OmniVelma

Radosław Świątkiewicz

12 czerwca 2016

Spis treści

Rozdział 1

Opis problemu

1.1 Cel

Celem tej pracy inżynierskiej jest budowa narzędzi do symulacji robota mobilnego w środowisku wirtualnym. Za zadanie jest stworzyć drivery i sterowniki do zwirtualizowanej podstawy jezdnej, poruszającej się za pomocą czterech kół szwedzkich. Powinna ona być dokładną kopią prawdziwego robota, aby zachowanie modelu w symulacji było jak najbardziej zbliżone do zachowania fizycznej formy. Ten robot będzie używany jako podstawa do przemieszczania innego robota manipulującego Velma.

Należy tak zaprogramować model, aby reagował na siły podobnie do prawdziwej wersji i przyjmował to samo sterowanie z zewnątrz, co prawdziwy obiekt. To spowoduje, że można będzie napisać wspólną logikę aplikacji sterowania i użyć jej zarówno w wirtualnej wersji, jak i fizycznej.

Testowanie niedoskonałego programu na prawdziwym obiekcie może prowadzić do jego uszkodzeń, dlatego wpierw trzeba się upewnić o poprawności rozwiązań na bezpiecznej kopii wirtualnej. Prawdziwy świat nie pozwala także na przeprowadzanie zaawansowanych scenariuszy i dowolnych ustawień środowiska testowego. Szybciej i taniej jest stworzyć wirtualne środowisko, niż fizyczne, w dodatku porażka przy symulacji nie wpływa na zniszczenie robota w rzeczywistości. Dopiero przy sukcesywnej symulacji wirtualnej można zastosować programy i przetestować obiekt w prawdziwym życiu.

Do obsługi są dane także odpowiednie czujniki, za pomocą których program orientuje się w przestrzeni i generuje sterowanie. Symulacja czujników polega na generowaniu danych na podstawie wirtualnej symulacji i dodaniu szumu, oraz celowych błędów dla przybliżenia prawdziwego zachowania czujnika.

1.2 Podstawa jezdna

Jest to duża, prostokątna podstawa dookólna jeżdżąca na czterech kołach szwedzkich. Koła są stałe, parami przyczepione do dwóch osi. Każde koło jest sterowane osobno przez serwomotor, może mieć prędkość i kierunek niezależny od reszty kół i kierunku poruszania się robota. Każdy z serwomotorów ma także wbudowany enkoder do sprawdzania rzeczywistego kąta obrotu.

Koła szwedzkie, zwane także kołami Mecanum, to specjalne koła z dodatkowymi rolkami na obwodzie ustawionymi pod kątem 45° do osi koła. Rolki nie mają silników i obracają się niezależnie od siebie. Ich osie ustawione są tak, że osie rolek dwóch kół z tej samej strony robota są pod kątem prostym i przecinają się pośrodku urządzenia. Innymi słowy, robot ma identycznie ustawione koła na przeciwległych wierzchołkach, i razem ustawione są w kształt litery *X*.

Odpowiedni obrót kół względem siebie pozwala na ruch platformy w dowolnym kierunku niezależnie od kąta obrotu robota. Za ich pomocą da się także obracać całością w czasie jazdy. Jeśli na przykład obracać tylko przeciwległymi kołami po przekątnej, system zacznie się poruszać po skosie. A jeśli do tego dodamy obrót kół drugiej przekątnej w odwrotnym kierunku, wtedy pojazd pojedzie w bok pomimo faktu, że koła nie zmieniły swojej osi i się nie przekręciły.

Podstawa ma za zadanie wozić wydziałowego robota manipulującego Velma. Jest to wysoki i bardzo ciężki robot wyposażony w dwa chwytaki na ramionach o licznych przegubach. Konstrukcja powoduje, że sama podstawa potrzebuje mieć sporą nośność i szerokość, aby zachować środek ciężkości bezpiecznie nisko. Jeżdżąc na tej podstawie robot może się przemieszczać i obracać w dowolnym kierunku, aby uzyskać lepszy dostęp do manipulowanych przedmiotów.

Z jednej strony platformy znajduje się zawias oddzielający główną część od drugiej nieco mniejszej. Części łączy wspólna oś, wygląda to tak, jakby przeciąć podstawę w poprzek i połączyć obie części przegubem na środku.

