Visual Navigation for Flying Robots
D. Cremers, J. Sturm, N. Engelhard
Summer Term 2012

Computer Vision Group
Institut für Informatik
Technische Universität München

Sheet 2

Topic: Motion Models and Robot Odometry

Submission deadline: Tue, 22.05.2012, 10:15 a.m.
Hand-in via email to visnav2012@cvpr.in.tum.de

Preliminaries: Setting up the drone

To simplify the setup, we created a SVN-Repository[1] with all the code needed. Make sure to clone it into your ROS_PACKAGE_PATH. The SVN contains the following folders:

- `ex_2`: The exercise stub for this sheet. Run `rosmake ex_2` to compile it together with the `joy`- and `ardrone_joystick`-nodes.
- `ardrone_brown`: Improved version of the Ardrone driver from Brown university. It now also sends the image from the down-facing camera if this camera is activated. There are two ways to use this node:
  - Binary version (recommended on lab computers): Just use the compiled version in the bin-folder.
  - Alternative: Build from Source by running `sudo ./build_sdk.sh` to compile the SDK library and subsequently `rosmake` to create the ROS node.
- `joy`: The ROS driver for a joystick[2].
- `ardrone_joystick`: This node translates the joystick commands into `cmd_vel` messages which are then interpreted by the Ardrone driver. The `ardrone_joystick`-node expects the PS3-joystick to be at `/dev/input/js0`. Some laptops however treat their accelerometer also as a joystick. The PS3 joystick will then probably be on `/dev/input/js1`. In this case just uncomment the third line in the `launch/teleop.launch` file. If there are still problems, check the tutorial here[3].
- `ar_recog`: The marker recognition. `./getAndBuildARToolkit.py` the Toolkit and `rosmake` the package. To run the node `cd` into `bin` and execute `./ar_recog image:=/ardrone/image_raw_small` to run the marker recognition on the down-facing camera. The ex2 stub expects the β- and ζ-marker, but of course

[1] `https://svncvpr.informatik.tu-muenchen.de/lecturematerial-visnav`
[2] `http://www.ros.org/wiki/joy`
[3] `http://www.ros.org/wiki/joy/Tutorials/ConfiguringALinuxJoystick`

you can also adapt the code if you want to use different markers.

- `bag_files`: These bags contain two flights of the quadrocopter.

  ROS Fuerte Users: See hints on last page

Preliminaries: First flight

(a) Read and memorize the safety warnings.

(b) Connect the battery to the quadrocopter. After a few seconds, the LEDs should switch to green. After this happens, you can connect with your computer over wireless to the quadrocopter. On the lab computer, ask Nikolas for a wireless dongle and then run `sudo ardrone-connect` to connect. On your private laptop, just use the network manager.

(c) Start the driver via `rosrun ardrone_brown ardrone_driver`.

(d) Start the `ardrone_joystick`-node via `roslaunch ardrone_joystick teleop.launch`. This launch-file[4] also starts the `joy`-node that is the interface between the joystick and the ROS-Network. The connect the joystick with the computer press the PlayStation-button in the middle of the controler. The axes and buttons are assigned as follows:

- The **R1 button** toggles the emergency state of the robot. Pressing R1 while flying will stop the rotors immediately. If the LED beneath the rotors are red (for example, after a crash), press R1 to reset the drone.
- The **L1 button** starts the motors of the quadrocopter. It also works as a deadman button so that the robot will land if you release it during flight. The quadrocopter will ascend to one meter above ground and tries to hold this position.
- 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. This can also be done by executing `rosservice call /ardrone/togglecam`.

Exercise 1: Record a bag file

In this exercise, you will take a group picture using the quadrocopter and record an own bag file.

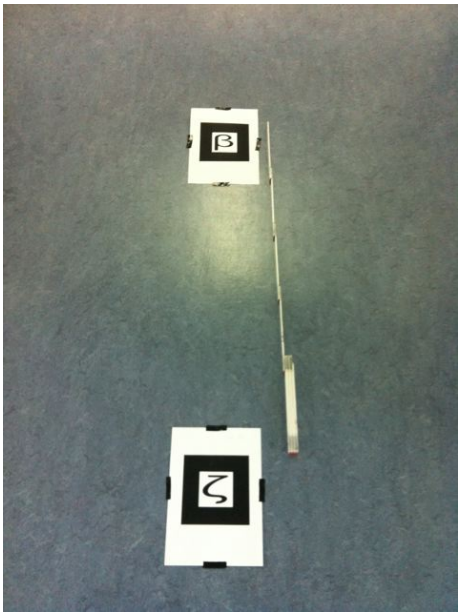
[4] `http://www.ros.org/wiki/roslaunch/`

(a) Setup your computer (as described above) and connect to the quadrocopter. Use the joystick to control the quadrocopter. Fly a short round in the robot lab and practice your flying skills.

