User Instructions for 31390

Unmanned Autonomous Systems 31390
June 2020
Department of Electrical Engineering • Technical University of Denmark

Introduction:

This document will give you a brief overview on ROS and Gazebo, and how to intefrace them with Simulink so that you can use this for the exercises and assignment project.

1 The system setup

This section will briefly explain how the simulation system will be run for the purpose of the course. You can follow along on Figure 1. The simulations will be run in Gazebo and Simulink i parallel.
ROS and Gazebo will be handling the physical model of the drone and a some level of drone control depending on what simulation is run.
Matlab will be used for defining parameters for the simulink model and for calculating routes from maps.
The simulink model is where the position control will be performed. Through simulink is where the communication with ROS is handled. The simulink model will have both publishers and subscribers. These will be the blocks handling the communication with ROS.

Running a simulations will thus work the following way:

- Run the start script in matlab to define simulink variables and calculates routes if necessary.
- You start ROS and Gazebo with a launch file. This starts Gazebo and all the programs nodes running the physical simulation and controls.
- Once Gazebo is up and running start the simulink model.
- The model will now run and the synchronizer will make sure timing is kept between Simulink and Gazebo
- Data sent from Gazebo is read by Simulink via the subscriber and data sent from simulink to Gazebo is handled by the publisher. All the data is sent through ROS.

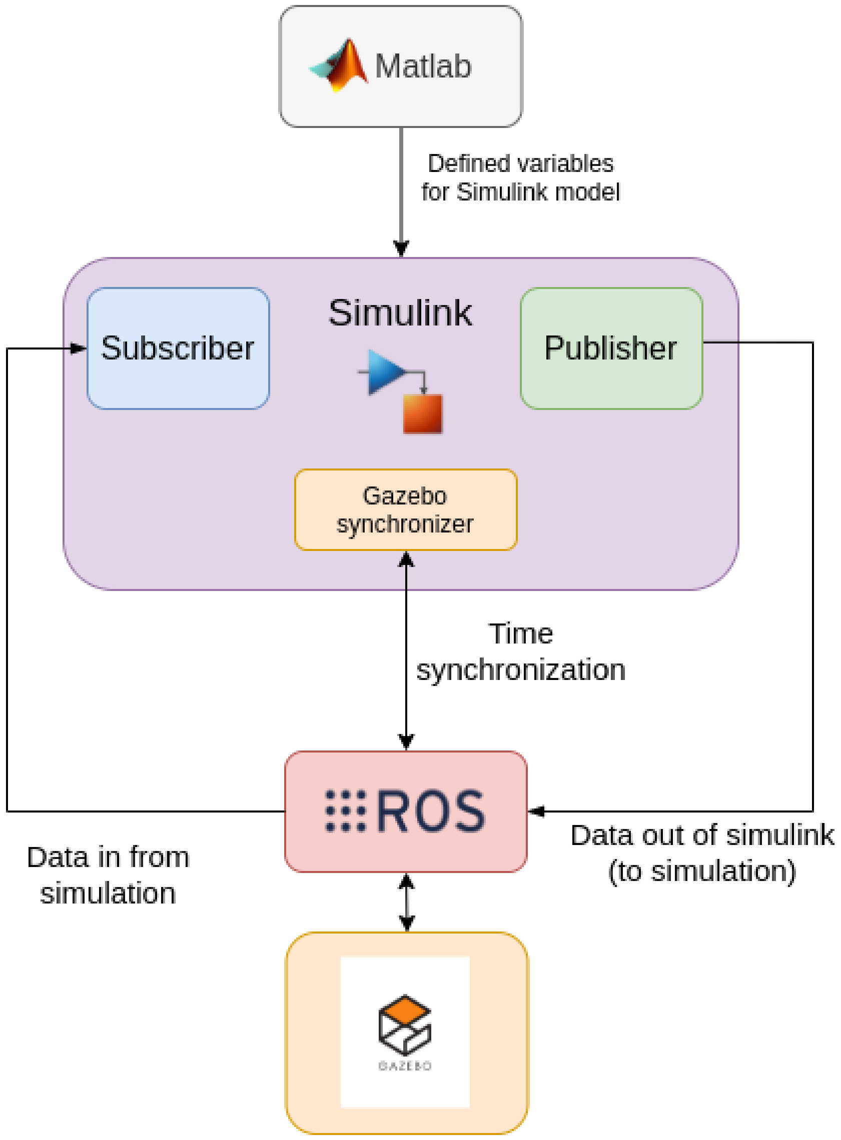


Figure 1: Diagram showing how the different programs will interact together

2 ROS + rotorS

This section will tell you what you need to know to use the Simulator running in ROS.

