrybować wiele tematów. Poszczególne węzły uczestniczące w komunikacji nie są świadome istnienia innych. Odpowiada to idei rozłączenia produkcji danych od ich konsumpcji.

Usługi (*services*)

Usługi udostępniają inny mechanizm komunikacji pomiędzy węzłami. Usługa jest zdefiniowana poprzez parę struktur wiadomości: jedną dla żądania, drugą dla odpowiedzi. W przeciwieństwie do prostego przesyłania wiadomości, które przekazywane są w jednym kierunku na zasadzie wiele-do-wielu, usługi pozwalają na interakcję żądanie/odpowiedź.

Takie podejście jest często wymagane w rozproszonym systemie. Węzeł-serwer oferuje usługę o danej nazwie, natomiast węzeł-klient korzysta z niej poprzez zgłoszenie wiadomości żądania, a następnie oczekiwanie na odpowiedź.

Worki (*bags*)

Worki są formatem zapisu i odtwarzania informacji zawartych w wiadomościach. Jest to istotny mechanizm służący do przechowywania danych, które mogą być trudne do zebrania np. danych sensorycznych. Umożliwiają również testowanie programów dla jednolitego, uprzednio zdefiniowanego, zestawu danych.

2.4. Qt



Qt jest to wieloplatformowy zestaw narzędzi i bibliotek dedykowany dla języka C++. Środowisko to jest dostępne dla takich platform jak X11 (m. in. GNU/LINUX, BSD, Solaris), Windows, Mac OS X oraz dla urządzeń wbudowanych opartych na linuxie, Windows CE, Symbian czy Android. Podstawą działania bibliotek Qt jest mechanizm slotów i sygnałów, które zapewniają obsługę wszelkich zdarzeń w obrębie aplikacji i służy komunikacji pomiędzy obiektami. Odbywa się to w sposób następujący, podczas działania naszej aplikacji pewne specyficzne akcje, predefiniowane bądź dodane przez użytkownika, emitują sygnał zawierający informacje o danym zdarzeniu. Slot natomiast jest to funkcja, która zostaje wywołana jako reakcja na wcześniej przypisany jej konkretny sygnał. Znaczącą część pracy wykonuje MOC (Qt Meta Object Compiler), generujący kod odpowiedzialny za obsługę wszystkich mechanizmów bibliotek Qt, z których w danej aplikacji korzystamy. Dzięki temu twórcy tego rozwiązania w bardzo dużym stopniu uprościli oraz przyspieszyli procedurę pisania kodu przez programistę.

Ważnym przy wyborze narzędzi do zaimplementowania prostego GUI na potrzeby realizowanego projektu był fakt, iż cały interfejs symulatora Gazebo został oparty właśnie o omawiane tutaj biblioteki graficzne. Dodatkowo posiada on dość duże wsparcie dla tworzenia własnych pluginów zintegrowanych z owym środowiskiem, co znacząco wpłynęło na komfort korzystania z naszej aplikacji. Należy również wspomnieć o prostocie wykorzystywania tego rozwiązania oraz dobrej dokumentacji zarówno jeżeli chodzi o same biblioteki Qt oraz ich składowe jak i poradniki dotyczące tworzenia pluginów do symulatora Gazebo i komunikacji między nimi. Posługując się owymi bibliotekami w wersji Qt4 i ich podstawowymi klasami stworzono kilka prostych okien, w których mieści się niemalże całość funkcjonalności naszego projektu. W ten sposób stworzona została również belka z przyciskami zakotwiczona w interfejsie symulatora a także wspomniane wcześniej okna dające użytkownikowi możliwość kontrolowania symulacji czy zarządzania naszymi modelami robotów bądź sali. Owe okna a także dostarczona przez nie funkcjonalność jest szczegółowo opisana w dalszej części raportu.



2.5. Inventor

Środowisko Autodesk Inventor Professional jest środowiskiem inżynierskim pozwalającym na wytwarzanie wirtualnych modeli 3D rzeczywistych obiektów. Inventor w swojej gamie posiada narzędzia pozwalające na szybkie wytwarzanie brył o precyzyjnie określonym kształcie i wymiarach. Właśnie dzięki takim możliwościom został wykorzystany przy tworzeniu modeli przedmiotów umieszczonych w sali. Budowa modelu konkretnego przedmiotu rozpoczyna się tu od się od narysowania kształtu przedmiotu w postaci szkicu 2D, a następnie wyciągnięcia tegoż szkicu do bryły trójwymiarowej. Aby rozwijać, wytworzoną w ten sposób, bryłę tworzy się kolejne szkice płaskie i również się je wyciąga. Dzięki takiemu rozwiązaniu możemy rozwijać nasz o kolejne elementy jego wyglądu. Po zakończeniu podstawowego wyciągania brył mamy do dyspozycji takie narzędzia jak np. fazowanie krawędzi, wiercenie otworów, zmiana materiałów wykorzystanych do produkcji przedmiotu i wiele innych. Dzięki tym funkcjonalnościom można doprowadzić model do stanu, w którym będzie on idealnie odwzorowywał prawdziwy przedmiot zarówno kształtem jak i wyglądem. Oczywiście Inventor Professional, jako profesjonalne narzędzie inżynierskie, posiada szereg zaawansowanych funkcji do badania modelu. Są to między innymi obliczenia bezwładności brył, wytrzymałości ram nośnych, wykrywanie kolizji w zaawansowanych układach kinematycznych, tworzenie rysunków poglądowych, montażowych czy przekroi itd. Z racji prostoty modeli przedmiotów w sali znaczna większość z tych narzędzi nie została użyta przy tworzeniu modeli do projektu.

2.6. Blender



Blender jest bezpłatnym narzędziem do tworzenia grafiki 3D. Umożliwia projektowanie oraz renderowanie zarówno statycznych modeli, jak i animacji, czy gier.

Oprogramowanie to posiada potężne możliwości, dzięki czemu z powodzeniem może konkurować z profesjonalnymi, płatnymi programami takimi jak 3DS Max, czy Maya.

Dzięki dużej popularności i ogromnej rzeszy użytkowników posiada wsparcie społeczności. W połączeniu z licencją open-source i możliwością edycji kodu daje on niemal nieokreślane możliwości przy tworzeniu wszelkiego rodzaju pluginów. Dzięki możliwości używania skryptów w Pythonie jego funkcjonalność można rozszerzać i automatyzować.

W przeciwieństwie do Inventora, modele od początku i w całości są projektowane w widoku 3D. Wynika to z faktu, iż Blender znajduje zastosowania głównie przy tworzeniu grafiki wirtualnej, a nie przy projektach technicznych. W takim przypadku tego typu podejście do projektowania lepiej się sprawdza.

Blender umożliwia eksport do formatu Collada (.dae). Funkcja ta była kluczowa z powodu konieczności wykorzystania tego właśnie formatu przy tworzeniu modeli do symulacji w Gazebo.

2.7. Zarządzanie projektem

Zarządzanie rozwojem projektu było realizowane zgodnie z metodologią Scrum. Zakłada ona pracę w krótkich okresach zwanych Sprintami, w których to realizowane są kolejne zadania wybrane przez zespół, a po jego zakończeniu zespół powinien móc dostarczyć kolejne działające funkcjonalności wchodzące w skład finalnego produktu. W przypadku naszego projektu Sprints te trwały dwa lub cztery tygodnie, a w czasie realizacji projektu odbyło się pięć takich iteracji.

Dzięki takiemu podejściu do organizacji pracy, udało się wyeliminować błędne założenia i wyklarować optymalną architekturę tworzonego systemu. Między innymi zrezygnowano z konfiguracji środowiska gazebo poprzez zewnętrzne skrypty, na rzecz wykorzystania pluginów udostępnionych przez to środowisko o których zespół dowiedział się w trakcie trwania drugiego sprintu. Co spowodowało radykalną zmianę założeń podczas planowania kolejnego sprintu.

2.7.1. Git

Git jest systemem kontroli wersji który bardzo ułatwia zespołową pracę nad projektem, którego główną wartością wytworzoną jest kod źródłowy.

Repozytorium projektu wirtualnego laboratorium L1.5 zostało umieszczone w serwisie Github, wybór ten był podyktowany darmowym dostępem narzędzia oraz mnogością dodatkowych możliwości udostępnionych przez serwis. W celu łatwego zarządzania wszelkimi treściami wytworzonymi w projekcie, założono grupę projektową (<https://github.com/ProjektPrzejscioy>) w której znajdują się wszelkie repozytoria, takie jak na przykład:

- projekt_przejscioy – repozytorium z główną częścią projektu, dodatkami do środowiska Gazebo,
- models – repozytorium zawierające modele pozwalające odwzorować laboratorium L1.5.

2.7.2. Trello

Trello jest to bardzo proste narzędzie wspomagające planowanie i zarządzanie projektami. Możliwe jest stworzenie w nim tablicy do której możemy dodawać listy z kartami, które z kolei można dowolnie edytować i przenosić między listami. Dzięki takiej funkcjonalności możliwe było wykorzystanie Trello jako tablicę Scrum'ową w wersji online. Dodatkowo przy pomocy wtyczki 'Scrum for Trello' było możliwe łatwe przeglądanie estymowanego oraz 'zużytego' czasu dla poszczególnych zadań. Wykorzystanie Trello bardzo pomogło w realizacji projektu, dzięki temu narzędziu członkowie zespołu mieli możliwość wzajemnego egzekwowania wykonywanych przez siebie zadań.

