**Описание проекта и алгоритма действий для создания нового проекта на базе данного**

В этом руководстве рассматривается интеграция URDF модели робота OpenMANIPULATOR-P в среду Unity-ROS для построения цифрового двойника.

Проект реализует 3 режима работы робота:

1. Передача в среду ROS и решатель MoveIt конфигурации робота (joinst), а также координаты объектов Target и TargetPlacement.
2. Решение и реализация задачи Pick and Place с помощью пакета MoveIt.
3. Передача из ROS в Unity конфигурации робота и ее визуализация в режиме реального времени.

**Основные этапы создания проекта:**

1. Установка необходимых программ и плагинов
2. Подготовка URDF модели манипулятора
3. Импорт и настройка URDF модели в Unity
4. Создание конфигурации в MoveIt Setup Assistant по URDF модели
5. Написание (правка) пользовательских скриптов в ROS, а также файлов msg и srv
6. Генерация с помошью ROS-TCP Connector скриптов в Unity по файлам msg и srv
7. Написание (правка) пользовательских скриптов в Unity и их привязка к сцене

**Структура проекта**

В проекте OpenMP\_Project используются пакеты-плагины Unity:

URDF Importer и ROS-TCP Connector.

На стороне Unity работают пользовательские скрипты (папка Scripts), а также скрипты, сформированные с помощью ROS-TCP Connector, описывающие с помощью классов С# данные (msg и srv) из среды ROS (папка RosMessages).

На стороне ROS Melodic:

* пакет ros\_tcp\_endpoint необходим для связи ROS-Unity
* пакет moveit\_msgs предоставляет функционал MoveIt.
* пакет openmp\_moveit содержит настройки MoveIt для конкретного робота, а также содержит пользовательские Hode, msg и srv.
* пакет open\_mp\_urdf содержит описание робота для работы MoveIt.

**Установка Unity** осуществляется с помощью Unity Hub. Проект протестирован на версии 2021.3.5f. При установке дополнительные компоненты необязательны, но для удобной работы с кодом может понадобиться текстовый редактор, например, VS Code.

**При первом запуске Docker Engine** (Docker Desktop) приложение может потребовать установить недостающие программы – WSL2 и Ubuntu – и предложит инструкцию по их установке.

**Установка XLaunch** (например, по ссылке <https://sourceforge.net/projects/xming/>)

Приложение необходимо для открытия окон внутри среды Docker, в частности, RViz.

1. Задайте следующие настройки:
2. 2.1. Display number 0
3. 2.2. Set Disable access control
4. 2.2. Unset Native opengl
5. 2.2. To additional parameters add: -xkblayout us,ru -xkbvariant winkeys -xkboptions grp:alt\_shift\_toggle
6. Либо запустите файл конфигурации config.xlaunch

В проекте используется **URDF** файлы.

Часто встречаются файлы с расширением .хacro – для перевода в .urdf можно использовать программу xacro2urdf или же самостоятельно, с помощью текстового редактора скорректировать файл.

При встраивании модели пользовательского робота, так или иначе придется вносить изменение в URDF. Особенно следует проверить правильность ссылок на файлы mesh.

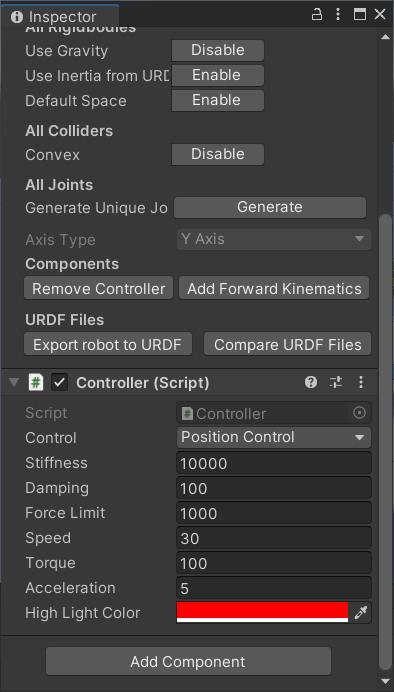
Если требуется создать файл .xacro достаточно изменить расширение.

