Flight camp part 1

Environment setup

All the instructions for the course will assume that you are using Linux (Ubuntu 18.04) as your operating system. You can use the computers in the computer rooms just like you did in course DD2410.

For the project part will will assume that at least one person in the group has access to a laptop which you can use when you want to perform experiments with the actual drone. **Instructions for how to set up your computer can be found here (https://kth.instructure.com/courses/17743/pages/linux-installation)** .

Setup a new catkin workspace

For this course it is a good idea to create a new catkin workspace, we will call this workspace `~/dd2419_ws`. Remember to source your ROS installation and your new workspace from your `.bashrc` file:

```
mkdir -p ~/dd2419_ws/src
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrcecho "source ~/dd2419_ws/devel/setup.bash" >> ~/.bashrc
```

Install/setup the simulator

To install the simulator run these commands:

```
cd ~/dd2419_ws
wstool init src
wstool merge -t src https://raw.githubusercontent.com/KTH-CAS-UAV/dd2419/master/.rosinstall --merge-replace
wstool update -t src
catkin build
source devel/setup.$(basename $SHELL)
roscd dd2419_simulation
mkdir build
cd build
cmake ../../sim_cf/crazyflie-firmware/sitl_make/
make
```

If you did not just blindly copy-paste the above commands you should have seen *wstool* a number of times. Wstool is great for maintaining a ROS workspace with packages from a number of different repositories and/or version-control systems. We will take advantage of it when we provide packages for you, but of course you can also use it for your own repositories. You can read more about wstool **here (http://wiki.ros.org/wstool)** .

Another tool that is useful is *rosdep* which can automatically find and install the dependencies of all the packages you have in your ROS workspace. It does not work on the computers in the computer rooms since it requires sudo permissions. However, you can use it to see what it would install by using the `-simulate` option. You can read about it **here** **(http://wiki.ros.org/rosdep)** .

If you open the `src` directory located inside `~/dd2419_ws` you should see a directory called `course_packages`. In the `course_packages` directory are all the packages that we have made for you. When you create your own packages later on in the course it is good practise to put them right under the `src` directory and not inside `course_packages`, such that `course_packages` only contains packages that we have provided for you.

If you are running on a school computer

```
cd ~
mkdir not
mv .nv .nvidia-settings-rc not
```

The last line fixes an OpenGL problem that students with an old account would otherwise encounter. ***It is fine if it gives an error and says that the file or folder does not exist****.*

Update and build the code

To get the latest version of the packages that we provide for you, you can run:

```
cd ~/dd2419_ws
wstool update -t src
catkin build
```

We are using `catkin build` for building/compiling the packages in the `~/dd2419_ws` workspace. You can type `catkin build` in any directory that is under `~/dd2419_ws` to build the packages, for example you can be in `~/dd2419_ws/src` and type it.