Stworzona podczas realizacji projektu tablica znajduje się pod adresem <https://trello.com/b/WGkwX6TM/backlog>. Według danych w niej zebranych, wszystkie zadania związane z powstawaniem projektu zostały wyestymowane na 926 roboczogodzin, a w rzeczywistości udało się zrealizować je w 592 roboczogodzin.

3. Opis systemu

Symulator robotów jest doskonałym narzędziem dla każdej osoby zajmującej się robotyką. Pozwala szybko przetestować różne algorytmy i konstrukcje oraz skomplikowane systemy realizujące niecodzienne scenariusze. Jednym z takich narzędzi jest darmowy program Gazebo, przeznaczony do tworzenia dokładnych i efektywnych symulacji robotów działających w złożonych środowiskach. Posiada zaawansowany silnik fizyki, wysokiej jakości grafikę oraz wygodne i programowalne interfejsy.

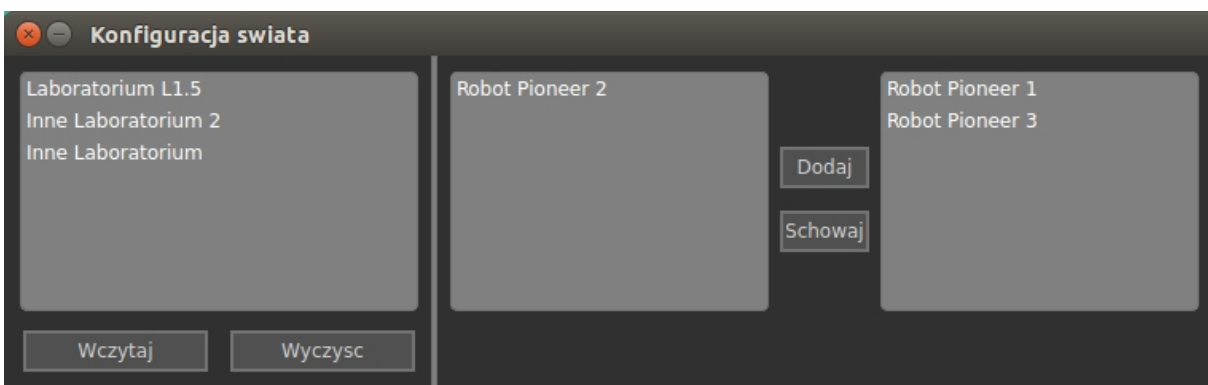
To powyżej to próba tłumaczenia opisu poniżej ze strony Gazebo :D

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community.

3.1. Okienka

W Gazebo do dyspozycji użytkownika udostępnione są trzy okienka stworzone w ramach projektu.

3.1.1. Konfiguracja świata



Rysunek 3.1. Okno konfiguracji świata

Okno konfiguracji świata pozwala zmodyfikować zawartość sceny. Domyślnie ładowany jest jedynie ground plane, czyli bazowe podłoże, na którym umieszczane są modele. Okno umożliwia szybkie rozpoczęcie pracy z wybranymi modelami. Użytkownik ma możliwość wyboru sali z listy dostępnej w lewej części okna. Lista tworzona jest na podstawie plików

konfiguracji sali z rozszerzeniem `.txt` znajdujących się w folderze `worlds`, w katalogu źródłowym Gazebo. Przykładowy zestaw danych ma następującą postać:

```

1 smietnik 4.6 1.3 0 0 0 0
  szafa 1.5 -3.6 0 0 0 0
3 kaloryfer -4.5 1.6 0 0 0 1.570796
  kaloryfer -4.5 -2.8 0 0 0 1.570796
5 stol -2.4 2.5 0 0 0 0
  stol -4 2.5 0 0 0 0
7 stol -0.7 -3.4 0 0 0 0
  krzeslo -1.7 -2.6 0 0 0 0
9 krzeslo -1 -2.6 0 0 0 0
  krzeslo -0.1 -2.6 0 0 0 0

```

W każdej linii dodawany jest jeden z dostępnych modeli. Trzy pierwsze liczby oznaczają pozycję modelu na scenie (X, Y, Z). Pozostałe określają orientację wokół każdej z osi.

Naciśnięcie przycisku „Wczytaj” powoduje wyczyszczenie sceny i załadowanie wybranej sali. Przycisk „Wyczyść” usuwa wszystkie statyczne modele.

Prawa strona okna konfiguracji świata pozwala wybrać roboty, które mają znaleźć się na scenie. Pozycje w lewym polu określają, które z Pioneerów są dostępne, natomiast pozycje w prawym polu definiują roboty znajdujące się na scenie. Zawartość pól, czyli wybór robotów aktywnych na scenie, można modyfikować za pomocą przycisków „Dodaj” i „Schowaj”.

3.1.2. Zarządzanie robotami

Okienko Zarządzanie robotami umożliwia ustawianie pozycji oraz orientacji wybranego robota na scenie.

Zakładki umożliwiają wybór *Pioneer*a, którego pozycję chcemy zmienić. Aktywne są tylko zakładki skojarzone z robotami dodanymi aktualnie do świata.

Po wyborze zakładki możliwe jest podanie nowych współrzędnych XY oraz orientacji θ robota. W polach odpowiedzialnych za pozycję i orientację kolor informuje o poprawności wprowadzonych danych – zielony oznacza poprawnie wypełnione pola, natomiast czerwony błędnie.

Przyciśnięcie przycisku *Ustaw* powoduje ustawienie *Pioneer*a zgodnie z żądaniem. Przycisk *Reset* przenosi robota do punktu $(0, 0)$ i ustawia jego orientację na 0.

Po zmianie zakładki lub ustawieniu/zresetowaniu pozycji, w polach wyświetlane są aktualne współrzędne robota.



Rysunek 3.2. Okno zarządzania robotami

Implementacja

Klasa odpowiadająca za całe okienko to `RobotManagementWindow`. Dziedziczy ona po klasie `QDialog`.

Sloty odpowiadające za to co dzieje się w oknie w momencie dodania lub schowania robota to:

```
public slots:
    void onAddRobot(int id);
    void onHideRobot(int id);
```

W momencie otrzymania odpowiednich sygnałów aktywowana bądź dezaktywowana jest stosowna zakładka, a także ustawiane są wartości w polach.

Wyświetlanie pozycji i orientacji robota możliwe jest dzięki subskrybowaniu odpowiedniego tematu i wywołaniu funkcji aktualizującej pozycję robota:

```
this->sub = node->Subscribe("~/pose/info",
                            &RobotManagementWindow::OnPoseMsg, this);
```

Klasa odpowiadająca za wygląd pojedynczej zakładki oraz główną funkcjonalność okienka to `RobotManagementTab`. Dziedziczy ona po klasie `QWidget`. Jej kluczowe metody to:

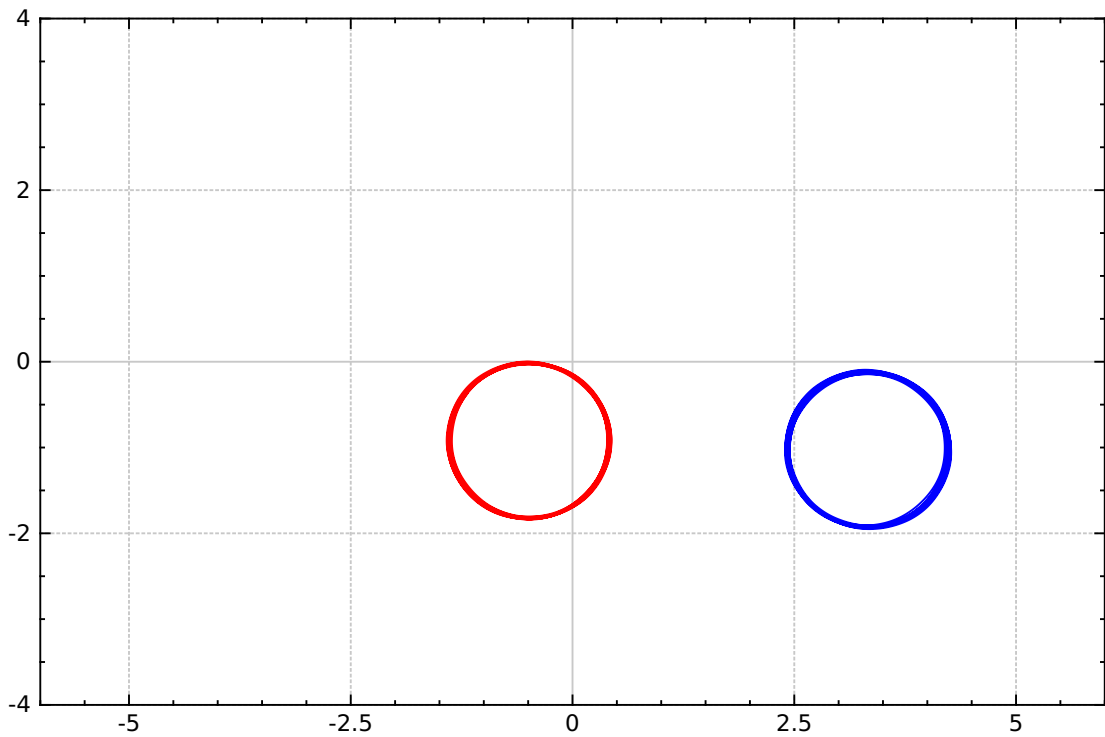
```
private slots:
    void on_pushButtonUstaw_clicked();
    void on_pushButtonReset_clicked();
```

Odpowiadają one za ustawienie bądź zresetowanie pozycji robota. Przyciśnięcie któregoś z przycisków skutkuje wywołaniem usługi `/gazebo/set_model_state`.

