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

Computer Vision Group
Institut für Informatik
Technische Universität München

Sheet 3

Topic: Robot Control

Submission deadline: Tue, 12.06.2012, 10:15 a.m.
Hand-in via email to visnav2012@cvpr.in.tum.de

Preliminaries: Setting up your installation

Similar to the previous exercise sheet, we provide you with all tools you need to implement the exercise. You can find all code in the course SVN-repository[1]. Run `svn update` in this folder to make sure that you have the latest version of all packages. Then run `rosmake ex_3` to compile the stub and all of its dependencies.

Exercise 1: Implementation of Position Control

In this exercise, you will enable the quadrocopter to hover on the spot without accumulating drift.

(a) Specify the control law $u = f(e)$ of a SISO (single input, single output) proportional controller, i.e., specify how the control command $u \in \mathbb{R}$ is computed from the error $e = x_{\text{des}} - x_{\text{current}} \in \mathbb{R}$.

(b) Implement this proportional controller in the file `pid_controller.cpp` in the function `PID_Controller::getCommand`. Note that this class has a member variable `c_proportional` that specifies the gain. Please use this variable because this will simplify to tune the gains during the flight later on.

(c) In this exercise, we want to control the x-position, y-position, and yaw-angle of the quadrocopter, using three independent controllers. Specify how the error signals $\mathbf{e} = (x_e, y_e, \psi_e)^\top$ for each of these controllers can be computed from the current pose $\mathbf{x} = (x_x, y_x, \psi_x)^\top$ and the goal pose $\mathbf{g} = (x_g, y_g, \psi_g)^\top$.

(d) Implement the function `sendNewCommand` in `main.cpp` as derived before. Pass the computed error values to three instances of the P-controller and store the resulting steering commands in the `linear` variable.

(e) **Optional:** Extend the proportional controller in a full PID controller. Please note that there are also variables `c_integral` and `c_derivative` for the controller gains (or coefficients).

[1] `https://svncvpr.informatik.tu-muenchen.de/lecturematerial-visnav`

Exercise 2: Autonomous Flight

In the second exercise, you will try out your P(ID) controller from the previous exercise and tune its gains.

(a) 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. Start your controller and replay the bag file `test_control.bag`. Check whether these arrows point in the right direction. Make a screen shot and attach it to your report.

(b) Run `rxplot` to visualize the current pose estimate and the current command. Replay the bag file again, make a screen shot of rxplot, and attach it to your report. 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
```

(c) 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.

(d) 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.

(e) **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.

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`.