Launch the simulator

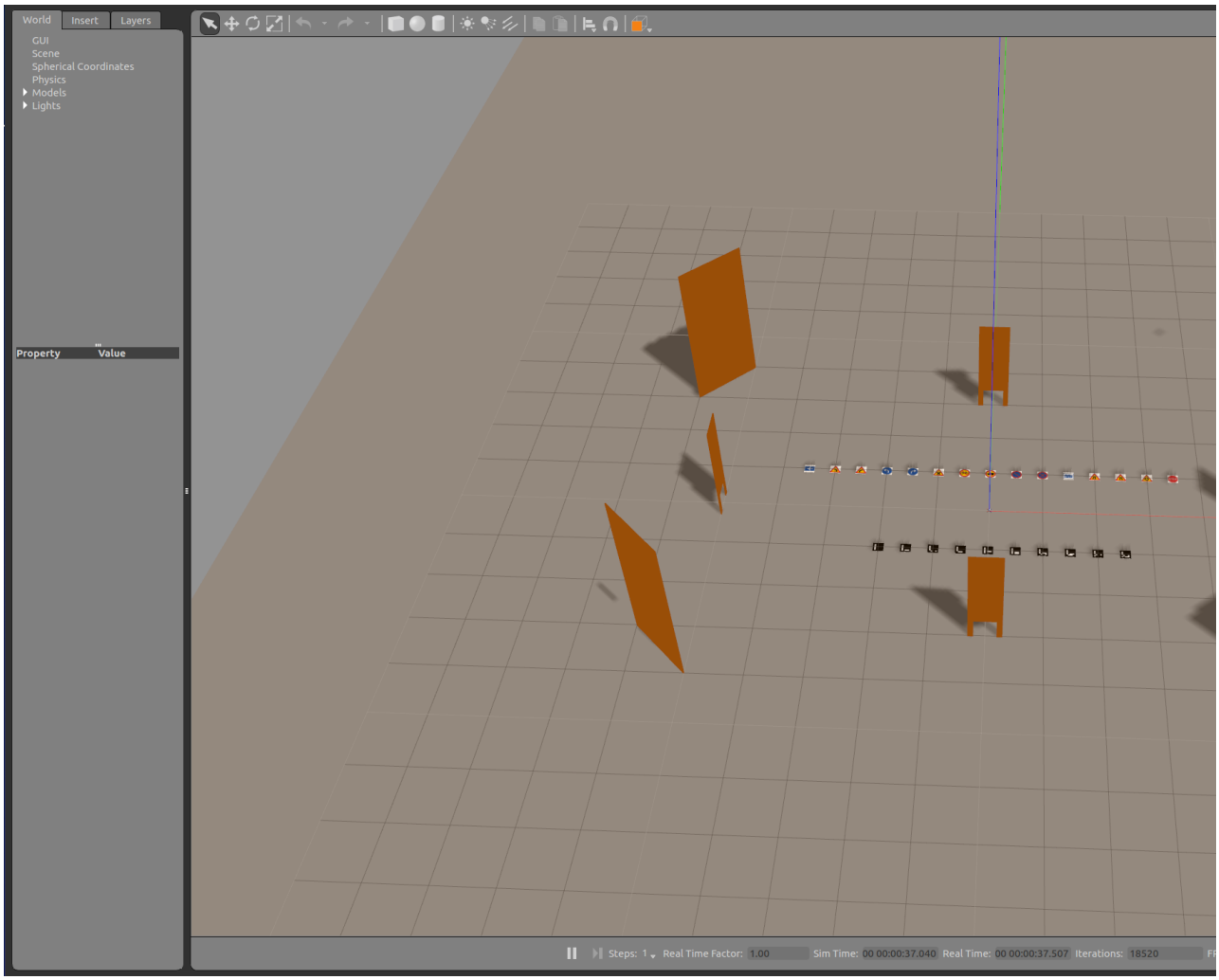
Now we have everything that we need for the **fun to begin**. We will start off by generating a Gazebo world:

```
rosrun dd2419_simulation json_to_world.py tutorial_1.world
```

Next, we will fire up the Gazebo simulation by typing the command:

```
roslaunch dd2419_simulation simulation.launch
```

The first time you run this command you will probably have to wait a couple of minutes for everything to show up (there are a lot of paper work when opening a new airport you know) and it will probably seem like Gazebo has frozen (which is has not (most likely (hopefully))). When everything has finally loaded you should see something like this:



After the initial launch of the simulator it should start up “faster”, but be ready for having to wait for like a minute or so.

You can use the mouse to move around in the Gazebo simulator. Have fun and try pressing the mouse buttons (and scroll wheel) when you are moving around to see what happens, **go wild**! You can also press `Help` -> `Hotkey Chart` to see the Gazebo hotkeys.

Inspect the world

When you are finally satisfied playing around with the mouse we can start talking about what you are seeing in the Gazebo simulator. There are three lines, the red line represents the x-axis, the green the y-axis, and the blue the z-axis. To remember the color mapping, think about xyz being mapped to RGB. Where the three lines meet is the origin and if you zoom in there you will see the simulated Crazyflie with an awesome white box representing the camera that is mounted on top of it.

The black and white squares that are on the ground are called ArUco markers and you will use them later on to localize yourself in the environment.