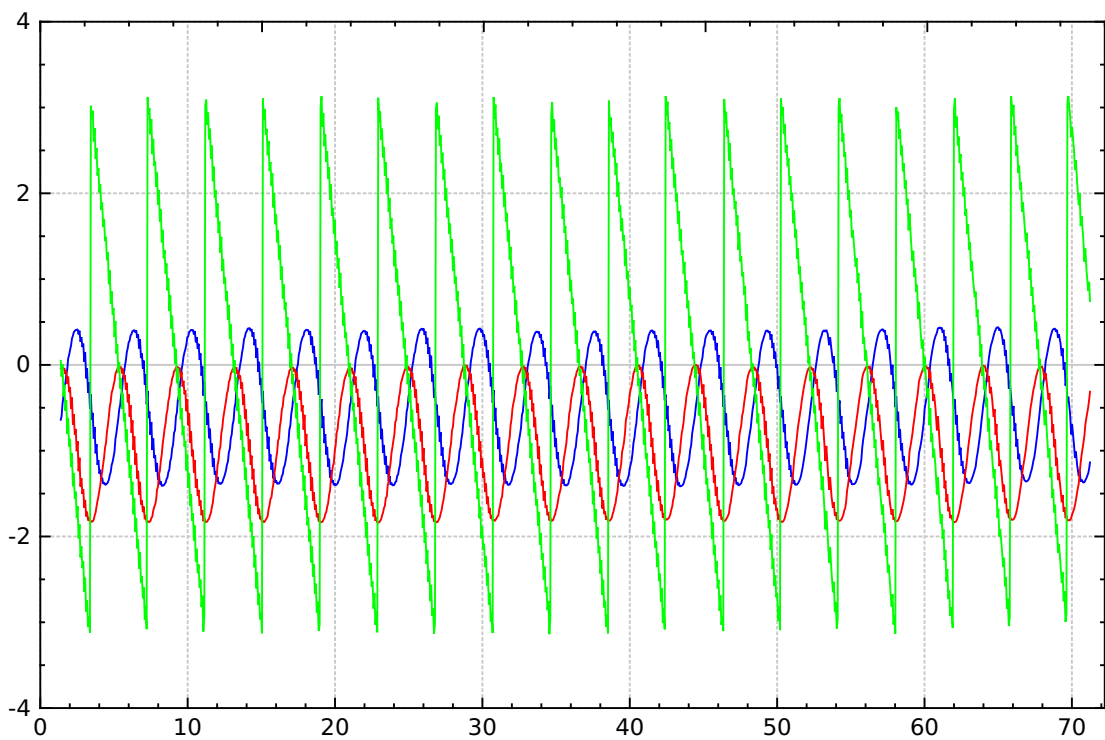
3.1.3. Wyniki symulacji

Funkcjonalność

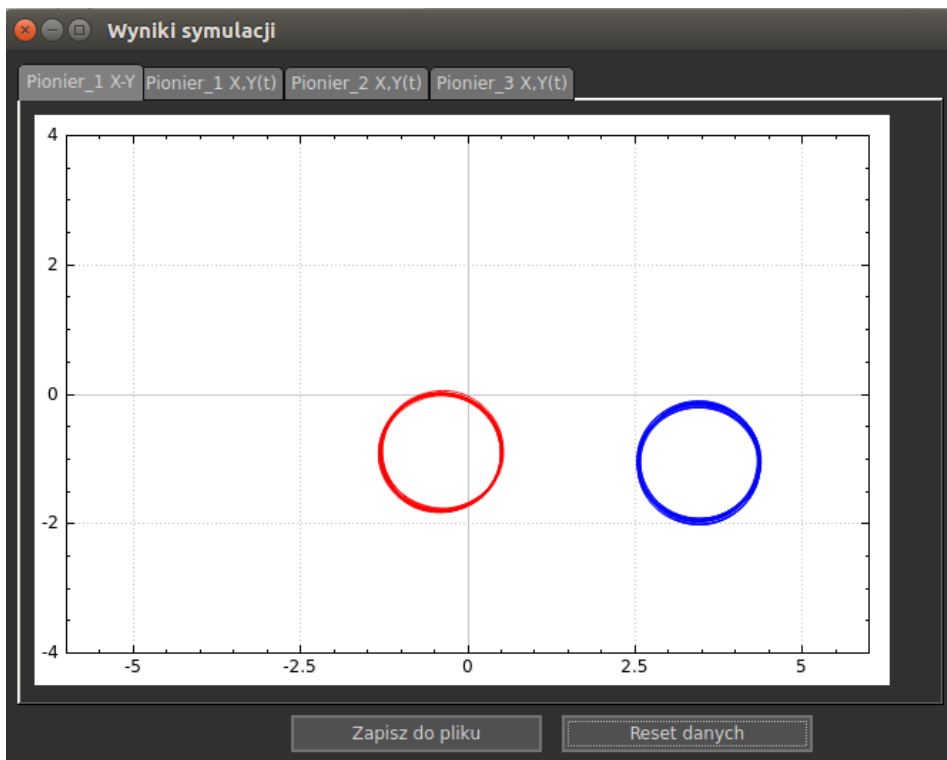
Okno wykresów oferuje w pierwszej zakładce podgląd na pozycję X-Y wszystkich robotów dostępnych na scenie.



Na kolejnych wykresach ilustrowana jest pozycja robota w osiach X,Y oraz orientację względem czasu symulacji.



Okno zawiera dwa przyciski pozwalające na zresetowanie danych o robotach (czas symulacji nie zostaje zresetowany). Oraz przycisk pozwalający na zapis danych.



Dane znajdują się w katalogu głównym kontenera. Wykresy są zapisywane w postaci plików pdf. Program generuje również plik csv z danymi uzyskanymi podczas symulacji.

```
#Pionier1 X,Pionier1 Y,Pionier1 W,Pionier2 X,Pionier2 Y,Pionier2 W,Pionier3 X,Pionier3 Y,Pionier3 W,time
```

```
2.4122,-1.15,1.6118,-0.63934,-0.040688,0.047668,0,1,1,1.389
```

```
2.4082,-1.1215,1.5783,-0.61012,-0.036784,0.01829,0,0,0,1.409
```

```
2.4072,-1.1129,1.5685,-0.60134,-0.035781,0.0094779,0,0,0,1.415
```

Implementacja

Do stworzenia wykresów wykorzystano biblioteki QT i QCustomPlot. Klasa ResultWindow odpowiedzialnej za rejestrowanie, wyświetlanie i zapisywanie informacji o robotach. W klasie z wykorzystaniem funkcji SubscriberPtr dostępnej w bibliotece gazebo/transport/transport.hh pobierane są z topic-ów informacje o pozycji robotów.

3.2. Pluginy

Plugin, czyli wtyczka, jest częścią kodu, który został skompilowany jako współdzielona biblioteka i dodany do symulatora. Ma on dostęp do wszystkich funkcjonalności Gazebo poprzez gotowe klasy języka C++. Pluginy są bardzo przydatne nie tylko ze względu na możliwość kontroli dowolnych modułów symulatora, ale również ze względu na swoją elastyczność. Z łatwością można je dodawać i usuwać z systemu. W Gazebo jest dostępnych 6 typów wtyczek:

- World
- Model
- Sensor
- System
- Visual
- GUI

Oprogramowanie utworzone w ramach projektu wykorzystuje dwie z nich: GUI i World.

3.2.1. GUI

Plugin GUI jest bezpośrednio związany z graficznym interfejsem użytkownika. Z jego poziomu można w prosty sposób dodawać elementy takie jak przyciski, okna, listy czy pola tekstowe. Wtyczka dodawana jest do pliku konfiguracji świata w następujący sposób:

```

1 ...
  <gui fullscreen='0'>
3 <plugin name='SimulationGUI' filename='lib_simulation_gui_plugin.so' />
  ...

```

Z wykorzystaniem pluginu GUI i bibliotek Qt utworzona została w górnej części okna symulacji belka z trzema przyciskami otwierającymi opisane wcześniej okna konfiguracyjne, zwiększające funkcjonalność symulatora.

Wtyczka interfejsu ma ograniczony dostęp do zasobów Gazebo. Chcąc z jej pomocą dokonywać modyfikacji na scenie konieczne jest wysyłanie odpowiednich komend do pluginu World. W projekcie wykorzystano w tym celu komunikację za pomocą topiców symulatora. Wtyczka GUI pełni rolę nadawcy (publisher), natomiast wtyczka World jest subskrybentem (subscriber). Wiadomości mogą mieć dowolny, wcześniej zdefiniowany format. Przykładowo, jeżeli wysyłane są informacje o pozycji robota, format wiadomości zawiera nazwę modelu i sześć zmiennych liczbowych przechowujących pozycję i orientację.

Tworzenie topiców i konfiguracja komunikacji realizowana jest z poziomu kodu, z wykorzystaniem bibliotek Gazebo.

3.2.2. World

World plugin pozwala na edycję różnych parametrów świata, przykładowo silnika fizyki lub oświetlenia. Ponadto umożliwia modyfikowanie sceny i znajdujących się na niej obiektów. Wtyczka jest bezpośrednio związana z plikiem ustawień z rozszerzeniem .world, ładowanym podczas uruchamiania Gazebo. Ten plik jest również odpowiedzialny za uruchomienie wybranej wtyczki:

```

<sdf version='1.6'>
2 <world name='default'>
  <plugin name='projekt_przejsciowy' filename='libprojekt_przejsciowy.so' />
4 ...

