

Exercises

Practical implementation in Robot Operating System

Tjark Schütte

Leibniz Institute for Agricultural Engineering and Bioeconomy e.V.
(ATB), Engineering for Crop Production - Agromechatronics,
Potsdam-Bornim

Germany

Part 3: Practical Exercises

- Let's bring it all together: remote sensing, UGVs and RFID sensors



The Scenario

- UGV as a fast and energy efficient vehicle to monitor and read out RFID sensors
- UGV to perform actions, only when the data collected by UGV indicates them
 - Before we dive into the practical work, some background on ROS.

Quadrotor – Simulation (prev. slide): Institute of Flight Systems and Automatic Control, Technische Universität Darmstadt.

ROS

● The Robot Operating System (ROS)

“ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.”

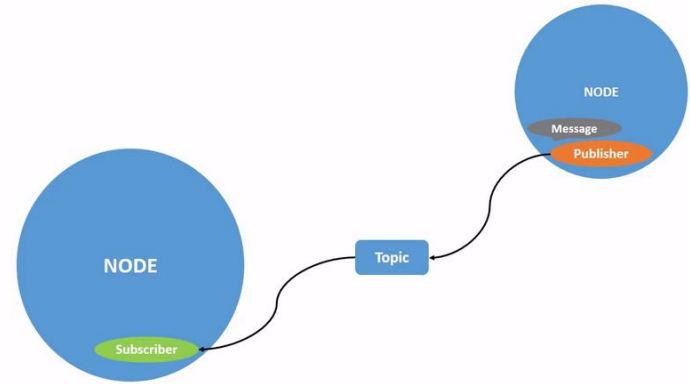
(<http://wiki.ros.org/ROS/Introduction>)

Handling Data in ROS

- ROS offers several functionalities and packages that make handling and transforming data easier:
 - **Topics** using **publishers** and **subscribers** to make data available
 - **TF** or **TF2** to store and update coordinate transformations between different frames (coordinate systems)

ROS - Topics

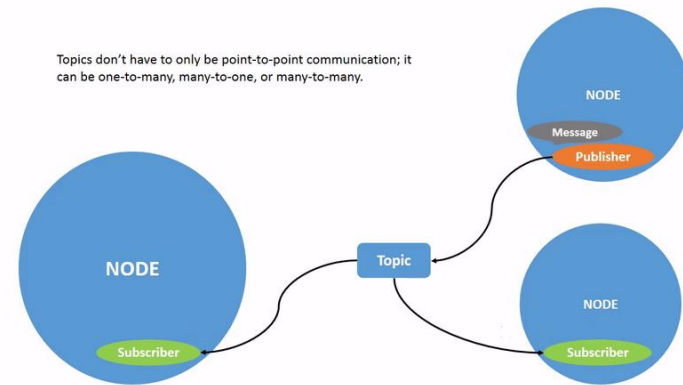
- ROS breaks down complex systems into independent software modules – so-called nodes
- For data transmission between nodes, it uses topics and messages



*ROS Publisher and Subscriber .
Source: <https://docs.ros.org/>*

ROS - Topics

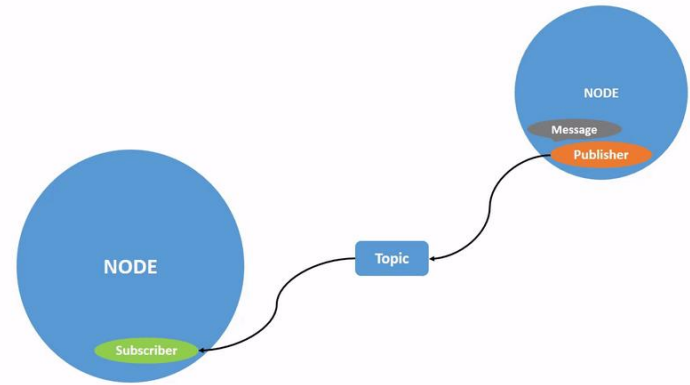
- ROS breaks down complex systems into independent software modules – so-called nodes
- For data transmission between nodes, it uses topics and messages
- Topics don't have to be point-to-point



