Лабораторная работа №2
«Построение подвижных твёрдотельных моделей робототехнических аппаратов»

Цель:
изучить основные принципы описания моделей роботов в симуляторе Rviz.

Задачи:

- научиться создавать собственные ROS-пакеты;
- изучить основы формата описания робототехнических аппаратов URDF;
- изучить способы описания физических свойств звеньев роботов;
- познакомиться с графическим приложением Rviz;
- создать простейшую модель робота-манипулятора.

Теоретическая часть

Введение

Уровень файловой системы ROS определяет ресурсы, которые хранятся на диске. Далее выделены следующие элементы файловой системы ROS (рис. 1):

- Пакеты (*packages*) – это набор логически связанных файлов, служащих определенной цели. Каждый пакет может содержать библиотеки, исполняемые файлы, скрипты и т. д. Все программное обеспечение ROS организовано в пакеты.

- Метапакеты (*metapackages*) являются специализированными пакетами, которые служат только для представления группы других пакетов, связанных одной задачей. Метапакеты не содержат кода, файлов или других элементов, обычно встречающихся в пакетах. Метапакет просто ссылается на один или несколько связанных пакетов, которые группируются вместе.

- Манифест пакета (*manifest*) – это специальный файл (package.xml), служащий для определения зависимостей между пакетами и содержащий данные о пакете (имя, версия, описание, информация о лицензии, зависимости и т. п.).

- Репозитории (*repositories*) – это коллекция пакетов, которые имеют общую систему управления версиями.

```
workspace_folder/       – рабочее пространство, например, catkin_ws
  src/                  – папка источников
    Project1            – репозиторий проекта, например, Myrobot
      CMakeLists.txt    – главный CMake-файл проекта, сгенерированный catkin
      package_1/
        CMakeLists.txt  – файл CMakeLists.txt для package_1
        package.xml     – манифест для package_1
        other folders   – папки для описания узлов и хранения различных данных
      ...
      package_n/
        CMakeLists.txt  – файл CMakeLists.txt для package_n
        package.xml     – манифест для package_n
        other folders   – папки для описания узлов и хранения различных данных
```

Рисунок 1 – Структура рабочего пространства ROS на диске

Любой написанный и запущенный код в рамках ROS должен принадлежать конкретному пакету, выполняющему важную функцию. Например, это могут быть пакеты для создания кинематики роботов, их моделирования в симуляторах, описания характеристик и возможностей систем управления, обработку сенсорных данных и т.д.

Чтобы создать новый пакет в текущем каталоге нужно выполнить команду:

```
$ roscreate-pkg [package_name] [depend1] [depend2] [depend3]
```

Утилита roscreate-pkg создает новый пакет ROS. Все пакеты ROS содержат набор схожих файлов: манифесты, CMakeLists.txt, mainpage.dox, и Makefile. Указанная утилита берёт на себя большую часть рутинных задач, которые нужно выполнить при создании нового пакета вручную, и тем самым устраняет распространённые ошибки, вызванные ручным созданием файлов сборки и манифестов. При наличии в командной строке дополнительных зависимостей roscreate-pkg настроит сгенерированный пакет для работы с ним.

Сгенерируем новый проект Lab_02 с помощью следующих команд:

```
$ mkdir -p ~/catkin_ws/src/Lab_02
$ cd ~/catkin_ws/src/Lab_02
$ roscreate-pkg myrobot_description std_msgs rospy roscpp
$ cd ~/catkin_ws
$ catkin_make
$ . ~/catkin_ws/devel/setup.bash
```

С помощью первой команды в пространстве catkin была создана папка с именем проекта Lab_02. Вторая команда переместила во вновь созданную папку. А третья создала пустой пакет с именем myrobot_description. Использованные зависимости std_msgs, rospy и roscpp требуются для того, чтобы будущие узлы могли понимать структуры сообщений ROS, а также их можно было модифицировать на языках C++ и Python. Далее, вернувшись в основную директорию рабочего пространства ROS командой catkin_make производится процесс окончательной сборки пакета. Последняя строчка позволяет записать имя и зависимости нового пакета в общий список зависимостей ROS.

rospy и roscpp являются клиентскими библиотеками (Client Libraries). Клиентские библиотеки позволяют различным языкам программирования, общаться через ROS. Указанные зависимости являются зависимостями первого порядка, поэтому после создания пакета их можно посмотреть с помощью утилиты rospack:

```
$ rospack depends1 myrobot_descriptions
std_msgs
rospy
roscpp
```

В большинстве случаев, зависимость также будет иметь свои собственные зависимости. Например, rospy зависит от других пакетов.

```
$ rospack depends1 rospy
roslib
roslang
```

После создания пакета, в директории myrobot_description будут находиться папки include (для header-файла языка C++) и src (место хранения исходных кодов), а также файлы

CMakeLists.txt и package.xml. В данной лабораторной работе мы будем использовать только внутренние скрипты ROS, поэтому папки include и src можно удалить.

