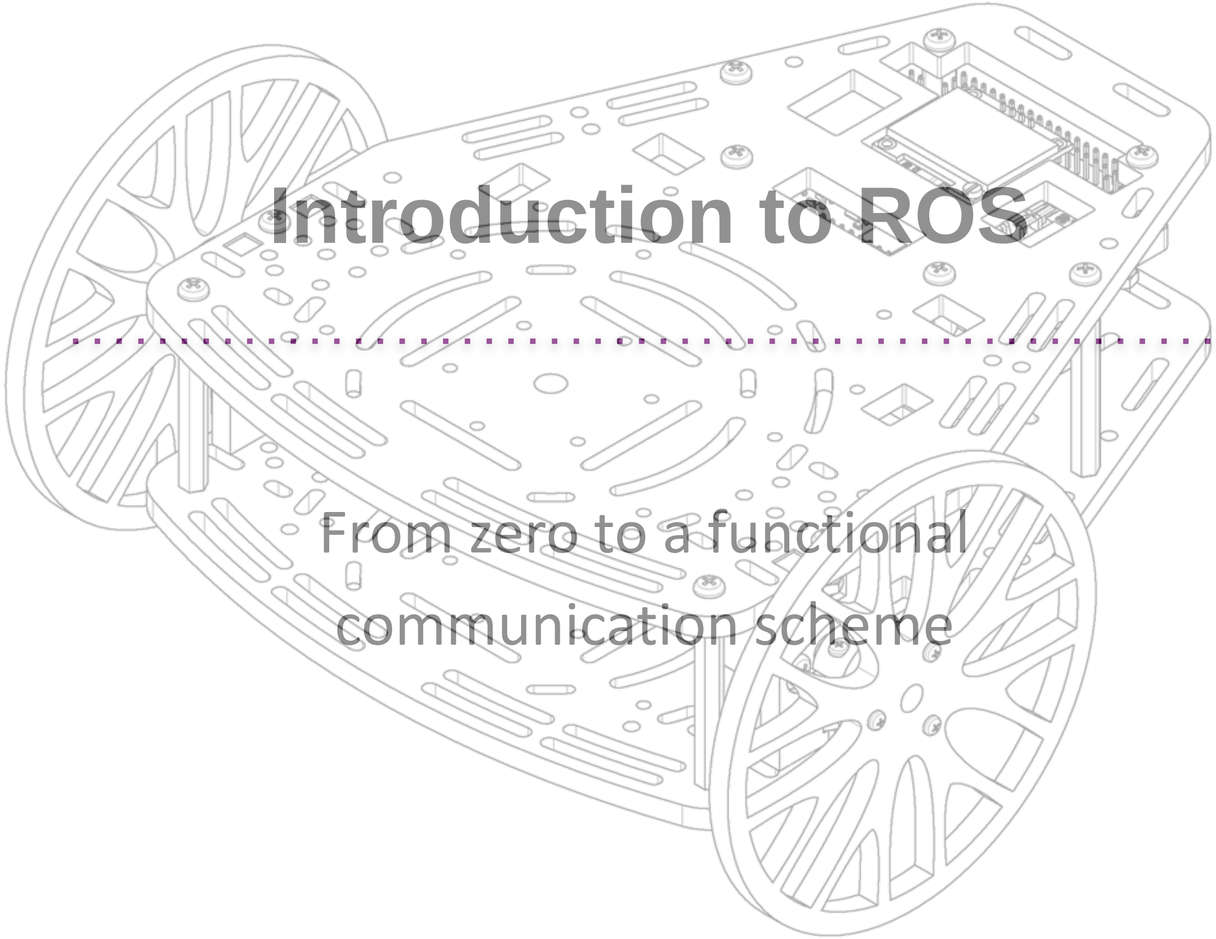
Introduction to ROS

From zero to a functional communication scheme

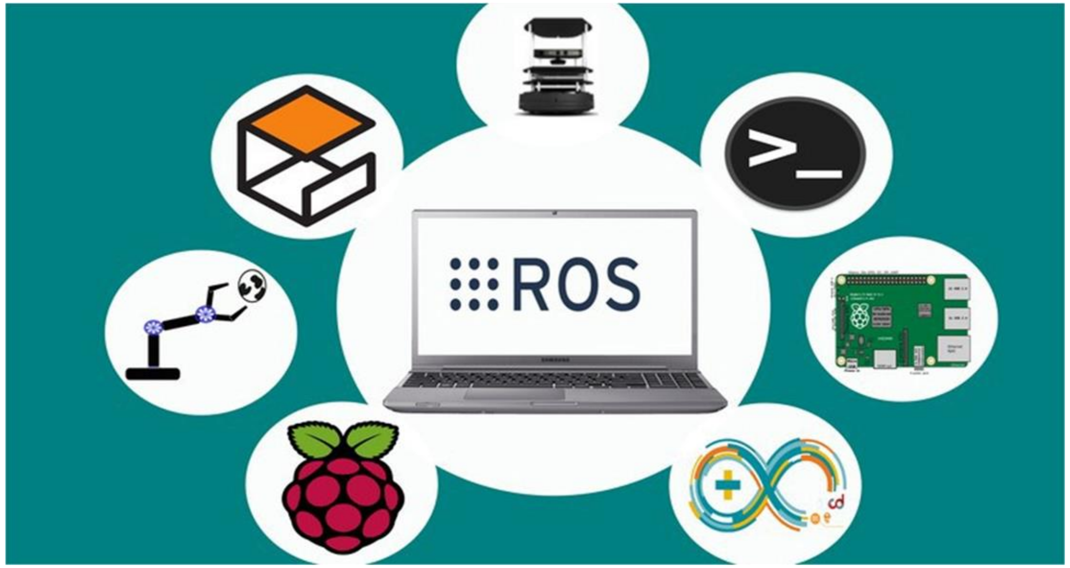


Schedule

- Day 1 content
  - What is ROS
  - How is ROS communication done
  - Terminal tools
  - Talker and listener
  - ROS Launch
- Day 2 content
  - More advanced ROS concepts
  - Activity
  - Questions

