Student ID:		**Lab Section:**	
Name:		**Lab Group:**	

Software Lab 2

ROS Packages and Topics

Topic Overview

This lab aims to acquaint students with the fundamentals of ROS packages and topics. It covers the concepts from building a basic *package* to running nodes in that *package* containing a simple ROS *topic*. This lab walks the students through those concepts utilizing the Husky simulator.

Learning Outcome

After this lecture, the students will be able to:

a. Know the concepts of workspace, package, and node.
b. Create and build a ROS workspace.
c. Write Python code for subscriber and publisher to a topic.
d. Run a ROS environment in Gazebo using the Husky simulator.
e. Control the Husky bot using a Python script.

Prepared by Shayekh Bin Islam and Mir Sayeed Mohammad

2.1 ROS Package

Packages are the software organization unit of ROS code. Each package can contain libraries, executables, scripts, or other artifacts. A **manifest** is a description of a *package*. It defines dependencies between packages and captures meta-information about the *package* like version, maintainer, license, and so on. The recommended method of working with catkin packages is using a catkin **workspace**. The package management (such as package building) tool for ROS is called **catkin**.

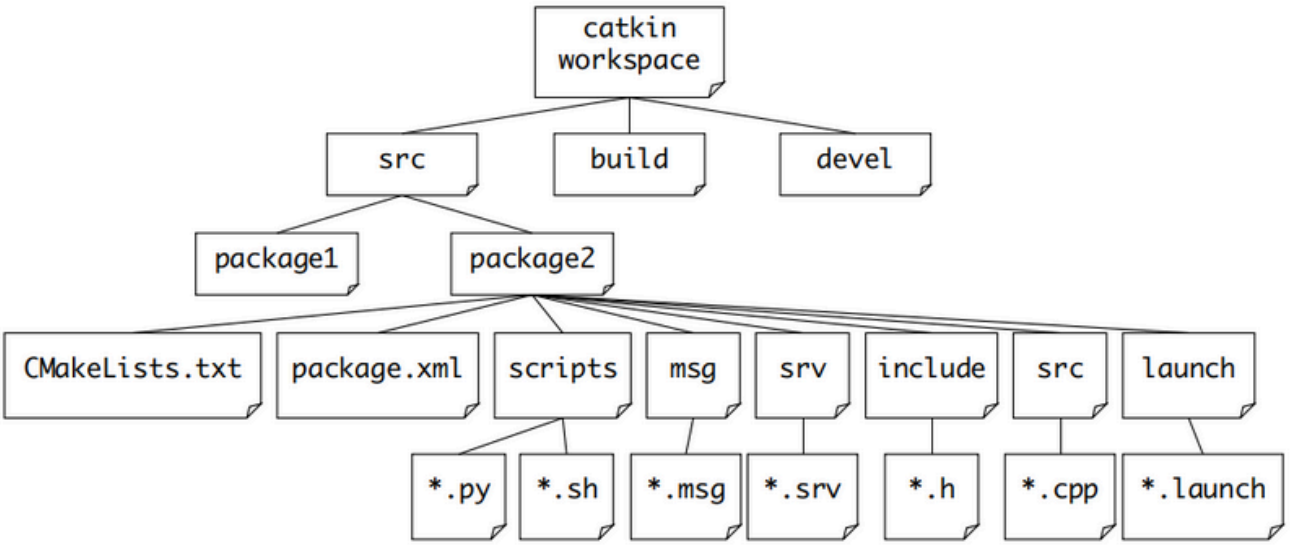


Figure: Organization inside a catkin workspace. Package source codes are organized under the src directory.

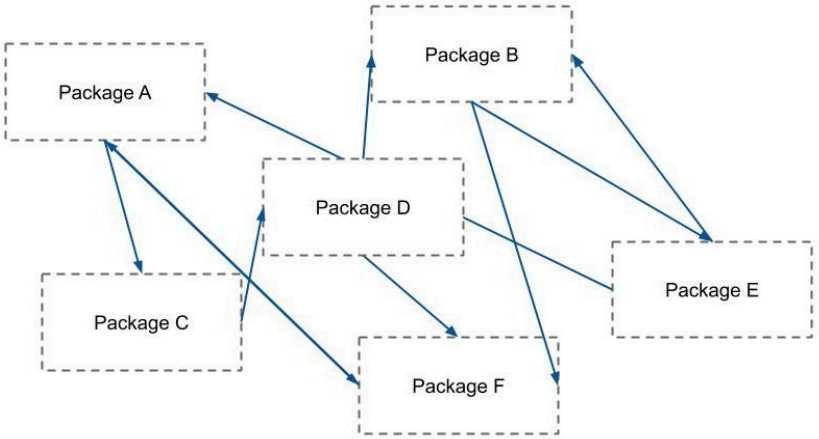


Fig: Packages and dependencies - different packages can depend on libraries contained within each other.