Файл package.xml содержит информацию о пакете: название, автора, лицензию, список зависимостей от других пакетов ROS. Для поддержания коммуникации в ROS-сообўестве введіте в соответствующие строки свою открытую личную информацию (рис. 2)

Второй обязательный файл для пакета – это файл CMakeLists.txt. Этот файл содержит инструкции для Catkin (системы сборки пакетов, которая использует Cmake). В этом файле содержатся инструкции на создание исполнительных файлов, очередность сборки проекта, создание символьных ссылок и тп. Откройте и посмотрите на содержание файла.

```
<?xml version="1.0"?>
<package format="2">
  <name>myrobot_description</name>
  <version>0.0.1</version>
  <description>The myrobot_description package</description>

  <maintainer email="cola@todo.todo">cola</maintainer>
  <license>TODO</license>
  <! - <url type="website">http://wiki.ros.org/test_package</url>  ->
  <! - <author email="jane.doe@example.com">Jane Doe</author>  ->

  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

  <! - The export tag contains other, unspecified, tags  ->
  <export>
    <! - Other tools can request additional information be placed here  ->

  </export>
</package>
```

Рисунок 2 – Описание манифеста сгенерированного пакета

Далее перейдём к функциональному наполнению сгенерированного пакета.

Спецификация грамматики URDF

URDF (Unified Robot Description Format – Объединенный Формат Описания Роботов, является спецификацией XML, используемой в академической среде и промышленности для моделирования систем, состоящих из большого числа подвижных частей и звеньев. Это могут быть звенья роботов-манипуляторов для производства сборочных конвейеров, ноги и руки аниматронных роботов для парков развлечений и тд. URDF является языком описания роботов в ROS. Этот формат признан многими организациями, он является основой для создания других подобных языков, а также поддерживается рядом других серьёзных систем

моделирования и разработки роботов, как например, MATLAB, в котором модели URDF можно конвертировать в среду Simscape Multibody.

Язык URDF представляет робота как совокупность звеньев (link) и сочленений (joint), что позволяет однозначно описать его кинематическую схему, а также сенсоров и ряда вспомогательных параметров. Для подробного ознакомления с данным форматом могут быть полезны ресурсы: руководство, спецификация элементов.

Описание звена в общем виде может быть представлено следующим образом. Блок visual содержит видимую модель робота, т.е. описывает его геометрию и свойства поверхности. Геометрия может быть задана как простейшими формами (параллелограмм, цилиндр и сфера), так и сеткой, полученной в CAD-системе. Блок collision описывает ту модель, которая будет использована при выявлении столкновений с другими объектами. Чаще всего эта часть совпадает с описанием блока visible, но может и отличаться. Например, для ускорения расчётов сложный профиль поверхности может быть аппроксимирован каким-либо геометричесеким примитивом. В блоке inertial содержатся физические свойства звена, такие как масса и инерция.

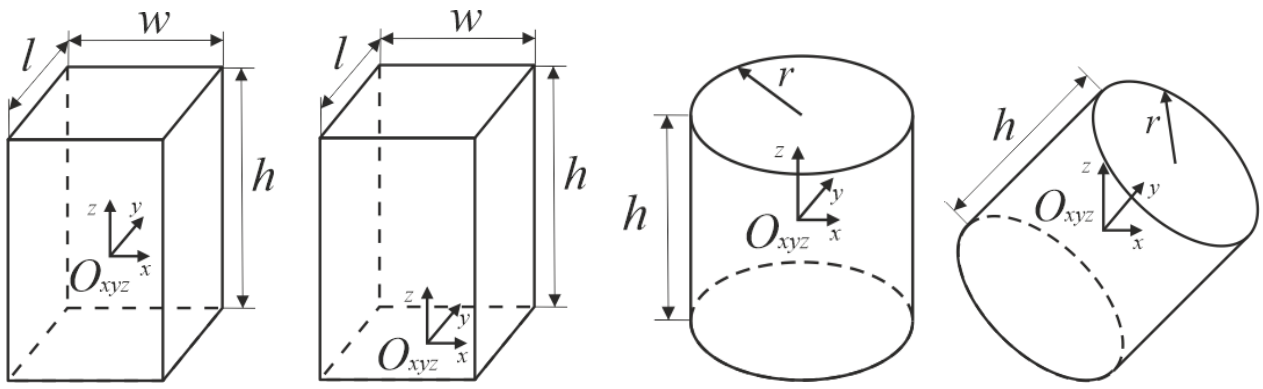
```
<link name="имя_звена">
  <visual>
    ... (геометрическая модель)
  </visual>
  <collision>
    ... (модель, используемая для определения взаимодействия)
  </collision>
  <inertial>
    ... (физические свойства модели)
  </inertial>
</link>
```