ROS basics

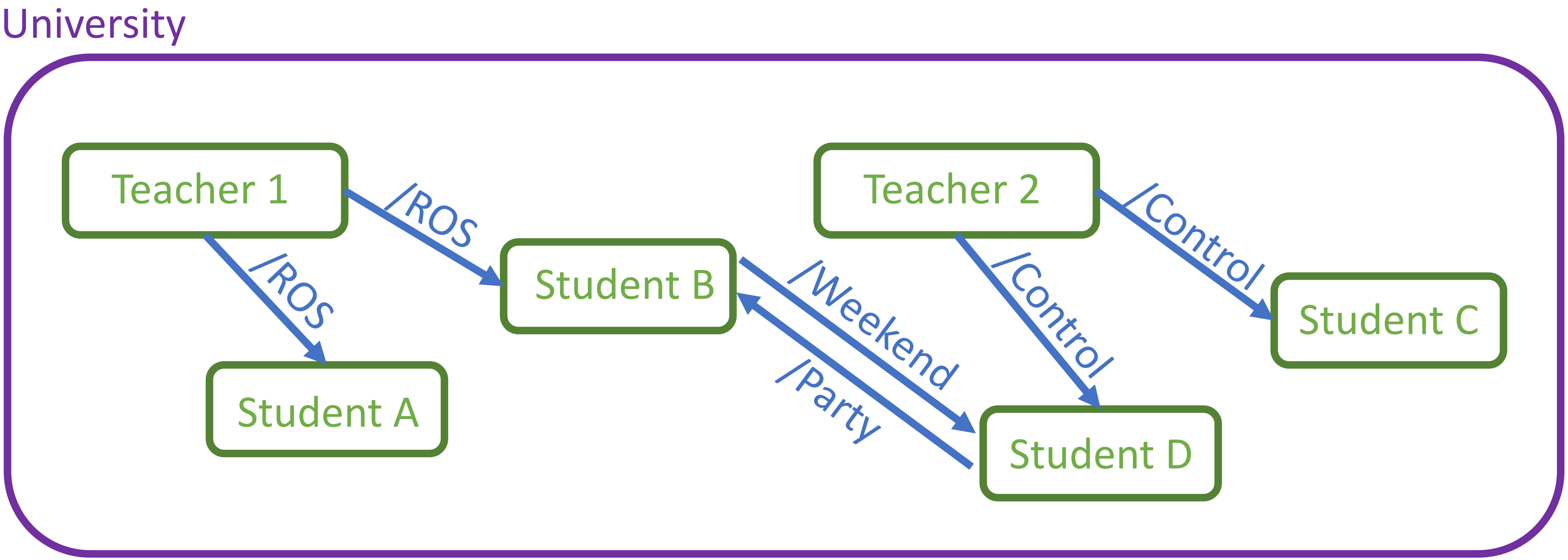
What is ROS ?



"ROS is an open-source framework that helps researchers and developers build and reuse code between robotics applications"

ROS basics

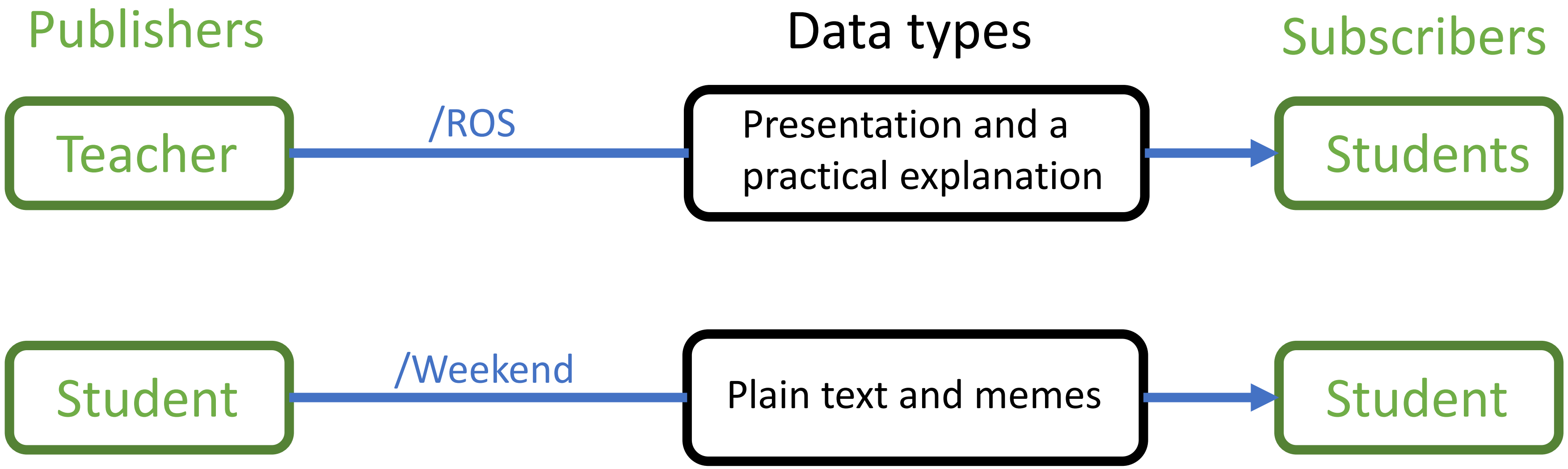
The conceptual idea behind ROS



ROS basics

What is the information delivered?

- Any class has a certain format, which both the teacher and the student know off and is expected to be followed.
- Between two friends you are not expecting a power point presentation but some plain text.