There are also 15 Swedish trafic signs on the ground. Later on you should be able to detect and classify these signs.

The walls that has a hole in them are called *gates* and you should be able to fly through the hole (in the correct direction). The other walls are just regular walls.

If you zoom out you will see a big orange semi-transparent box. This box represents the airspace in which you are allowed to fly in. You should **never** fly outside of this. If you do fly outside of it you should be ready to kill your Crazyflie such that it does not harm anyone.

Modify the world

During the course you might want to modify or create your own world. You can easily create new worlds by simply putting a JSON file in the `~/dd2419_ws/src/course_packages/dd2419_resources/worlds_json/`. If you look in that directory you will find a file called `tutorial_1.world.json`, this is the world that you are seeing right now in Gazebo. You can open that file and inspect and modify it if you like.

Gazebo can not read the JSON file straight away though, so we have provided you with a Python script that converts the JSON world to a Gazebo world. If you run the command:

```
rosrun dd2419_simulation json_to_world.py
```

You will see something like this:

```
USAGE: rosrun dd2419_simulation json_to_world.py FILENAME

Where FILENAME is one of:
    *  tutorial_1.world
```

This informs you how you should use this script. The script automatically finds all JSON files that are located directly under the `worlds_json` directory and lists them for you there.

Try creating your own world by creating a new file inside the `worlds_json` directory and paste this inside that file:

```
{
  "airspace": {
    "min": [-10, -10, 0.0],
    "max": [ 10,  10, 5.0]
  },

  "gate_size": [0.40, 0.40],
  "marker_size": [0.197, 0.197],
  "roadsign_size": [0.20, 0.20],

  "markers": [
    {"id": 0, "pose": {"position": [1.0, 0.0, 0.00],  "orientation": [90.0, 0.0, 0.0]}},
    {"id": 1, "pose": {"position": [4.99, 0.0, 0.105], "orientation": [90.0, -90.0, 0.0]}},
    {"id": 2, "pose": {"position": [7.0, 0.0, 0.00],  "orientation": [90.0, 0.0, 45.0]}}
  ],

  "gates": [
    {"id": 0, "heading":   0.0, "position": [ 5.0, 0.0,  0.00]}
  ],

  "walls": [

    {"plane": {"start": [6.0,   3.0, 0.0], "stop": [5.0,   4.0, 3.0]}},
    {"plane": {"start": [-5.0,  4.0, 0.0], "stop": [-6.0,  3.0, 3.0]}},
    {"plane": {"start": [-6.0, -3.0, 0.0], "stop": [-5.0, -4.0, 3.0]}},    {"plane": {"start": [5.0,  -4.0, 0.0], "stop": [
```

```
    {"plane": {"start": [9.0,  -2.0, 0.0], "stop": [9.0,   2.0, 3.0]}}

  ],

  "roadsigns": [
    {"sign": "stop", "pose": {"position": [8.99,  0.0, 0.4], "orientation": [90.0, -90.0, 0.0]}}
  ]
}
```

Save the file as `awesome.world.json`. All the files in this directory should end with ".world.json", otherwise they will not be found by the converter.

Now run the command:

```
rosrun dd2419_simulation json_to_world.py
```

You should now see `awesome.world.json` as one of the listed files:

```
USAGE: rosrun dd2419_simulation json_to_world.py FILENAME

Where FILENAME is one of:
    *  awesome.world
    *  tutorial_1.world
```

If you run:

```
rosrun dd2419_simulation json_to_world.py awesome.world
```

The script will convert your JSON file to a Gazebo world file and put it inside the `~/dd2419_ws/src/course_packages/dd2419_simulation/worlds/` directory with the name `awesome.world`.

If you close the simulator (by pressing `CTRL+C` in the terminal) and run:

```
roslaunch dd2419_simulation simulation.launch
```

You will see that… **NOTHING CHANGED?!?!** What the frick? Are you kidding me? Why? Oh, sorry. We forgot to change the parameter in the launch file that decides which world to load (would be pretty awesome with that kind of parameter in the *real* world too huh?).