Рассмотрим описание указанных блоков более подробно.

```
<visual>
  <origin xyz="0 0 0" rpy="0 0 0" />
  <geometry>
    <box size = "0 0 0" />
    <cylinder radius= "0" length= "0" />
    <sphere radius = "0" />
    <mesh filename= "PATH_TO_FILE" scale="1 1 1" />
  </geometry>
  <material name>
    <color rgba = "r g b a" />
    <texture filename />
  </material>
</visual>
```

Тэг <origin> указывает точное положение звена относительно его собственной системы координат O_{xyz}, которая для всего робота является одной из множества локальных. Параметр *xyz* указывает на линейное перемещение звена вдоль осей *x*, *y* и *z*, а *rpy* – на последовательные повороты вокруг соответствующих осей своей собственной локальной системы координат. Как правило, локальная система координат O_{xyz} каждого звена находится в его геометрическом центре. Если звено представляет собой параллелограмм, то в блоке <geometry> требуется

указать тэг <box>, параметры которого укажут на его высоту вдоль оси O_z, ширину вдоль оси O_x и длину вдоль оси O_y (рисунок 3 *а*). Если требуется перенести его локальную систему координат в плоскость своего основания, то в значениях вектора *xyz* следует указать следующий вектор “0.5*h* 0 0” (рисунок 3 *б*). Аналогично, строиться и следующий геометрический примитив – цилиндр, высота которого по умолчанию расположена также вдоль оси O_z (рисунок 3 *в*). Если значения вектора *rpy* изменить на “0 0.7854 0”, то цилиндр повернётся вокруг оси O_y на 45° по правилу правой системы координат (рисунок 3 *г*).



а) *б*) *в*) *г*)

Рисунок 3 – Описание звена отсносительно его собственной системы координат

Следует указать, что в URDF-скриптах используются следующие единицы измерения: длины – в метрах, углы поворотов – в радианах, угловая скорость – рад/с, линейная скорость – м/с, масса – кг.

Геометрические модели звеньев могут быть представлены как геометрическими примитивами, так и твердотельными моделями сложных объектов, например, 3Д-файлами с расширением *.STL, *.DAE и *.OBJ, адреса расположения которых указываются в теге <mesh>. Помимо адреса файла требуется указать значение масштабирующего коэффициента *scale* по всем трём осям:

```
<mesh filename="package://[PACKAGE_NAME]/[3D-FILE_PATH]" scale = "1 1 1" />
```

Тэг <color> позволяет не только задать любой цвет через указание значений яркости его основных трёх цветов R (красный), G (зелёный) и B (синий), но и влиять на степень его прозрачности (четвёртый компонент). Все четыре компонента имеют значения от 0 до 1. Тэг <texture> также более детальна будет рассмотрен в следующих лабораторных работах.

Как уже говорилось выше, тэг <collision> практически полностью совпадает по описанию с тегом <visual>. Так как его основным назначением является описание физических границ звена, которые будут учитываться при фиксировании факта столкновения с другими объектами, то единственным отличием является отсутствие тэга <texture> и более упрощённое описание геометрии самого звена.

```
<collision>
  <origin xyz="0 0 0" rpy="0 0 0" />
  <geometry>
    <box size = "0 0 0" />
```

```
      <cylinder radius= "0" length= "0" />
      <sphere radius = "0" />
      <mesh filename= "PATH_TO_FILE" scale="1 1 1" />
    </geometry>
  </collision>
```

