Installation and Programming

manual Multidrone Project

Goetz Winterfeldt

TH Deggendorf

Installation and Test of Components

# Be aware that the manual is assembled form official wiki sources!

Copyright remains with the authors

# Indigo Installation on Ubuntu

## Setup your sources.list

Setup your computer to accept software from packages.ros.org.

* sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

|  |  |
| --- | --- |
| [Mirrors](http://wiki.ros.org/ROS/Installation/UbuntuMirrors) | [Source Debs](http://wiki.ros.org/DebianPackageSources) are also available |

## Set up your keys

* sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

If you experience issues connecting to the keyserver, you can try substituting hkp://pgp.mit.edu:80 or hkp://keyserver.ubuntu.com:80 in the previous command.

## Installation

First, make sure your Debian package index is up-to-date:

* sudo apt-get update

There are many different libraries and tools in ROS. We provided four default configurations to get you started. You can also install ROS packages individually.

In case of problems with the next step, you can use following repositories instead of the ones mentioned above [ros-shadow-fixed](http://wiki.ros.org/ShadowRepository)

**Desktop-Full Install: (Recommended)** : ROS, [rqt](http://wiki.ros.org/rqt), [rviz](http://wiki.ros.org/rviz), robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception

sudo apt-get install ros-indigo-desktop-full

## Initialize rosdep

Before you can use ROS, you will need to initialize rosdep. rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS.

sudo rosdep init

rosdep update

## Environment setup

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

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

source ~/.bashrc

If you have more than one ROS distribution installed, *~/.bashrc* must only source the *setup.bash* for the version you are currently using.

If you just want to change the environment of your current shell, instead of the above you can type:

source /opt/ros/indigo/setup.bash

## Getting rosinstall

[rosinstall](http://wiki.ros.org/rosinstall) is a frequently used command-line tool in ROS that is distributed separately. It enables you to easily download many source trees for ROS packages with one command.

To install this tool on Ubuntu, run:

sudo apt-get install python-rosinstall

# Install the Simulator

## Create a workspace for the simulator

mkdir -p ~/tum\_simulator\_ws/src

cd ~/tum\_simulator\_ws/src

catkin\_init\_workspace

## Autonomy Driver

git clone <https://github.com/AutonomyLab/ardrone_autonomy.git>

cd ..

rosdep install --from-paths src --ignore-src --rosdistro indigo -y

catkin\_make

## TUM Simulator (make sure to install the patch from basti35)!!!

https://github.com/dougvk/tum\_simulator/tree/5f758305d42a9f9e6fdd7447b3f494e5085486d7

cd ..

rosdep install --from-paths src --ignore-src --rosdistro indigo –y

catkin\_make

## ARDrone Helper

AR\_Helper for the joystick

# cd into ros root dir

roscd

# clone repository

svn checkout https://svncvpr.informatik.tu-muenchen.de/cvpr-ros-pkg/trunk/ardrone\_helpers

# add to ros path (if required)

export ROS\_PACKAGE\_PATH=$ROS\_PACKAGE\_PATH:`pwd`/ardrone\_helpers

# build package

rosmake ardrone\_joystick

rosmake joy

Source the environment

source devel/setup.bash

## Launch a Simulation

### Launch

How to run a simulation. Run a simulation by executing a launch file in cvg\_sim\_gazebo package:

roslaunch cvg\_sim\_gazebo ardrone\_testworld.launch

### Control using the Terminal

* take off

rostopic pub -1 /ardrone/takeoff std\_msgs/Empty

* land

rostopic pub -1 /ardrone/land std\_msgs/Empty

* switch camera

rosservice call /ardrone/togglecam

* motion (must be called frequently)

# fly forward

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 1.0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

# fly backward

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: -1.0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

# fly to left

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.0, y: 1.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

# fly to right

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.0, y: -1.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

# fly up

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.0, y: 0.0, z: 1.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

# fly down

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.0, y: 0.0, z: -1.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

# counterclockwise rotation

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 1.0}}'

# clockwise rotation

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: -1.0}}'

# stop

rostopic pub -r 10 /cmd\_vel geometry\_msgs/Twist '{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'

### Control using Joystick

You can manipulate the quadrocopter with joysticks after launching:

roslaunch ardrone\_joystick teleop.launch

* The L1 button starts the quadrocopter. It also works as a deadman button so that the robot will land if you release it during flight.
* The left stick can be used to control the vx/vy-velocity. Keep in mind that these velocities are given in the local frame of the drone!
* The right stick controls the yaw-rate and the altitude.
* The select button can be used to switch between the two cameras.

# Setting Up the Development Environment

# Hardware Control (under test)

## Run the ROS Driver

ardrone\_autonomy driver accepts a command line argument named -ip to connect to a drone configure with an IP address other than default 192.168.1.1.