2.1.1 Creating a Catkin Package

For creating a package we need to define a workspace. We keep our workspace in the user home directory (~).

```
ubuntu@ubuntu2004:~$ cd ~
ubuntu@ubuntu2004:~$ mkdir -p catkin_ws/src
```

We keep our packages inside the workspace ~/catkin_ws. Inside the ~/catkin_ws/src folder, the packages are located. To create a package, we change the directory to the ~/catkin_ws/src folder.

```
ubuntu@ubuntu2004:~$ cd ~/catkin_ws/src/
ubuntu@ubuntu2004:~/catkin_ws/src$
```

Now we create a package named topic_demo. Notice that this package depends on rospy and std_msgs (standard libraries for most tasks in ROS).

```
ubuntu@ubuntu2004:~/catkin_ws/src$ catkin_create_pkg topic_demo rospy
std_msgs

# catkin_create_pkg <package_name> [depend1] [depend2] [depend3] ...
```

We see the following directory structure is created.

```
# Before
ubuntu@ubuntu2004:~/catkin_ws$
.
└── src

1 directory, 0 files
```

```
# After
ubuntu@ubuntu2004:~/catkin_ws$
.
└── src
    └── topic_demo
        ├── CMakeLists.txt
        ├── package.xml
        └── src
3 directories, 2 files
```

Now, we have to build the workspace. We move to the base directory of the workspace and run the build command.

```
ubuntu@ubuntu2004:~/catkin_ws/src$ cd ~/catkin_ws/
ubuntu@ubuntu2004:~/catkin_ws$ catkin_make
Base path: /home/ubuntu/catkin_ws
Source space: /home/ubuntu/catkin_ws/src
Build space: /home/ubuntu/catkin_ws/build
Devel space: /home/ubuntu/catkin_ws/devel
Install space: /home/ubuntu/catkin_ws/install
Creating symlink "/home/ubuntu/catkin_ws/src/CMakeLists.txt" pointing to
"/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
####
#### Running command: "cmake /home/ubuntu/catkin_ws/src
-DCATKIN_DEVEL_PREFIX=/home/ubuntu/catkin_ws/devel
-DCMAKE_INSTALL_PREFIX=/home/ubuntu/catkin_ws/install -G Unix Makefiles" in
"/home/ubuntu/catkin_ws/build"
####
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
```

```
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/ubuntu/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/noetic
-- This workspace overlays: /opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is
"3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Found PY_em: /usr/lib/python3/dist-packages/em.py
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/ubuntu/catkin_ws/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Found gmock sources under '/usr/src/googletest': gmock will be built
-- Found PythonInterp: /usr/bin/python3 (found version "3.8.10")
-- Found Threads: TRUE
-- Using Python nosetests: /usr/bin/nosetests3
-- catkin 0.8.10
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~ traversing 1 packages in topological order:
-- ~~ - topic_demo
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'topic_demo'
-- ==> add_subdirectory(topic_demo)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ubuntu/catkin_ws/build
####
#### Running command: "make -j2 -l2" in "/home/ubuntu/catkin_ws/build"
####
```

Now, we have to configure the terminal so that ROS is aware of our package every time we open a new terminal.

```
ubuntu@ubuntu2004:~/catkin_ws$ source ./devel/setup.bash
ubuntu@ubuntu2004:~/catkin_ws$ echo "source $PWD/devel/setup.bash" >> $HOME/.bashrc
```