```

Budowa pluginu world w pliku źródłowym jest dość prosta i intuicyjna. Minimalny kod pozwalający skompilować bibliotekę ma następującą postać:

```

#include <gazebo/gazebo.hh>
2
namespace gazebo
4 {
  class WorldPluginTutorial : public WorldPlugin
6 {
  public: WorldPluginTutorial() : WorldPlugin()
8 {
    printf("Witaj!\n");
10 }

  public: void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf)
12 {
14 }
};
16 GZ_REGISTER_WORLD_PLUGIN(WorldPluginTutorial)

```

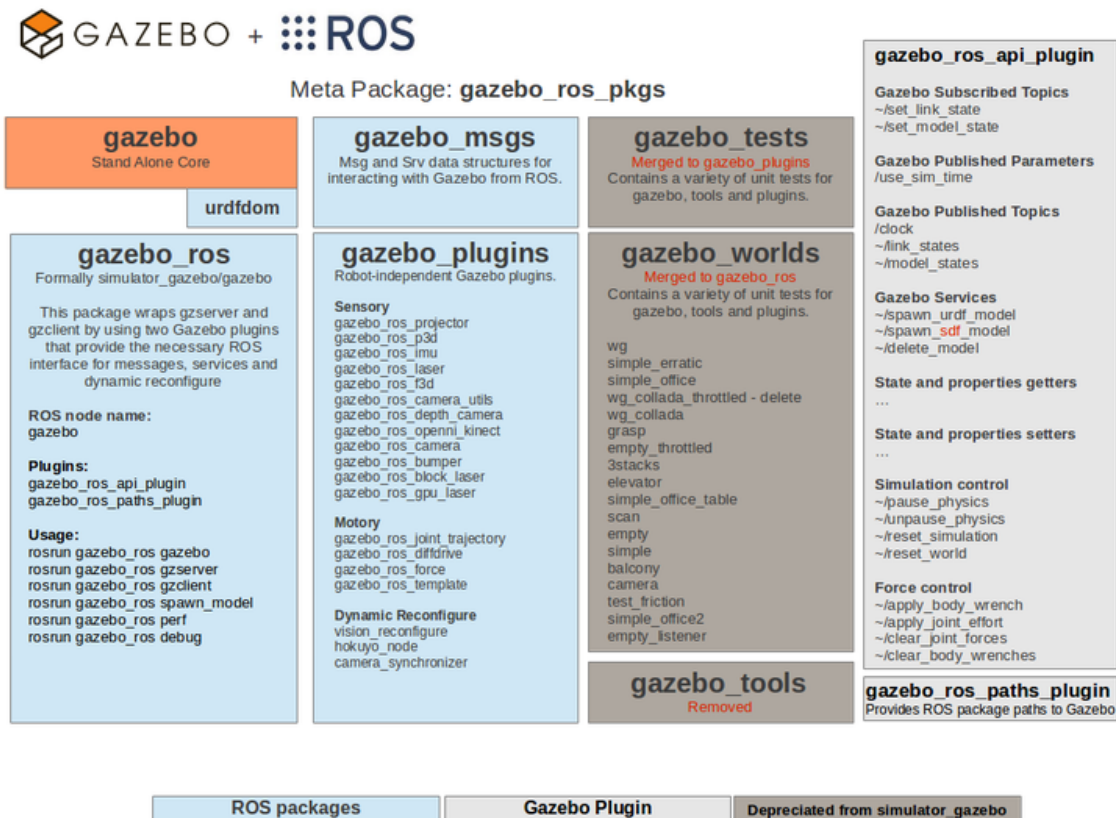
}

Na podstawie powyższego kodu, przy odpowiedniej konfiguracji plików makefile można zbudować plik z rozszerzeniem `.so`, który jako biblioteka jest dodawany do programu.

3.3. Integracja Gazebo z ROS

Głęboka integracja Gazebo z ROS jest największą zaletą tego symulatora. Umożliwia ją zestaw dostarczanych pluginów do ROS w pakiecie `gazebo_ros_pkgs`. Podstawowe funkcjonalności dostarczane razem z pakietem:

- pomimo integracji Gazebo pozostaje nadal samodzielnym systemem,
- możliwość zbudowania pakietów Gazebo w catkin,
- zmniejszenie ilości kodu potrzebnego do symulacji,
- udostępnia użytkownikowi szereg usług i tematów ROS do zarządzania symulacją (wymienione na Rysunku 3.3)



Rysunek 3.3. Schemat prezentujący funkcje udostępniane przez `gazebo_ros_pkgs`

3.3.1. Kompilacja pakietów Gazebo przy użyciu catkin

Jak zaznaczono wcześniej, możliwe jest bezpośrednie użycie systemu catkin do budowania pakietów napisanych dla Gazebo. Wymagało to stworzenia catkin workspace –

katalogu, w którym budowane są pakiety catkin. Stworzone przez nas pluginy Gazebo zostały zebrane i tak skonfigurowane, że tworzą pakiet kompatybilny z ROS.

Po uruchomieniu, Gazebo tworzy własny węzeł do komunikacji z ROS. Dzięki kompilacji poprzez catkin, pluginy Gazebo mają dostęp do ROSa, bezpośrednio w kodzie możemy odwoływać do jego funkcji. Aby się o tym upewnić, w World plugin umieszczono poniższy kod, który zatrzymuje działanie Gazebo w przypadku braku komunikacji z ROS.

```

1 ...
  // Make sure the ROS node for Gazebo has already been initialized
3 if ( !ros::isInitialized() )
  {
5   ROS_FATAL_STREAM("A ROS node for Gazebo has not been initialized");
     return;
7 }
  ...

```

3.3.2. Uruchamianie Gazebo przy użyciu narzędzi z ROS

Dzięki użyciu catkin'a możemy uruchamiać Gazebo za pomocą narzędzia roslaunch, które pozwala na automatyczne wywoływanie węzłów ROS oraz wstępną konfigurację. Ułatwia to start skonfigurowanego do pracy środowiska, co sprowadza się do jednego polecenia:

```
$ roslaunch projekt_przejsciowy l15.launch
```

Aby to umożliwić, przygotowano odpowiedni plik l15.launch w podkatalogu stworzonego pakietu Gazebo *launch*. Plik launch może przy uruchomieniu zaczytywać plik world z konfiguracją Gazebo lub umieszczać modele w odpowiednich miejscach.

3.3.3. Udostępnione funkcjonalności do komunikacji ROS

Jak widać na Rysunku 3.3, Gazebo udostępnia szereg funkcjonalności których można użyć do integracji z ROS.

Dla projektu największe znaczenie miały pluginy wysyłające dane o czujnikach na robocie, takich jak IMU, skaner laserowy czy kamera oraz umożliwiające sterowanie napędem robota. Zostały one zintegrowane z modelem Pioneera.

Węzeł tworzony przez Gazebo publikuje informacje na zewnątrz za pomocą tematów, np. `/model_states`, wysyłający informacje o stanie modeli aktualnie używanych w symulacji. Węzeł dostarcza również usługi ROS'a, jak `/spawn_urdf_model` do ładowania modeli robotów w formacie URDF, czy `/delete_model` do usuwania modeli. Mogą zostać one wykorzystane przez użytkownika naszego systemu do kontroli symulacji i zbierania danych symulacyjnych.

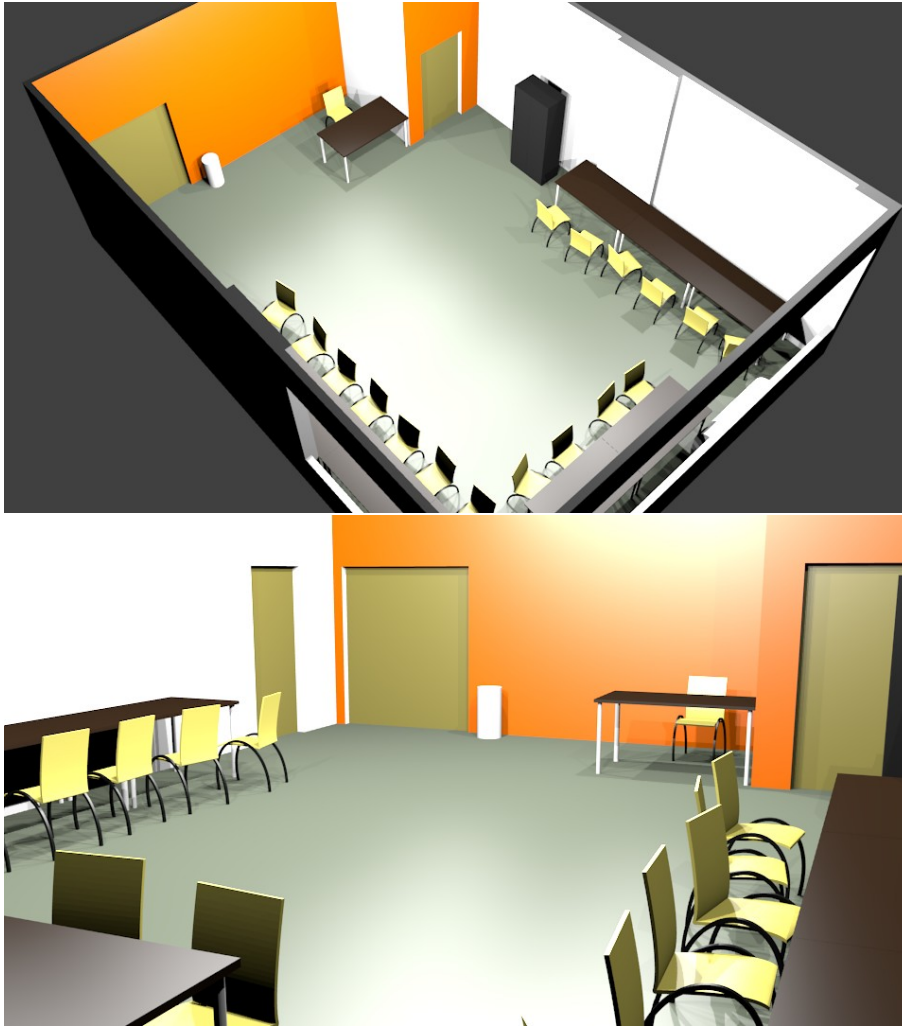
3.4. Modele

Modele użyte do stworzenia laboratorium L1.5 zostały wykonane w programach Inventor Professional oraz Blender.