* From Command Line:

rosrun ardrone\_autonomy ardrone\_driver

## Multidrone (under test)

### Step 1: Configure wifi Network

Configure your wireless router to create a managed wifi network without security (e.g. dronenet). In my experiments I was not able to make drones connect to WEP/WPA secured networks. This might be because of either a bug in firmware or in my connection script. Configure the router to have the IP address of 192.168.1.1 on subnet 255.255.255.0. Configure the router's DHCP server to give clients IP addresses starting from 192.168.1.100, we will reserve smalled IP addresses for drones.

### Step 2: Configure Each Drone

In this step we create a file called wifi.sh on the drone which will make the drone connect to pre-defined wifi network whenever triggered. The steps are as follows:

* Connect to your AR-Drone's ad-hoc wifi network
* Telnet to your drone

mani@pc# telnet 192.168.1.1

Trying 192.168.1.1...

Connected to 192.168.1.1.

Escape character is '^]'.

BusyBox v1.14.0 () built-in shell (ash)

Enter 'help' for a list of built-in commands.

* Create a file named wifi.sh in /data/ directory on the drone:

vi /data/wifi.sh

* Change network SSID in line 3 and IP number in line 4 of the following code, then copy & paste it into the file. The SSID is what you set up in step 1. The IP should be unique for each drone from the reserved range.

killall udhcpd

ifconfig ath0 down

iwconfig ath0 mode managed essid dronenet

ifconfig ath0 192.168.1.10 netmask 255.255.255.0 up

Hint if you are not familiar with vi: Press i in vi to go to insert mode, paste the code by pressing shift-ctrl-v, press ESC to exit insert mode. Type :wq to write and quit.

* Make the newly created file executable

chmod +x /data/wifi.sh

* Close the telnet connection

exit

### Step 3: Run the wifi.sh from PC

While Steps 1 & 2 should only be done once, this step should be executed whenever you want to make the Drone connect to your wireless network (for example everytime you turn the drone on).

* Connect to your drone's ad-hoc wireless network
* Remotely execute wifi.sh

mani@pc# echo "./data/wifi.sh" | telnet 192.168.1.1

* Connect your PC to the hidden wireless network
* Test the connection to the drone by pinging its IP address

ping 192.168.1.10

### Step 4: Run the ROS Driver

ardrone\_autonomy driver accepts a command line argument named -ip to connect to a drone configure with an IP address other than default 192.168.1.1.

* From Command Line:

rosrun ardrone\_autonomy ardrone\_driver -ip 192.168.1.10

* In a launch file:

<node name="ardrone\_driver" pkg="ardrone\_autonomy" type="ardrone\_driver" args="-ip 192.168.1.10" />

Setting Up the Development Environment

# Project

## Objective

We are developing a Choregraphic Suite to control multiple drone using a ROS environment. We have chosen to realize the Suite using JavaScript (RobotWebTools). Users will be able to edit object timelines. We are organizing Commands in these timelines to control the object behavior.

|  |  |  |  |
| --- | --- | --- | --- |
| Object: Drone1 | Time In | Time Out | Param |
| Takeoff | 0.0 | 0.3 |  |
| Hover | 0.3 | 1.0 |  |
| Forward | 1.0 | 2.0 | Speed: 0.1 |

Table Timeline (table describtion)

## Overall Architecture

# Setup

## Installation of Components

Download Netbeans ([www.netbeans.org)](http://www.netbeans.org))

Run installer

## Download JavaScript Libraries

We are using roslibjs

<https://github.com/RobotWebTools/roslibjs>

## Run the roslibjs

Code can be copied into the Netbeans Web Project:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<script type="text/javascript" src="http://cdn.robotwebtools.org/EventEmitter2/current/eventemitter2.min.js"></script>

<script type="text/javascript" src="http://cdn.robotwebtools.org/roslibjs/current/roslib.min.js"></script>

<script type="text/javascript" type="text/javascript">

// Connecting to ROS

// -----------------

var ros = new ROSLIB.Ros({

url : 'ws://localhost:9090'

});

ros.on('connection', function() {

console.log('Connected to websocket server.');

});

ros.on('error', function(error) {

console.log('Error connecting to websocket server: ', error);

});

ros.on('close', function() {

console.log('Connection to websocket server closed.');

});

// Publishing a Topic

// ------------------

var cmdVel = new ROSLIB.Topic({

ros : ros,

name : '/cmd\_vel',

messageType : 'geometry\_msgs/Twist'

});

var twist = new ROSLIB.Message({

linear : {

x : 0.1,

y : 0.2,

z : 0.3

},

angular : {

x : -0.1,

y : -0.2,

z : -0.3

}

});

cmdVel.publish(twist);

// Subscribing to a Topic

// ----------------------

var listener = new ROSLIB.Topic({

ros : ros,

name : '/listener',

messageType : 'std\_msgs/String'

});

listener.subscribe(function(message) {

console.log('Received message on ' + listener.name + ': ' + message.data);

listener.unsubscribe();

});

</script>

</head>

<body>

<h1>Simple roslib Example</h1>

<p>Check your Web Console for output.</p>

</body>

</html>

Run roscore:

* roscore&

The rosbridge v2.0 server with the following:

* roslaunch rosbridge\_server rosbridge\_websocket.launch

## Control a drone using HTML5

(Work in progress)