Ma to za zadanie zapewnić, aby każde koło dociskało do podłoża z taką samą siłą, jak to po drugiej stronie osi. Pozwala to także na pokonywanie drobnych nierówności terenu. Bez tego zawiasu nierówna podłoga spowodowała by, że na przykład tylko trzy koła dotykałyby podłoża w danym momencie uniemożliwiając sprawne sterowanie. Niedeterministyczne tarcie kół jest niewykrywalne w bezpośredni sposób, więc należy je wyeliminować w jeden z tych sposobów.

1.3 Składniki systemu

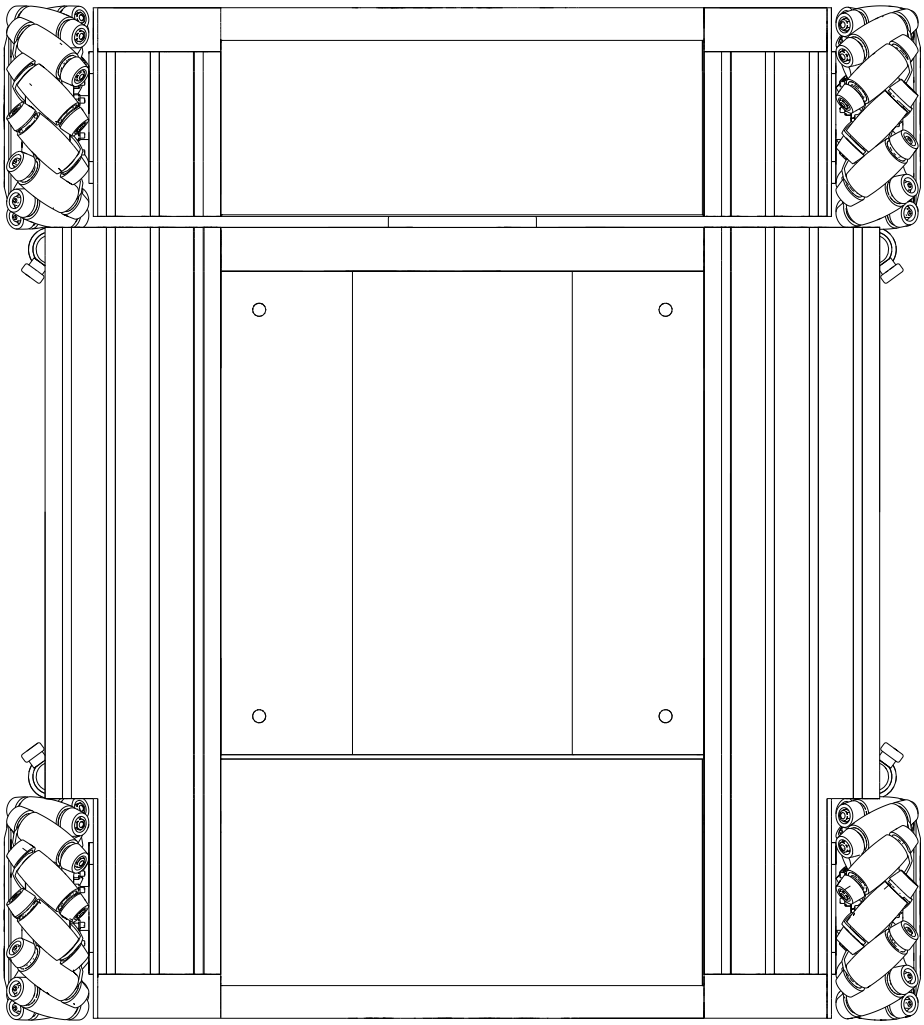
Symulacja składa się z kilku odrębnych składników, które komunikują się ze sobą poprzez specjalne interfejsy oparte o kolejki wiadomości. To pozwala zamieniać i przepisywać elementy zachowując tę samą komunikację między składnikami i nie tracąc kompatybilności.