*ROS Publisher and Subscriber .
Source: <https://docs.ros.org/>*

ROS – Callbacks

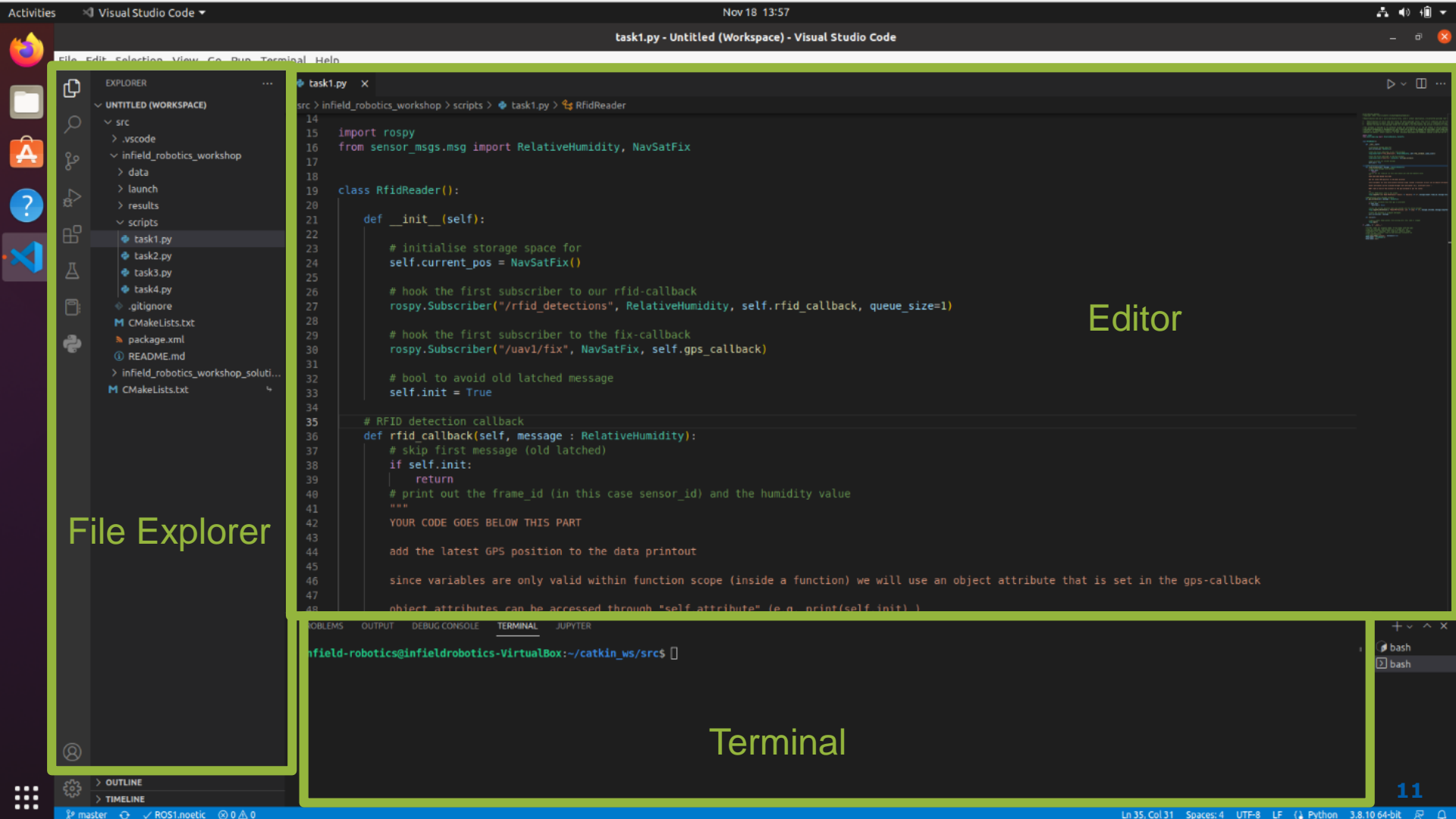
- For handling/manipulation of the incoming data, callback functions are used.
- The subscriber hooks a callback to a specific topic
- The callback is executed, whenever a new message is read by the node.



