ardrone_autonomy Documentation

Release indigo-devel

Mani Monajjemi

October 18, 2015

Contents

ardrone_autonomy is a ROS driver for Parrot AR-Drone 1.0 & 2.0 quadrocopter. This driver is based on official AR-Drone SDK version 2.0.1. *ardrone_autonomy* is a fork of AR-Drone Brown driver. This package is developed in Autonomy Lab of Simon Fraser University by Mani Monajjemi and other Contributors .

External Links: Source code and issue tracker | ROS wiki page | Code API

CHAPTER 1

Updates

- **April 2014**: 1.4
  - Publish Odometry (#123)
  - Support for multiple instances of the driver on a single machine (#98 and ardronelib/#2)
  - Use reception time for video streams (#89)
  - Refactoring of source code and build system
  - Deprecated setting TF root frame (6afa19)
  - Deprecated auto IMU calibration (6afa19)
- **September 3 2014** : 1.3.5: Bug Fixes & Minor Improvements
- **March 14 2014**: The binary packages of the driver are now built on ROS build farm. You can install the driver for ROS *Indigo*, *Hydro* and *Groovy* using `apt-get` on *Ubuntu*.
- **January 17 2014**:
  - Fully *catkinized* package (#75 & #79).
  - ARDroneLib has been configured to be built as an external project. ARDroneLib is replaced by the vanilla SDK's stripped tarball. (More info).
- **October 22 2013**: Update to Parrot SDK 2.0.1 (Fixes crashes on 2.4.x firmwares, no support for flight recorder (yet).
- **February 13 2013**: Support for USB key recording (More info). Motor PWM added to legacy Navdata.
- **January 9 2013**:
  - ROS Groovy support.
  - Support for zero-command without hovering (More info).
  - Fully configurable Navdata support (More info).
  - Support for *Flight Animations*.
  - Support for Real-time navdata and video publishing (More info).
  - Support for configurable data publishing rate.
- **November 9 2012**: Critical Bug in sending configurations to drone fixed and more parameters are supported (More info). Separate topic for magnetometer data added (More info).
- **September 5 2012**: Experimental automatic IMU bias removal.
- **August 27 2012**: Thread-safe SDK data access. Synchronized *navdata* and *camera* topics.

- **August 20 2012**: The driver is now provides ROS standard camera interface.
- **August 17 2012**: Experimental `tf` support added. New published topic `imu`.
- **August 1 2012**: Enhanced *Navdata* message. *Navdata* now includes magnetometer data, barometer data, temperature and wind information for AR-Drone 2. (Issue #2)
- **July 27 2012**: *LED Animations* Support added to the driver as a service
- **July 19 2012**: Initial Public Release

CHAPTER 2

Table of Contents

2.1 Installation

2.1.1 Binary install

On supported *Ubuntu* platform and for ROS *Indigo*, *Hydro* and *Groovy* you can install the driver by running `apt-get install ros-*-ardrone-autonomy` e.g. `apt-get install ros-hydro-ardrone-autonomy` in a terminal.

2.1.2 Compile from source

The bundled AR-Drone SDK has its own build system which usually handles system wide dependencies itself. The ROS package depends on these standard ROS packages: `roscpp`, `image_transport`, `sensor_msgs`, `tf`, `camera_info_manager`, *nav_msgs* and `std_srvs`.