Elementy wyposażenia wnętrza takie jak stoły, krzesła, grzejniki, szafa oraz śmietnik zaprojektowane zostały w Inventorze, natomiast sama sala (ściany) w programie Blender. Wszystkie stworzone elementy opracowane zostały w oparciu o pomiary rzeczywistych mebli, przedmiotów i ścian w sali L1.5 w budynku C-16. Następnie wszystkie elementy

zostały przekonwertowane przy użyciu programu Blender do formatu Collada (.dae), aby możliwe było wykorzystanie ich w Gazebo.

Modele zostały również poskładane w Blenderze w celu wyrenderowania widoków z różnych miejsc sali:



Aby stworzone modele mogły być użyte w Gazebo, konieczne było jeszcze napisanie plików konfiguracyjnych "model.config" oraz "model.sdf".

Plik "model.config" zawiera takie informacje jak:

- nazwa obiektu
- lista wersji pliku .sdf
- informacje o autorze

W pliku "model.sdf" przechowywane są informacje na temat:

- właściwości fizycznych obiektu (masa, momenty bezwładności)
- statyczności obiektu (możliwości poruszenia przez inny obiekt)
- geometrii obiektu wykorzystywanej przy kolizji z innymi obiektami
- wizualizacji obiektu w symulacji

W przypadku modeli złożonych, takich jak np. roboty, plik "model.sdf" może zawierać również:

- informacje o właściwościach fizycznych poszczególnych członów obiektu
- parametry połączeń kinematycznych między członami
- pluginy wykorzystywane w modelu
- dane o czujnikach i ich umieszczeniu

Do stworzenia modelu robota Pioneer wykorzystano pliki zawarte w Gazebo, jednak aby można było go użyć w symulacjach konieczne było zaimplementowanie jego funkcjonalności w pliku .sdf. W tym celu dodano pluginy do obsługi napędów oraz sensorów, które zostały dodane do modelu.

Dzięki tym zabiegom podczas dodawania robota w symulacji tworzone są tematy pozwalające na komunikację z modelem robota.

4. Testy

W ramach testów systemu przygotowano środowisko zgodnie z instrukcją użytkownika (zawarta w dodatku A). Po uruchomieniu systemu symulacyjnego wykonano czynności sprawdzające działanie podstawowych funkcji oprogramowania, wymaganych do bezproblemowego użytkowania go.

W pierwszej kolejności sprawdzono poprawność wczytywania się modelu laboratorium, oraz umieszczania i chowania robotów ze sceny, test ten wypadł pomyślnie. Kolejnym ważnym aspektem jest to czy w systemie ROS, po dodaniu robota pojawiają się odpowiednie tematy. Dla robota 'pioneer_1' są one następujące:

- /pioneer_1/RosAria/cmd_vel
- /pioneer_1/RosAria/pose
- /pioneer_1/camera/camera_info
- /pioneer_1/camera/image_raw
- /pioneer_1/camera/parameter_descriptions
- /pioneer_1/camera/parameter_updates
- /pioneer_1/scan

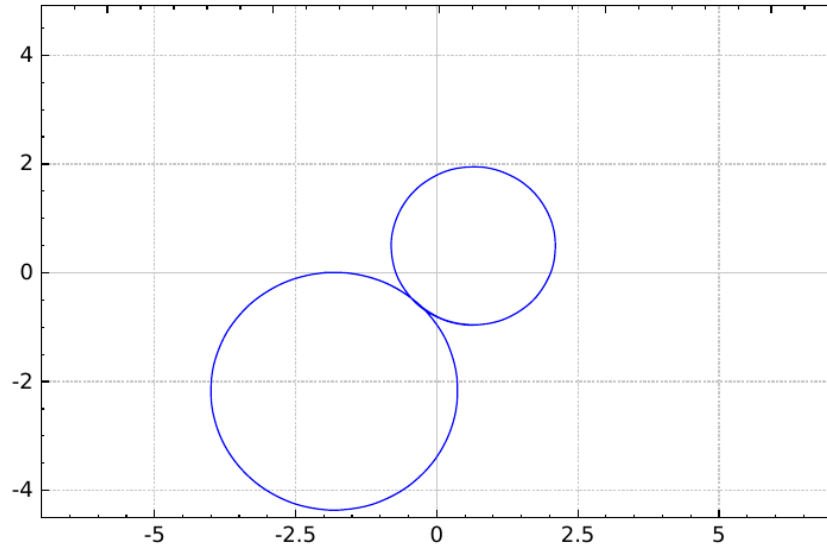


Następnie sprawdzono możliwość zarządzania pozycją robotów z poziomu okienka 'Zarządzanie robotami'. Przeprowadzone testy wykazały, że pozycja robota ustawiana jest w sposób odpowiedni, a błędnie wprowadzone dane są odpowiednio sygnalizowane. Jedynym problemem okazał się brak weryfikacji czy miejsce w którym chcemy postawić robota jest wolne i czy nie wyląduje on np. na innym robocie.

Ostatnią grupą funkcjonalności jest możliwość rejestracji pozycji robota i zapisanie jej do pliku. W celu jej weryfikacji do systemu ROS opublikowano dwie wiadomości, które w efekcie powinny umożliwić narysowanie robotem ósemki:

- rostopic pub /pioneer_1/RosAria/cmd_vel geometry_msgs/Twist - - '[0.5, 0.0, 0.0]'
- rostopic pub /pioneer_1/RosAria/cmd_vel geometry_msgs/Twist - - '[0.0, 0.0, 0.2]'
- rostopic pub /pioneer_1/RosAria/cmd_vel geometry_msgs/Twist - - '[0.5, 0.0, 0.0]'
- rostopic pub /pioneer_1/RosAria/cmd_vel geometry_msgs/Twist - - '[0.0, 0.0, -0.3]'

Po wykonaniu powyższych poleceń robot pokonał zamierzoną trasę, a otrzymane wykresy ruchu zgadzały się z rzeczywistością.



5. Podsumowanie i wnioski

Realizacji tego projektu miała dwa główne cele. Pierwszym z nich było niejako opracowanie systemu pozwalającego na wstępne przeprowadzenie ćwiczeń laboratoryjnych jeszcze przed przystąpieniem do zajęć. Drugim aspektem realizowanego zadania było pokazanie studentom, uczestniczącym w tymże kursie, jak w praktyce wygląda praca nad produktem w stosunkowo dużych grupach projektowych oraz nauczenie ich korzystania z narzędzi do zarządzania pracą, planowania zadań, kontrolowania poczynionych postępów czy zapewniających komunikację wewnątrz tychże grup.

5.1. Serwer

Początkowo równocześnie z wersją aplikacji wykorzystującą Dockera rozwijana była wersja aplikacji działająca na serwerze. Według założeń Gazebo, Ros oraz Ubuntu było zainstalowane na jednym komputerze a klienci łączyli by się z nim za pomocą klienta X11. Podejście to zapewniało zarówno kompatybilność z systemem klienta działającym pod Unixem przy wykorzystaniu systemowego terminalu jak i systemem Windows przy wykorzystaniu np Putty i Xminga. Jednakże napotkane problemy sprawiły, iż zaprzestano dalszego rozwoju aplikacji w wersji serwerowej.

Problemy które wykluczyły taką wersję aplikacji to:

- Gazebo źle współpracuje ze środowiskiem graficznym X11. Zarówno na kartach graficznych AMD jak i NVIDIA występował problem z przesłaniem obrazu. Żeby aplikacja u klienta działała poprawnie trzeba było wyłączyć sprzętową akcelerację GPU co przy środowisku 3d Gazebo znacznie wpływało na wydajność.
- Do komfortowej pracy z systemem potrzeba co najmniej 15 klatek na sekundę. Taka ilość obrazu do przesłania generowała bardzo dużą ilość danych, która musiała przejść przez sieć. Dla zapewnienia 15 klatek na sekundę dla pojedynczego klienta serwer wysyłał ok 300Mb danych. Taką ilość musiał być też w stanie odebrać komputer klienta do ograniczało użycie do gigabitowego ethernetu w sieci lokalnej i nie więcej niż trzech klientów na jeden serwer. Łączność przez internet wymagała by od serwera uploadu rzędu gigabita na łączu używanym tylko do tego celu, aby być w stanie obsłużyć trzech klientów. Korzystając z sieci wifi Politechniki Wrocławskiej nie udało się osiągnąć więcej niż 0.5klatki na sekundę.