2.1 Basic ROS

ROS, or Robot Rperating System, is a software framework that can be used to handle many different processes and programs needed for running and controlling robots. ROS offers many different features, but as this course is not for learning ROS we will just use it as a tool.

2.1.1 Starting a ”ROS program”

In this course we will use pre-programmed bundles of ROS programs to handle all the simulation. When we wish to start such a collection we use `launch` files. It is not necessary to understand how they work, you just need to learn how to use them in order to setup your simulation.

To run a launch file you must first open a terminal. In the terminal there exist the command: `roslaunch`

```
roslaunch <package> <launch file> <extra>
```

To use it we need to pass at least 2 arguments. First the package name, which in your case will be `rotors_gazebo`. Next we need the launch file for this course. We have prepared 4 files for you:

- `mav_hovering_example.launch`
  This spawns a drone and starts the simulation and flies it up to 1 meter and stays there. This is just for testing if ROS and Gazebo is installed correctly.
- `mav_with_position_controller.launch`
  This spawns the drone in Gazebo with all the controllers available so that all position and attitude control is done in the simulation. This is to be used when only path planning is run in Simulink.
- `mav_rpy_thrust_controller.launch`
  This spawns the drone in Gazebo with attitude control. This is a stabilizing controller so that we can send it attitude commands. This means that position control and path planning should be done in Simulink but attitudes will be handled in the simulator.
- `mav_without_controller.launch`
  This spawns the drone in Gazebo but doesn’t add any controllers. This is to be used when everything is controlled from Simulink.

An example of a launch command could be:

```
roslaunch rotors_gazebo mav_hovering_example.launch
```

When you want to stop the program agian and reset the simulation, all you need to dois to terminate the program in the terminal you started it in by pressing `Ctrl + c`.

Lastly there is also the option for extra command arguments. The relevant arguments here are:

- `mav_name`
  This option determines what drone model should be used in this course we will use the following:

  - `hummingbird` This is a small quadcopter in the + configuration. This is the default drone when launching without this argument.
  - `ardrone` This is a small quadcopter in the X configuration

- `gui`
  This option is whether the graphical interface of Gazebo should open. The default value is true, but if you don't need the graphics we can set `gui` to `false`. Not using gui may speed up simulations and make resets faster.

- `world_name`
  This options is used to choose what world you want to simulate. For the course we will be using 3 different worlds:
  - `basic` The basic world is an empty world with a ground plane. This is the default world used if no argument is passed
  - `maze_1` A world with a simple maze used for 2D path planning. The maze is 2 m tall and 12x12 m large with an inner grid of 10x10 m.
  - `maze_3D` A world with a 3 level maze used for 3D path planning. Each level of the maze is 2 m tall and 12x12 m large. The inner moveable grid of this maze is 10x10x3.

As an example, if you want to launch a simulation that starts without graphics (background), and with the ardrone using the position controller in the maze_1 world, type in a terminal:

```
roslaunch rotors_gazebo mav_with_position_controller.launch
 ↪  mav_name:=ardrone gui:=false world_name:=maze_1
```

