LR_1 Запуск ROS пакета

Чтобы работать с пакетами нам необходим интрумент catkin

Установка catkin выполняется через `apt-get install python3-catkin-tools`

❓ Важно!!! Мы будем пользоваться **catkin build** вместо catkin_make не смотря на то, что во всех туториалах предлагается catkin_make !!!

В чем отличие?

Основное отличие — изолированная среда, которую вы получаете при сборке Catkin. Это делает всю конфигурацию сборки более разделенной и устойчивой к изменениям в конфигурации (добавление/удаление пакета, изменение переменной cmake и т. д.).

Кроме того, мы получаем более структурированный и легко читаемый цветной вывод командной строки, что намного приятнее.

Также вместе с `catkin` мы получаем много полезных команд, таких как `catkin clean` для очистки `build`, `devel`, `install`, `catkin list` и т.д.

Ну и **catkin build** работает в любом месте рабочего пространства, не только в верхнеуровневой директории + можно собирать пакеты по отдельности, не только все сразу.

Обязательным требованием является наличие src папки внутри рабочего пространства! Если вы один раз использовали catkin_make то в дальнейшем нельзя будет просто воспользоваться **catkin build**!

Пример создания окружения в папке catkin_ws

```
mkdir catkin_ws
cd catkin_ws
mkdir src
catkin init
catkin build
```

Для справок по catkin: https://catkin-tools.readthedocs.io/en/latest/index.html

Справочник команд по catkin: https://catkin-tools.readthedocs.io/en/latest/cheat_sheet.html

После каждой сборки необходимо обновить файл запуска пакета `source devel/setup.bash`

`catkin config` - покажет параметры catkin workspace

1. Самый простой способ создания пакета, это скачать исходный код пакета в наш workspace сделаем клон репозитория и скопируем из папки `practice_1` папку `my_robot_controller`

`git clone https://gitlab.com/likerobotics/ros_course_2023.git`

Теперь нам предстоит его изучить и исправить в соответствии с заданием.

2. Исследование пакета

В нашем минимально жизнеспособном пакете мы увидим 2 папки и два файла:

- **scripts**
- **src**
- CMakeLists.txt
- package.xml

В папке *scripts* мы найдем *.py* файлы

3. Нас интересует файл my_node.py содержащий необходимый минимум
    a. окружение и версия python
    b. импорт библиотеки для работы python
    c. инициализация ноды
    d. задаем частоту обновления
    e. и запускаем бесконечный цикл средствами ROS

Таким образом, наш пакет содержит ноду, которая публикует в консоь сообщения с заданной частотой.

Если файл не является исполняемым, то вы получите сообщение об ошибке… тогда делаем этот файл исполняемым при помощи `chmod +x my_node.py`

4. Запускаем наш исполняемый файл `rosrun my_robot_controller my_node.py`

> 💡 **А все ли получилось?**

Теперь было бы здорово остановить его, сделать это можно через CTRL+C или при помощи команды `rosnode kill test_node_1`

Обратите внимание, что название исполняемого файла и название ноды могут не совпадать!

Для вывода информации в терминале мы моежем воспользоваться `rospy.loginfo("Информация я вывода в терминале")`

💡 Пункты 5 и 6 выполняются вместе с преподавателем!

4. **Создание Паблишера**

Создадим своего паблишера, который будет управлять черепашкой (публиковать сообщения “правильного” типа в “правильный” топик.

1. Создаем новый файл с названием my_publisher.py
2. Копируем из файла нашей ноды все, и меняем название ноды.
3. Импортируем библиотеку из пакета geometry_msgs, и из него нужный нам класс.
4. Все внешние пакеты и модули должны быть указаны в package.xml поэтому добавляем этот пакет туда + turtlesim который мы будем использовать.

```
<build_depend>geometry_msgs</build_depend>
<build_depend>turtlesim</build_depend>


<exec_depend>turtlesim</exec_depend>
<exec_depend>geometry_msgs</exec_depend>
```

5. перед входом в бесконечный цикл создаем объект класса Publisher, передаем в него название топика, в который будет публиковать, тип сообщений и размер очереди.

   `pub = rospy.Publisher("/turtle1/cmd_vel", Twist, queue_size=10)`
6. также создаем экземпляр класса Twist, который по сути будет нашим сообщением, который и будем отправлять в топик.

   `msg = Twist()`
7. Внутри бесконечного цикла мы можем менять значение параметров экземпляра класса и публиковать его с частотой, который мы задали в Rate.

   `pub.publish(msg)`

5. **Создание Сабскрайбера**

   a. При создании сабскрайбера используется немного другой механизм, тут мы не входим в бесконечный цикл, вместо этого используем rospy.spin()

   b. Создаем новый файл my_subscriber.py

   c. Создаем функцию callback_function принимает на вход один параметр msg и выводит в лог консоли.

   `def pose_callback(msg):`

   …..

   d. В инициализации __main__ меняем название ноды на pose_subscriber

e. После этого создаем экзепляр класса Subscriber и передаем туда название топика, который будем слушать, тип сообщений, и указываем callback функцию.

```
subscriber = rospy.Subscriber("/turtle1/pose", Pose, callback=pose_callback)
```

f. Чтобы callback вызывался постоянно, необходимо указать `rospy.spin()`

Запускаем в отдельных консолях Паблишер и Сабскрайбер.

Запускаем еще в отдельной консоли rqt_graph и посмотрим на наши ноды и топики.

6. **Немного про типы сообщений**

У нас в ROS уже установлен пакет `std_msgs` в котором имеются примитивные типы сообщений, которыми тоже можно пользоваться.

Задание

Вам необходимо исправить содержимое пакета таким образом, что в ней будет только 2 ноды (их названия указаны в таблице), которые отправляют по одному числу в другую ноду (в таблице последний столбик), который уже публикует результат в топик, название которого состоит из слова “result_” + ваш ID в ISU.

Номера вариантов для студентов соответствуют последней цифре ISU ID:

Последняя цифра ису ID	node1	node2	node3	операция	Результат вычисляется в
1	+	+		+	node3
2	+		+	+	node2
3		+	+	+	node1
4	+	+		-	node3
5	+		+	-	node2
6		+	+	-	node1
7	+	+		*	node3
8	+		+	*	node2
9		+	+	*	node1
0	+	+		/	node3

Как отправить свое решение на проверку?

1. Зарегистрируйтесь на gitlab
2. Сделать форк репозитория ros_course_2023
3. сделайте этот форк приватным

4. добавьте в форк контрибютора `@likerobotics` и задайте для него роль `Maintainer`
5. После выполнения своего задания скопируйте свое решение в папку `practice_1` и сделайте коммит, а затем отправьте свое решение на gitlab при помощи `git push`.

> 💡 Если вы используете несколько веток, делайте пожалуйста коммиты в деволтную ветку.