5.2. Zastosowanie kontenerów

Branymi pod uwagę opcjami udostępnienia tworzonej aplikacji i jej składowych było spakowanie całości do obrazu maszyny wirtualnej bądź kontenera. Jako lżejszą i bardziej przystępną wybraliśmy opcję numer 2. Pozwoliło to w znaczący sposób ograniczyć wielkość wynikowej paczki a dzięki bardzo szybkiemu rozwojowi tejże technologii w ostatnich latach cieszy się ona ogromnym zainteresowaniem i dobrym wsparciem technicznym. Bogata dokumentacja i zasoby tutoriali pozwoliły na szybkie wdrożenie owych mechanizmów

do naszego projektu, dodatkowo bardzo pomocnym było internetowe repozytorium poświęcone na przechowywanie właśnie takich kontenerów, dzięki czemu każda modyfikacja sprowadzała się jedynie do zaaktualizowania obrazu na serwerze i pobrania przez wszystkich pracujących z danym kontenerem nowych plików i zasobów zamiast każdorazowego pobierania całej paczki z zewnętrznego, często wolniejszego serwera. Takie rozwiązanie pociąga za sobą wielką wygodę dla osób korzystających z naszego projektu - udostępniona przez nas paczka zawiera kompletne środowisko, zainstalowane wszystkie niezbędne pakiety i aplikacje, które próbując uruchomić nasze dzieło na czystym systemie należałoby pobrać i zainstalować. Cały obraz kontenera zajmuje zaledwie 2.4Gb, z czego znaczącą większość stanowi środowisko ROS, Gazebo oraz pakiety wymagane dla ich wzajemnej komunikacji. Paczka ta ostatecznie została odchudzona o wszelkie zbędne aplikacje w celu zaoszczędzenia miejsca, w zamian czego opracowano wiele rozwiązań pozwalających na przykład na dzielenie folderów z systemem hostem, co pozwala na edytowaniu kodu źródłowego z jego poziomu oszczędzając tym samym zasoby kontenera. Niestety jak się okazało w trakcie realizacji projektu mimo, iż zastosowanie kontenerów miało pozwolić na przenoszenie projektu między platformami, lecz niestety możliwości te zostały zweryfikowane w trakcie realizacji projektu, a badaniu poddano środowisko MS Windows, które niestety - mimo wsparcia ze strony technologii kontenerów nie podołało zadaniu - przy pracy z interfejsem graficznym doznawano znaczne opóźnienia jeżeli chodzi o wizualizację aplikacji poprzez środowisko X-Window. Lepszym rozwiązaniem, jeżeli chodzi o wydajność, okazało się postawienie maszyny wirtualnej z systemem Linux, na którym następnie uruchomiono dockera.

5.3. Zawartość projektu

W skład naszego projektu wchodzi kilka składowych. Mamy tutaj na przykład wiernie odwzorowujące rzeczywiste laboratorium robotyczne L1.5 model sali wraz ze wszystkimi obiektami w niej się znajdującymi, a dzięki łatwości w tworzeniu i dodawaniu kolejnych modeli użytkownik nie będzie miał problemu z tworzeniem własnych rozwiązań, czy dodawaniem obiektów o przeróżnych kształtach celem testowania algorytmów sterowania robotami i wykorzystania ich czujników. Odwzorowanie rzeczywistego laboratorium ma pozwolić użytkownikowi na implementację i testowanie algorytmów sterowania w warunkach bezpiecznych korzystając z wirtualnych robotów jeszcze zanim jego program znajdzie się na docelowej maszynie.

W ramach projektu powstały również pluginy do środowiska Gazebo - pozwalają one na proste zmienianie aktualnie wybranego modelu sali czy zarządzanie robotami, których zachowanie chcemy symulować. Z ich poziomu możemy również obserwować przebiegi dla używanych robotów, jednocześnie mając możliwość ich zapisu do plików tekstowych czy jako pliki graficzne w formie wykresów, co może być niezwykle użyteczne w pisaniu przeróżnego typu raportów, sprawozdań bądź na potrzeby zwykłej analizy uzyskanych danych.

Korzystając ze środowiska Gazebo otrzymaliśmy możliwość jego łatwej integracji z systemem ROS, który to funkcjonuje również na rzeczywistych robotach modelowanych w naszych symulacjach. Pozwoliło to na implementację kodu i rozwiązań, które w sposób bezpośredni można przenieść z symulacji do rzeczywistych obiektów bez większych modyfikacji.

5.4. **Możliwości rozwoju**

Projekt został wyposażony w specjalnie skonstruowane pliki dockerfile, dzięki którym w łatwy sposób można zaktualizować obraz do najnowszej wersji pluginów oraz modeli, a także dokonać ewentualnych ulepszeń w przypadku ukazania się nowszych wersji zastosowanych środowisk - jak na przykład aktualizacja środowiska Gazebo do wersji 7.5. Upraszcza to znacząco również procedurę eksportu naszej aplikacji do innej wersji Systemu linux, środowiska ROS bądź Gazebo jeżeli tylko zajdzie taka potrzeba.

A. Instrukcja dla użytkownika

A.1. Wstęp

W niniejszej instrukcji znajduje się opis czynności potrzebnych do uruchomienia i użytkowania systemu wirtualnego laboratorium.

A.2. Przygotowanie systemu do uruchomienia

Do uruchomienia systemu wymagane jest zainstalowanie lokalnie Dockera. W przypadku braku tego pakietu na komputerze należy postąpić zgodnie z instrukcją zamieszczoną na stronie Dockera:

<https://docs.docker.com/engine/installation/linux/ubuntu/>

Aby nie musieć za każdym razem używać *sudo* w poleceniach dotyczących Dockera należy po zakończeniu instalacji wykonać polecenie:

```
sudo usermod -aG docker [nazwa twojego użytkownika]
```

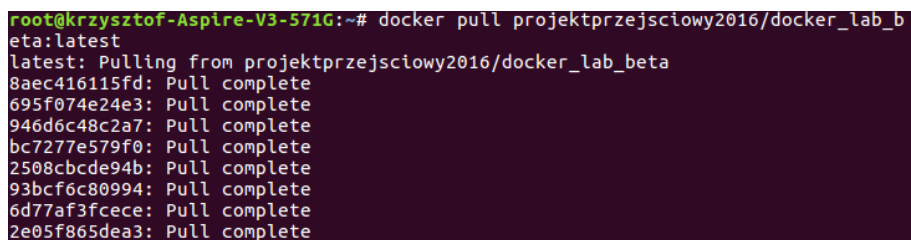
i potwierdzić hasłem użytkownika.

Następnie należy pobrać na dysk lokalny obraz systemu znajdujący się w repozytorium DockerHub. Należy pobrać najnowszy obraz wersji beta o nazwie :

```
projektprzejsciowy2016$/docker_lab_beta:latest
```

Możliwe jest to za pomocą polecenia:

```
docker pull projektprzejsciowy2016/docker_lab_beta:latest
```



```
root@krzysztof-Aspire-V3-571G:~# docker pull projektprzejsciowy2016/docker_lab_beta:latest
latest: Pulling from projektprzejsciowy2016/docker_lab_beta
8aec416115fd: Pull complete
695f074e24e3: Pull complete
946d6c48c2a7: Pull complete
bc7277e579f0: Pull complete
2508cbcde94b: Pull complete
93bcf6c80994: Pull complete
6d77af3fcede: Pull complete
2e05f865dea3: Pull complete
```

Przed uruchomieniem obrazu należy zezwolić wszystkim aplikacjom działającym z poziomu Dockera na uruchamianie się w trybie graficznym:

```
xhost +
```

Powinien pojawić się następujący komunikat: "access control disabled, clients can connect from any host"

Uruchomienie obrazu systemu odbywa się poprzez wpisanie w konsoli polecenia (powoduje stworzenie nowego kontenera, na podstawie wywołanego i otwiera go - wszelkie zmiany będą zapisywane w nowym kontenerze) :

```
docker run -it -e DISPLAY=unix$DISPLAY -v=/tmp/.X11-unix:/tmp/.X11-unix:rw \
projektprzejsciowy2016/docker_lab_beta:latest
```

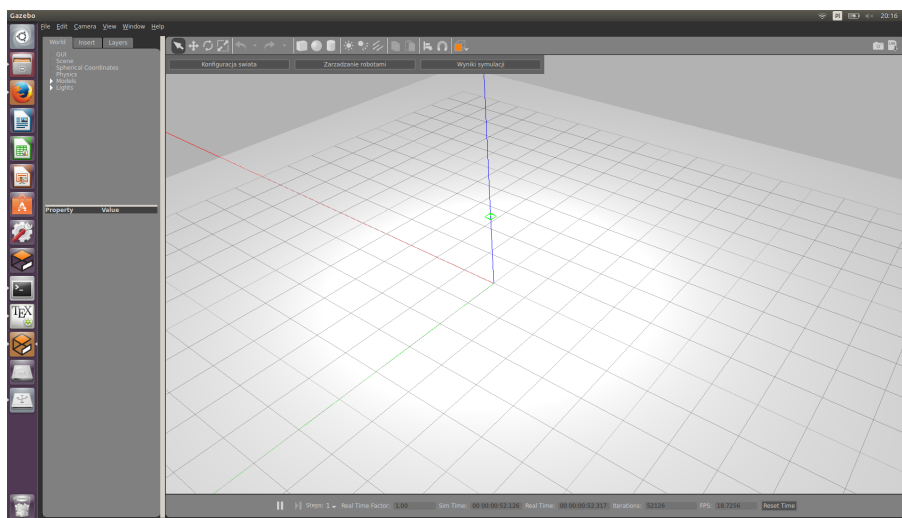
W przypadku gdy potrzebujemy współdzielić pewne katalogi lokalne z Dockerem należy w poleceniu uruchamiającym obraz systemu "docker run" skorzystać z opcji -v:

```
docker run -it -e DISPLAY=unix$DISPLAY -v=/tmp/.X11-unix:/tmp/.X11-unix:rw \
-v[ścieżka_do_katalogu_lokalnego]:[ścieżka_gdzi_ma_być_montowany_w_obrazie] \
projektprzejsciowy2016/docker_lab_beta:latest
```

System ROS wraz z symulatorem - Gazebo, uruchamiany jest poleceniem wykonywanym już z poziomu obrazu Dockera:

```
roslaunch projekt_przejsciowy 115.launch &
```

Po chwili powinien uruchomić się symulator Gazebo:



Aby przetestować czy wszystko działa poprawnie należy dodać na scenę robota jak w punkcie 3.2.