(b) Use `image_view` to watch the live images from the quadrocopter. Use this to take a group picture of your team and add it to your report.

(c) Use `rostopic list` to see what topics are currently being published. Which of these topics do you need for Exercise 2?

(d) Print the markers attached to this sheet on A4 paper and attach them in a distance of 1m to the floor in the following orientation:



(e) Use `rosbag record /topic1 /topic2` to record the relevant topics. Use `rosbag info file1.bag` to see the recorded topics in your bag file and send us the output. Important: To run the marker detection on the vertical camera, you will also have to record the `/ardrone/camera_info`-topic.

Exercise 2: Extended Kalman filter

In this exercise, you will learn how to use a Kalman filter to estimate the pose of the robot from a bag file or live data. You can either use the bag file from the repository, or your own bag file recorded in Exercise 1. We provide you with a C++ framework of an extended Kalman filter for which you will have to implement the correction step. For simplicity, we model the quadrocopter only in the 2D plane, i.e., its state at time t is described by $\mathbf{x}_t = \begin{pmatrix} x_t & y_t & \psi_t \end{pmatrix}^\top$ and its odometry is given by $\mathbf{u}_t = \begin{pmatrix} x_t & y_t & \psi_t \end{pmatrix}^\top$.

(a) Start `rviz` and add a grid, tf and marker display as in the previous exercise sheet.

(b) Compile the Kalman filter using `rosmake ex_2`.

(c) Start the Kalman filter by running `rosrun ex_2 visnav_2`. Replay the bag file using `rosbag play` and watch the result in RVIZ. Take a screen shot of RVIZ after a few seconds with the covariance ellipse.

(d) Assume that the quadrocopter drifts more (say two times more) than this in the global x-direction, for example, because there is strong wind in this direction. Modify the process noise matrix Q accordingly in the source code and re-run the experiment. Take another screen shot (on the same bag file) and send us the result.

(e) What would be a good way to determine the noise values empirically? Describe briefly an experimental setup that could be used to determine these values.

(f) The framework detects markers in the environment and provides (in the case of a detection) an observation $\mathbf{z} = \begin{pmatrix} x & y & \psi \end{pmatrix}^\top$ *relative* to the frame of the quadrocopter. Specify the observation function $\mathbf{z} = h(\mathbf{x})$ and compute its derivative (Jacobian) $H = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}$.

(g) Implement the correction step in the Kalman filter using the observation function and the Jacobian. Take care when you compute the difference between angles (as required for the computation of $\mathbf{z}_t - h(\mu_t)$ in the correction step). One simple solution is to normalize the angle afterwards, for example, using $\psi_{\text{normalized}} = \text{atan2}(\sin(\psi), \cos(\psi))$.

Submission instructions

A complete submission consists both of a PDF file with the solutions/answers to the questions on the exercise sheet and a TGZ (or ZIP) file containing the source code that you used to solve the given problems. Make sure that your TGZ file contains all files necessary to compile and run your code, but it should not contain any build files or binaries (`make clean`, `rm -rf bin`). Please submit your solution via email to `visnav2012@cvpr.in.tum.de`.

Hints: ROS Fuerte

Although not officially supported, here are some hints for the ROS Fuerte users:

- To compile `ex_2`, you will have to adapt the `CMakeLists.txt` and `manifest.xml`.
  - `CMakeLists.txt`: add
    find_package(Eigen REQUIRED)

include_directories(${EIGEN_INCLUDE_DIRS})

- `manifest.xml`: replace
  <depend package="eigen" />
  with
  <rosdep name="eigen" />

- If you have problems to link the OpenCV libraries, check the `manifest.xml` of your `opencv2`-pkg. You will find something like
  `<cpp cflags="‘pkg-config opencv-2.3.1 --cflags‘" lflags="‘pkg-config opencv-2.3.1 --libs‘"/>` Make sure that the `pkg-config opencv-2.3.1 --libs` creates the right linker-flags. (Something like `-lopencv_contrib -lopencv_legacy -lopencv_objdetect ...`)

Hints: Compiling ar_recog on the laptop

Some of the participants had problems to compile the `ar_recog` on their own machine. There is one solution which worked at least on one computer and is worth a try: It can be found here[5]. Copy the `ARToolKit-2.72.1.tgz` directly to the `src`-folder (don't use the script). Untar and apply the patch. During `./Configure`, the answers were "3, yes(!), no, no". Then `make` and `rosmake ar_recog`.

[5] `http://ubuntuforums.org/showthread.php?t=1690625&page=2`