**Для загрузки URDF** модели в среду Unity выполните:

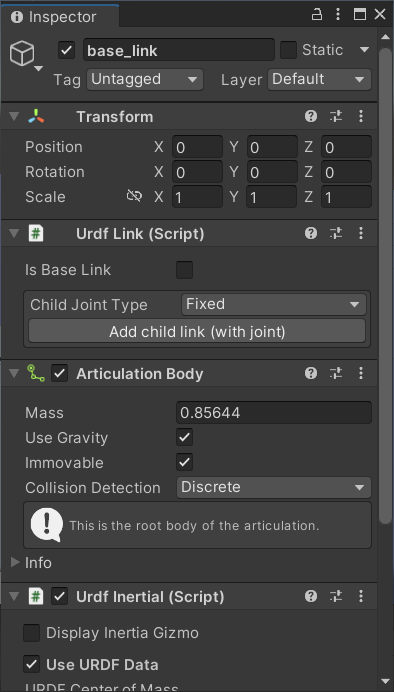
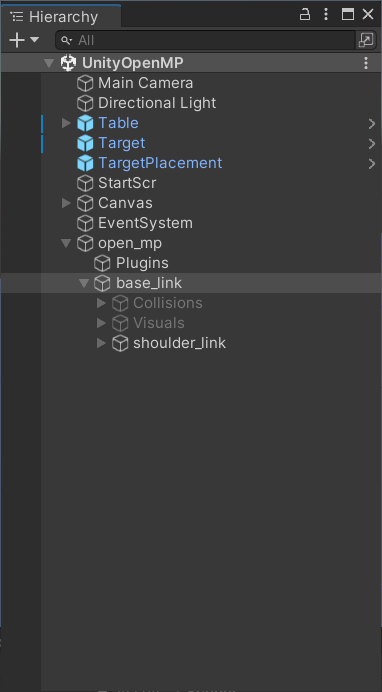
GameObject -> 3D object -> URDF model (import)

Выберете файл URDF, настройки оставьте стандартные – Y Axis, VHACD.

При успешном импорте выберете получившийся объект в Hierarchy, в настройках Inspector выставите **нулевые координаты** (Transform), а также **настройки Controller**, например:



В дереве робота выберете первое звено и задайте ему свойство **Immovable**.



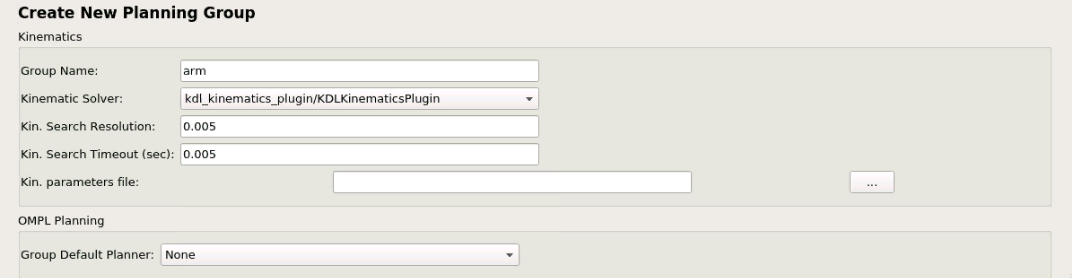
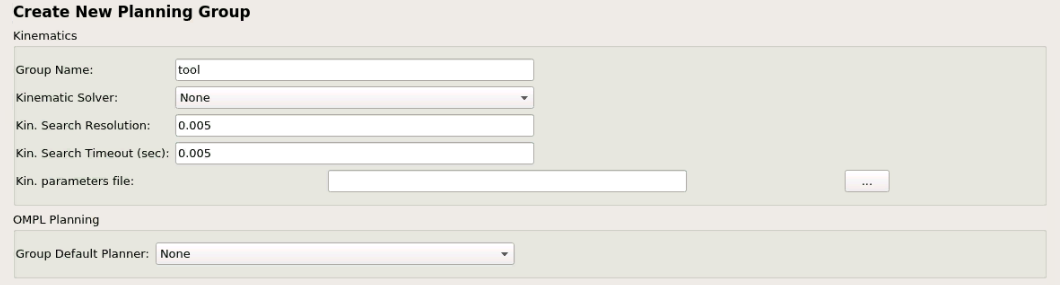
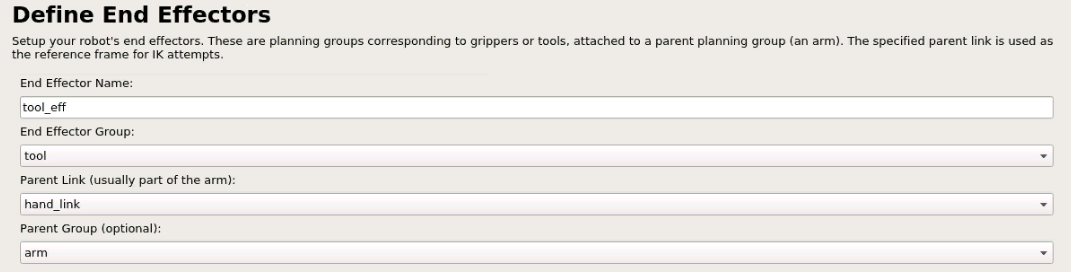
**Для проверки корректности импорта URDF модели** запустите сцену (кнопка Play).