2.1.2 Topics

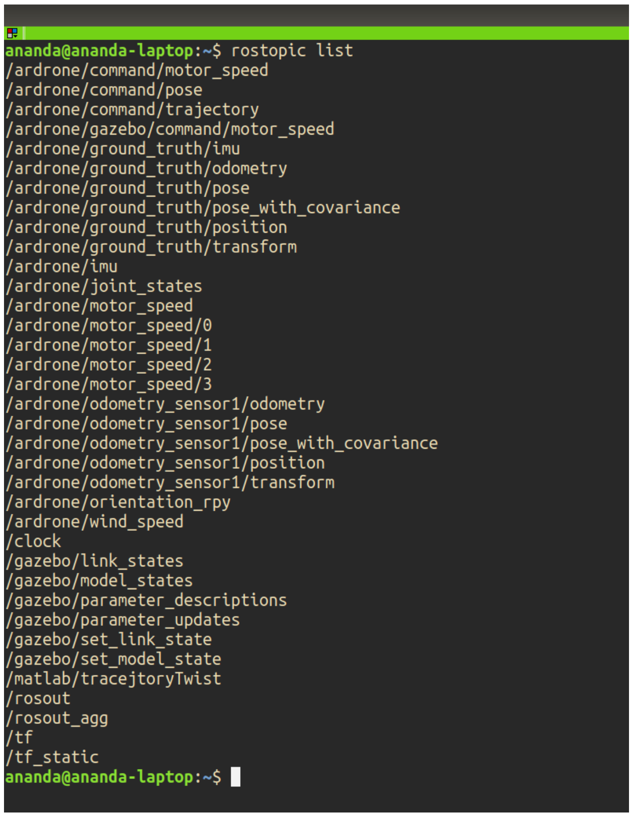
As ROS is used to handle communication between Simulink and the Gazebo simulator the most useful thing is knowing how to check the information that is transmitted and whether the transmitted data look correct.

The communication in ROS is handled with so called *topics*. A topic is a message link that a program can either read from or write to. If a program is reading from a topic it is called a *subscriber*. If a program on the other hand is writing to a topic we call it a *publisher*.

To see a list of all the available topics run the following command:

```
rostopic list
```

This will produce an output like on Figure 2:



```
ananda@ananda-laptop:~$ rostopic list
/ardrone/command/motor_speed
/ardrone/command/pose
/ardrone/command/trajectory
/ardrone/gazebo/command/motor_speed
/ardrone/ground_truth/imu
/ardrone/ground_truth/odometry
/ardrone/ground_truth/pose
/ardrone/ground_truth/pose_with_covariance
/ardrone/ground_truth/position
/ardrone/ground_truth/transform
/ardrone/imu
/ardrone/joint_states
/ardrone/motor_speed
/ardrone/motor_speed/0
/ardrone/motor_speed/1
/ardrone/motor_speed/2
/ardrone/motor_speed/3
/ardrone/odometry_sensor1/odometry
/ardrone/odometry_sensor1/pose
/ardrone/odometry_sensor1/pose_with_covariance
/ardrone/odometry_sensor1/position
/ardrone/odometry_sensor1/transform
/ardrone/orientation_rpy
/ardrone/wind_speed
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/matlab/tracejtoryTwist
/rosout
/rosout_agg
/tf
/tf_static
ananda@ananda-laptop:~$
```

Figure 2: Output of `rostopic list`

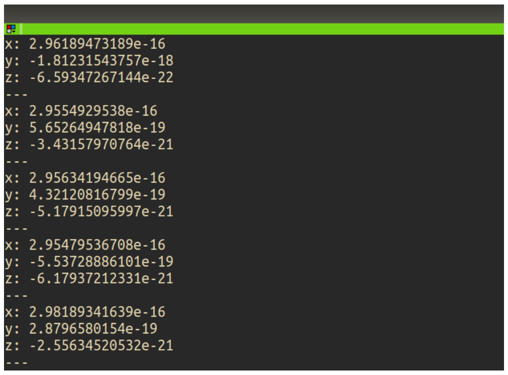
Now if we wish to see what is being published to a topic we can use the command:

```
rostopic echo <topic name>
```

To see the drones orientation you could run:

```
rostopic echo /ardrone/orientation_rpy
```

This will produce the output in Figure 3



```
x: 2.96189473189e-16
y: -1.81231543757e-18
z: -6.59347267144e-22
---
x: 2.9554929538e-16
y: 5.65264947818e-19
z: -3.43157970764e-21
---
x: 2.95634194665e-16
y: 4.32120816799e-19
z: -5.17915095997e-21
---
x: 2.95479536708e-16
y: -5.53728886101e-19
z: -6.17937212331e-21
---
x: 2.98189341639e-16
y: 2.8796580154e-19
z: -2.55634520532e-21
---
```

Figure 3: Output of `rostopic echo`

3 MatLab

All the programming and controller implementation you will be doing will be performed in MatLab and Simulink. Therefor you need to know how to interface your simulink designs with ROS. To do so several publisher and subscriber blocks have been created that you can attach to your systems to interface them with ROS.