Следующий блок необходим для описания физических свойств звена. В частности, в теге <mass> указывается его масса. Тэг <inertia> содержит только шесть из девяти элементов матрицы момента (тензора) инерции, так как матрица является симметричной относительно главной диагонали. Формулы вычисления моментов инерции для ряда твёрдых тел различной формы приведены в следующей ссылке.

```
  <inertial>
    <origin xyz = "0 0 0" rpy = "0 0 0" />
    <mass value = "0" />
    <inertia ixx iyy izz ixy ixz iyz />
  </inertial>
```

Если робот содержит более одного звена, его элементы должны быть соединены. Для этого служат сочленения (шарниры) – тэг <joint>, минимальное описание которых включает в себя указание родительского звена, а также звена-потомка. Также, необходимо определить тип соединения: revolute – для управляемых вращательных движений, prizmatic – для управляемых поступательных движений, continuous – для неуправляемых вращений, например, колёс, fixed – если закрепление неподвижное, например, сварное соединение.

```
<joint name="имя_сочленения" type="тип_сочленения">
  <parent link="звено-родитель" />
  <child link="звено-потомок" />
  <origin xyz = "0 0 0" rpy = "0 0 0" />
  <axis xyz = "0 0 0" />
  <limit lower="0" upper="0" effort="0" velocity="0"/>
</joint>
```

Тэг <origin> в блоке <joint> хоть полностью и совпадает по описанию с одноимённым тэгом в блоке <link>, но имеет другое назначение. Он указывает на смещение локальной системы координат O_{xyz} текущего звена относительно предыдущего по иерархии всех элементов моделируемой системы (рисунок 4).

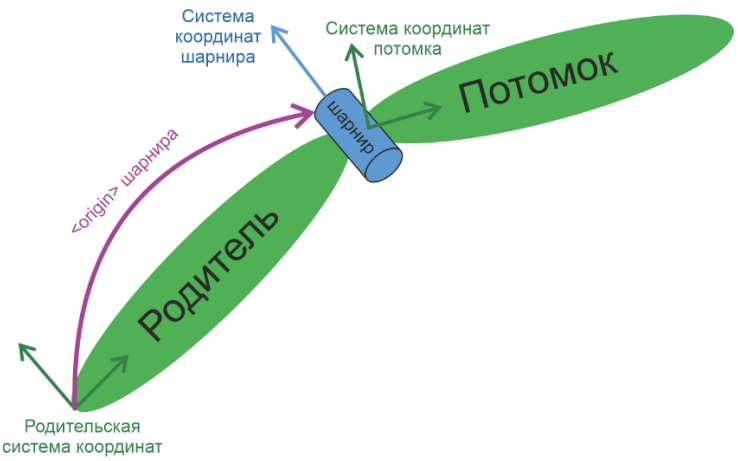
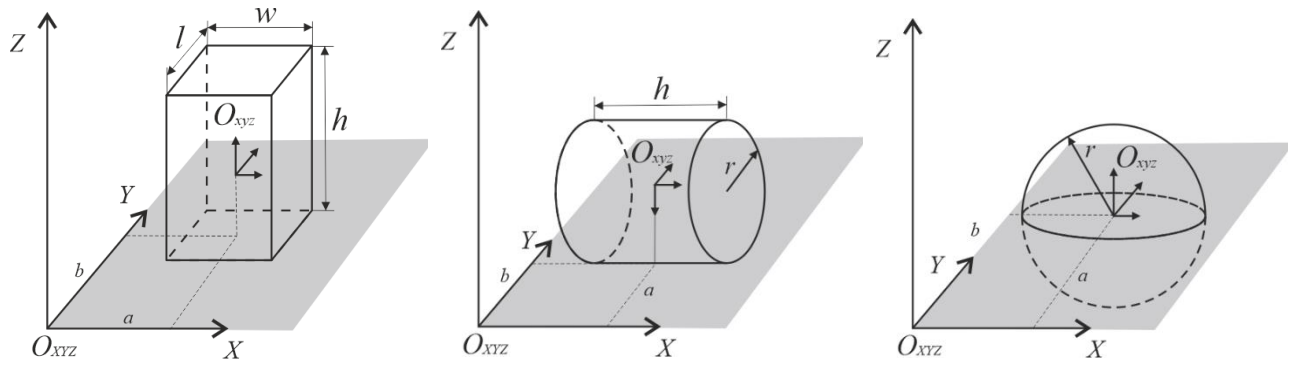