Запуститься скрипт Controller из пакета URDF Importer. Используйте клавиши-стрелки для управление роботом.

Чтобы **создать конфигурацию** робота по URDF модели в **MoveIt Setup Assistant** необходимо для удобства отладки иметь доступ к среде ROS с возможностью загрузки и скачивания файлов. Если выполнить настройку в контейнере Dockerпредлагаемого проекта, то после каждого удаления контейнера настройку нужно будет выполнять заново.

Как один из способов, воспользуемся **The ROS Development Studio by The Construct.**

В этом онлайн сервисе необходимо зарегистрироваться, создать новый Rosject на ROS Melodic. Пакет MoveIt в нем предустановлен.

* Загрузите в папку ROS/src ваши ROS файлы – в ROS Dev Studio через Code Editor
* При каждом изменении пакетов выполняйте команды:  
   cd /catkin\_ws  
   catkin\_make  
   source devel/setup.bash
* Для запуска MoveIt Setup Assistant выполните:   
   roslaunch moveit\_setup\_assistant setup\_assistant.launch
* Сгенерируйте матрицу Self-Collision
* Создайте Planning Group под именем **arm**Добавьте joints до захвата, так чтобы звенья образовывали кинематическую цепь  
  В данном проекте добавлены 6 joints – от joints\_1 до joints\_6
* Создайте Planning Group для описания конечного звена манипулятора  
    
  В проекте это сочленение «end\_link\_fixed»
* (Необязательно) Создайте еще две Planning Group для правого и левого захватов по аналогии с arm
* Настройте End Effector  
  
* Внесите информацию об авторе и сгенерируйте пакет (virtual joints необязательны)
* Сгенерированные файлы вручную скопируйте в ROS/src/openmp\_moveit  
  Обратите внимание, что необходимо сохранить три launch файла и один yaml:  
  publish\_joint.launch  
  pick\_and\_place.launch  
  digital\_clone.launch  
  params.yaml

Для конвертации данных ROS-Unity используются **RosMessages**, которые формируют классы на языке C# по файлам msg и srv из ROS.

**Для формирования кода RosMessages выполните:**

Robotics - > Generate ROS Message

В ROS message path выберете путь до папки с файлами ROS.

В built message path - папку RosMessage.

Выполните:

ROS -> moveit\_msgs -> msg -> Robot Trajectore.msg - > build msg

ROS -> openmp\_moveit - >msg -> OpenmpMoveitJoints.msg -> build msg

ROS -> openmp\_moveit - >msg -> OpenmpTrajectory.msg -> build msg

ROS -> openmp\_moveit - >srv ->MoverService.msg -> build msg

**При изменении или добавлении файлов msg и srv** необходимо повторно выполнить генерацию кода. Для некоторых пакетов ROS данные скрипты уже сформированы и генерировать их заново не нужно.

**Кроме того, необходимо будет корректировать файлы CMakeLists и скрипты, если изменить название файлов msg и srv.**

**При замене модели робота на подобный по кинематике, изменять msg и srv необязательно, но при этом нельзя менять их названия.**

**Изменение скриптов Unity и ROS**

**В режиме I** скрипт **SourceDestinationPublisher** считывает конфигурацию робота, а также позиции Target и TargetPlacement и отправляет их в ROS Topic "/openmp\_joints".

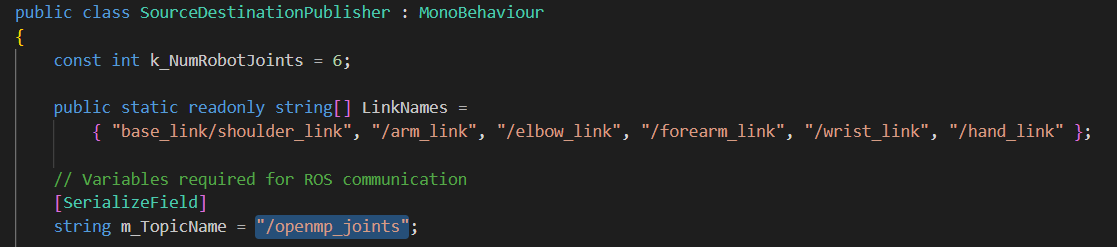
На стороне ROS запускается **publish\_joint.launch**, который создает server endpoint и выполняет trajectory\_subscriber.py.

**Для обеспечения режима I необходимо:**

Выполнить настройку робота по URDF модели в MoveIt Setup Assistant.

В SourceDestinationPublisher – изменить количество сочленений, их название, а также название Topic.