Load a different world

You can load a different world in two different ways. The first one is good as a more permanent solution, and it entails opening the `~/dd2419_ws/src/course_packages/dd2419_simulation/launch/simulation.launch` file and change the line that says:

```
<arg name="world_name" default="tutorial_1"/>
```

If you change `tutorial_1` to `awesome` and run:

```
roslaunch dd2419_simulation simulation.launch
```

You should see that the world you created is loaded.

You can also load a world by typing:

```
roslaunch dd2419_simulation simulation.launch world_name:=NAME
```

Where `NAME` is the name of the world you want to load. In the launch file you could also see other parameters that you can change. One of high interest might be:

```
<arg name="gui" default="true"/>
```

If you set this to `false` the Gazebo GUI will not turn on, this can be good if you do not have a powerful computer. You can also set this like:

```
roslaunch dd2419_simulation simulation.launch gui:=false world_name:=NAME
```

Make the Crazyflie move

Okay okay, take it easy! You do **not** have to *scream*, I know you want to fly. So lets fly. We will now go though two different types of controls you can use to make the Crazyflie move. In this tutorial we will use rqt since it enables us to quickly change the input we give to the Crazyflie on-the-fly, such that it does not land. The Crazyflie will land if it does not recieve any control input for a certain period of time. You can of course use the terminal instead or by creating a ROS node (which you will do in part 2 of the flight camp).

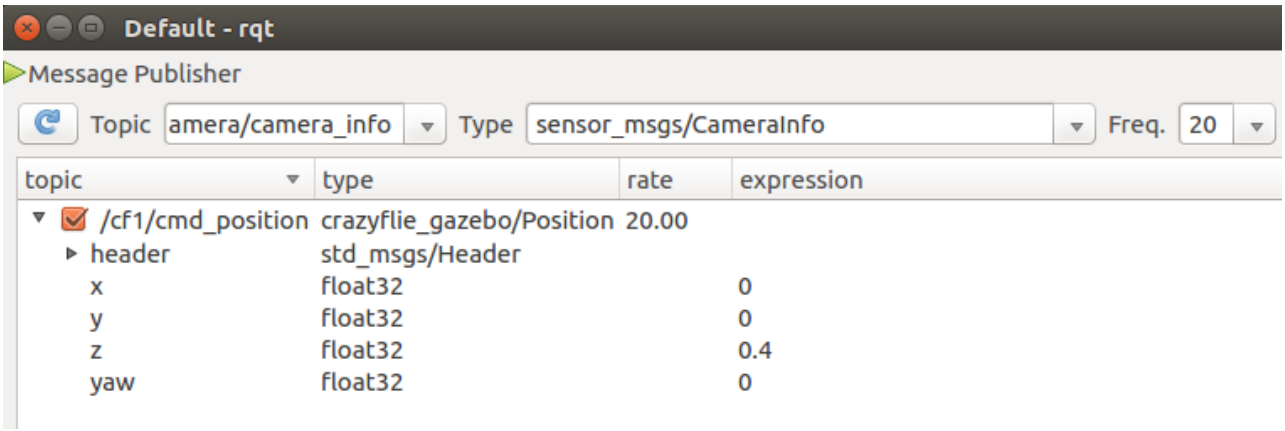
When you start the Gazebo simulator wait a couple of seconds before you start publishing setpoints and/or velocities to the Crazyflie. If you start publishing them too early the Crazyflie will get out of control.