3.1 Provided files

The `31390_matlab` folder which you should have found provides some different files that you will use in the course. If you ever need a fresh copy download it from `https://github.com/AnandaNN/31390_matlab`.

- `generateMsg.m` Simple script for ROS message generation
- `simulink_ROS_interface_blocks.slx` This simulink model contains all the different subscribers and publishers you could need for your models. Available blocks:
  - Position subsrciber
  - Attitude subsrciber
  - IMU subsrciber (accelerations and gyroscope)
  - Odometry velocity subsrciber (linear and angular velocities)
  - Position command publisher
  - Attitude command publisher
  - Motor velocities publisher
- `maze` This folder contains the mazes used for the simulation exercises as well as matlab functions for plotting the maps and the routes found. Run the script `maze_plotting_script.m` to get aquinted
- `initial_test` This folder contians the script and model for the initial test simulation to check if Simulink and Gazebo are working together
- `rotorS_attitude_interface` Contains a script and simple simulink model ready to be used with the attitude controlled drone (position and attitude subscriber and attitude control publisher)
- `rotorS_position_interface` Contains a script and simple simulink model ready to be used with the position controlled drone (position and attitude subscriber and position control publisher)
- `rotorS_motor_velocity_interface` Contains a script and simple simulink model ready to be used with the motor velocity controlled drone (position and attitude subscriber and motor velocities publisher)

3.2 Synchronization

They way we will be running the simulations Simulink and Gazebo will be running one after the other in iterations. E.g. Simulink will perform its calculations for some *dt* simulation time and then Gazebo will do it so that it in practice is two discrete systems running in parallel.

To make sure this works synchronization is needed between the two programs. This is done by the Simulink block seen in Figure 4. This block just needs to be present in the simulink model to function.

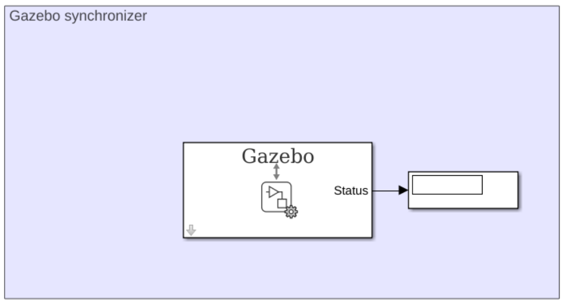


Figure 4: The gazebo synchronization block

3.3 Publisher and subscriber

Next we need to have a subscriber and publisher so we can communicate with ROS. All the needed subscribers and publishers are provided in their respectable interface models for the exercises. Figure 5 and Figure 6 shows an example of a subscriber and publisher respectively that can be used in the Simulink model.

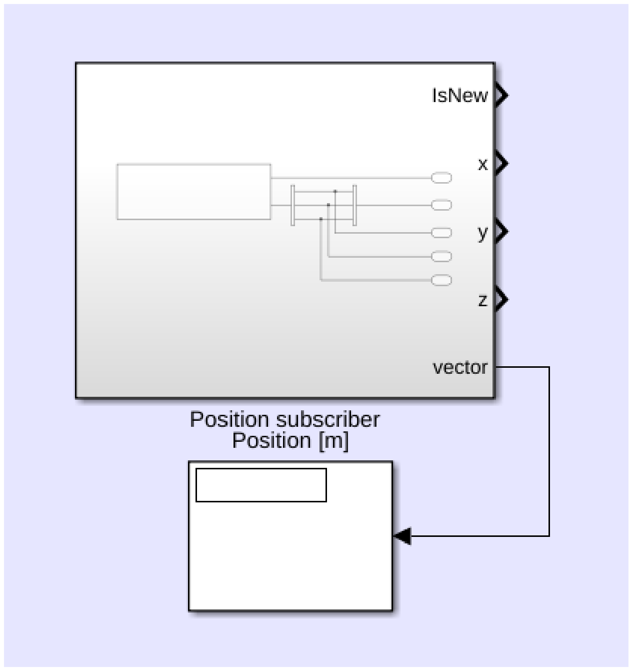


Figure 5: Example of a subscriber box created for the course. It will have the relevant values of subscriber as individual output as well as the values in a vector form. Also the IsNew is a boolean indicating if the value is new at the time of sampling.