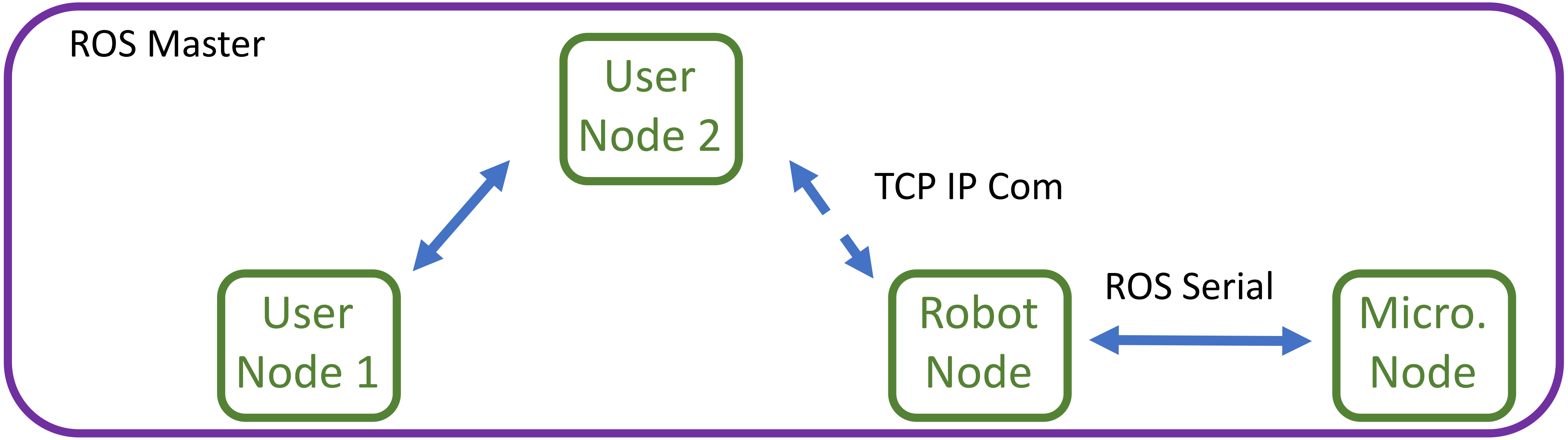
ROS basics

ROS versions and installation

- New versions of ROS are released with each Linux distributions, due to compatibility in this course we are going to be working with ROS melodic (released with Ubuntu 18.04).
- Currently there's another version of ROS available for Ubuntu 20.04 and a revision of the ROS structure, known as ROS2, that aims to increase the robustness of the framework for industrial applications and distributed systems. Furthermore, ROS2 allows real time applications.

ROS Architecture
ROS Master

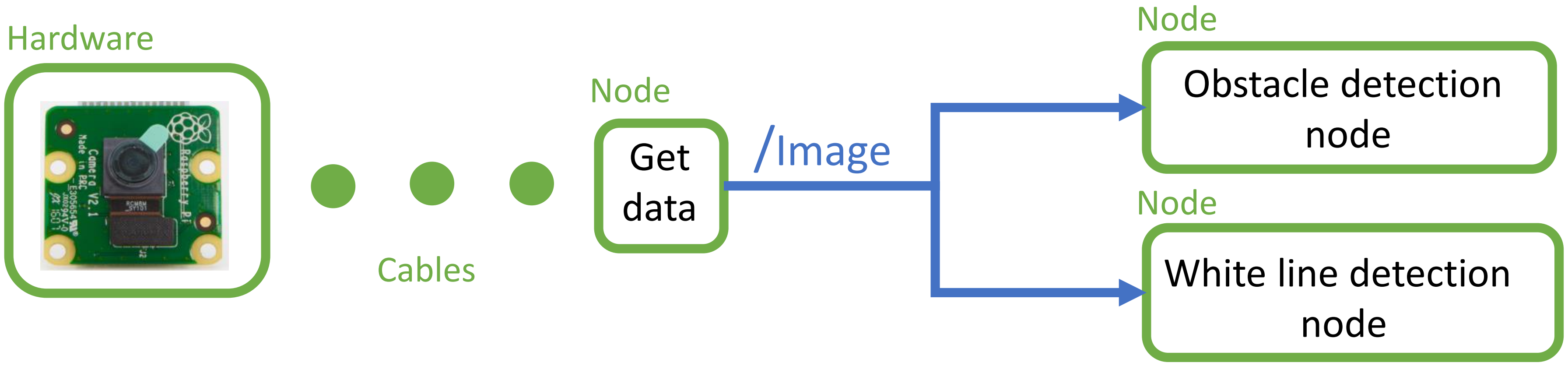
- Brain of ROS -> Process manager that enables the communication between different agents in the network.
- Allows communication between different computers or robots by means of a Server – Client architecture



ROS Architecture

nodes and topics

- Piece of software that acts as an element in the network.
- It is in charge of executing a part of the code and can be programmed in Cpp, Python or Lisp.
- Each topic has a unique msg type.



ROS Architecture
messages

std_msgs/Float32

float 32 data

sensor_msgs/Image

std_msgs/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data

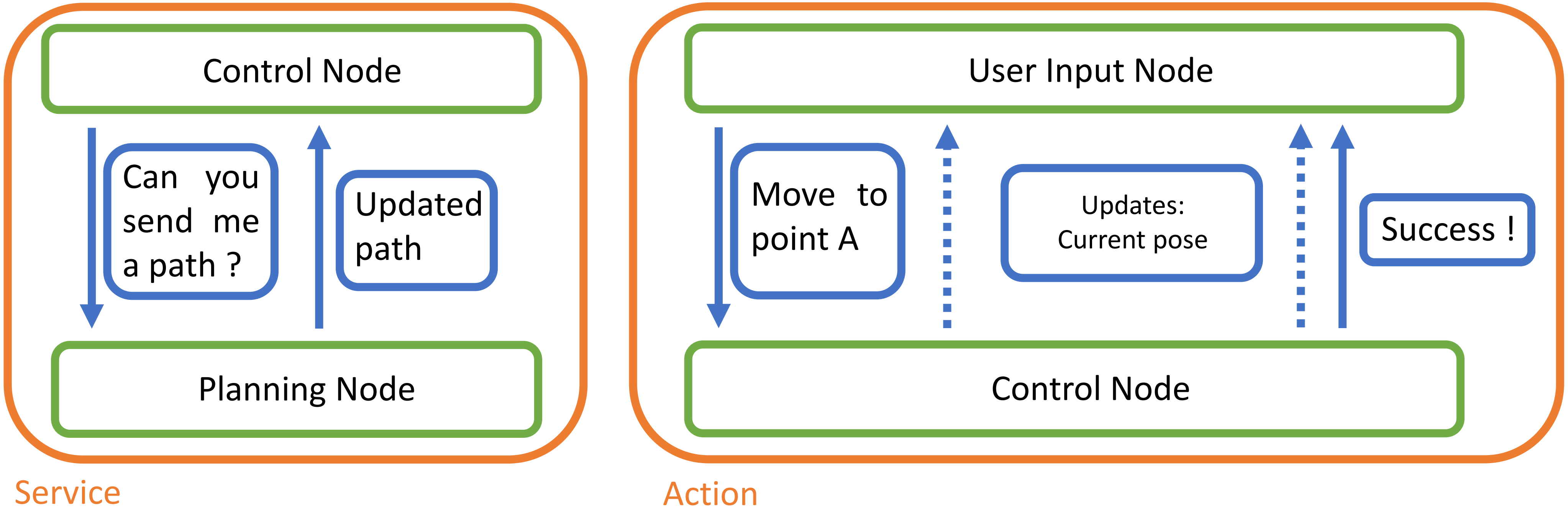
geometry_msgs/PoseStamped

std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
geometry_msgs/Pose pose
    geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
    geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w

