Mobile Robotics Practicals
Winter Semester 24/25

Prof. Dr. Stefan Leutenegger
Teaching Assistants: Dr. Jaehyung Jung, Jiaxin Wei

Practical 1: Manual Steering of an MAV

Start: 17/10/2024
Hand-In: 31/10/2024, 10:00

Introduction

This coursework is geared towards getting to know the AR Drone 2 and its Robot Operating System (ROS) interfaces in simulation and for real. We have adopted a simulation environment based on Gazebo-ROS that maximally mirrors the real-world in terms of AR drone functionality.

Real Hardware Set-Up

Here we describe the set-up of hardware and software.

Hardware

To fly the drone, you will need the following components:

- a drone,
- 2 batteries,
- a wall charger,
- a USB WiFi dongle.

Software Interface to Hardware

The drone is operated over WiFi connection using the vendor's open-source library `ARDroneLib`[1]. The library is interfaced via the open-source software package `ardrone_autonomy`[2] that is based on the Robot Operating System (ROS)[3]. From their webpage: "ROS is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source."

Note: the infrastructure provided here is supposed to build on Ubuntu 18.04 and 20.04. As a warning for those tempted to use their Macs: ROS on MacOS is experimental, and getting the `ARDroneLib` running on MacOS may be cumbersome.

[1] See http://developer.parrot.com/docs/SDK2/ARDrone_SDK_2_0_1.zip – do not build this, just for reference. It also contains the documentation.
[2] See http://wiki.ros.org/ardrone_autonomy. Note the documentation link on that page!
[3] See www.ros.org

Hardware Safety

Warning: read this before using the drone.

Read the safety instructions provided with the drone package. Note that some of the rules below are stricter, not permitting, for instance, outdoor use of the drones.

Make sure to keep the batteries charged sufficiently when operating the drones (charge one while the other is in use).

By connecting the battery to the drone, it will be powered on automatically. A small motion of the propellers and an associated sound indicates it is up and running. Note that the motors won't start, unless you command the drone to do so. You are allowed to switch on the drones and work with them **with motors switched off** at your desks.

Warning: never obstruct the path of the rotating blades with objects or parts of your body, regardless of whether or not the blades are turning.

Note: We have the Drone Lab in 02.05.014 to fly the AR Drones. This is the only area you are allowed to turn on the motors and fly the drones. Also, you must fly the drone within the netted area, while everyone stays outside, with the net closed. You must not fly the drone without the protective styrofoam. By operating the drone using your own code, you confirm that you undertook the best endeavours to put the following safety mechanisms in place:

- pressing the 'L'-key sends the drone into landing mode (as specified by the vendor),
- pressing the ESCAPE key immediately shuts off all rotors (as specified by the vendor).

Note: By operating the drones, you agree to ensure at all times that people in the same room are aware of the activity and to undertake the best endeavours to guarantee their safety and to keep the drone intact.

You agree to hold TU Munich and the course supervisors entirely free from any liability, including financial responsibility for injuries incurred, regardless of whether injuries are caused by negligence. By operating the drone, you acknowledge the risks involved. These include but are not limited to drones crashing into people, rotating propellers causing cuts, and batteries catching fire.

Gazebo-ROS Simulation Environment

We have adopted the Gazebo[4] simulator with ROS[5] interface and a provided model of the AR Drone 2, which includes commanding it in the same way through ROS messaging, as well as receiving simulated sensory information through ROS messaging, i.e. images from the camera and IMU measurements.

While Gazebo is part of the ROS distribution, the AR drone model and virtual interfaces are provided by the `parrot_ardrone` package.

In other words, the full simulation environment replaces the real hardware and `ardrone_autonomy` package. We have set up this practical in a way that will allow seamless transition to the real-world set-up once it is working in simulation.

[4] See http://gazebosim.org/, note however that it comes already as part of ROS 1, so you do not need to separately install it.
[5] See www.ros.org.

Computer setup

While in principle you could use a machine and log in via `ssh -X -C`, we strongly recommend against this and to only use it as a last resort, since the interactive and 3D graphics based interfaces will be very laggy at best, even with an outstanding internet connection.

Therefore, we advise to set up your own computer. As a recommended and tested setup, Ubuntu 18.04 or 20.04 LTS is used. In this case follow these steps:

- Install system dependencies:

```
sudo apt-get install libsdl1.2-dev libsdl2-dev
```

- Install ROS 1:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu \
   $(lsb_release -sc) main" > \
   /etc/apt/sources.list.d/ros-latest.list'
sudo apt install curl # if you haven't already installed curl
curl -s \
   https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc \
   | sudo apt-key add -
sudo apt-get update
```

- Then, on Ubuntu 20.04:

```
sudo apt-get install ros-noetic-desktop-full
```

  Or on Ubuntu 18.04:

```
sudo apt-get install ros-melodic-desktop-full
```

In case you use other systems: make sure to have SDL 1.2 and 2 installed. Regarding ROS 1, please see http://wiki.ros.org/noetic/Installation.

Note that we cannot provide individual support if you cannot get things to work on your system. We recommend using VirtualBox if you cannot easily switch to a fully ROS-supported setup.

Task 1: Hello AR Drone

Set up ROS Throughout the practicals, we assume you run bash – if you prefer a different shell, it is your responsibility to set up ROS appropriately. To get the ROS tools and build system to work, you need to (assuming ROS noetic):

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Get the code Create a Catkin (ROS building infrastructure) workspace:

```
mkdir -p ~/ardrone_ws/src
cd ~/ardrone_ws/src
catkin_init_workspace
```

Next, clone the `ardrone_autonomy` ros (Catkin) package, which also obtains the `ARDroneLib` automatically:

```
cd ~/ardrone_ws/src
git clone -b indigo-devel \
   https://github.com/sleutenegger/ardrone_autonomy.git
```

Note that your lecturer needed to hack the `ardrone_autonomy` and `ARDroneLib`, in order to get the accurate timestamps from the drone, and he had to fix actual bugs in ARDroneLib. So please use the fork provided above and no other repository.

Now get the `parrot_ardrone` package that simulates the AR Drone 2 in Gazebo:

```
cd ~/ardrone_ws/src
git clone -b kinetic-gazebo9 \
   https://sleutenegger@bitbucket.org/sleutenegger/parrot_ardrone.git
```

Again, your lecturer had to fix some issues and provide some additions to the original repository, so please use the fork provided above and not the original.

Next, let the TAs know your GitHub user names, so you can be given read-only access to https://github.com/smartroboticslab/ardrone_practicals. There, you can now fork the repository and clone it (assuming you called it `ardrone_practicals`, and assuming SSH keys are set up):

```
cd ~/ardrone_ws/src
git clone git@github.com:<username>/ardrone_practicals.git
```

Building Now you are ready to build the Catkin workspace:

```
cd ~/ardrone_ws
catkin_make -DCMAKE_BUILD_TYPE=Release
```

You are now ready to run the demo program. Before you do, however, please carefully study the code in `~/ardrone_ws/src/ardrone_practicals` – as it forms the basis of all your developments.

WiFi connection If running the real-world set-up, your drone (named after a famous researcher) can be simply connected to by selecting the WiFi network network `ardrone2_<famousresearcher>`.

Running Before running the application with `roslaunch`, you need to set up the current workspace once per terminal:

```
source ~/ardrone_ws/devel/setup.bash
```

Feel free to add this to your `.bashrc` – otherwise you will have to re-run this command whenever you open a new terminal.

Now launch (auto-completion should work on these):

```
roslaunch ardrone_practicals arp_sim.launch
```

You should now see a window with the live video from the forward-facing camera. The launch file sets necessary parameters and launches a set of executables, importantly the `arp_node` that you will be extending in these practicals.

Warning: if you fly the real drone: before you take off, make sure to be cleared to do so and you are in the designated area with enough space and people around being aware of what you are doing. Running the same application in the real-world is done using the `arp.launch` file instead. Takeoff by pressing 'T' and the drone should go into hover mode. Land it with 'L'. You can shut-off the propellers at any time pressing the ESCAPE key – which may, however, cause damage when pressed in-flight. Note that the keyboard is only read when the graphics window is active – otherwise you cannot command the drone with keys.

Task 2: Teleoperation application

Here, you should extend the "Hello AR Drone" application (adapting `arp_node.cpp`) and the `arp::Autopilot` class to include the following features allowing safe teleoperation in future:

- move the drone forward/backward and left/right with the arrow keys,
- move the drone up/down and yaw left/right with the 'W'/'S' and 'A'/'D' keys, respectively,
- display instructions of use onto the graphics window.
- display the battery state of charge in the graphics window,
- display the current drone status in the graphics window.

Note that we use `SDL2`[6] for keyboard access and to render the live view. For drawing additional information onto the view, you should use `OpenCV4`[7] (which is already used).

To send move commands, you are expected to implement the method `arp::Autopilot::move`. Modify the `Autopilot.hpp` and `Autopilot.cpp` files appropriately. Similar to the already implemented commands, you will have to set up a `ros::Publisher` (see publisher in the header file). Follow the instructions on composing the respective `geometry_msgs::Twist` ros message from https://ardrone-autonomy.readthedocs.io. Note that the publishing topic needs to be `/cmd_vel` (without `/ardrone` prefix). Also note that your desired tilt/velocity outputs should all be in the interval $[-1, \ldots, 1]$, so check for that.

Note: it is good practice to check the mode in which the drone is currently in to determine whether or not move commands should actually be sent to the drone.

Note: make sure to handle several keys pressed at once and assemble that into **one** velocity command message! The messages should be sent at fixed rate, even if no keys are pressed (in which case you should send all zeros to stop the drone from moving and get it to hover). The drone will use whatever you last sent as a control reference...

Warning: the launch file contains parameters for maximum/minimum roll/tilt angles. Do not alter them!

Once you can demonstrate correct operation in simulation to the Teaching Assistants, you can test things in the Drone Lab!

Assessment

Demonstrate correct functionality (teleoperation, display overlays) in simulation and on the real drone to the Teaching Assistants.

[6] Documentation see https://wiki.libsdl.org

[7] Documentation see https://docs.opencv.org/4.2.0/