2.1.2 Creating a ROS Publisher Node

We create a file at **~/catkin_ws/src/topic_demo/src/publisher.py**

```
#!/usr/bin/python3
# ^^ The line above mentions the Python interpreter for this script.
# license removed for brevity
import rospy # For creating the publisher
from std_msgs.msg import String # For the message

def publisher():
   # This node publishes to 'topic_test' using message type 'String'
   pub = rospy.Publisher('topic_test', String, queue_size=10)
   # Register the name of the node
   # 'anonymous' means we allow more than one node using this script
   rospy.init_node('publisher', anonymous=True)
   rate = rospy.Rate(10) # The following loop runs at 10 Hz
   while not rospy.is_shutdown():
       hello_str = "hello world %s" % rospy.get_time()
       rospy.loginfo(hello_str) # Print the message to screen/logs
       pub.publish(hello_str) # Publish the message now
       rate.sleep() # Sleep to maintain the Rate (Here 10 Hz)

if __name__ == '__main__':
   try:
       publisher()
   except rospy.ROSInterruptException:
       pass # Exit when the node is shutdown
```

Then, after making the script executable, we can run the publisher node. Notice that in ROS, nodes are uniquely named.

```
ubuntu@ubuntu2004:~/catkin_ws/src/topic_demo/src$ chmod +x publisher.py
ubuntu@ubuntu2004:~/catkin_ws$ rosrun topic_demo publisher.py
[INFO] [1711179043.967943]: hello world 1711179043.967851
[INFO] [1711179044.068741]: hello world 1711179044.068594
[INFO] [1711179044.168727]: hello world 1711179044.1682181
...
```

We can observe that our new node and topic are active. If two nodes with the same name are launched, the previous one is kicked off. The `anonymous=True` flag means that `rospy` will choose a unique name (e.g. **publisher_18124_1711179100928**) for our 'publisher' node so that multiple publishers can run simultaneously. To see the list of active nodes and topics, run the following in a new terminal window.

```
ubuntu@ubuntu2004:~/catkin_ws$ rosnode list
/publisher_18124_1711179100928
/rosout
ubuntu@ubuntu2004:~/catkin_ws$ rostopic list
/rosout
/rosout_agg
/topic_test
```

2.1.3 Creating a ROS Subscriber Node

Similarly, we create a file at **~/catkin_ws/src/topic_demo/src/subscriber.py**

The final directory structure looks like this:

```
├── devel
│   └── setup.bash
└── src
    ├── CMakeLists.txt
    └── topic_demo
        ├── CMakeLists.txt
        ├── package.xml
        └── src
            ├── publisher.py
            └── subscriber.py

47 directories, 98 files
```

```
#!/usr/bin/python3
# ^^ The line above mentions the Python interpreter for this script.
# license removed for brevity
import rospy # For creating the subscriber node
from std_msgs.msg import String # For the message

def callback(data): # Call this function, when message received
    rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)

def subscriber():
    rospy.init_node('subscriber', anonymous=True)
    # When a message is received, the callback function is called.
    rospy.Subscriber('topic_test', String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    subscriber()
```

```
ubuntu@ubuntu2004:~/catkin_ws/src/topic_demo/src$ chmod +x subscriber.py
ubuntu@ubuntu2004:~/catkin_ws$ rosrun topic_demo subscriber.py
[INFO] [1711179870.267201]: /subscriber_18983_17111798639481 heard hello world 1711179870.2639139
[INFO] [1711179870.367688]: /subscriber_18983_17111798639481 heard hello world 1711179870.3649414
[INFO] [1711179870.468510]: /subscriber_18983_17111798639481 heard hello world 1711179870.464787
```

The *rqt_graph* of the system:

2.2 The Task - Driving *Husky*

1. Create a workspace named `CSE461_<section#>_<group#>` (e.g. `CSE461_07_03`).
2. Create a package named `husky_driver`
3. Launch the Husky node. Launch files are very common in ROS to both users and developers. They provide a convenient way to start up multiple nodes and a master, as well as other initialization requirements such as setting parameters. **`roslaunch`** is used to open launch files.