ROS Architecture

Services and actions

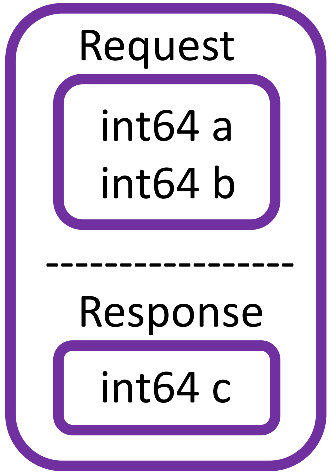
- While topics are one way communication, services and actions allow feedback and prevent unnecessary load in the network.
- Services follow a question-answer structure, while actions are designed to execute a task and give feedback during the process.



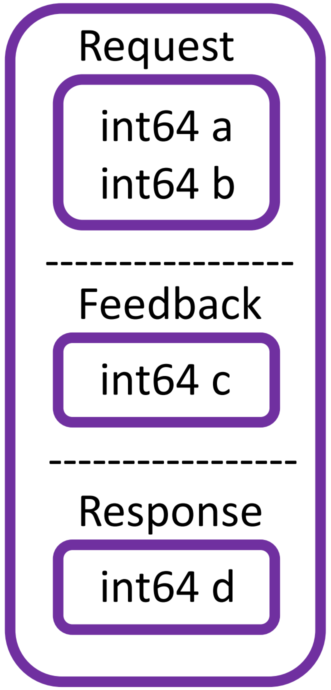
ROS Architecture
Services and actions

- When defining either an action or a service we need to stablish each member involved in the communication. Usually both are project dependent, so users define their own structures.

