UR-3 ROS - Podsumowanie

1 Wstęp

Pracę rozpoczęto aby określić możliwości sterowania robotem UR-3 za pomocą platformy ROS. Zidentyfikowano następujące paczki umożliwiające współpracę z robotem, które przetestowano i określono ich użyteczność.

- ros-industrial/ur_modern_driver - okazała się przestarzała i prace nad nią zostały szybko zakończone
- UniversalRobots/Universal_Robots_ROS_Driver - mimo, że ta paczka pochodzi oficjalnego repozytorium *Universal Robots A/S* nie badano jej zbyt dokładnie ze względu na brak gotowych połączeń z Gazebo i MoveIt
- ros-industrial/universal_robot - Ta paczka mimo, że nie jest już aktualizowana daje największe możliwości jeśli chodzi o komunikację między ROS-owymi węzłami. Gotowe jest połączenie MoveIt, Gazebo i RViz. W sieci znalazło się też kilka przykładów i skryptów z pomocą których udało się stworzyć prosty kontroler do sterowania robotem w RViz i Gazebo.

2 Zalecenia i wymagania

Przed przystąpieniem do rozbudowy projektu zaleca się zapoznać z ROS Tutorials , a minimum, aby uruchomić nasz przykład jest stworzenie biblioteki Catkin - czyli punkty od 1.1 do 1.4 na powyższej stronie.

Demo działa na Ubuntu 18.04 i ROS Melodic, a instrukcja instalacji jest zawarta w katalogu.

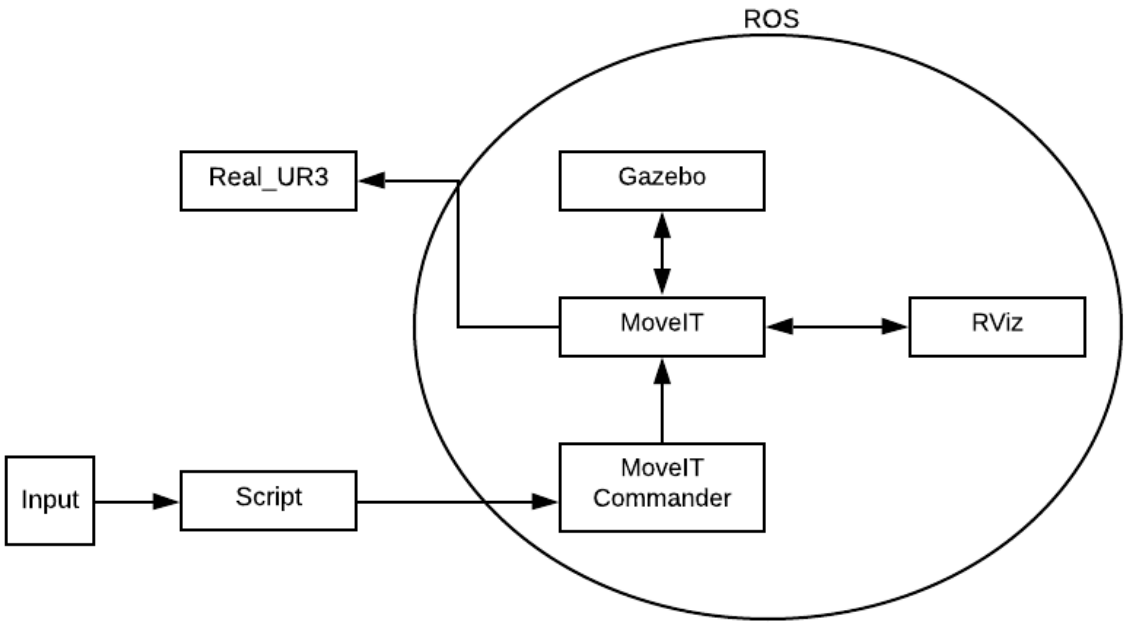
Aby rozszerzyć robota o efektor należy zapoznać się z GitHub Maroine Trabelsi , gdzie jest opisane jak należy dodawać obiekty do robota, ale na potrzeby tego dema powinno wystarczyć przekopiowanie pliku universal_robot\ur_description, wklejenie w te same miejsce w folderze biblioteka_robocza_ros\src i wykonanie polecenia catkin_make

3 Universal_Robot

Wszystkie kolejne badania skupiły się na universal_robot, więc dalej w pozostawiam w domyśle to, że chodzi o tą paczkę.

Paczka pozwala na pełną symulację robota, ale potrzebne są 3 węzły główne ROS do poprawnego działania.