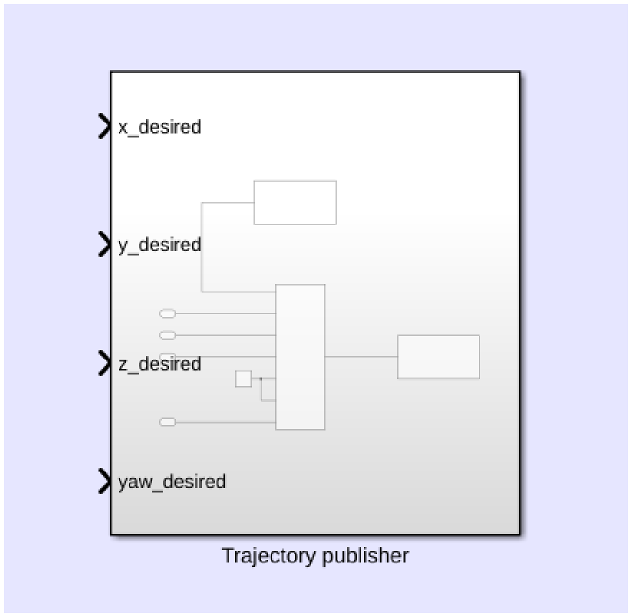


Figure 6: Example of publisher used in the course. It will have the input values needed as single input values.

3.4 Blocks and topic names

The subscriber blocks are tied to a specific topic name, but when we change the drone model the name of the topic will also change. This means that we need to change the topic name of the subscriber and publisher for the simulink model.

To do so open a subscriber or publisher subsystem by double clicking on it. Then open the subscriber or publisher parameter options by double clicking on the ROS subscriber or publisher block, again by double clicking it. This will open a window like in Figure 8.

In the *topic* the drone name needs to be the same as the model used in Gazebo, `ardrone` or `hummingbird`.

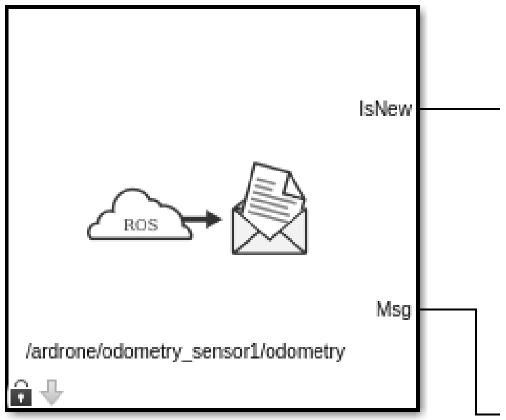
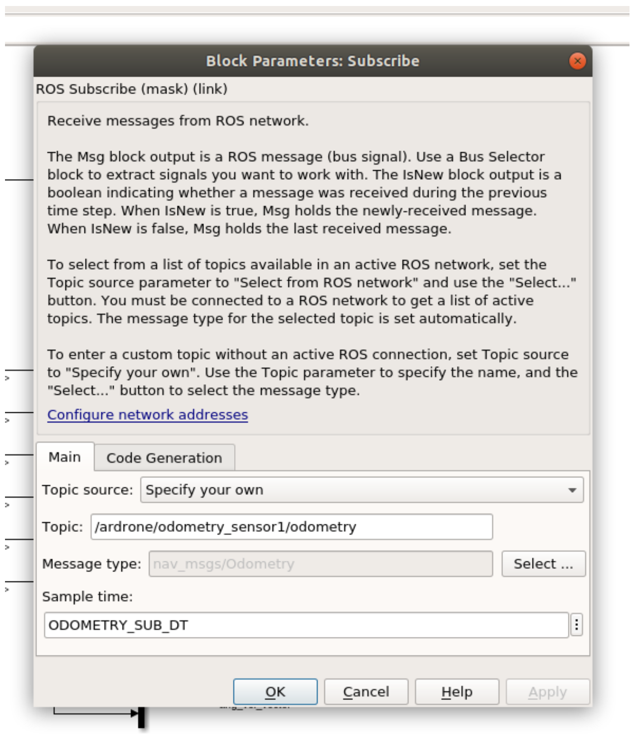


Figure 7: A basic ROS subscriber block from MatLab



Receive messages from ROS network.