При изменении msg использовать новый класс взамен OpenmpMoveitJointsMsg.



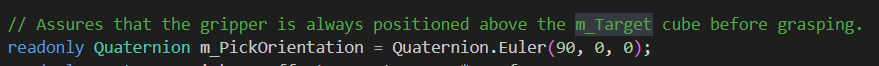
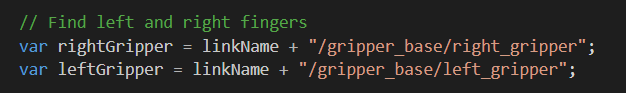
Также в trajectory\_subscriber.py заменить OpenmpMoveitJoints на название нового msg и изменить название Topic.

**В режиме II** скрипт **TrajectoryPlanner** формирует данные для отправки в MoverService, а также принимает от этого сервиса траекторию в пространстве joints, рассчитанную в решателе MoveIt. На стороне ROS пользователь запускает **pick\_and\_place.launch**, который выполняет mover.py, а также запускает среду RViz и пакет MoveIt.

Движение манипулятора осуществляется по фазам. Например, в конце первой фазы манипулятор останавливается чуть выше Target.

**Для обеспечения режима II необходимо:**

В **TrajectoryPlanner:**

* Выполнить изменения в **режиме I**
* Изменить количество сочленений <k\_NumRobotJoints>
* Изменить название сервиса < m\_RosServiceName >
* Изменить ориентацию захвата  
  
* Изменить путь до звеньев захвата  
  
* В функциях CloseGripper и OpenGripper при необходимости изменить значение target.
* Заменить OpenmpMoveitJointsMsg на новый класс при изменении msg в режиме I.
* Заменить MoverServiceRequest и MoverServiceResponse при изменении MoverService.

В **mover.py:**

* Изменить названия msg и srv, если они были изменены
* Изменить поле joint\_name
* Изменить константы в полях <pre\_pose.position.z>, <pick\_pose.position.z>,

< place.position.z>.

Для отладки и выбора этих констант возможно «выключение» некоторых фаз движения манипулятора путем их комментирования. В окне RViz отображается последняя фаза движения робота.

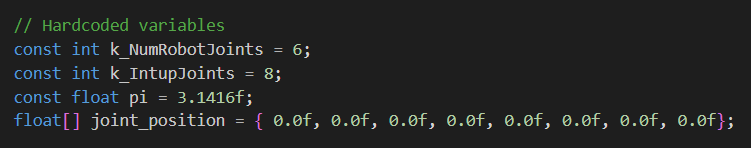
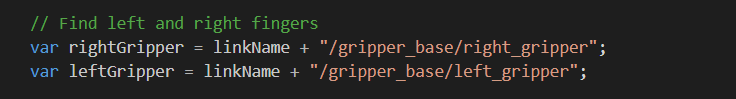
**В режиме III** скрипт **JointsSubscriber** после запуска периодически принимает данные из ROS Topic "/joint\_states" и визуализирует принятую конфигурацию на сцене Unity. На стороне ROS необходимо выполнить **digital\_clone.launch**,которыйзапускает среду RViz и пакет MoveIt. Среда RViz выбрала как источник данных – joints – для построения цифрового двойника. Topic /joint\_states запускается вместе с MoveIt из пакета Sensor.

В окне RViz выберете вкладку Planning, нажмите кнопку Plan&Execute для отправки конфигурации робота в среду Unity. Возможно переключить Planning Group, а также использовать вкладку Joints для точной настройки каждой кинематической пары.

**Для обеспечения режима III необходимо:**

Выполнить настройку робота по URDF модели в MoveIt Setup Assistant.

В **JointsSubscriber:**

* Изменить количество сочленений до захвата и общее их количество
* Выполнить изменения в режиме I  
  
* Изменить путь до захвата  
  
* Изменить название Topic и заменить JointStates\_Msg на новый msg при изменении источника joints

**Для переключения между режимами** на стороне Unity необходимо в Hierarchy выбрать элемент Canvas -> StartScr. В окне Inspector найдите поле On click(), затем нажмите на правое поле под ним и выберете исполняемую функцию.

Так осуществляется настройка кнопки UI Button.   
**Для привязки нового скрипта** к сцене добавьте его к какому-либо объекту (в том числе и к манипулятору) через Inspector -> Add Component.

**Для создания быстро развертываемой модели** манипулятора создайте Prehabs.

Перетащите объект в Project -> Assest для создания Prehabs.  
Для создания быстро развертываемого Unity проекта создайте Assets Package.

Assests -> Export Package.

Пакеты-плагины в Package Manager необходимо будет установить в новом проекте.

@git: FIComapny