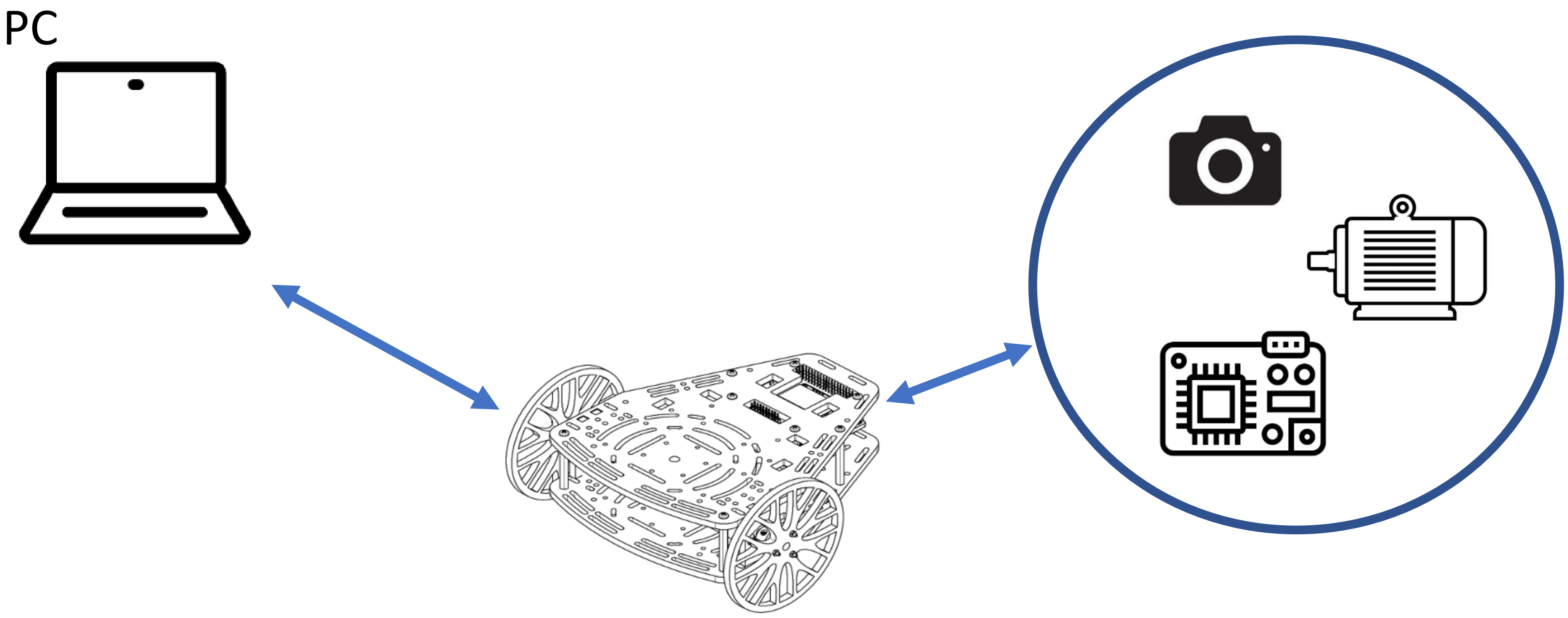
Service



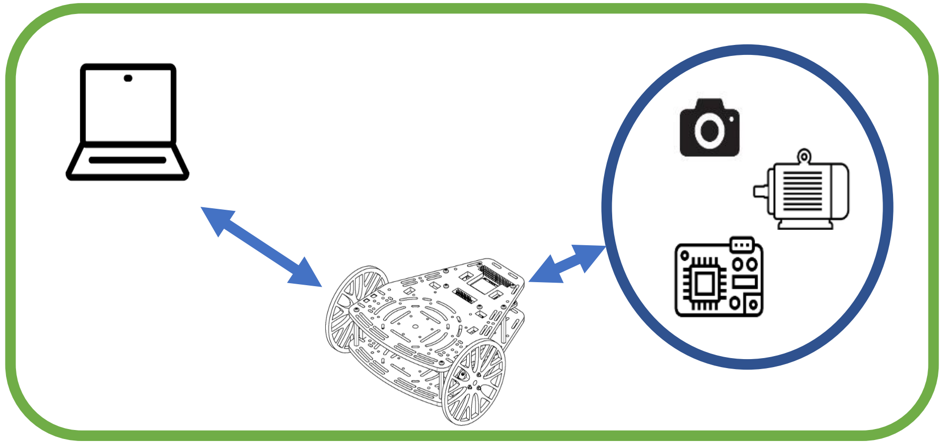
Actions



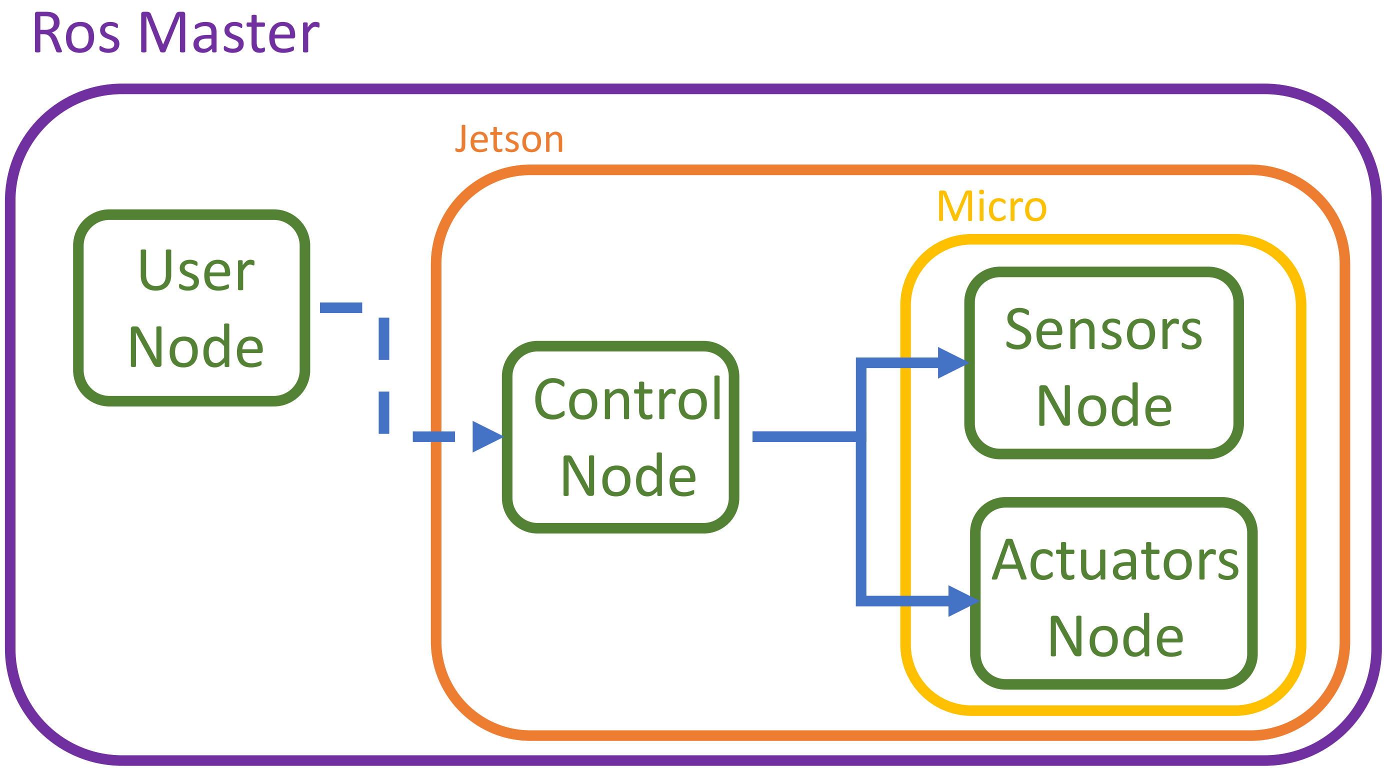
ROS Architecture
A practical example



ROS Architecture
Overview



Physical
System

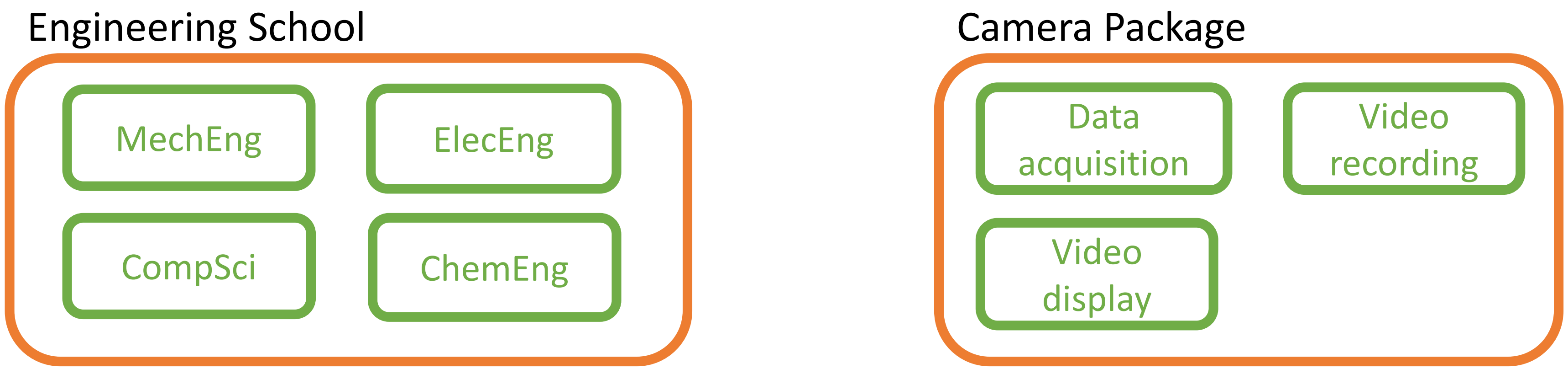


ROS Implementation

ROS Architecture

ROS Packages

- Ros Packages are a way of organizing codes related between each other.
- The same way teachers are gathered within the Engineering school nodes are gathered in ROS packages



ROS Code
Talker and listener framework

Simplest communication scheme.