The installation follows the same steps needed usually to compile a ROS driver using *catkin <http://wiki.ros.org/catkin>*. Clone (or download and unpack) the driver to the `src` folder of a new or existing catkin *workspace <http://wiki.ros.org/catkin/Tutorials/create_a_workspace>* (e.g `~/catkin_ws/src`), then run `catkin_make` to compile it. Assuming you are compiling for ROS *Indigo*:

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/AutonomyLab/ardrone_autonomy.git -b indigo-devel
$ cd ~/catkin_ws
$ rosdep install --from-paths src -i
$ catkin_make
```

2.2 Usage

The driver's executable node is `ardrone_driver`. You can either run `rosrun ardrone_autonomy ardrone_driver` directly or use a custom launch file with your desired parameters. Example launch files are located in the `launch` directory.

2.3 Reading from AR-Drone

2.3.1 Update frequencies

Drone Update Frequencies: The drone's data transmission update frequency depends on *navdata_demo* parameter. When it is set to *1*, the transmission frequency is set *15Hz*, otherwise transmission frequency is set to *200Hz*. (*navdata_demo* is a numeric parameter not Boolean, so use 1 and 0 (not True/False) to set/unset it)

Driver Update Frequencies: The driver can operate in two modes: real-time or fixed rate. When the *realtime_navdata* parameter is set to True, the driver publishes any received information instantly. When it is set to False, the driver caches the received data first, then sends them at a fixed rate. This rate is configured via *looprate* parameter. The default configuration is: `realtime_navdata=False` and `looprate=50`.

Please note that if *looprate* is smaller than the drone's transmission frequency, some data is going to be lost. The driver's start-up output shows the current configuration. You can also use *rostopic hz* command to check the publish rate of the driver.

```
# Default Setting - 50Hz non-realtime update, the drone transmission rate is 200Hz
$ rosrun ardrone_autonomy ardrone_driver _realtime_navdata:=False  _navdata_demo:=0