Jeśli potrzebujemy uruchomić drugą konsolę działającą na tym samym kontenerze to należy uruchomić drugą konsolę (można to wykonać skrótem klawiszowym "shift" + "ctrl" + "t", a następnie sprawdzamy nazwę kontenera, który mamy już uruchomiony:

```
docker ps -a
```

Powyższe polecenia wyświetli listę wszystkich dostępnych lokalnie kontenerów, w której można sprawdzić nazwy poszczególnych kontenerów:

```
micha@micha-X550JK:~$ docker ps -a
```

CONTAINER ID	IMAGE	STATUS	COMMAND
d68060f00592	projektprzejsciowy2016/docker_lab_beta	Exited (0) 6 seconds ago	"/ros_entryp oint.sh b"
41c7481247b4	projektprzejsciowy2016/docker_lab_beta	Up 7 minutes	"/ros_entryp oint.sh b"

```

names:
  goofy_varahamihira
  stupefied tesla
```

Na przedstawionej na rysunku liście przykładowe nazwy to goofy_varahamihira oraz stupefied_tesla. Podpięcie się konsolą pod otwarty już kontener opiera się o polecenie:

```
docker exec -it Nazwa_Kontenera bash
```

Uruchamiając kontener po raz kolejny mamy możliwość zachowania historii wykorzystywanych poleceń. Przy uruchamianiu kontenera odwołujemy się do ostatniego otwartego dockera:

```
docker start -i $(docker ps -q -l)
```

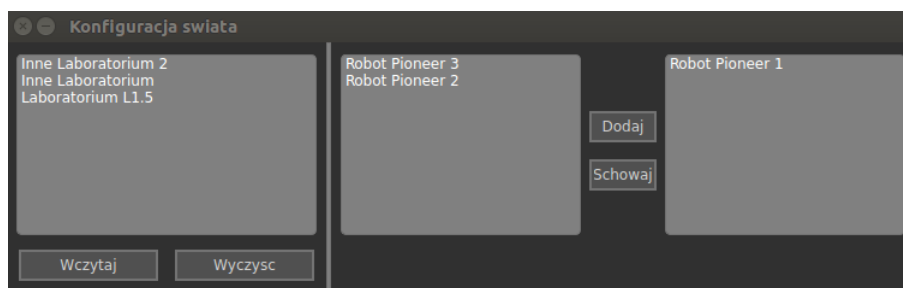
A.3. Obsługa systemu

A.3.1. Dodawanie modelu sali

Na obrazie systemu, w chwili obecnej, znajdują się trzy przykładowe modele świata:

- model sali L1.5 w budynku C-16
- dwa dodatkowe modele składające się z ułożonych w pewien sposób mebli

Dodanie jednego z tych światów możliwe jest z poziomu okna "Konfiguracja Świata". Dostęp do wspomnianego okna jest z poziomu widoku głównego Gazebo pod przyciskiem umieszczonym z lewej strony listwy na górze ekranu.



W oknie tym, po lewej stronie, znajduje się lista dostępnych światów oraz dwa przyciski "Wczytaj" i "Wyczyść". Wybraną salę można dodać wybierając ją z lisy i klikając pierwszy przycisk. Jeśli wczytamy inną salę w momencie, gdy jakaś jest już wczytana, scena zostanie automatycznie wyczyszczona i wczytany aktualnie wybrany świat. Przycisk "Wyczyść" służy do usunięcia wszystkich obiektów ze sceny z wyjątkiem robotów.

A.3.2. Dodawanie modeli robotów

Z okna konfiguracji świata można również dodawać na scenę roboty (jak narazie dostępne są trzy roboty Pioneer). Służą do tego listy na środku oraz z prawej strony okienka. Lista środkowa zawiera roboty, które są dostępne do dodania, a prawa te, które są już dodane. Aby dodać robota, należy wybrać go z listy środkowej i kliknąć przycisk "Dodaj". Roboty dodane pojawią się na scenie w momencie kliknięcia przycisku "Zatwierdź". W tym samym momencie pojawiają się w systemie ROS topiki związane z dodawanymi robotami. Listę topików można zobaczyć wpisując w terminalu:

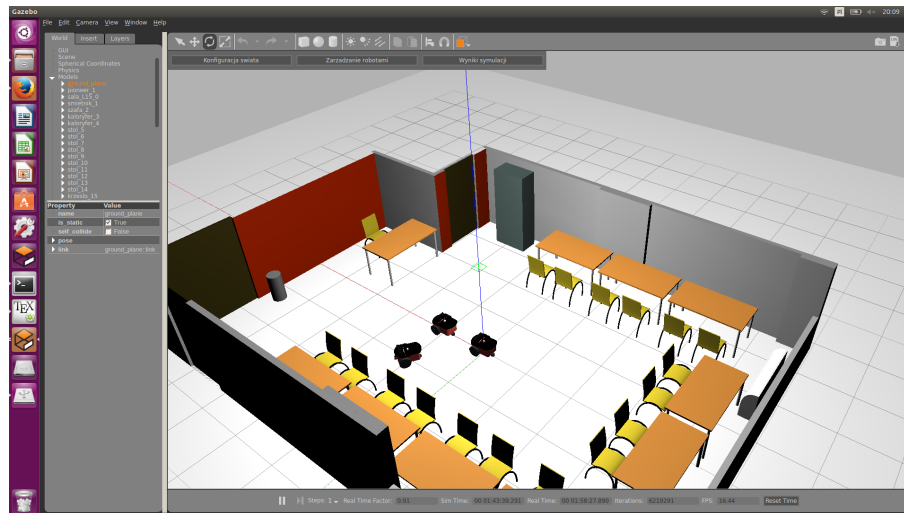
```
rostopic list
```

Przycisk "Schowaj" umieszcza wybranego robota poza salą w tak zwanym schowku.

Aby przetestować czy Gazebo dobrze współpracuje z systemem ROS należy dodać robota a następnie w konsoli wydać polecenie publikujące w topicu ROSa odpowiadającym za prędkość dodanego robota:

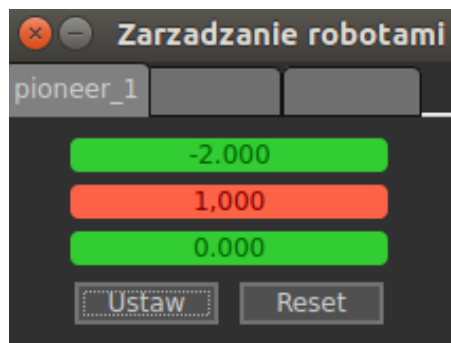
```
rostopic pub /pionner_1/RosAria/cmd_vel geometry_msgs/Twist \
-- '[1.0, 0.0, 0.0]' '[0.0, 0.0, -0.5]
```

W efekcie robot powinien zacząć poruszać się po okręgu.



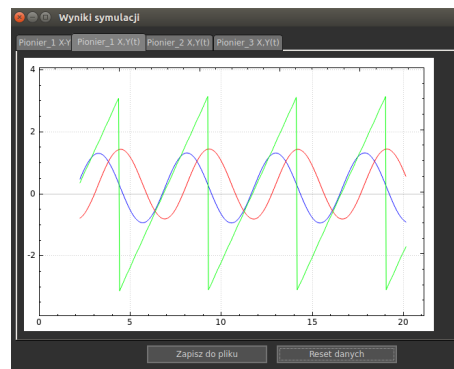
A.3.3. Resetowanie robotów i ustawianie ich w zadanych miejscach

Do resetowania i ustawiania w zadanej pozycji robotów służy okno "Zarządzanie robotami". Okno podzielone jest na zakładki - każda z nich dotyczy osobnego robota. Umieszczony w każdej zakładce przycisk "Reset" ustawia odpowiedniego robota w punkcie (0,0) sali z orientacją równą 0. Pola tekstowe służą do zadawania pozycji (X,Y) i orientacji robota. Wartości z tych pól są wykorzystywane przy używaniu przycisku "Ustaw". W trakcie kliknięcia wartości te są sprawdzane i ich ewentualna niepoprawność jest sygnalizowana czerwonym podświetleniem pola. Dopiero po sprawdzeniu robot przemieszczany jest na ustaloną pozycję.