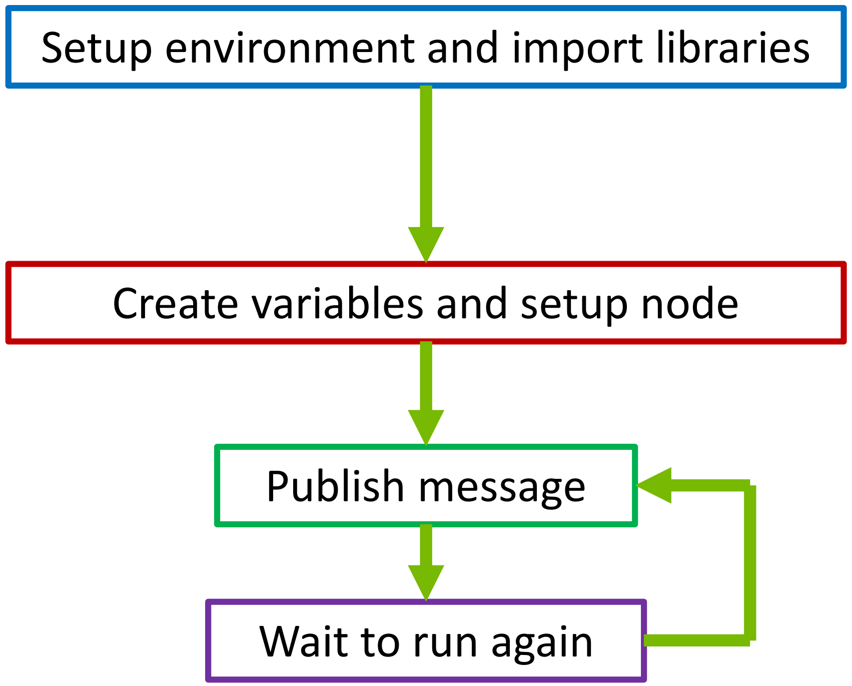
ROS Code
Talker


```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

if __name__ == '__main__':
    pub = rospy.Publisher("chatter", String, queue_size=10)
    rospy.init_node("talker")
    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        pub.publish(hello_str)

        rate.sleep()
```



Terminal Commands

```
roscore
rosrun basic_comms talker.py
rostopic echo /chatter
```

ROS Code
Listener


```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(msg):
    rospy.loginfo("I heard %s", msg.data)

if __name__ == '__main__':
    rospy.init_node('listener')
    rospy.Subscriber("chatter", String, callback)

    rospy.spin()
```

Setup environment and import libraries

↓

Define Callback

↓

Create variables and initialise node

↓

Continuously check for new messages

Terminal Commands

```
roscore
rosrun basic_comms listener.py
rostopic pub /chatter <tab complete>
```

ROS Tools
ROS Terminal tools

- roscore
  - Must be executed before working with ros
  - Handles the correct functionality of the network
- rostopic
  - Contains useful tools related with topics
  - *rostopic echo [topic name]*
  - *rostopic list*
  - *rostopic hz [topic name]*
- rosrun
  - Executes a given node
  - *rosrun [package] [node name]*
- rosnode
  - Contains useful tools related with nodes
  - *rosnode list*
  - *rostopic info [node]*

ROS Tools

ROS Terminal tools II

- roslaunch
  - Allows to launch multiple nodes with a single command by calling .launch files
  - *roslaunch [package] [.launch]*
- rosbag
  - Records the activity of the topics in the network
  - rosbag record
  - *rosbag info [bag file]*
  - *rosbag play [bag file]*
- config files
  - Load parameters into the system that can be accessed by any node
- rqt
  - ROS visualization tool that provides graphical information of the system status