# 200Hz real-time update
$ rosrun ardrone_autonomy ardrone_driver _realtime_navdata:=True _navdata_demo:=0

# 15Hz real-rime update
$ rosrun ardrone_autonomy ardrone_driver _realtime_navdata:=True _navdata_demo:=1
```

2.3.2 Legacy navigation data

Information received from the drone is published to the `ardrone/navdata` topic. The message type is `ardrone_autonomy::Navdata` and contains the following information: (Full specifications)

- `header`: ROS message header
- `batteryPercent`: The remaining charge of the drone's battery (%)
- **`state`: The Drone's current state:**
  - 0: Unknown
  - 1: Inited
  - 2: Landed
  - 3,7: Flying
  - 4: Hovering
  - 5: Test (?)
  - 6: Taking off
  - 8: Landing
  - 9: Looping (?)
- `rotX`: Left/right tilt in degrees (rotation about the X axis)
- `rotY`: Forward/backward tilt in degrees (rotation about the Y axis)
- `rotZ`: Orientation in degrees (rotation about the Z axis)
- `magX`, *magY*, *magZ*: Magnetometer readings (AR-Drone 2.0 Only) (TBA: Convention)

- `pressure`: Pressure sensed by Drone's barometer (AR-Drone 2.0 Only) (Pa)
- `temp` : Temperature sensed by Drone's sensor (AR-Drone 2.0 Only) (TBA: Unit)
- `wind_speed`: Estimated wind speed (AR-Drone 2.0 Only) (TBA: Unit)
- `wind_angle`: Estimated wind angle (AR-Drone 2.0 Only) (TBA: Unit)
- `wind_comp_angle`: Estimated wind angle compensation (AR-Drone 2.0 Only) (TBA: Unit)
- `altd`: Estimated altitude (mm)
- `motor1..4`: Motor PWM values
- `vx`, `vy`, `vz`: Linear velocity (mm/s) [TBA: Convention]
- `ax`, `ay`, `az`: Linear acceleration (g) [TBA: Convention]
- `tm`: Timestamp of the data returned by the Drone returned as number of micro-seconds passed since Drone's boot-up.

Note: The legacy Navdata publishing can be disabled by setting the *enable_legacy_navdata* parameter to *False* (legacy navdata is enabled by default).

2.3.3 IMU data

Linear acceleration, angular velocity and orientation of the drone is published to a standard ROS sensor_msgs/Imu message. The units are all metric and *TF* reference frame is set to drone's *base* frame. The covariance values are specified through `cov/imu_la`, `cov/imu_av` and `cov/imu_or` parameters. For More information, please check the Parameters section.

2.3.4 Magnetometer data

The normalized magnetometer readings are published to `ardrone/mag` topic as a standard ROS geometry_msgs/Vector3Stamped message.

2.3.5 Odometry data

New in version 1.4.

The driver calculates and publishes Odometry data by integrating velocity estimates reported by the drone (which is based on optical flow). The data is published as nav_msgs/Odometry messages to `ardrone/odometry` topic. The corresponding *TF* transform is also published as *odom* -> *base* transformation.

2.3.6 Selective Navdata (advanced)

You can access almost all sensor readings, debug values and status reports sent from the AR-Drone by using *Selective Navdata*. If you set any of following parameters to *True*, their corresponding *Navdata* information will be published to a separate topic. For example if you enable `enable_navdata_time`, the driver will publish AR-Drone time information to `ardrone/navdata_time` topic. Most of the names are self-explanatory. Please consult AR-Drone SDK 2.0's documentation (or source code) for more information. All parameters are set to False by default.

enable_navdata_trims	enable_navdata_rc_references	enable_navdata_pwm	enabl
enable_navdata_vision_raw	enable_navdata_vision_of	enable_navdata_vision	enabl
enable_navdata_trackers_send	enable_navdata_vision_detect	enable_navdata_watchdog	enabl
enable_navdata_video_stream	enable_navdata_games	enable_navdata_pressure_raw	enabl
enable_navdata_wind_speed	enable_navdata_kalman_pressure	enable_navdata_hdvideo_stream	enabl

Note: You can use `rostopic type ardrone/navdata_time | rosmsg show` command for each topic to inspect its published message's data structure.

2.3.7 Cameras

Both AR-Drone 1.0 and 2.0 are equipped with two cameras. One frontal camera pointing forward and one vertical camera pointing downward. This driver will create three topics for each drone: `ardrone/image_ra`, `ardrone/front/image_raw` and `ardrone/bottom/image_raw`. Each of these three are standard ROS camera interface and publish messages of type image transport. The driver is also a standard ROS camera driver, therefor if camera calibration information is provided either as a set of ROS parameters or through `ardrone_front.yaml` and/or `ardrone_bottom.yaml` files, calibration information will be also published via *camera_info* topics. Please check the FAQ section for more information.

- The *ardrone/** will always contain the selected camera's video stream and information.

The way that the other two streams work depend on the type of Drone.

AR-Drone 1

AR-Drone 1 supports four modes of video streams: Front camera only, bottom camera only, front camera with bottom camera inside (picture in picture) and bottom camera with front camera inside (picture in picture). According to active configuration mode, the driver decomposes the PIP stream and publishes pure front/bottom streams to corresponding topics. The *camera_info* topic will include the correct image size.

AR-Drone 2

AR-Drone 2 does not support PIP feature anymore, therefore only one of *ardrone/front* or *ardrone/bottom* topics will be updated based on which camera is selected at the time.

2.3.8 Tag detection

The `Navdata` message also contains information about the special tags that are detected by the drone's on-board vision processing system. To learn more about the system and the way it works please consult AR-Drone SDK 2.0's developers guide. These tags are detected on both video cameras on-board at *30fps*. To configure (or disable) this feature check the Parameters section.

Information about these detected tags are published through the following field of the *Legacy Navigation data* message.

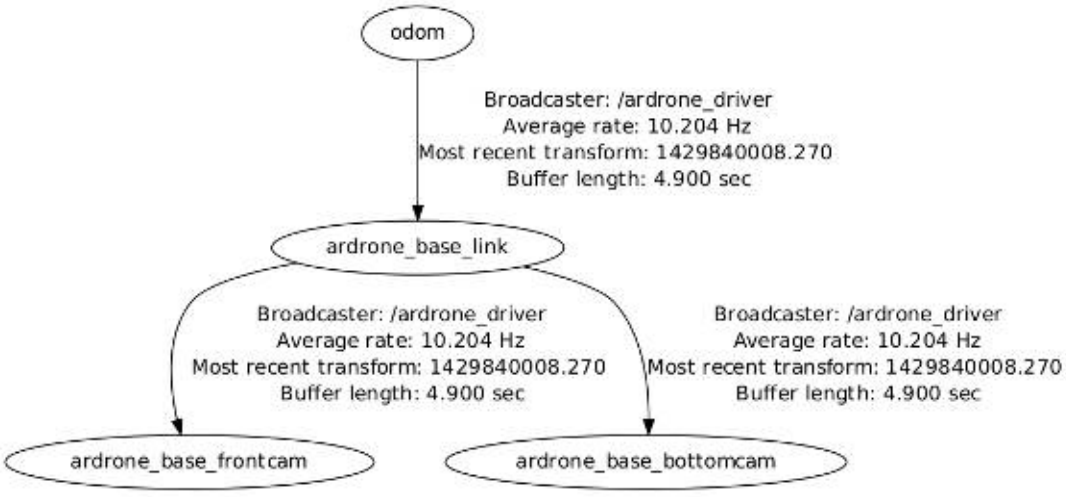
- `tags_count`: The number of detected tags.
- `tags_type[]`: Vector of types of detected tags (details below)
- `tags_xc[]`, `tags_yc[]`, `tags_width[]`, `tags_height[]`: Vector of position components and size components for each tag. These numbers are expressed in numbers between [0,1000]. You need to convert them back to pixel unit using the corresponding camera's resolution (can be obtained front *camera_info* topic).

- `tags_orientation[]`: For the tags that support orientation, this is the vector that contains the tag orientation expressed in degrees [0..360).

By default, the driver configures the drone to look for *oriented roundels* using bottom camera and *2D tags v2* on indoor shells (*orange-yellow*) using front camera. For information on how to extract information from *tags_type* field. Check the FAQ section in the end.

2.4 Coordinate frames

The driver publishes three TF transforms between these frames: `odom`, `${base_prefix}_link`, `${base_prefix}_frontcam` and `${tf_prefix}/${base_prefix}_bottomcam`. The *${base_link}* is the shared name prefix of all three reference frames and can also be set using Parameters, by default it has the value of *ardrone_base*.



The *frame_id* field in header of all published topics (navdata, imu, cameras) will have the appropriate frame names. All frames are ROS REP 103 compatible.

2.5 Sending Commands to AR-Drone

The drone will *takeoff*, *land* or *emergency stop/reset* if a ROS std_msgs/Empty message is published to `ardrone/takeoff`, `ardrone/land` and `ardrone/reset` topics respectively.

In order to fly the drone after takeoff, you can publish a message of type geometry_msgs::Twist to the `cmd_vel` topic:

```
-linear.x: move backward
+linear.x: move forward
-linear.y: move right
+linear.y: move left
-linear.z: move down
+linear.z: move up

-angular.z: turn left
+angular.z: turn right
```

The range for each component should be between -1.0 and 1.0. The maximum range can be configured using ROS **Parameters_** discussed later in this document.

2.5.1 Hover Modes

`geometry_msgs::Twist` has two other member variables `angular.x` and `angular.y` which can be used to enable/disable "auto-hover" mode. "auto-hover" is enabled when all six components are set to **zero**. If you want the drone not to enter "auto hover" mode in cases you set the first four components to zero, set *angular.x* and *angular.y* to arbitrary **non-zero** values.

2.6 Services

Note: You can find the API documentation of all services here.

2.6.1 Toggle Camera

Calling `ardrone/togglecam` service with no parameters will change the active video camera stream. (e.g `rosservice call /ardrone/togglecam`).

`ardrone/setcamchannel` service directly sets the current active camera channel. One parameter (**"**uint8 channel **"**) must be set when calling this service. For AR-Drone 1.0, valid values are *{0,1,2,3}* while for AR-Drone 2.0 these values are *{0,1}*. The order is similar to the order described in *Cameras* section.

2.6.2 LED Animations

Calling `ardrone/setledanimation` service invokes one of 14 pre-defined LED animations for the drone. The parameters are

- `uint8 type`: The type of animation which is a number in range [0..13]
- `float32 freq`: The frequency of the animation in Hz
- `uint8 duration`: The duration of the animation in Seconds.

The `type` parameter will map [in order] to one of these animations (srv/LedAnim.srv for more details):