Рисунок 4 – Пример соединения звеньев через шарниры

Другим словами, если требуется в глобальной системе координат O_{XYZ} поставить на плоскость O_{XY} параллелепипед высотой h, то в позиции смещения его локальной системы координат O_{xyz} по координате z требуется указать значение $0{,}5h$ (рисунок 5 *а*). При этом, как указывалось выше, если элементы тэга <origin> в блоке <link> все являются нулями, то собственная система координат параллелепипеда будет всегда находиться в геометрическом центре указанной фигуры. Это правило относится и к остальным используемым примитивам цилиндр и сфера (рисунок 5 *б* и *в*).



<box size = "w l h" />
<origin xyz="a b h/2"
rpy="0 0 0"/>

а)

<cylinder radius= "r" length= "h" />
<origin xyz="a b r"
rpy="0 1.5708 0" />

б)

<sphere radius = "r">
<origin xyz="a b 0"
rpy="0 0 0" />

в)

Рисунок 5 – Способы задания размеров и ориентации геометрических примитивов в пространстве

Тэг <axis> указывает одну из трёх осей, вдоль которой будет производиться поступательное движение или вокруг которой будет реализовываться выбранное вращательное движение. Например, на рисунке 5 *б* видно, что цилиндр был повёрнут вокруг оси O_y на угол 90°. В результате, его локальная ось O_z стала параллельна плоскости O_{XY} в глобальной системе координат. Теперь, чтобы его заставить катиться по плоскости требуется указать, чтобы его вращение было вокруг оси O_z. Для этого в тэге <axis> в поле *xyz* требуется указать вектор "0 0 1". Свойства тэга <limit> позволяют ограничить не только конечные значения вращения/передвижения шарниров, но также передаваемые через них момент силы и скорость.

В итоге описание всего робота имеет следующий вид:

```
<robot = "ROBOTNAME">
  <link name = "LINKNAME1">
  …
  </link>
  <joint name="JOINT1" type="JOINTTYPE">
  …
  </joint>
  …
  <link name = "LINKNAMEn">
  …
```

```
  </link>
  <joint name="JOINTn" type="JOINTTYPE">
  …
  </joint>
</robot>
```

Следует указать, что часто разработчиками ROS рекомендуется первое звено реализовать в виде «бестелесного», не имеющего массы и инерцию, который соединяет второе «действующее» звено через шарнир с типом соединения “fixed”. С целью повышения уровня читаемости кода перед тэгом <robot> следует указать строчку-пролог <?xml version="1.0"?>. Это подскажет текстовому редактору, какая кодировка используется в XML документе, что добавит цветовую раскраску тэгов и переменных. После того, как робот будет полностью описан по указанным правилам данный текстовый файл требуется сохранить с расширением *.urdf в новой директории urdf текущего проекта Lab_02. Например, для указанного файла выберем имя myrobot. Чтобы проверить корректность описания робота в URDF-формате требуется воспользоваться следующей командой:

```
$ check_urdf myrobot.urdf
```

Если в системе ROS ещё нет такой команды, то её требуется установить следующей строкой

```
$ sudo apt install liburdfdom-tools
```

Если всё сделано корректно, то в командной строке должна появиться надпись «Successfully Parsed XML» и соответствующее текстовое описание дерева связей звеньев. Следующая команда

```
$ urdf_to_graphiz myrobot.urdf
```

сгенерирует PDF-документ, в котором будет находиться описание кинематики робота в виде графа.

Визуализация роботов в симуляторе Rviz

После этого уже можно приступить к визуализации проектируемого робота в трёхмерном виде. Для этого требуется создать новую директорию launch, в которой будут содержаться скрипты для запуска созданных узлов. В указанной директории создадим новый файл с именем, например, showrobot.launch со следующим содержанием:

```
<?xml version="1.0"?>
<launch>
  <param name="robot_description" textfile="$(find
myrobot_description)/urdf/myrobot.urdf"/>

  <node name="rviz" pkg="rviz" type="rviz" args=" -d $(find
myrobot_description)/rviz/config.rviz" />
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"/>
  <node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui"
type="joint_state_publisher_gui"/>

</launch>
```

Данный launch-файл запускается с помощью типовой команды

```
$ roslaunch [package_name] [launch_name]
```