1.3.1 Model

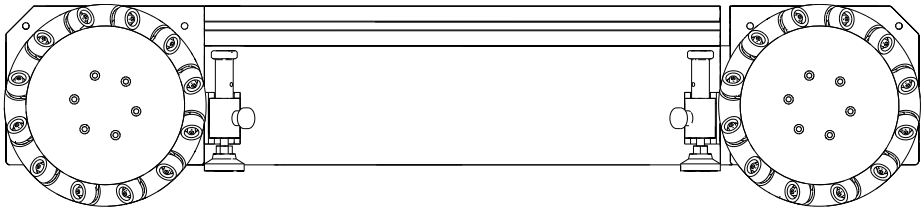
Odpowiednio opisany równaniami powinien być jak najdokładniejszą kopią fizycznego robota. Musi brać pod uwagę masy i momenty bezwładności elementów składowych, a także możliwe tarcia. Model posiada także więzy na ruchome elementy, jak koła i rolki, aby obracały się w określony sposób.

Ta część systemu oddziałuje bezpośrednio z silnikiem fizycznym. To kształt, masy i momenty bezwładności brył są argumentami funkcji liczących. Także silnik manipuluje z powrotem podanymi obiektami nadając im odpowiednie prędkości w czasie.

Do modelu doczepia się wirtualne czujniki generujące odpowiednie dane na podstawie symulacji i rozkładu losowego. Nie są to pełne dane o stanie modelu, jakie posiada silnik, gdyż w rzeczywistości również nigdy nie mamy pełnej informacji o stanie urządzenia.



Rysunek 1.1: Baza jezdna widziana od góry.



Rysunek 1.2: Baza jezdna widziana od prawej strony.

Rysunek 1.3: Baza jezdna widziana od tyłu.

Dla ozdoby można mu nadać wygląd zbliżony do prawdziwego robota poprzez wymodelowanie siatki w programie graficznym i nałożeniu wymaganych materiałów.

1.3.2 Driver silników

Ma taką samą funkcjonalność, jak pokładowy driver niskopoziomowy na robocie. W rzeczywistości najczęściej implementowany w formie mikrokontrolera, lub podobnego systemu czasu rzeczywistego. Zadaniem pokładowego sterownika jest przyjęcie wartości sterowania od zewnętrznego programu i podanie odpowiednich wartości napięcia na silniki kół. Może także wprowadzać niezależne zabezpieczenia przed zniszczeniem urządzenia przez nieprawidłowe wejście. Taki program jest najczęściej dostarczony przez producenta robota i nie znany użytkownikowi. Zależy ściśle od budowy urządzenia i nie może być prosto wymienialny z innym obiektem.

Cel polega na stworzeniu alternatywnego programu pobierającego podobne, co w fizycznej wersji dane i obracającego kołami w odpowiedni sposób. Tutaj polega to na wywoływaniu funkcji silnika fizycznego z odpowiednio przygotowanymi argumentami. Silnik fizyczny symulatora przekłada te siły na odpowiednie prędkości w przestrzeni wirtualnej biorąc pod uwagę charakterystykę modelu i otoczenia.

1.3.3 Driver czujników

Normalny driver na urządzeniu pobiera surowe dane i zamienia je na format możliwy do odczytania przez zewnętrzny program. Najczęściej polega to na próbkowaniu sygnału i wysyłaniu go dalej. Driver może także robić wstępne obrabianie danych w celach usuwania szumów, korekcji błędów, albo szybkiej obróbki na wyższy format, jak ma to miejsce na przykład w kamerze Kinect.

Symulując ten element budujemy program generujący dane na podstawie aktualnego stanu symulacji. W celu przybliżenia go do realizmu powinno także dodawać się do generowanych danych sztuczny szum i błędy, aby łatwiej można było go zamienić później na prawdziwy czujnik.

1.3.4 Program sterujący

Cześć odpowiedzialna za logikę aplikacji. Tutaj podejmowane są decyzje, jakie wymusić prędkości kół, aby pojechać po wymaganej krzywej. Ten program także

pobiera, obrabia i interpretuje przetworzone przez driver dane z czujników.

W tej części używanych jest wiele zaawansowanych algorytmów, jak budowanie mapy, szukanie ścieżki, unikanie kolizji, wyznaczanie obiektów na obrazie itp. Zwykle działa to na dużych, wielowątkowych maszynach ze względu na spore wymagania obliczeniowe. Jeśli robot komunikuje się z użytkownikiem, to ma to miejsce tutaj.

Sterownik składa się z wielu modułów, każdy odpowiedzialny za coś innego. Może być stworzony w językach wysokopoziomowych, a nawet po części skryptowych, gdyż nie ma wymagań czasowych. Zazwyczaj program ma wiele poziomów. Nadrzędne algorytmy decyzyjne wołają niższe funkcje odpowiedzialne np. za planowanie, a te z kolei jeszcze niższe od np. ruchu itp.