Navigate using setpoints

One way of controlling the Crazyflie is by giving it setpoints. This means that you input a position and an orientation. First you launch rqt by typing:

```
rqt
```

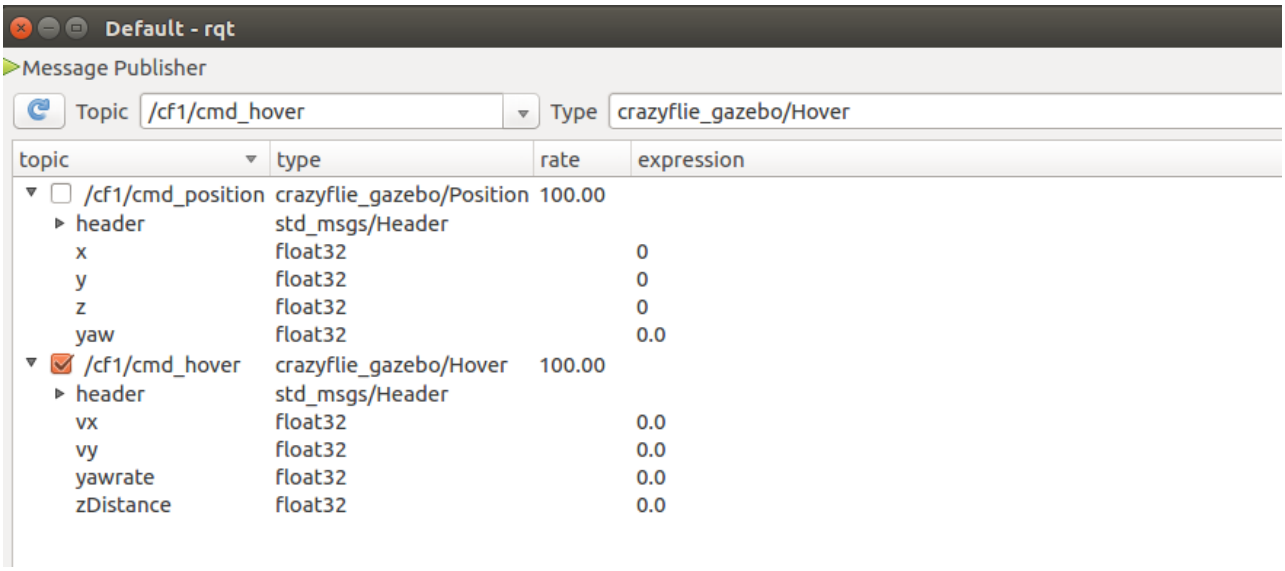
A window will open. At the top of the window press *Plugins* → *Topics* → *Message Publisher*. Then select the topic `/cf1/cmd_position`, and do *not* change type. Set the *Freq.* to *20* and press the *green* + sign, it should then look something like this:



If you set *z* to `0.4` and then mark the checkbox the Crazyflie should fly up 0.25 meters. If you now change the values of *x*, *y*, *z*, and *yaw* you will see that the Crazyflie will start flying around. The *yaw* is given in degrees here.

Navigate using velocity control

Another way of controlling the Crazyflie is by giving it velocities. In rqt select the topic `/cf1/cmd_hover`. Set the *Freq.* to *20* and press the *green* + sign, then it should look as in the image below:



If you unmark the checkbox for the `cmd_position` topic and instead mark the checkbox for the `cmd_hover` topic you can control the Crazyflie with velocities. As you can probably understand it is **not** a good idea to both send setpoints and velocities at the same time, that is why you should only have one of the checkboxes marked at a time in rqt.

Launch the ArUco marker detections

We have placed a number of ArUco markers in the world in order for you to be able to localize yourself. While you have the Gazebo simulator and rqt open run this command:

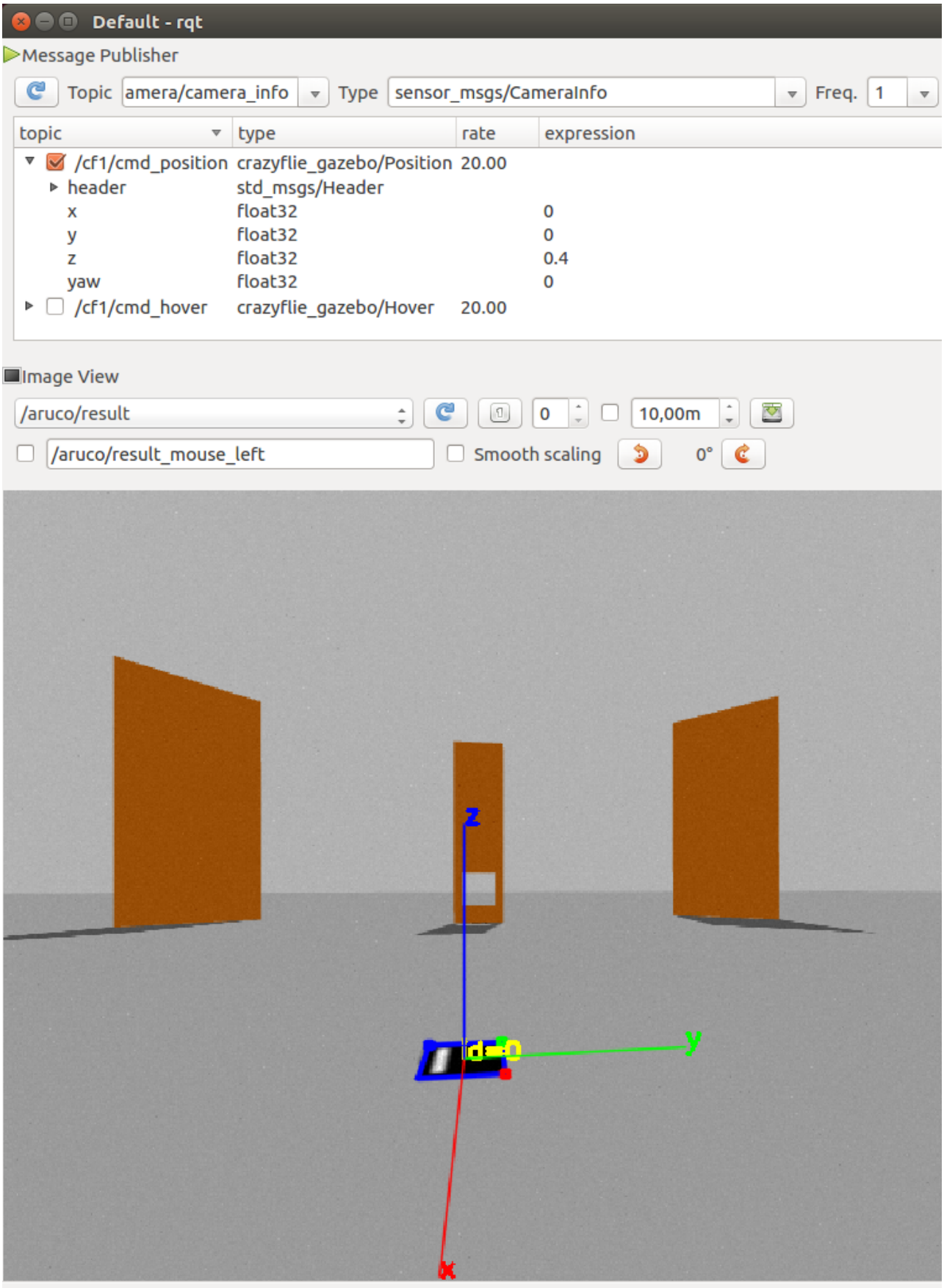
```
roslaunch dd2419_simulation aruco.launch gui:=false
```

Now the ArUco detector is running. **Reset the Gazebo simulator** by running the command:

```
rosservice call /gazebo/reset_world "{}"
```

Or if you have the Gazebo GUI running you can press *SHIFT+CTRL+R* while the Gazebo window is active.

In rqt *Plugins -> Visualization -> Image View* and the select the topic `/aruco/result`. It should now look like the image below:



If start up the Gazebo simulator with the `awesome.world` you created before and give the Crazyfly the setpoint `x=8`, `y=0`, `z=0.4`, and `yaw=0`, you will see the Crazyflie fly over the three ArUco markers and through the gate and then stop before the wall that has a stop sign on it.

ArUco markers

The ArUco markers have an orientation and an ID. Once you have run the `aruco.launch` you should be able to listen to a topic called `/aruco/markers` which publishes the pose and ID of the markers that it detects.