В нашем случае это будет команда

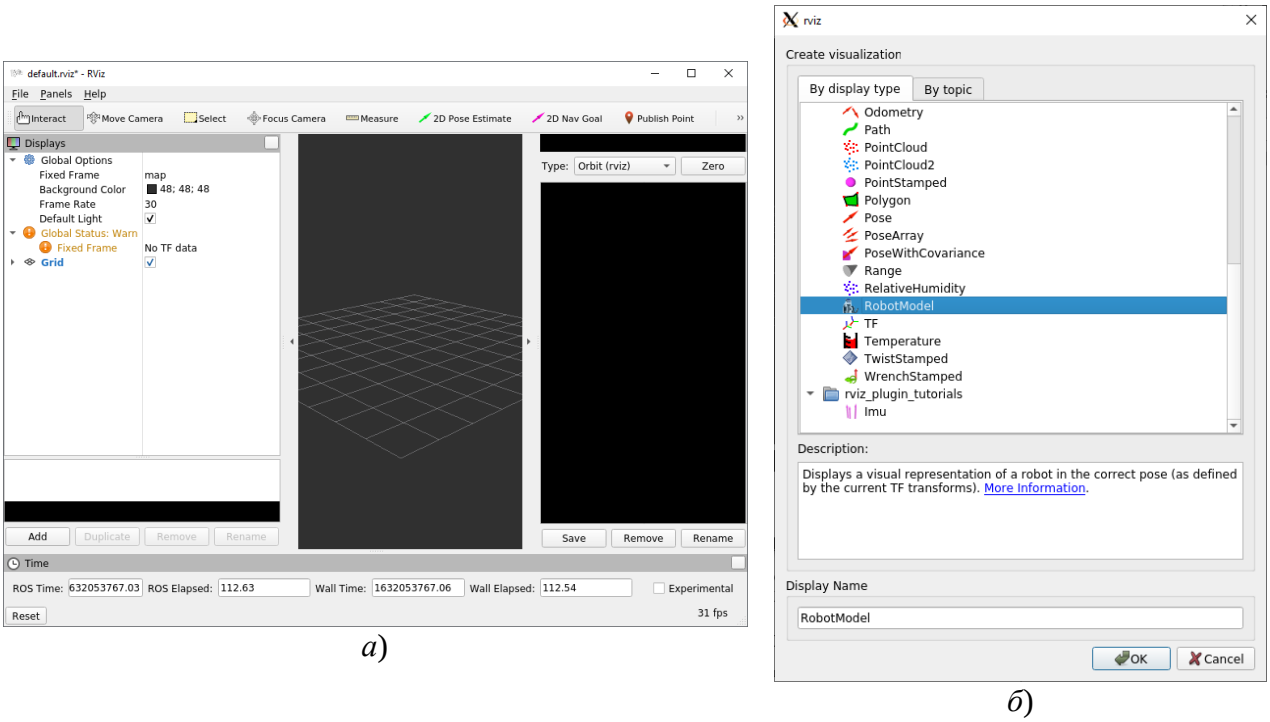
```
$ roslaunch myrobot_description showrobot.launch
```

Данный launch-файл запустит roscore, распарсит указанный в нём URDF-файл, а также активирует 3 узла: запустит симулятор Rviz, в графическом окне которого будет отображаться проектируемый робот, создаст узел /robot_state_publisher, который через тему /tf публикует геометрические параметры звеньев, и узел /joint_state_publisher_gui, который через тему /joint_states публикует в узел /robot_state_publisher положения шарниров. Данный набор узлов и тем является минимальным необходимым, для отображения роботов в симуляторе Rviz.

Однако, перед первым запуском launch-фала требуется выполнить ряд действий. Для начала надо в отдельном терминале запустить roscore, а в другом ввести команду

```
$ rosrun rviz rviz
```

После этого откроется графическое окно, в котором будет отображаться симулятор Rviz (рисунок 6 а). Так как на данный момент в ядре ROS не содержится описание какого-либо робота, то его центральное окно является пустым. Перед первым запуском требуется сконфигурировать отображение требуемых функциональных элементов. Для этого требуется в левом нижнем углу нажать кнопку «Add», которая активизирует диалоговое окно (рис. 6 *б*). Данное окно содержит список различных функциональных элементов, с которыми мы будем знакомиться по мере необходимости. В данной лабораторной работе требуется выбрать из списка пункты «RobotModel» и «Axes». После этого, через меню «File»-«Save Config As» требуется сохранить полученные настройки в директорию rviz в файл config.rviz. Именно путь к этому файлу указан в поле «args» при описании первого узла рассматриваемого launch-файла.