```
BLINK_GREEN_RED, BLINK_GREEN, BLINK_RED, BLINK_ORANGE,
SNAKE_GREEN_RED, FIRE, STANDARD, RED, GREEN, RED_SNAKE,BLANK,
LEFT_GREEN_RIGHT_RED, LEFT_RED_RIGHT_GREEN, BLINK_STANDARD`
```

You can test these animations in command line using commands similar to:

```
$ rosservice call /ardrone/setledanimation 1 4 5
```

2.6.3 Flight Animations

Warning: Be extra cautious about using animations, especially flip animations.

Calling `ardrone/setflightanimation` service executes one of 20 pre-defined flight animations for the drone. The parameters are:

- `uint8 type`: The type of flight animation, a number in range [0..19]
- `uint16 duration`: The duration of the animation. Use 0 for default duration (recommended)

The `type` parameter will map [in order] to one of these pre-defined animations (check srv/FlightAnim.srv for more details):

```
ARDRONE_ANIM_PHI_M30_DEG, ARDRONE_ANIM_PHI_30_DEG, ARDRONE_ANIM_THETA_M30_DEG, ARDRONE_ANIM_THETA_30_
ARDRONE_ANIM_THETA_20DEG_YAW_200DEG, ARDRONE_ANIM_THETA_20DEG_YAW_M200DEG, ARDRONE_ANIM_TURNAROUND,
ARDRONE_ANIM_TURNAROUND_GODOWN, ARDRONE_ANIM_YAW_SHAKE, ARDRONE_ANIM_YAW_DANCE, ARDRONE_ANIM_PHI_DANC
ARDRONE_ANIM_THETA_DANCE, ARDRONE_ANIM_VZ_DANCE, ARDRONE_ANIM_WAVE, ARDRONE_ANIM_PHI_THETA_MIXED,
ARDRONE_ANIM_DOUBLE_PHI_THETA_MIXED, ARDRONE_ANIM_FLIP_AHEAD, ARDRONE_ANIM_FLIP_BEHIND, ARDRONE_ANIM_
ARDRONE_ANIM_FLIP_RIGHT
```

You can test these animations in command line using commands similar to:

```
rosservice call /ardrone/setflightanimation 1 0
```

while drone is flying.

2.6.4 Flat Trim

Calling `ardrone/flattrim` service without any parameter will send a "Flat Trim" request to AR-Drone to re-calibrate its rotation estimates assuming that it is on a flat surface. Do not call this service while Drone is flying or while the drone is not actually on a flat surface.

2.6.5 Record to USB Stick

Calling `ardrone/setrecord` service will enable and disable recording to the USB stick. Pass *1* to enable or *0* to disable this feature.

2.7 Parameters

2.7.1 AR-Drone Specific Parameters

The parameters listed below are named according to AR-Drone's SDK 2.0 configuration. Unless you set the parameters using `rosparam` or in your `launch` file, the default values will be used. These values are applied during driver's initialization phase. Please refer to AR-Drone SDK 2.0's developer's guide for information about accepted values. Not all the parameters are needed during regular usage of the AR-Drone, please consult the example launch file `launch/ardrone.launch` for frequently used ones:

```
altitude, altitude_max, altitude_min, ardrone_name, autonomous_flight, bitrate, bitrate_ctrl_mode,
bitrate_storage, codec_fps, com_watchdog, control_iphone_tilt, control_level, control_vz_max,
control_yaw, detect_type, detections_select_h, detections_select_v, detections_select_v_hsync,
enemy_colors, enemy_without_shell, euler_angle_max, flight_anim, flight_without_shell, flying_mode,
groundstripe_colors, hovering_range, indoor_control_vz_max, indoor_control_yaw, indoor_euler_angle_ma
latitude, leds_anim, longitude, manual_trim, max_bitrate, max_size, navdata_demo, navdata_options,
nb_files, outdoor, outdoor_control_vz_max, outdoor_control_yaw, outdoor_euler_angle_max, output,
owner_mac, ssid_multi_player, ssid_single_player, travelling_enable, travelling_mode, ultrasound_freq
ultrasound_watchdog, userbox_cmd, video_channel, video_codec, video_enable, video_file_index,
video_live_socket, video_on_usb, video_slices, vision_enable, wifi_mode, wifi_rate
```

This wiki page includes more information about each of above parameters.

2.7.2 Other Parameters

These parameters control the behavior of the driver.

- `drone_frame_id` - The "frame_id" prefix to be used in all *tf* frame names - default: *ardrone_base*
- `cov/imu_la`, `cov/imu_av` and ‘‘cov/imu_or‘: List of 9 covariance values to be used to fill *imu*‘s topic linear acceleration, angular velocity and orientation fields respectively - Default: 0.0 for all members (Please check the FAQ section for a sample launch file that shows how to set these values)
- `enable_legacy_navdata`: Enables *Legacy navigation data* publishing - Default: True

2.8 License

- The Parrot's license, copyright and disclaimer for ARDroneLib
- Other parts of the code are subject to BSD license

2.9 Contributors

Note: List of all commiters to the source repository

- **@mikehamer**
  - Added support for proper SDK2 way of configuring the Drone via parameter (critical bug fix) (More Info).
  - Support for zero-command without hovering (More info).
  - Full configurable Navdata support (More info).
  - Support for Real-time navdata and video publishing (More info).
  - Support for configurable data publishing rate.
- @JakobEngel
- **@sameerparekh**
  - Turn on and off USB stick recording
  - Seperate Magnetometer Topic
- **@devmax**
  - Flat trim service
  - Various comments for enhancements
- **@younata**
  - Enhanced Navdata for AR-Drone 2.0
- **@boris-il-forte & @lesire**
  - Catkinization (+)
- **@kbogert**
  - Move ARDroneLIB to an external project

    - Minimal changes to enable running multiple instances of driver on a single machine (+)
- **@garyservin**
    - Fix ffmpeg library link order
    - Moved header files to include directory
    - Add pressure unit
- **@v01d**
    - Odometry from optical flow
    - Use reception time for video streams

2.10 FAQ

2.10.1 Where should I go next? Is there any ROS package or stack that can be used as a tutorial/sample to use ardrone_autonomy?

Absolutely. Here are some examples:

- falkor_ardrone

"falkor_ardrone" is a ROS package which uses the "ardrone_autonomy" package to implement autonomous control functionality on an AR.Drone.

- tum_ardrone

State Estimation, Autopilot and GUI for ardrone.

- arl_ardrone_examples

This ROS stack includes a series of very basic nodes to show users how to develop applications that use the ardrone_autonomy drivers for the AR drone 1.0 and 2.0 quadrotor robot.

- AR Drone Tutorials

This repository contains the source-code for the Up and flying with the AR.Drone and ROS tutorial series, published on [Robohub](http://www.robohub.org).

- tum_simulator

AR Drone simulation in Gazebo, compatible with *ardrone_autonomy*.

2.10.2 How can I report a bug, submit patches or ask for a feature?

github offers a nice and convenient issue tracking and social coding platform, it can be used for bug reports and pull/feature request. This is the preferred method. You can also contact the author directly.

2.10.3 Why the *ARDroneLib* has been patched?

The ARDrone 2.0.1 SDK has been patched to 1) Enable the lib only build 2) Make its command line parsing compatible with ROS and 3) To fix its weird *main()* function issue. The patched SDK is being hosted on an external repository.

2.10.4 Why the wifi bandwidth usage is too much?

The driver has been configured by default to use the maximum bandwidth allowed to ensure the best quality video stream possible (please take a look at default values in parameters section). That is why the picture quality received from Drone 2.0 using this driver is far better than what you usually get using other software. If for any reason you prefer the lower quality* video stream, change *bitrate_ctrl_mode*, *max_bitrate* and *bitrate* parameters to the default values mentioned in the AR-Drone developer guide.

Note: Please note that lower quality does not mean lower resolution. By configuring AR-Drone to use bitrate control with limits, the picture gets blurry when there is a movement.

2.10.5 What is the default configuration for the front camera video stream?

Drone 1: *320x240@15fps UVLC Codec Drone 2*: *640x360@20fps H264 codec with no record stream*

2.10.6 How can I extract camera information and tag type from *tags_type[]*?

tag_type contains information for both source and type of each detected tag. In order to extract information from them you can use the following c macros and enums (taken from *ardrone_api.h*)