*ROS Publisher and Subscriber .
Source: <https://docs.ros.org/>*

Time for the First Exercise

- Inside the virtual machine you have been handed, a ros-node is playing back data of the UAV flying around and reading in RFID sensors. It publishes:
 - The topic `/uav1/fix` → the GPS coordinates
 - The topic `/rfid_detections` → the RFID sensor ID and humidity data



File Explorer

Editor

Terminal

The Code Explained

- The programs mainly consist of a single class, the `RfidReader` class, and a short `main` function
 - Inside the main we initialise a new ros-node and instantiate (create) an object of the class

```
rospy.init_node(['listener', anonymous=True])

# Instantiate object of the RfidReader() class
RFID_reader = RfidReader()


# execute the objects run() method
RFID_reader.run()
```

- The object is where the magic happens

The Code Explained

- The `RfidReader` class consist of a few attributes (class-specific variables) and 4 methods (functions):
 - The `__init__` function, that initialises the attributes and the subscribers
 - The `run()` method that simply keeps the program running
 - The `gps_callback()` that is executed when new GPS-data comes in
 - The `rfid_callback()` that is executed each time our UAV has detected a new RFID-sensor

First Exercise: Reading and Printing Out Data

- Start the virtual machine (if you're using your own ROS installation launch *workshop.launch* from the *infield_robotics_workshop* package)
- Open Visual Studio Code 
- Task1.py should already open up
- Type "roslaunch infield_robotics_workshop task1.py" in the terminal and hit enter
 - You should see printouts of the current gps position of the UAV
 - To terminate press Ctrl+C

Let's start! Task1 : Reading and Printing Out Data

- Go to the editor and find the `rfid_callback()` function
- Edit the printout statement in such way that apart from the RFID-data we also print out the latest GPS position
 - There is more info in the commented code
- To test your changes (in terminal):
 - `roslaunch infield_robotics_workshop task1.py`

Task2: Reading and Saving Data

- Open task2.py in the editor
- Go to the editor and find the `rfid_callback()` function
- Edit the printout statement in such way that apart from the RFID-data we also print out the latest GPS position
 - There is more info in the commented code

Back to theory: Transforming Data

- So now let's say we do not just want to store the data but directly tell our UGV to act based on it
- Remember: ROS enables the sharing of topics and messages across different machines, as long as they are part of the same network
- In order to send goals to a second robot, we need to send it in a common coordinate frame

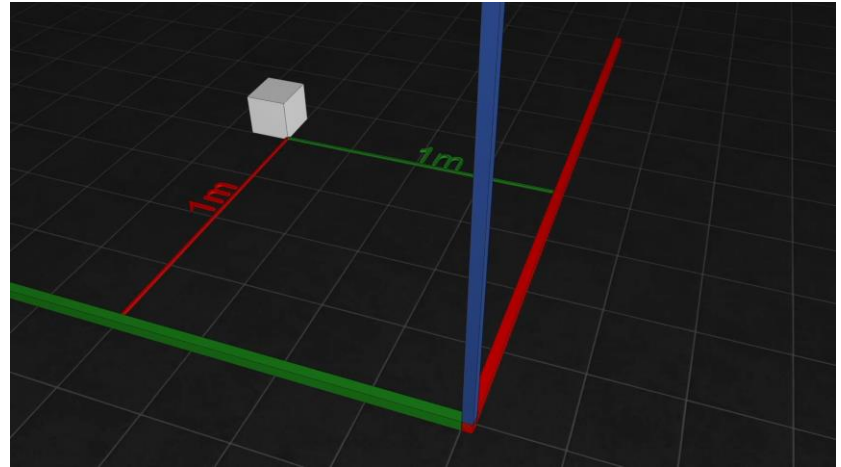
For this we use TF (transform library)

Measurement Registration and Georeferencing

- **Measurement registration** is the process of transforming data measured at different locations into a common coordinate system
- In **Georeferencing** this coordinate frame is world-fixed
 - “[...]georeferencing means to associate a digital image file [or other measurement data] with locations in physical space”. (Wikipedia, Georeferencing)

Transforming Data

- In order to transform data from a local into a global coordinate system we need to know the rotation and the translation between the two frames



Transforming Data - Rotation

- Rotation can be described by Rotation Matrices
- A vector is rotated by multiplying the rotation matrix with it
- General rotation matrices can be derived as matrix product of elemental rotations
 - Order of multiplication matters!