а) *б*)

Рисунок 6 – Настройка симулятора Rviz

Следует указать, что при первом запуске Rviz в меню «Global Options»-«Fixed Frame» в выпадающем списке справа следует указать звено, от которого будет строиться дерево вывода кинематической схемы конкретного URDF-фала. Как правило, здесь указывается корень графа кинематической схемы.

В самых новых версиях ROS требуемый узел, передающий положения шарниров, первоначально не установлен в сборку и его требуется устанавливать отдельно. Для этого в командной строке требуется ввести команду

```
$ sudo apt install ros-noetic-joint-state-publisher-gui
```

В более ранних версиях используется узел joint_state_publisher, который уже есть по умолчанию. Узел joint_state_publisher_gui запускает отдельное графическое окно, которое содержит каретки, управляющие положением всех подвижных шарниров.

После выполнения описанных выше действий в главном окне симулятора Rviz появится отображение проектируемого робота. Благодаря широким настройкам Rviz пользователь может вручную отобразить или скрыть любое из имеющихся звеньев. Также для каждого звена можно отобразить оси его локальной системы координат и вывести подпись его имени. Это позволяет в ручном режиме промоделировать возможности поведения проектируемого робота.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Целью данной лабораторной работы является реализация в симуляторе Rviz подвижной модели робота-манипулятора с 5 степенями свободы, оснащённым рабочим органом – схватом.

Для этого требуется решить следующие задачи:

1. в зависимости от полученного варианта описать в формате URDF робот-манипулятор с требуемой кинематической системой;
2. поместить в рабочую область манипулятора красную сферу небольшого размера;
3. с помощью графического приложения joint_state_publisher_gui подобрать такие значения поворотов/перемещений шарниров, чтобы указанная сфера оказалась между губками схвата;
4. с помощью команд, изученных в предыдущей лабораторной работе, определить начальные и конечные значения шарниров;
5. создать скрипт, в котором теме-получателю последовательно отсылаются значения шарниров, равномерно распределённые между начальным и конечным положениями звеньев.

Типовая команда, изменяющая положения *n* шарниров, выглядит следующим образом:

```
$ rostopic pub -r 10 /joint_states sensor_msgs/JointState '{header: auto, name: ['joint1','joint2',… , 'jointn'], position: [pos_1, pos_2, …, pos_n], velocity: [], effort: []}'
```

где ключ “–r 10” указывает на скорость обновления потока данных со скоростью 10 раз в секунду. Следует также указать, чтобы приведённая команда не конфликтовала с узлом joint_state_publisher_gui, (после того, как будет запущен Rviz через указанный выше launch-файл) предварительно его требуется выключить командой

```
$ rosnode kill /joint_state_publisher_gui
```

Кинематика манипулятора определяется полученным вариантом задания, который соответствует одной из четырёх систем координат (рисунок 7). Робот состоит из трёх основных

подвижных звеньев, которые задают специфику системы координат, а схват в своём основании дополнительно имеет две степени свободы – вращение вокруг осей *x* и *z*.