The Msg block output is a ROS message (bus signal). Use a Bus Selector block to extract signals you want to work with. The IsNew block output is a boolean indicating whether a message was received during the previous time step. When IsNew is true, Msg holds the newly-received message. When IsNew is false, Msg holds the last received message.

To select from a list of topics available in an active ROS network, set the Topic source parameter to "Select from ROS network" and use the "Select..." button. You must be connected to a ROS network to get a list of active topics. The message type for the selected topic is set automatically.

To enter a custom topic without an active ROS connection, set Topic source to "Specify your own". Use the Topic parameter to specify the name, and the "Select..." button to select the message type.

Configure network addresses

Topic source: Specify your own

Topic: /ardrone/odometry_sensor1/odometry

Message type: nav_msgs/Odometry

Sample time: ODOMETRY_SUB_DT

Figure 8: Subscriber block parameters. In *topic* the name should be the name of the drone model used (*ardrone* or *hummingbird*)

4 Model parameters:

Depending on the drone that you choose to use, you might need to know some of the drone parameters to model and optimize the control gains. The parameters for the different drones that you can choose are listed below.

4.1 Hummingbird:

- mass: $m = 0.716$ kg
- arm length: $L = 0.17$ m
- rotor radius: $r = 0.1$ m
- body inertia: $Ixx = 0.007$, $Iyy = 0.007$, $Izz = 0.012$
- max rotor velocity: $\Omega_{max} = 838$ rad/s

4.2 Ardrone:

- mass: $m = 1.52$ kg
- arm length: $L = 0.127$ m
- rotor radius: $r = 0.1$ m
- body inertia: $Ixx = 0.0347563$, $Iyy = 0.0458929$, $Izz = 0.0977$
- max rotor velocity: $\Omega_{max} = 838$ rad/s

4.3 Frame of reference:

On Figures 9 and 10 you can see the rotor configuration and numbering of the drones running in the simulation. The Figures also show the frame of reference for the drone with Z pointing upwards. The world frame of reference is also a right hand coordinate system with z pointing upwards.

On Figure 11 and 12 you can see the rotation directions for roll, pitch, and yaw as they are defined in the Gazebo simulation.

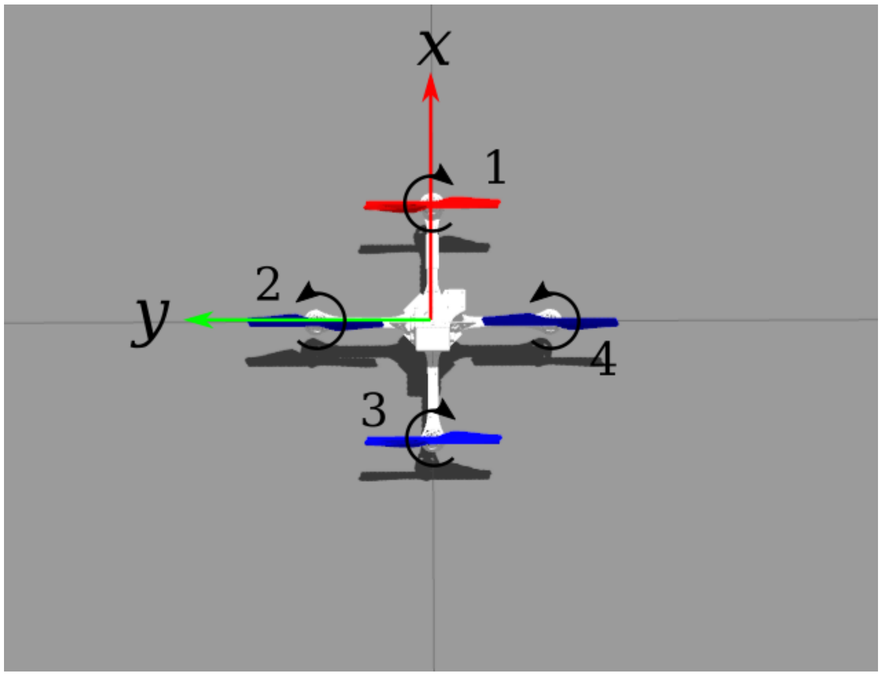


Figure 9: The Hummingbird drone with coordinate frame, motor numbering, and rotation directions. z is pointing upwards