```
#define DETECTION_EXTRACT_SOURCE(type)   ( ((type)>>16) & 0x0FF )
#define DETECTION_EXTRACT_TAG(type)      ( (type) & 0x0FF )

typedef enum
{
  DETECTION_SOURCE_CAMERA_HORIZONTAL=0,   /*<! Tag was detected on the front camera picture */
  DETECTION_SOURCE_CAMERA_VERTICAL,       /*<! Tag was detected on the vertical camera picture at fu
  DETECTION_SOURCE_CAMERA_VERTICAL_HSYNC, /*<! Tag was detected on the vertical camera picture inside
  DETECTION_SOURCE_CAMERA_NUM,
} DETECTION_SOURCE_CAMERA;

typedef enum
{
  TAG_TYPE_NONE              = 0,
  TAG_TYPE_SHELL_TAG         ,
  TAG_TYPE_ROUNDEL           ,
  TAG_TYPE_ORIENTED_ROUNDEL  ,
  TAG_TYPE_STRIPE            ,
  TAG_TYPE_CAP               ,
  TAG_TYPE_SHELL_TAG_V2      ,
  TAG_TYPE_TOWER_SIDE        ,
  TAG_TYPE_BLACK_ROUNDEL     ,
  TAG_TYPE_NUM
} TAG_TYPE;
```

2.10.7 How can I calibrate the ardrone front/bottom camera?

It is easy to calibrate both cameras using ROS Camera Calibration.

First, run the camera_calibration node with appropriate arguments: (For the bottom camera, replace front with bottom)

```
rosrun camera_calibration cameracalibrator.py --size [SIZE] --square [SQUARESIZE] image:=/ardrone/fro
```

After successful calibration, press the *commit* button in the UI. The driver will receive the data from the camera calibration node, then will save the information by default in `~/.ros/camera_info/ardrone_front.yaml`. From this point on, whenever you run the driver on the same computer this file will be loaded automatically by the driver and its information will be published to appropriate *camera_info* topic. Sample calibration files for AR-Drone 2.0's cameras are provided in `data/camera_info` folder.

2.10.8 Can I control multiple drones using a single PC? or can I make my drone connect to a wireless router?

Since version 1.4, the driver supports connecting to multiple AR-Drones from a single PC. Thanks to efforts and patches provided by @kbogert. For more information please check this wiki page.

2.10.9 Is there any support for GPS (Parrot Flight Recorder)

Yes but it is experimental. The code is maintained in a separate branch (gps-waypoint). For more information see this documentation.