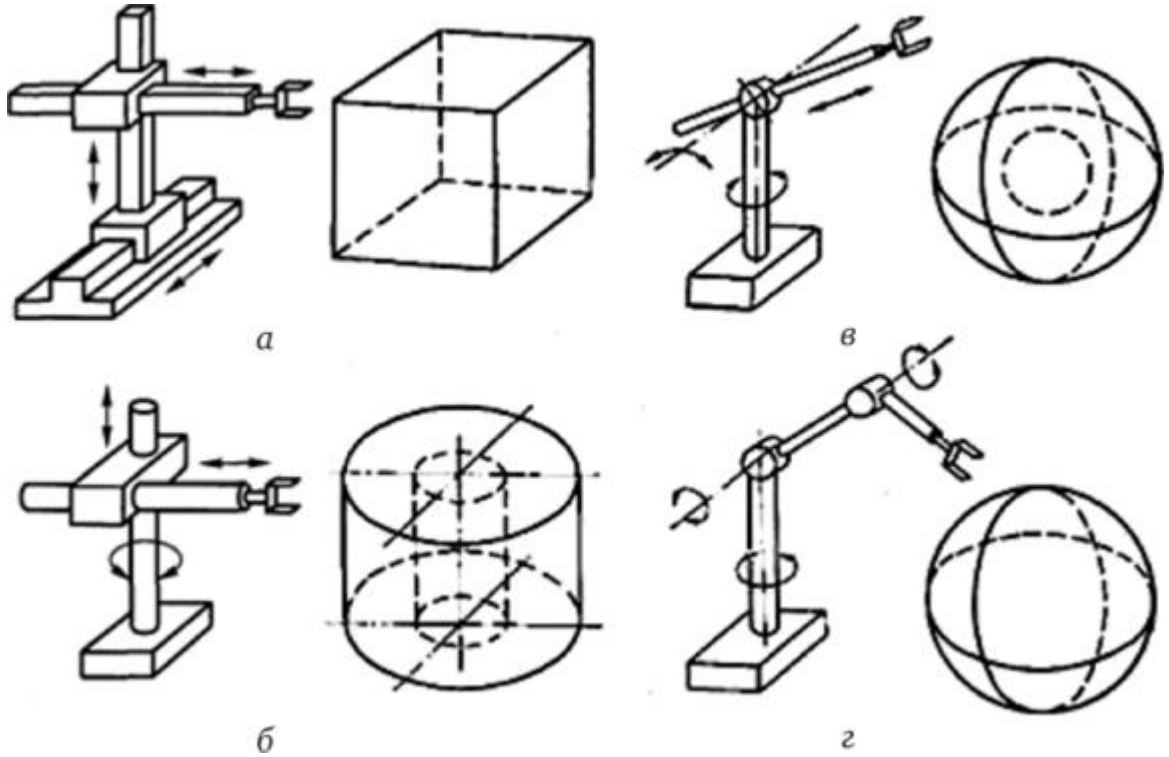


Рисунок 7 – Геометрия зон обслуживания промышленных роботов, работающих в разных системах координат: *а* – прямоугольная; *б* – цилиндрическая; *в* – сферическая; *г* – угловая

По условию задания три звена манипулятора требуется создать из геометрических примитивов, а схват представляет собой уже готовый 3Д-файл в формате *.STL, у которого центр локальной системы координат находится в центре основания (рисунок 8). Как правило, все 3Д-файлы помещают в специальную директорию текущего пакета с именем «meshes».

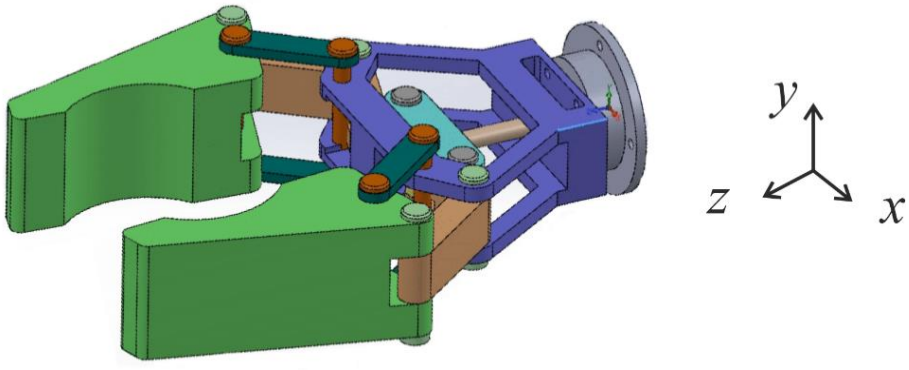


Рисунок 8 – Внешний вид 3Д-модели схвата манипулятора

Контрольные вопросы

1. В чём разница между пакетами и метапакетами?
2. Сколько узлов и тем может содержать пакет?
3. В каком файле/лах можно найти описание пакета?
4. С помощью какой команды можно узнать зависимости конкретного пакета?
5. Какие узлы должны быть активизированы, чтобы отобразить трёхмерную модель робота в симуляторе Rviz?
6. Какие дополнительные возможности предлагает Rviz для удобного отображения роботов?

Требования к отчету по работе

Отчет по выполнению лабораторной работы должен содержать:

1. титульный лист принятого в образовательной организации образца с названием лабораторной работы, фамилиями исполнителя и проверяющего работу;
2. цель и задания лабораторной работы;
3. листинги urdf-файла и управляющего скрипта, графы связей между узлами и элементами кинематической схемы (скриншоты);
4. выводы по лабораторной работе.