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation matrices for rotation around the three main-axes, x, y, z (basic/elemental rotation). Source Wikipedia – Rotation Matrix

Transforming Data - Rotation

- In robotics, the more compact representation of rotation, quaternions is used:
- Quaternions can be interpreted as describing a new axis and an angle of rotation around that axis
- Rotation using a Quaternion is done through:

Transforming Data - Rotation

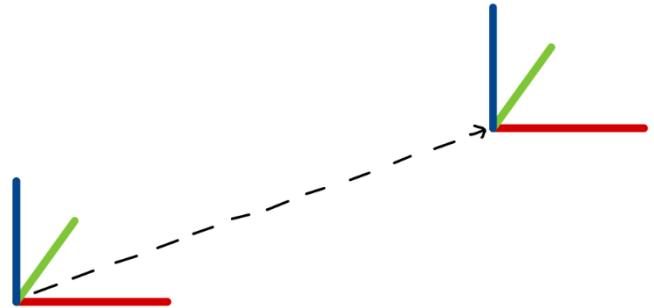
For a nice, interactive tutorial and visualization of quaternion operations see:
<https://eater.net/quaternions/video/intro>



<https://eater.net/quaternions/video/intro>

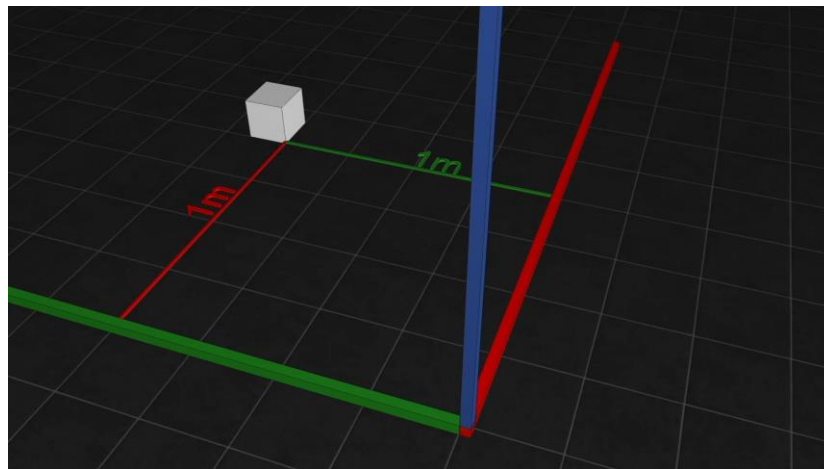
Transforming Data - Translation

- Translation of a point is achieved by simply adding the position of the origin of the local coordinate system to any coordinates in the local coordinate system



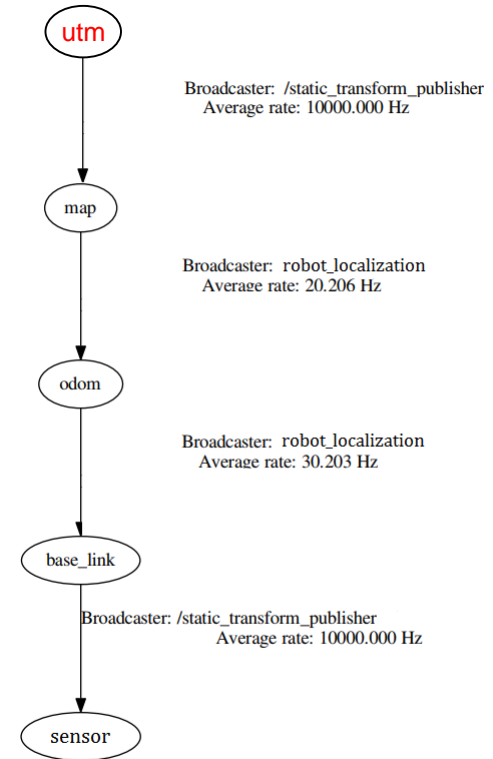
Transforming Data

- For the combined rotation and translation the position of the point is first rotated, then translated
- Continuously computing the rotation and translation of a robot-fixed coordinate system in a world-fixed coordinate system is the task of robot localization