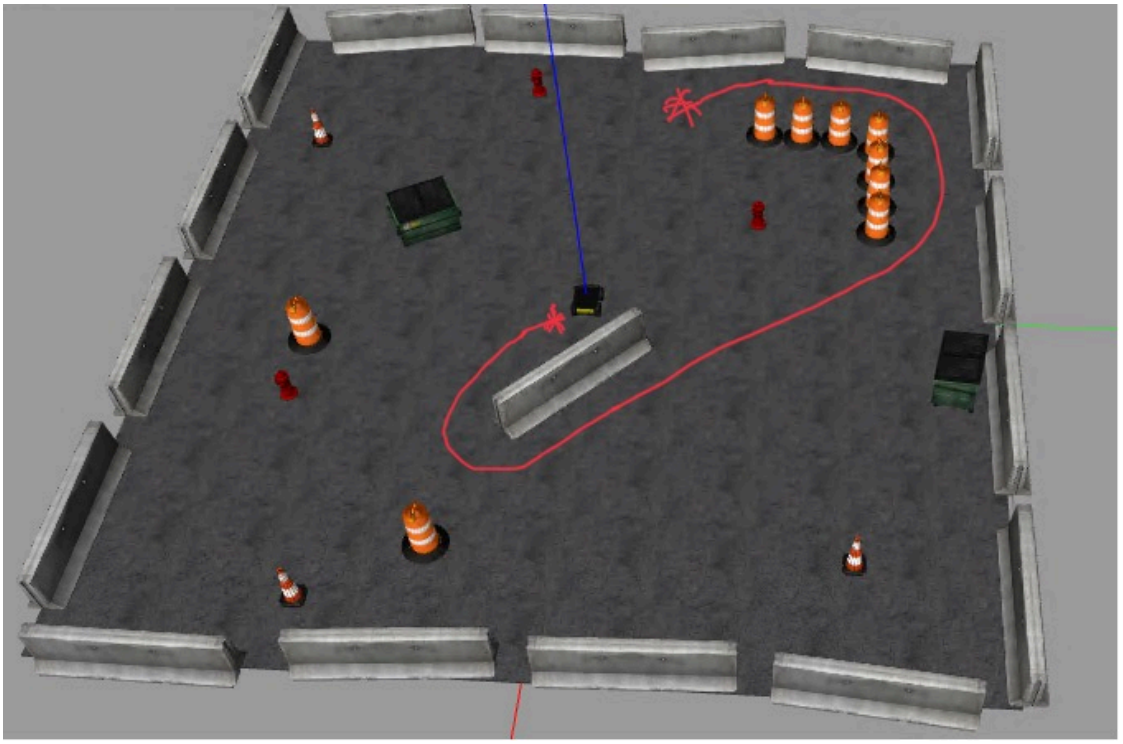
```
ubuntu@ubuntu2004:~/catkin_ws$ roslaunch husky_gazebo husky_playpen.launch
```

4. Create a node named **`follow_path.py`**
5. Modify the following code so that the Husky follows the given path without any collision with any objects.

```
#!/usr/bin/python3
import rospy
from geometry_msgs.msg import Twist

def move():
   # Starts a new node
   rospy.init_node('follow_path', anonymous=True)
   velocity_publisher = rospy.Publisher('/husky_velocity_controller/cmd_vel',
                                        Twist, queue_size=10)
   vel_msg = Twist()
   # Receiveing the user's input
   print("Let's move your robot")
   speed = input("Input your speed:")
   distance = input("Type your distance:")
   isForward = input("Foward?: ") # True or False
   speed = float(speed)
   distance = float(distance)
   isForward = int(isForward)

  # Checking if the movement is forward or backward
   if(isForward):
       vel_msg.linear.x = abs(speed)
   else:
       vel_msg.linear.x = -abs(speed)
   # Since we are moving just in the x-axis
   vel_msg.linear.y = 0
   vel_msg.linear.z = 0
   vel_msg.angular.x = 0
   vel_msg.angular.y = 0
   vel_msg.angular.z = 0

   while not rospy.is_shutdown():
       # Setting the current time for distance calculation
       t0 = rospy.Time.now().to_sec()
       current_distance = 0
       # Loop to move the turtle in a specified distance
       while(current_distance < distance):
```

```
            # Publish the velocity
            velocity_publisher.publish(vel_msg)
            # Takes actual time for distance calculation
            t1 = rospy.Time.now().to_sec()
            # Calculates distancePoseStamped
            current_distance = speed*(t1-t0)
        # After the loop, stop the robot
        vel_msg.linear.x = 0
        # Force the robot to stop
        velocity_publisher.publish(vel_msg)

if __name__ == '__main__':
    try: move()
    except rospy.ROSInterruptException: pass
```

The Path to Follow:



Resources

1. https://app.theconstruct.ai/Desktop Free virtual machine online hosted by The Construct. Very useful for running experiments out of the box without the hassle of installing ROS on your own system. They also have an extensive list of tutorials and courses on ROS. https://app.theconstruct.ai/courses/
2. ROS official tutorials: https://wiki.ros.org/ROS/Tutorials
3. Clearpath Robotics: http://www.clearpathrobotics.com/assets/guides/noetic/ros/index.html