Computer Vision Group
Department of Informatics
Technical University of Munich

Sheet 2

Topic: Motion Models and Robot Odometry

Submission deadline: Tue, 21.05.2013, 10:15 a.m.
Hand-in via email to visnav2013@vision.in.tum.de

Introduction: Please read first!

In this exercise sheet, you will learn how to use a Kalman filter to estimate the pose of the AR.Drone from it's sensor measurements, using visual markers attached to the ground: In the **first exercise**, you will set up everything required to fly the drone with a joystick, and record a .bag file of the drone flying above visual markers. In the **second exercise**, you will develop and implement the Kalman filter, and use it to estimate the drone's trajectory from the recorded .bag file from exercise one.

As we have more groups than AR.Drones we provide an example .bag file, such that some of you can start with the second exercise (for which you do not require an AR.Drone or joystick).

Exercise 1: Manual Flight

(a) Set up the joystick: If you are working on your own laptop, you first have to install the ROS joystick packages using

```
sudo apt-get install ros-fuerte-joystick-drivers
```

Now plug in the joystick, and start `roscore` and the joystick driver using `rosrun joy joy_node`. Inspect the topic `/joy`, and verify that the joystick works - you might have to press the P button once to initialize it.

(b) Set up the control node: We provide a node that translates the raw joystick messages to the correct control commands sent to the drone, which you can find here:

```
git://github.com/tum-vision/ardrone_joystick.git
```

Clone it into your workspace, run `rosmake ardrone_joystick`, and start the node using `rosrun ardrone_joystick ardrone_teleop`. 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 switch 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 roll and pitch angle of the drone. 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`.
- The triangle button can be used to switch off the on-board position stabilization: Per default, the drone hovers (i.e. stabilizes it's horizontal position by keeping v_x and v_y at zero) when you do not touch the left control stick. It even actively decelerates when letting go of the left control stick.

  Pressing and holding the triangle button will switch this off, i.e. give you direct control over roll and pitch at all times – note how this leads to rapid drift in horizontal position.

(c) First Flight: Connect the drone as in the very first exercise sheet. Display the camera image using

```
rosrun image_view image_view image:=ardrone/image_raw
```

Fly a short round in the robot lab and practice your flying skills. **Take a group picture of your team and add it to your report**.

(d) Graph Visualization: Use `rxgraph` to visualize the running nodes and used topics. **Take a screenshot, and attach it to your report.**

(e) Record a .bag File: [put marker on ground]. Switch to the bottom camera (the AR.Drone only streams one of the two camera images to the PC), and use rosbag to record approximately a 30s-flight. The marker should regularly be visible in the camera image, but should leave it temporarily. You will need to record at least the camera image topic, the camera info topic, and the ardrone navdata topic, i.e.:

```
rosbag record /ardrone/navdata /ardrone/image_raw
        /ardrone/camera_info -O flight.bag
```

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 website, or your own bag file recorded in Exercise 1. For the screenshots in your report, please use the bag file from our website. **Exit all running nodes and disconnect the drone and joystick, you will not need them for this exercise.**

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 $\mathbf{x}_t = \begin{pmatrix} x_t & y_t & \psi_t \end{pmatrix}^\top$. The prediction function is given by the odometry from the previous exercise sheet – using the measured yaw angle and horizontal velocities.

(a) Download the C++ framework for the Kalman Filter from

```
git://github.com/tum-vision/visnav2013_exercise2.git
```

Rename the folder to `ex_2` and build it using `rosmake`.

(b) Download and compile the visual marker detection node from

```
http://robotics.ccny.cuny.edu/git/ccny-ros-pkg/ccny_vision.git
```

Launch the marker detection node using the launch file provided in `ex_2`.

(c) Start `rviz` and add a grid, tf display, a marker-display and a markerArray-display as in the previous exercise sheet. Remember to change the base coordinate system to `/world`, and to select the correct topics.

(d) 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 screenshot after a couple of seconds with the covariance ellipse**. The Kalman filter also publishes the estimated pose on `/ardrone/filtered_pose`. Visualise the full estimated two-dimensional trajectory from the bag file using

```
rxplot -p 48 /ardrone/filtered_pose/linear/x:y
```

i.e., restart the playback after starting `rxplot`. **Take a screenshot and attach it to your report.**

(e) 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, re-run the experiment, **and take another two screenshots.**

(f) 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.**

(g) 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) $H = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}$.**

(h) 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 $\psi_{\text{normalized}} = \operatorname{atan2}(\sin(\psi), \cos(\psi))$.

(i) use `rxplot` again to visualize the - now corrected - estimated two-dimensional trajectory from the bag file, **take a screenshot of the trajectory, and add it to your report.**

(j) From the screenshots, **estimate roughly how far the pose-estimate drifted** without visual markers over the 48s flight (in meters).

Submission instructions

A complete submission consists both of a PDF file with the solutions/answers to the **bold** 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 `visnav2013@vision.in.tum.de`.