Second Session
Visual Navigation for Flying Robots Workshop

Jürgen Sturm, Jakob Engel, Daniel Cremers
Computer Vision Group, Technical University of Munich

Bergendal Meetings
17.06.2013

In this session, we will implement position control for the AR.Drone. We provide two alternative exercises that you can follow depending on your interests:

- In continuation of the morning session, we use the pose estimate from the EKF and feed it into a simple PID controller. The PID controller generates motion commands that it sends to the quadrocopter. As a result, the AR.Drone can hover robustly above a visual marker.
- Alternatively, you can try out the `tum_ardrone` package that features a marker-less visual SLAM front-end (based on PTAM), an EKF that properly handles time delays, and a PID controller. A high-level controller allows you to specify a flight figure using a simple scripting language.

Exercise 1: Simple position control using a PID controller

In this exercise, you will enable the quadrocopter to hover on the spot without accumulating drift. To get started, open the file `controller.cpp` in the `visnav_exercise` package and inspect its functions.

(a) Check out to which topics the controller subscribes to and to which topics it publishes.

(b) The PID controller is implemented in the function `PidController::getCommand`. Inspect this function.

(c) The function `ArdroneController::calculateContolCommand` computes the next control command for three axes (x/y/yaw) based on the current pose estimate. Check out which error signal is fed into the three PID controllers.

(d) Start RVIZ and add a marker display for `cmd_marker`. On this topic, the controller will publish an arrow pointing in the direction of the steering command.

(e) Start your controller using

```
$ rosrun visnav_exercise ardrone_controller
```

(f) Replay the `control_flight.bag` file using

```
$ rosbag play control_flight.bag
```

Check visually in RVIZ whether the arrows point in the right direction.

(g) Run `rxplot` to visualize the current pose estimate and the current command. Replay the bag file again. You can use the following line for accomplishing this for the xy-pose and velocity commands

```
$ rxplot /quadcopter_state/x:y /cmd_vel/linear/x:y
```

(h) Run `rosrun dynamic_reconfigure reconfigure_gui` to inspect and change the coefficients of your controllers. Play around with different values to understand their effect on the control commands.

(i) Now try your controller on the quadrocopter. Connect to the quadrocopter via wifi and launch the ROS driver as explained on the previous exercise sheet. Start with a P-gain of 0.5 for the translational controllers, a P-gain of 0.1 for the yaw controller, and set the I- and D-gains (initially) to zero. Modify the coefficients until you are satisfied with the resulting behavior.

(j) **Optional:** Instead of using a static goal location, implement a function that slowly shifts the goal location from the first marker to the second marker. Alternatively, implement a function that slowly moves the goal location around the first marker along a (small) circle. To update the goal location, you can use the `setGoalPose` method.

Exercise 2: Marker-less autonomous flight

In this exercise, we will use the `tum_ardrone` package which provides (1) a visual SLAM module based on PTAM, (2) implements an EKF which considers the time delays, and (3) a simple PID controller to keep positions or follow waypoints. More information on the approach can be found in the following paper[1] and on the corresponding ROS wiki page[2]:

> J. Engel, J. Sturm, D. Cremers. **Camera-Based Navigation of a Low-Cost Quadrocopter**, In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012.

[1] `http://vision.in.tum.de/_media/spezial/bib/engel12iros.pdf`
[2] `http://www.ros.org/wiki/tum_ardrone`

Abstract In this paper, we describe a system that enables a low-cost quadrocopter coupled with a ground-based laptop to navigate autonomously in previously unknown and GPS- denied environments. Our system consists of three components: a monocular SLAM system, an extended Kalman filter for data fusion and state estimation and a PID controller to generate steering commands. Next to a working system, the main contribution of this paper is a novel, closed-form solution to estimate the absolute scale of the generated visual map from inertial and altitude measurements. In an extensive set of experiments, we demonstrate that our system is able to navigate in previously unknown environments at absolute scale without requiring artificial markers or external sensors. Furthermore, we show (1) its robustness to temporary loss of visual tracking and significant delays in the communication process, (2) the elimination of odometry drift as a result of the visual SLAM system and (3) accurate, scale-aware pose estimation and navigation.

Installation This sheet assumes that all preparatory steps from the "installation instructions" sheet have been done (i.e., a working ROS environment, pre-compiled AR.Drone drivers, etc). In particular, you should have the `tum_ardrone` package contained in the git repository - you only need to build it:

```
$ rosmake tum_ardrone
```

The following is a condensed version of the full documentation available at the ROS wiki pages[3]. An offline-version is saved in `tum_ardrone/doku`:

Run

(a) Start a ROS core

```
$ roscore
```

(b) Open four new consoles, and run the following four ROS nodes

```
$ rosrun ardrone_autonomy ardrone_driver
$ rosrun tum_ardrone drone_stateestimation
$ rosrun tum_ardrone drone_autopilot
$ rosrun tum_ardrone drone_gui
```

Manual Keyboard control

- Focus the `drone_gui` window
- Press ESC to activate keyboard control
- Fly around with the keyboard (q,a: fly up and down; i,j,k,l: fly horizontally; u,o: rotate yaw ; F1: toggle emergency; s: takeoff; d: land)

[3] `http://www.ros.org/wiki/tum_ardrone`

Manual Joystick control Assuming a plugged-in PS3 six-axis controller, with set rights, run `$ rosrun joy joy_node`

- left joystick is horizontal position control
- right joystick is height and yaw control.
- L1 to take off, release L1 to land.
- R1 to toggle emergency state.

By moving any of the two joysticks, the Control Source is immediately set to Joystick. This can be used for safety (autopilot does wired stuff → immediately take over by hand, disabling the autopilot and enabling manual control).

Using the Autopilot

- Place the quadrocopter on the ground, with enough open space around it. There should be some structure with enough keypoints in front of it, ideally at a distance of 2m to 10m.
- Load contents of file initDemo.txt (left, below the big text field in the GUI) click Clear and Send (best to click Reset first). The quadrocopter will takeoff and initialize PTAM, then fly a small figure (1m up, 1m down, 1x1m horizontal square).
- You can interrupt the figure anytime by interactively setting a relative target: click on video (relative to current position); see here. First fly up at least 1m to facilitate a good scale estimate, do not start e.g. by flying horizontally over uneven terrain.
- Always have a finger on ESC or on the joystick for emergency-keyboard control.