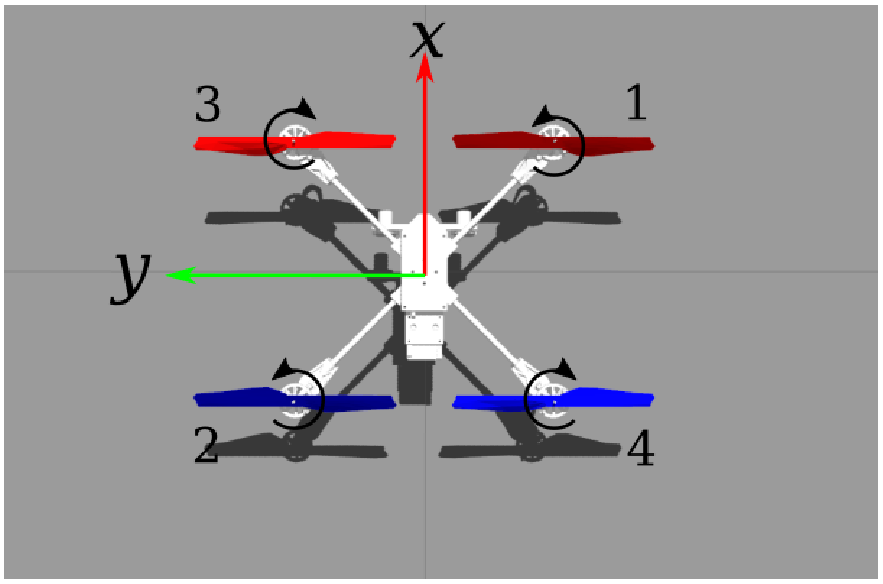


Figure 10: The Ardrone with coordinate frame, motor numbering, and rotation directions. z is pointing upwards

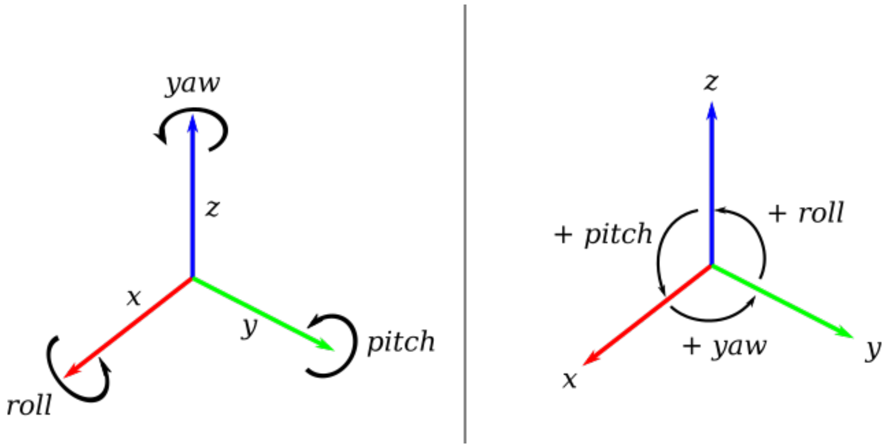


Figure 11: The frame of the drone as it is defined in the simulation. Left frame indicates the rotation direction around the axis and the left shows the positive direction of roll, pitch, and yaw.

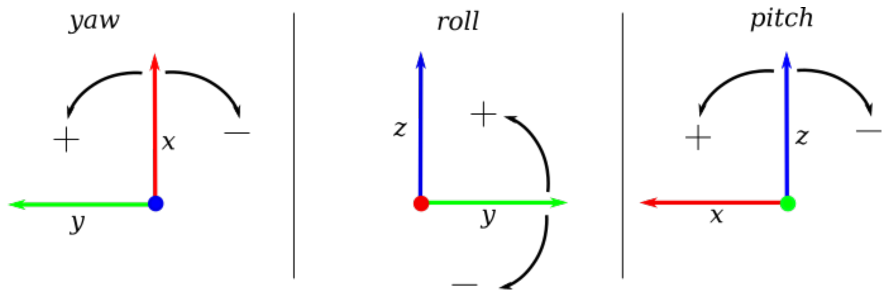


Figure 12: Figure showing the sign of rotations around all 3 axis. We are looking into the orthogonal axis.