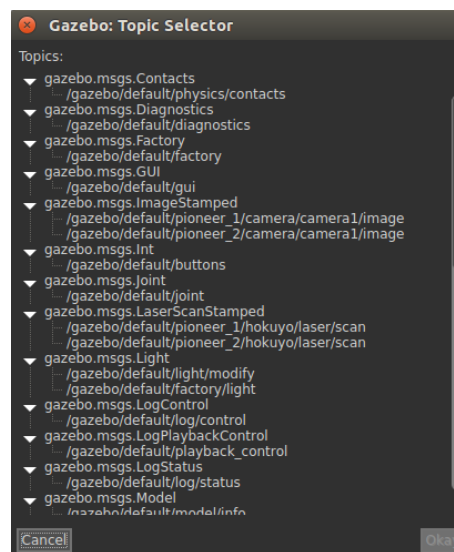
A.3.4. Wyświetlanie i zapisywanie wyników

Do wizualizacji i zapisu danych z eksperymentu przygotowane zostało okno "Wyniki symulacji". Okno podzielone jest na zakładki, w których można otrzymać podgląd trajektorii wszystkich robotów oraz przebiegi położenia X, Y i orientacji. W pierwszej zakładce są trasy przebyte przez wszystkie roboty a każda kolejna zakładka odnosi się do konkretnego robota prezentując jego położenie i orientację. Przycisk "Reset danych" pozwala wyczyścić wykresy i zacząć zbierać je od nowa. Okno daje możliwość zapisania przebiegów do plików, które znajdują się potem w folderze /root/. Ważne jest też aby przed rozpoczęciem zbierania danych do konkretnego wykresu za pomocą przycisku z dolnego paska Gazebo "PLAY/PAUSE" zatrzymać czas symulacyjny następnie przyciskiem "Reset Time" aby wykres był czytelny.

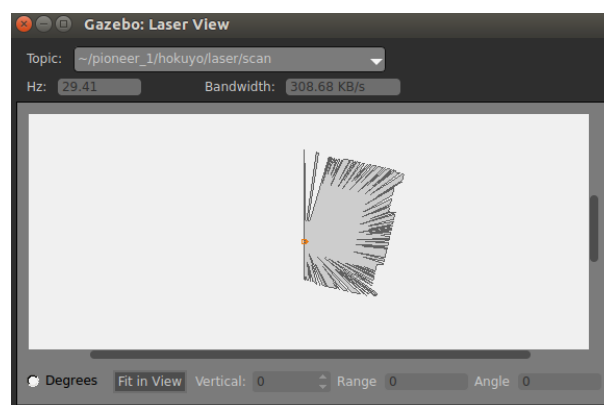


A.3.5. Wyświetlanie odczytów z czujników

Aby uzyskać dostęp do pomiarów z czujników należy z paska narzędzi Gazebo wybrać zakładkę *Window >> Topic Visualization*. Uruchomi się okno:

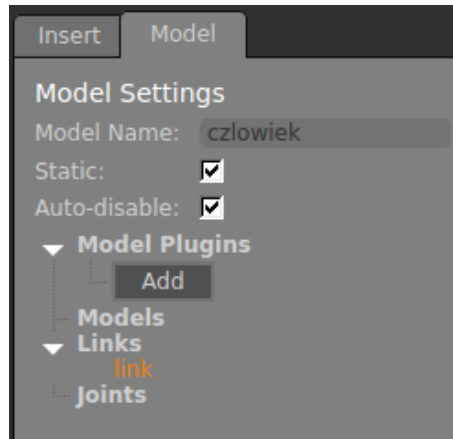


Następnie wybieramy topic czujnika, z którego chcemy wyświetlać dane. Dla skanera laserowego wykres wygląda następująco:



A.3.6. Dodawanie i edycja modeli przeszkód na scenie

Aby dodać przeszkodę na scenę należy z lewej strony ekranu wejść w zakładkę "Insert", wybrać kliknięciem model i kliknąć na miejsce na scenie, w którym chcemy postawić przeszkodę. Proste modele przeszkód dostępne są z paska narzędzi Gazabo. Aby zmienić statyczność przeszkody należy kliknąć na nią prawym przyciskiem myszy (PPM) a następnie z rozwiniętego menu wybrać opcję "Edit Model". Gazebo przejdzie do trybu edycji modelu. Wybieramy zakładkę "Model" i zaznaczamy lub odznaczamy opcję "Static":



B. Instrukcja dla dewelopera

B.1. Wstęp

W niniejszej instrukcji znajduje się opis wstępnych czynności wymaganych do dalszego rozwoju symulatora. Prezentuje konfigurację narzędzi, środowiska oraz obrazu a także automatyzację procesu budowy obrazu dla wersji deweloperskiej oraz testowej/finalnej przy pomocy pliku Dockerfile.

Instrukcja opisuje stan na styczeń 2017 roku, przez co informacje w niej zawarte mogą być nieaktualne. Dotyczy to systemów operacyjnych wspierających Dockera, dostępnej wersji ROS'a oraz Gazebo, a także API, z którego korzystano.

B.2. System operacyjny

Wymagany jest system Linux w architekturze 64 bitowej. Wybrane wspierane dystrybucje **Ubuntu**:

- Yakkety 16.10
- Xenial 16.04 (LTS)
- Trusty 14.04 (LTS)

Debian:

- Jessie 8.0 (LTS)
- Wheezy 7.7 (LTS)

Pełna lista wspieranych dystrybucji dostępna jest pod adresem <https://docs.docker.com/engine/installation/linux/>. Praca z Dockerem możliwa jest również na systemie Windows, ale niniejsza instrukcja nie zawiera opisu dla tego systemu, gdyż nie udało się skonfigurować środowiska do pracy z Dockerem w trybie graficznym.

B.3. Docker

Oprogramowanie należy zainstalować zgodnie z instrukcją przygotowaną dla wybranego systemu operacyjnego umieszczoną na stronie dostawcy (link z podpunktu B.2).

B.4. Obraz

Symulator można rozwijać lub modyfikować, dzięki przygotowanemu obrazowi, który jest dostępny na platformie **DockerHub** pod adresem https://hub.docker.com/r/projektprzejsciowy2016/docker_lab/. Składa się z systemu operacyjnego **Ubuntu Xenial 16.04 LTS**, systemu **ROS Kinetic** oraz **Gazebo 7.5** oraz pozostałych narzędzi do komunikacji ROS-Gazebo. Jeżeli zaprezentowana konfiguracja jest akceptowalna przez dewelopera, należy pobrać obraz. W przeciwnym wypadku należy zapoznać się z rozdziałem B.7, w którym przedstawiono, jak przy pomocy pliku Dockerfile można wygenerować nowy obraz.

B.5. Repozytoria

Do pracy nad symulatorem wymagane są dwa repozytoria: **kod źródłowy pluginu** zapewniającego wszystkie funkcjonalności prezentowanego symulatora oraz **modele** obiektów wymaganych przez program Gazebo.

Kod źródłowy pluginu należy pobrać lokalnie. Dostępny jest pod adresem https://github.com/ProjektPrzejsciowy/projekt_przejsciowy. Repozytorium będzie dostępne w obrazie za pomocą mechanizmu współdzielenia katalogów. Dzięki temu deweloper może pracować z pluginem lokalnie, korzystając z wybranego IDE a wszystkie zmiany są widoczne z poziomu kontenera. Informację o tym, jak współdzielić katalog z obrazem Dockera umieszczono w podpunkcie B.6.

Modele obiektów znajdują się pod adresem <https://github.com/ProjektPrzejsciowy/models>. Obraz zaprezentowany w podpunkcie B.4 zawiera już wymagane modele, zatem nie trzeba ich ponownie dodawać. Jeżeli obraz pozyskano z innego źródła, należy każdy utworzony kontener uzupełnić o modele. Informacja o tym, jak dodać modele do kontenera umieszczono w podpunkcie B.6.

B.6. Uruchamianie obrazu i konfiguracja

Przed uruchomieniem obrazu należy zezwolić na pracę obrazu Dockera z X serwerem poleceniem:

```
xhost +
```

Obraz należy uruchomić przy pomocy komendy:

```
docker run -it -e DISPLAY=unix$DISPLAY -v=/tmp/.X11-unix:/tmp/.X11-unix:rw \
-v [ścieżka do katalogu lokalnego]:/root/catkin_ws/src/projekt_przejsciowy \
[nazwa obrazu]
```

W miejsce [ścieżka do katalogu lokalnego] należy podać ścieżkę do katalogu z pobranym kodem źródłowym pluginu. W przypadku korzystania z przygotowanego obrazu w wersji deweloperskiej w miejsce [nazwa obrazu] należy podstawić:

```
projektprzejsciowy2016/docker_lab:latest
```

W analogiczny sposób przy pomocy opcji -v można współdzielić dowolny katalog z kontenerem Dockera.

Następnie, należy upewnić się, że plik /root/.bashrc/ w obrazie zawiera następujące linie:

```
export GAZEBO_PLUGIN_PATH=/root/catkin_ws/devel/lib/
source ~/catkin_ws/devel/setup.bash
```

Pierwsza linia odpowiada za dołączenie pluginu do Gazebo. Jeżeli po uruchomieniu Gazebo plugin jest niewidoczny, świadczy to o braku linii we wspomnianym pliku lub nie skompilowaniu plików źródłowych pluginu w katalogu /root/catkin_ws/. Druga linia odpowiada za widoczność przygotowanych pakietów i możliwość uruchamiania ich z dowolnej lokalizacji w terminalu. Jeżeli przygotowane pakiety są niewidoczne w terminalu, należy sprawdzić czy plik /root/.bashrc zawiera tę linię i czy pakiet został zbudowany w katalogu /root/catkin_ws/.

Jeżeli obraz nie zawierał modeli wymaganych przez Gazebo to należy dodać je do katalogu /root/.gazebo/models/ np. pobierając je z repozytorium.

Przed uruchomieniem symulatora należy skompilować plugin poleceniem:

```
cd /root/catkin_ws/  
catkin_make
```

B.7. Dockerfile

Pod adresem <https://github.com/ProjektPrzejsciowy/docker> dostępne są pliki Dockerfile, które umożliwiają wygenerowanie nowego obrazu dla wersji deweloperskiej i testowej/finalnej. Na ich podstawie można wygenerować własne obrazy, które następnie można umieścić na platformie **DockerHub**.