1.4 Technologie

Symulator daje użytkownikowi do dyspozycji odpowiedni silnik fizyczny w którym odbywa się symulacja modelu, oraz API do obsługi. Zaawansowany silnik powinien obsługiwać odpowiednie tarcia, więzy, siły, materiały fizyczne i wszystko, co potrzebne do jak najwierniejszego odtworzenia prawdziwego zachowania obiektu. Na rynku jest wiele różnych silników zarówno do symulacji w czasie rzeczywistym, jak i do wyznaczania tras obiektów po długich obliczeniach. Jedne z technologii są otwartoźródłowe, inne mniej.

1.4.1 Gazebo

Ten silnik jest dość prosty w obsłudze i wydaje się mieć mało funkcji ze względu na prosty interfejs. Jednak jego potencjał tkwi w argumentach linii poleceń i w czytaniu podanych plików.

Program symuluje podane obiekty używając jednego z czterech popularnych silników fizycznych: ODE, Bullet, Simbody lub DART. Wszystkie te silniki są wolnym oprogramowaniem i używane także w innych programach, jak na przykład Blender.

Program oprócz symulatora ma wbudowany edytor modeli i budynków w którym możemy tworzyć nasze dzieła od razu w przestrzeni trójwymiarowej. Jakość wykonania tych składników jest bardzo słaba, brak jest tak podstawowych funkcjonalności, jak cofanie ruchu. Również tworząc modele w ten sposób nie mamy nad nimi pełnej kontroli i dokładności.

Zatem najlepszym sposobem jest tworzenie modelu w specjalnym formacie SDF. Jest to ustandaryzowany zdefiniowany zewnętrznie format do opisywania składników robotów i czujników. Dzięki temu napisany w ten sposób model może być użyty także gdzie indziej, pod warunkiem przestrzegania standardu. Składnia jest zwykłym plikiem XML, co znaczy, że może być tworzony na każdym edytorze tekstowym.

Wtyczka do sterowania modelem jest zwyczajną biblioteką dołączaną w czasie symulacji. Pisze się ją w C++ jako własna klasa dziedzicząca po klasie wtyczki Gazebo. Dodatkowo tym sposobem może się łączyć z programem sterującym poprzez kolejki wiadomości udostępnione przez Gazebo, lub inne mechanizmy, nawet systemowe, jak gniazda, czy pamięć współdzielona.

Program działa domyślnie na dystrybucji Ubuntu, ale bez problemu można go także skompilować pod inne systemy. Interfejs jest dopracowany i przestrzega

wielu ustawień systemowych, jak DPI. Uruchamianie jest proste i nie wymaga dodatkowych ustawień, tworzenia odpowiednich katalogów, czy definiowania zmiennych systemowych. Podobnie do wszystkich tworzy ukryty katalog w katalogu domowym użytkownika, gdzie znajdują się wszystkie modele i logi.

1.4.2 V-Rep

Duży i skomplikowany silnik reklamujący się wieloma zaawansowanymi mechanizmami. Bogaty interfejs graficzny zakłada budowę i symulację wszystkiego w tym jednym programie.

Używa dwóch z silników z Gazebo, czyli ODE i Bullet, oraz Vortex i Newton. Z tej czwórki tylko Vortex ma zamknięty kod. Podobno symulacja fizyczna jest gorszej jakości, niż w Gazebo.

Problemem jest także zapisywanie utworzonych w programie modeli. Program zapisuje drzewiastą strukturę modelu w pliku binarnym własnego formatu, co uniemożliwia edycję i oglądanie modelu bez posiadania całego programu i wczytywania pliku. Brak przenośności, czy wsparcia kontroli wersji dla takich zbiorów bajtów także jest problemem.

Pisanie wtyczek najczęściej odbywa się w C. Są też dostępne inne języki skryptowe, jak Lua, Matlab, Java itp. Komunikacja z innymi programami odbywa się poprzez specjalne mosty w formie dodatków. API pozwala nam stworzyć mały interfejs graficzny do sterownia symulacją poprzez przyciski i suwaki.

