Visual Navigation for Flying Robots
D. Cremers, J. Stückler, V. Usenko, J. Engel
Summer Term 2015

Computer Vision Group
Department of Informatics
Technical University of Munich

Sheet 4

Topic: Robot State Estimation and Control

Submission deadline: Monday, 18.05.2015, 12:00
Hand-in via email to visnav15@vision.in.tum.de

General Notice

The exercises should be done in teams of two to three students. Each student in a team must be able to present the solution to the tutors during the exercise sessions on a lab PC in room 02.05.014. The presentations and solutions will be graded and will count for the final grade of the lab course. If you have not yet done so, please register yourself together with your team members on the team list in room 02.05.14.

We will use ROS Indigo on Ubuntu 14.04 in this lab course. It is already installed on the lab computers. If you want to use your own laptop, you will need to install these versions of Ubuntu and ROS. Please read the ROS and OpenCV documentation for further reference.

Introduction

The goal of this exercise is to acquire practical experience with controlling a flying robot in a simulator. You will estimate the robot position using IMU and 6D pose sensors and write a simple PID controller to make the robot hover on spot and resist external disturbances.

Exercise 1:

Download the code sample for this exercise provided on the course website:

- `https://vision.cs.tum.edu/teaching/ss2015/visnav_ss2015/slides`

To get you started, it contains a flying robot simulator and a solution skeleton code.

In this exercise you will implement a position PID controller for the flying robot assuming the ground truth pose of the robot's center of mass is available.

(a) Get familiar with the simulator and the skeleton code of the exercise. Launch the simulator by running the following command:

```
roslaunch euroc_simulation_server ex4_setting1.launch
```

Launch the exercise solution in a different terminal:

```
roslaunch ex4_solution ex4.launch
```

Verify that launching the solution unpauses the simulator and it starts publishing messages. What messages does the simulator publish? What are the frequencies of these messages?

(b) Inspect the uav_controller.hpp class in the solution source code. In the constructor this class initializes some constants, such as gravity, robot mass, noise characteristics of the sensor. After that it initializes publishers and subscribers and unpauses the simulator. The groundTruthPoseCallback function receives the ground truth pose from the simulator, computes the velocity and stores pose and velocity in the class variables. The getPoseAndVelocity function returns the current estimate of pose and velocity. It will be used later to switch easily between ground truth measurements and filter output.

(c) Fill the computeDesiredForce function with the code to compute the desired force using the PID controller for hovering at (0,0,1) with zero velocity. The desired force can be computed using the following formula:

$$\ddot{x} = k_p(x_d - x) + k_d(\dot{x_d} - \dot{x}) + k_i \int (x_d - x)dt,$$

where x_d and $\dot{x_d}$ are desired position and velocity, x and $\dot{x}$ are current position and velocity and k_p, k_d, k_i are proportional, differential and integral gains of the PID controller.

(d) The simulator provides a low-level controller for the robot that expects a command consisting of desired roll, pitch angles, trust values and yaw rate and control the motors to maintain the desired values. Fill the computeCommandFromForce function with the code to compute desired roll, pitch angles and thrust. For this exercise you can set the yaw rate to zero. To obtain roll pitch and thrust you can use the following equations:

$$\begin{aligned}
\phi_d &= \frac{1}{g}(\ddot{x}_1 \sin\psi - \ddot{x}_2 \cos\psi), \\
\theta_d &= \frac{1}{g}(\ddot{x}_1 \cos\psi + \ddot{x}_2 \sin\psi), \\
T_d &= \ddot{x}_3 + mg,
\end{aligned}$$

where ϕ_d is desired roll angle, θ_d desired pitch angle, T_d desired thrust and ψ current yaw angle.

(e) Write a sendControlSignal function that will obtain a current pose and velocity estimate from getPoseAndVelocity function, compute the desired force with computeDesiredForce function, transform it into the message with computeCommandFromForce and publish it.

(f) Publish the control message with the rate of the most high frequency sensor. You can for example call the sendControlSignal in the end of IMU callback.

(g) Test the controller in different settings (ex4_setting1.launch, ex4_setting2.launch, ex4_setting3.launch). In the setting 1 no external forces are applied to the robot, in the setting 2 at 40s second of the simulation a constant wind starts blowing and in setting 3 the wind gust blows just several seconds and then stops. Tune the PID controller such that the robot maintains a stable flight in all those settings.

Exercise 2:

In this exercise you will implement a UKF filter that will fuse noisy measurements from the sensors running at different frequencies to get a reliable estimate of the robot state. The simulator will provide you messages from two sensors: IMU that is mounted approximately in the center of mass and provides high-frequency measurements that are subject to Gaussian noise and have a constant bias; Generic 6D pose sensor which is mounted with some offset from the center of mass, provides low-frequency measurements that are subject to Gaussian noise.

(a) Inspect the se3ukf.hpp class in the solution source code. In particular have a look at compute_sigma_points, compute_mean and compute_mean_and_covariance functions. The first of them computes sigma points from the current mean and covariance, the other two perform reverse operation. They compute new mean and covariance from the current sigma points.

(b) Implement the predict function that computes the predicted state distribution from the current state distribution and IMU measurements. To do that you should first compute the sigma points of the current state, for each of the sigma points apply the IMU motion model, compute new mean and covariance from the transformed sigma points. Assuming the IMU is located close to the center of mass you can use the following IMU model:

$$\begin{aligned} p_{t+1} &= p_t + v_t \Delta t, \\ v_{t+1} &= v_t + (R_t(a - b_{at}) - g)\Delta t, \\ R_{t+1} &= R_t \exp((\omega - b_{\omega t})\Delta t), \end{aligned}$$

where p is position, $R \in SO(3)$ is orientation and v is velocity of the IMU expressed in the world coordinate frame; a and ω represent accelerometer and gyroscope measurements, and b_a and b_ω represent their biases accordingly.

(c) Fill the measurePose function to apply 6D measurements to the filter.

(d) Add UKF filter to the UAVController class. Initialize it with provided initial pose and covariance. Fill imuCallback and pose1Callback to fuse the sensor measurements using the UKF. Please note that the coordinate frame of the 6D pose sensor is not the same as the IMU coordinate frame, so you should first transform the measurements to the IMU frame using the transformation T_imu_cam that is provided in the UAVController class.

(e) Change the getPoseAndVelocity function to provide the pose and velocity estimates from UKF when use_ground_thruth_data variable is set to false.

(f) Test your controller from the Exercise 1 with the UKF state estimation instead of ground truth values. Verify that it works in different settings. What difference do you observe compared to the controller that works with ground truth?

(g) Verify that your controller now works without ground truth pose. For that you can launch the simulator as follows:

```
roslaunch euroc_simulation_server ex4_setting1.launch
    enable_ground_truth:=false
```

Submission instructions

A complete submission consists both of a PDF file with the solutions/answers to the questions on the exercise sheet and a ZIP file containing the source code that you used to solve the given problems. Note all names of your team members in the PDF file. Make sure that your ZIP file contains all files necessary to compile and run your code, but it should not contain any build files or binaries. Please submit your solution via email to `visnav15@vision.in.tum.de`.