Activity

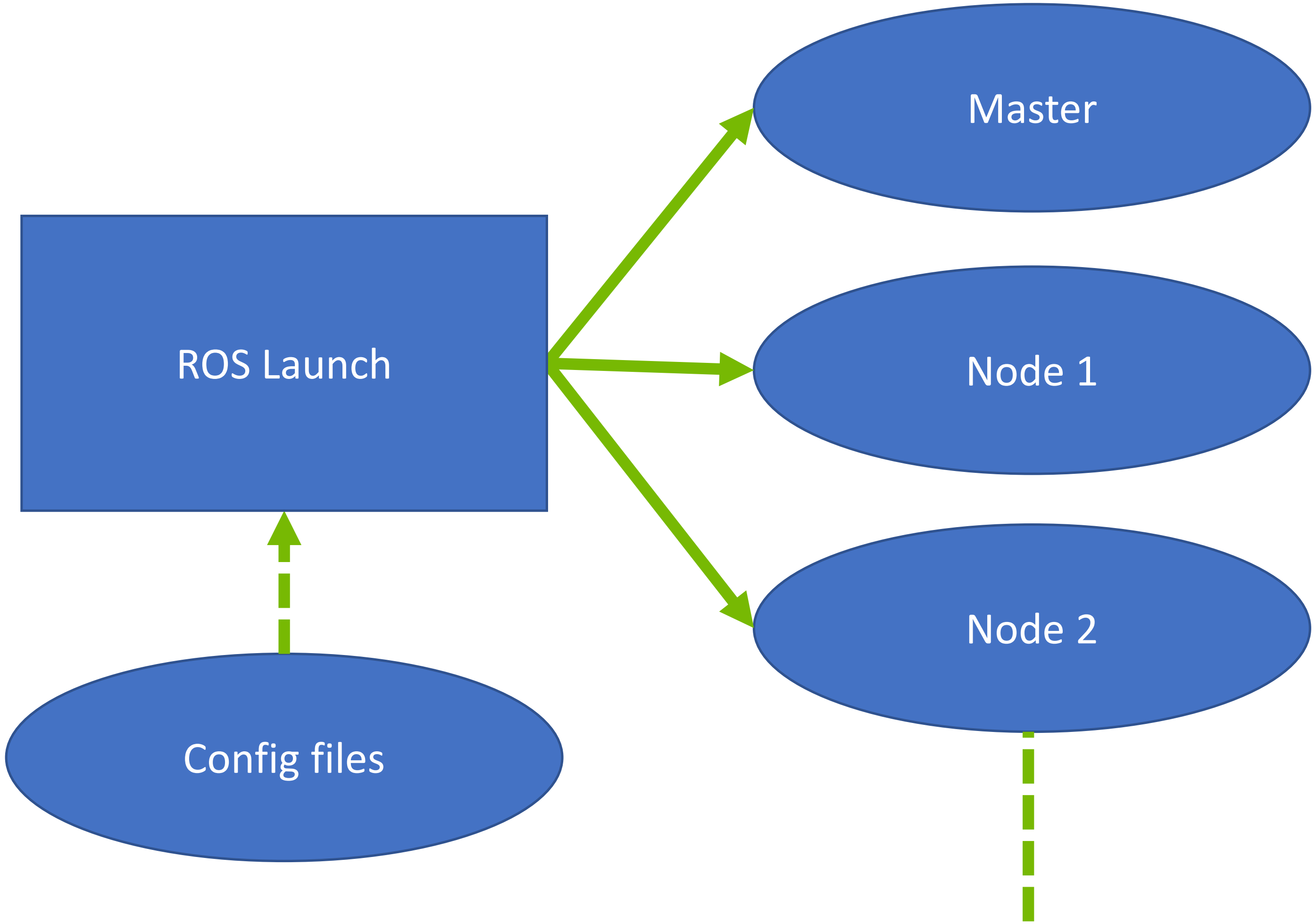
Implement a listener-talker

- Implement the code in either Python or Cpp.
- Use some of the command line tools to verify the correct functioning of the system.

ROS Tools
ROS Parameter Server

- ROS allows to load variables into the master which are accessible by all the nodes of the system.
- Typically, a .yaml file is loaded into the system through a launch file and configuration parameters are stored there.
- They are used them when loading robot models or constants that may vary in different scenarios where the same code is applied.
- You will see an example of them in further days of the course

ROS Tools
ROS Launch



ROS Tools
ROS Launch Syntax

- Launch files are sets of commands written in xml that allow executing various scripts at the same time.
- The general syntaxis is the following

```
<?xml version="1.0"?>
<launch>
    [Body of the launchfile]
</launch>
```

- This syntaxis allows to run any object used within the ROS architecture and has a wide variety of tools that allow to parametrize the launch file so that it can be adapted to the requirements of you project.
- An extensive documentation can be found in http://wiki.ros.org/roslaunch

ROS Tools
ROS Launch code tools

- Running a node

```
<node name=“listener” pkg=“basic_comms” type=“listener.py” output=“screen”/>
```

- Running another file or launch file

```
<include file="$(dirname)/other.launch" />
```

- Set parameters

```
<param name="publish_frequency" type="double" value="10.0" />
```

- Pass args to the launch file

```
<arg name=”camera_id" value=”cam_3" />
```

- Load files into the system

```
<rosparam command="load" file="$(find package_name)/config/file_name.yaml" />
```

ROS Tools

ROS File structure

- ROS projects are organized using workspaces, which are a collection of grouped codes called packages.
- Instructions for the compiler need to be allocated in .cmake and package files

```
example_workspace
├── build
├── devel
└── src
    ├── example_package1
    │   ├── CMakeLists.txt
    │   ├── package.xml
    │   └── src
    │       └── my_code.py
    └── example_package2
        ├── CMakeLists.txt
        ├── package.xml
        └── src
            └── my_code2.py
```

- Package files are exportable between projects
- Configuration files used to stablish code dependencies
- Code that we will execute
- Catkin_make will generate 'src' and 'devel' for you when compiling the workspace

ROS Tools

ROS Compilation tools

- ROS requires to compile each package, generating dependencies related with other packages, external libraries or custom messages, services and actions.
- The preferred compilation tool is known as `catkin` and uses two separated files, `package.xml` and `CMakeLists.txt`.
- A useful command to create packages is:

  ```
  catkin_create_pkg [name] [list of dependencies]
  ```

- Which generates an empty package and templates of both CMakeLists and package files.
- More information about the syntaxis of this files can be found In http://wiki.ros.org/ROS/Tutorials/CreatingPackage
- Projects are compiled with the instruction `catkin_make` and the instruction `source [project]/devel/setup.bash` needs to be executed to load the changes.

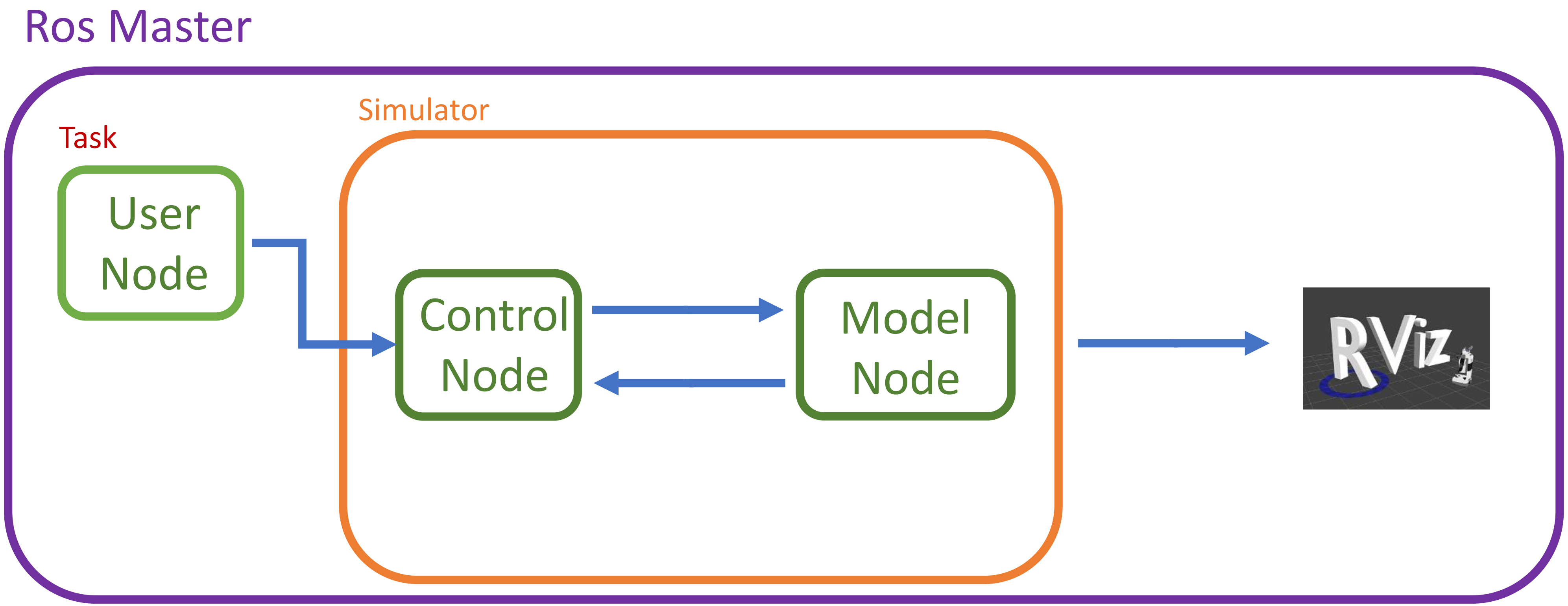
Example
Using custom messages

1. Create a folder called `msg` in your workspace.
2. Make a custom msg definition with a Header and a Float32.
3. Open both CMakeLists and package.xml and modify them to compile our new msg.
4. Modify the code of the previous activity using the new msg definition.