ROS - TF

- TF keeps track of all the coordinate transformations in our systems and stores them in a tree structure – the TF-tree
- Transformations can either be static (e.g. from the base of a robot to a rigidly attached sensor) or dynamic (e.g. mobile robot position in a georeferenced frame)



Task3: Using TF to get Coordinate Transforms

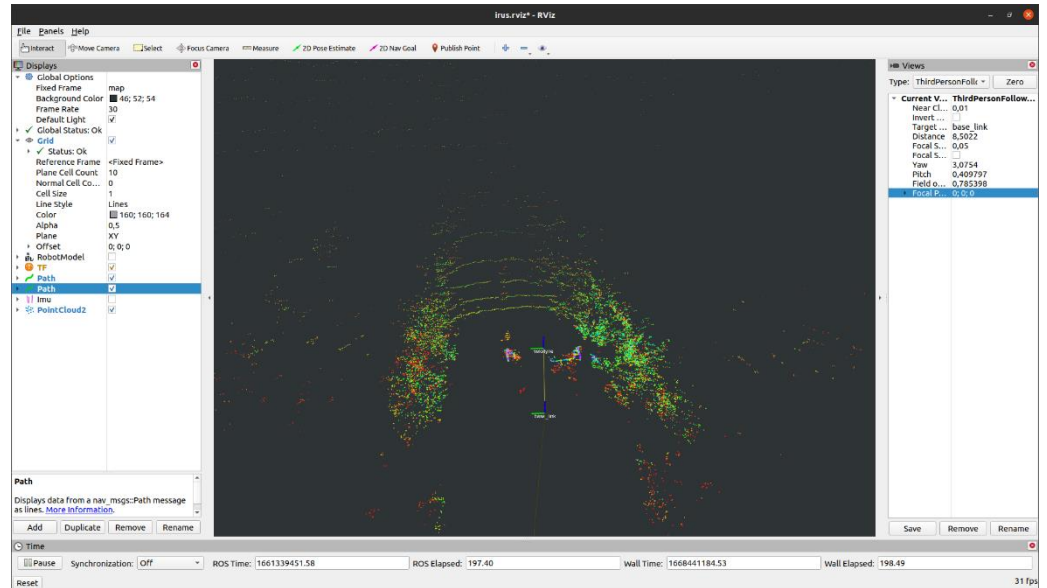
- Open task3.py in the editor
- Go to the editor and find the `rfid_callback()` function
- This function is now calling an additional function:
`send_current_position_as_goal()`
- Edit this function in such way that you receive and print out the transformation between the UAVs current position ("base_link" frame) and the global "map" frame
 - There is more info in the commented code

References

- Quadrotor – Simulation: Institute of Flight Systems and Automatic Control, Technische Universität Darmstadt. Meyer, Johannes, et al. "Comprehensive simulation of quadrotor uavs using ros and gazebo." International conference on simulation, modeling, and programming for autonomous robots. Springer, Berlin, Heidelberg, 2012.
- Simulation Environment: Gazebo-11 (Koenig, Nathan, and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator."2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS))
- ROS – documentation: <https://docs.ros.org/>
- ROS-wiki: <https://wiki.ros.org/>

ROS - RVIZ

- RVIZ is a tool that enables us to visualize several standard ROS-messages



Python – Variable Scope

- Variables are generally only available within function scope in python:
 - A variable created inside a function is only available inside that function
- Global variables can be used to overcome this but are generally not recommended

Python – Objects

- Object Attributes are available within the scope of the object