- MoveIt - sterowanie robotem
- Gazebo - symulacja środowiska
- RViz - wizualizacja środowiska



Rysunek 1: Komunikacja między węzłami i elementami zewnętrznymi

Z racji, że w naszym przypadku MoveIt steruje wszystkimi węzłami ROS kluczową rolę w skryptach sterujących pełni *MoveIt Commander*. *MoveIt Commander* pozawala na poprawną komunikację pomiędzy skryptem, a węzłem MoveIt za pomocą *tematów*.

4 Komunikacja wewnętrzna

W naszym przypadku stworzono wirtualny joystick do poruszania robotem i na tym przykładzie zostanie przedstawiona komunikacja między węzłami.

Cała komunikacja w między głównymi węzłami jest wykonana wg standardu ROS Action Protocol czyli komunikacja przebiega na pięciu tematach:

- goal
- result
- status
- feedback
- cancel

Poziom skomplikowania tej struktury jest dość duży i odradza się ingerencję bezpośrednio na tematach.

w przypadku naszej aplikacji komunikacja odbywa się automatycznie między poszczególnymi węzłami. Skrypt wirtualnego joysticka jedynie komunikuje się z node move_group za pomocą
/rviz_Pielnik_xxx/motionplanning_planning_scene_monitor/parameter_descriptions
i */rviz_Pielnik_xxx/motionplanning_planning_scene_monitor/parameter_updates*
które są w pełni kontrolowane przez bibliotekę MoveIt_Commander i wywoływane przez funkcję *set_pose_target* i *go*.

5 Demo

Zadaniem dema jest ukazanie przykładu programu, który za pomocą platformy ROS może poruszać fizycznym robotem UR-3.
Demo to program, który spełnia funkcję prostego joysticka do sterowania robotem i pozwala na podnoszenie przedmiotów w środowisku symulatora RViz.

Do poruszania robotem w układzie kartezjańskim wykorzystuje się klawisze WSADRF, a do zmiany rotacji IKJLUO. Odkładanie i podnoszenie przedmiotu realizowane jest przez klawisze X i Y. Aby wyjść z programu należy wcisnąć V.

Program pisany w języku Python głównie korzysta z bibliotek MoveIt i MoveIt_Commander

6 Inne

Warto zwrócić uwagę na kilka funkcji *MoveIt Commander*, które nie zostały wykorzystane i w pełni przetestowane, ale wydają się mieć duże zastosowanie.

- pick/place - Te funkcje wymagają podania listy ruchów i nazwy obiektu do podniesienia. Jej zadaniem jest zautomatyzowanie procesu podnoszenia i odkładania przedmiotów.
- set_path_constraints - pozwala na ograniczenie ruchów robota w celu wymuszenia przez plannera obrania właściwej ścieżki.
- set_workspace - pozwala na określenie przestrzeni pracy robota.
- compute_cartesian_path - wymusza na plannerze ruch po lini prostej.

Ważne jest też na zauważanie różnic w sterowaniu robotem za pomocą standardowego oprogramowania, a poprzez planera MoveIt. W środowisku MoveIt nie są wyróżniane ruchy typu moveJ, moveI i TCP. Jesli zarządzony zostanie ruch z punktu A do punktu B planner obliczy ilość ścieżek określoną w programie i wybierze którąś z nich.

6.1 Kooperacja z innymi robotami

Wnioski z wdrożenia dodatkowego robota: Dodawanie drugiego robota do przestrzeni gazebo/rviz okazało się być problematyczne ze względu na topic "robot_description" który może pobierać i ładować do programu wizualnego jedynie jeden .urdf file, ten stanowi konstrukcje robota opisaną w języku macro. To oznacza że zapisując do topic'u

robot_description model robota X a następnie model robota Y, następuję nadpisanie i zachowanie modelu robota Y.

Możliwym rozwiązaniem jest użycie namespace'ów, czyli grupowanie robotów do podzbiorów a następnie wywołanie po kolej każdego z podzbiorów.

Takie rozwiązanie okazało się być łatwe w implementacji i obsłudze w przypadku mniej złożonych robotów, jak turtlebot, natomiast w przypadku robotów firmy universal robots jest to rozwiązanie którego implementacja jest trudna w obsłudze, ze względu na złożoność plików launch i ich powiązania

6.2 Odnośniki

Paczka została zbudowana na podstawie ros-industrial/universal_robot oraz GitHub Michael Ferguson

Wykonana w ramach Projektu Zespołowego dla Politechniki Wrocławskiej przez Maroine Trabelsi, Krzysztof Siera i Krzysztof Biały