package/time_msg.msg

```
String date
String hour
```

Activity
Problem architecture

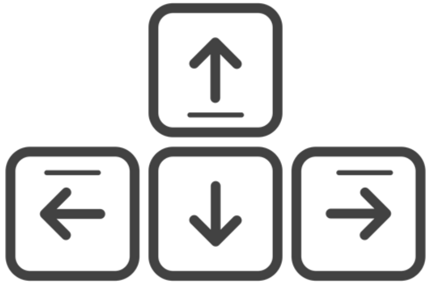


ROS Implementation

Activity

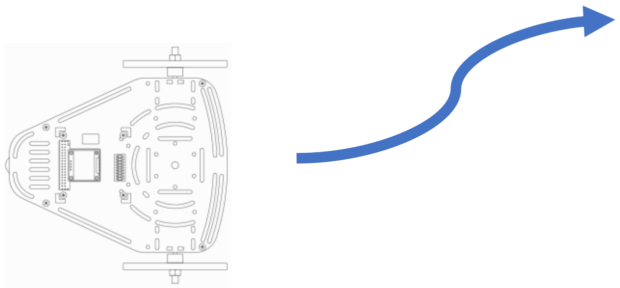
Teleoperate a PuzzleBot

Teleoperation



Input in the terminal the commands to move the robot

Square

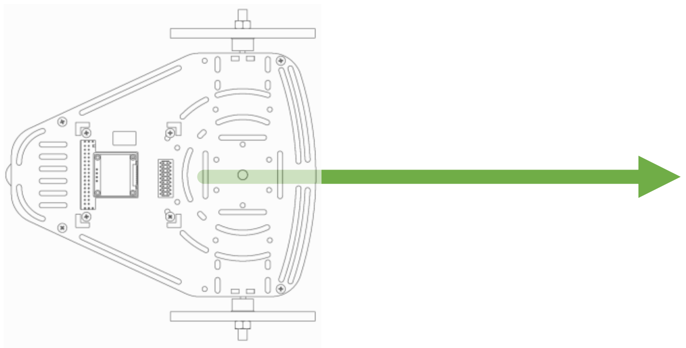


See how the robot traverses accordingly the environment.

Activity

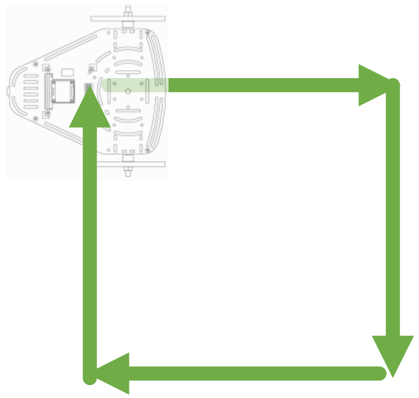
Moving a PuzzleBot

Straight line



Drive the robot in a straight line.

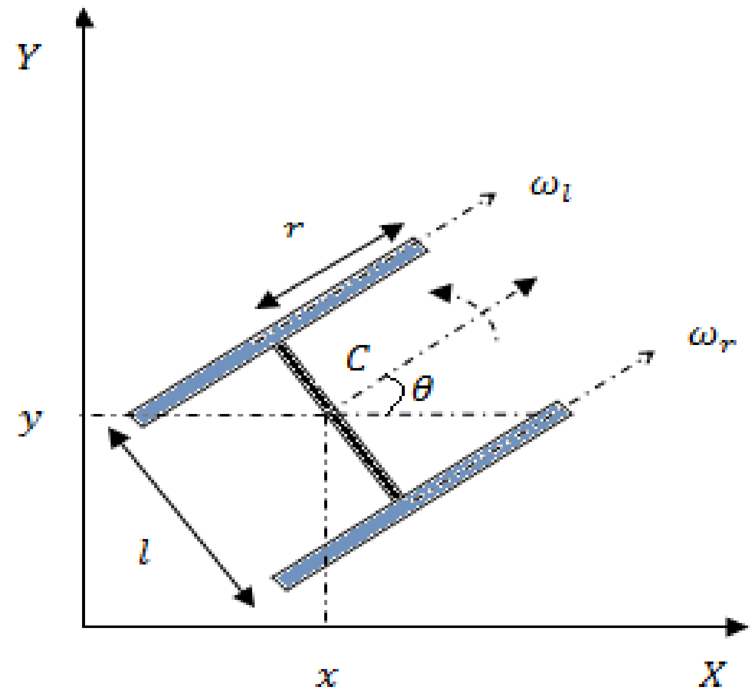
Square



Drive the robot making a square with a side length of 1m.

Activity

How is the robot modelled?



- The resultant forward velocity through C (the centre of mass) is $r(\frac{\omega_r+\omega_l}{2})$.
- The steering velocity is $r(\frac{\omega_r-\omega_l}{l})$.

Activity
Some tips and tricks

- One publisher `cmd_vel`
- `cmd_vel` – `from geometry_msgs.msg import Twist` – 3 linear and 3 angular velocities.
  - `msg.linear.x`, `msg.linear.y`, `msg.linear.z`
  - `msg.angular.x`, `msg.angular.y`, `msg.angular.z`
- Use the equations below to compute the distance moved and the angle turned by the robot

$$d = r\left(\frac{\omega_r + \omega_l}{2}\right)dt$$

$$\theta = r\left(\frac{\omega_r - \omega_l}{l}\right)dt$$

where r is the radius of the wheels (=0.05m) and l is the distance between the wheels (=0.18m)

- Use `rospy.get_time()` to measure the time dt between each loop
- If the robot is not moving, check your topics with `rostopic echo` and `rostopic pub`
- Ensure your python file is executable: `sudo chmod +x <path_to_file>.py`