Ze strony producenta pobieramy gotowe archiwum z programem, który nie wymaga żadnej instalacji i posiada wszystkie potrzebne zasoby do pracy i nauki, jak przykładowe modele istniejących komercyjnych robotów. Program działa na trzech Najpopularniejszych systemach operacyjnych — Windows, Linux i OS X.

Lepiej jednak jest używać do symulacji Gazebo głównie ze względu na jego otwartość, prostotę i elastyczność.

1.4.3 ROS

Najpopularniejsza biblioteka i gotowe algorytmy do budowania logiki sterowania. Dostępne są tutaj programy do obróbki danych, wyznaczania ścieżek, tworzenia map itp.

Właściwie nie ma dobrej alternatywy do zbioru tych bibliotek, poza implementacją wszystkiego ręcznie na nowo i po swojemu. Programy dla ROS pisze się w C++ i integruje z robotem za pomocą kilku gotowych struktur.

Instalacja programu na komputerze jest dużym problemem. Z wyjątkiem Ubuntu nie ma łatwego sposobu na wgranie go do innych systemów. Na przeszkodzie stoją błędy kompilacji dla nowszych wersji kompilatorów i inne problemy wykonania, jak naruszenie ochrony pamięci. Niektórym studentom wydziału się udało tego z wielkim trudem dokonać, lecz dużo prościej jest użyć Ubuntu na maszynie wirtualnej, bądź dysku USB. Takie rozwiązanie także daje dostęp do najnowszej wersji *Kinetic Kame* niedostępnej jeszcze na innych dystrybucjach.

Uruchomienie systemu wymaga wielu dodatkowych komend inicjalizujących, a także dopisywania wielu plików konfiguracyjnych za pomocą wielu skryptów do tworzonych projektów. Używanie linii poleceń wymaga ustawienia kilku zmiennych systemowych. Użycie niektórych funkcji wymaga uruchomionego demona

serwera w tle. Ogólnie instalacja i używanie ROS na systemie dużo śmieci, dlatego lepiej jest trzymać ją z dala od codziennego systemu operacyjnego na maszynie wirtualnej, lub dysku.

1.5 Ogólny zarys pracy

1. Należy stworzyć model w SDF zachowując wszystkie rozmiary i momenty prawdziwego robota. Należy ustawić bryły, aby przypominały kształtem podstawę i nadać im parametry fizyki.

2. Trzeba zdefiniować wszystkie więzy na koła i przegub i zamodelować je, aby silnik fizyczny dobrze współpracował. Taki model powinien na tym stanie reagować na zewnętrzne siły, ale nie ruszać się własnymi silnikami. Można go prosto przetestować działając siłą na elementy i patrząc, czy reaguje w poprawny sposób.

3. Wtyczka sterująca w Gazebo odczytuje dane z zewnątrz i odpowiednio obraca kołami. Na tym poziomie można dobudować zamiennik programu sterującego jedynie do podawania prostego sterowania na koła, bez żadnej logiki. Do komunikacji użyć struktur Gazebo, a nie ROS. Model powinien poprawnie reagować na proste wejście, jak naprzemienne koła itp.

4. Wtyczki dla czujników, aby generowały dane z enkoderów, oraz ewentualnie innych urządzeń, dodawały błędy pomiarowe i wysyłały je na zewnątrz, aby program sterujący miał dane do działania. Taki model jest w tym momencie kopią prawdziwego robota.

5. Program sterujący w ROS. Największy i najbardziej skomplikowany element, na szczęście wspólny dla obu bytów — wirtualnego i rzeczywistego. Zazwyczaj nie jest to praca jednego człowieka, a jego rozwój nie ustaje przez długi czas. Ten program dostarczy funkcji ruchów wywoływanych przez wyższy sterownik Velmy.

1.6 Pomocne implementacje

Istnieją już wcześniejsze modele jeżdżących robotów na kołach szwedzkich. Można z nich brać przykład i sugerować się źródłami kodu i modelów.

Kuka Youbot jest popularnym robotem tego typu. Jego modele są domyślnie dostępne zarówno w Gazebo, jak i w V-Rep. Tylko w przypadku V-Rep mamy wstępny sterownik do którego kierujemy zadane prędkości kół za pomocą graficznego interfejsu. Wersja dla Gazebo symuluje kłodę drewna.

Te profesjonalne modele także pomogą przy wstępnej weryfikacji zachowania się naszego modelu, czy nie zachowuje się nadzwyczaj dziwnie.