Robot Localization

Master Semester Project LCAV

Frederike Dümbgen

frederike.duembgen@epfl.ch

R. Scheibler, M. Vetterli

December 11, 2015

1 Introduction

2 Visual Localisation

Introduction to visual localisation, explanation of procedure.

2.1 Intrinsic Calibration

Implemented algorithm ...

2.2 Extrinsic Calibration

Reference points Checkerboard vs. 6 points.

Algorithm SolvePnP: Description Levenberg-Marquardt algorithm Why not RANSAC.

2.3 Triangulation

Introduction

DLT fixed height using SVD, Hartley-Zissman algorithm.

DLT free height

2.4 Image Processing

In order to localize the robot 3D, the position of the location of the robot and of the N reference points need to be determined. The implemented procedure is based on bright colored, circular reference points and is semi-automatic.

While the precise location of the centers of the reference points is calculated automatically, the user defines the regions of interest and the relative position of the reference points by clicking in order on these points. (see Figure). Examining the picture's HSV representation (Figures and), one can see that a combination of the H and S values gives can deliver a good criteria for the extraction of the bright colored points. Hence, the colors are extracted in two steps: first a rough filter is applied to the image, which is then refined by only keeping the main part of the colors left. (FORMULAS) (Figure, green lines mark the contours of the extracted color regions).

-> algorithms: color extraction contours circles

3 Echo-SLAM

3.1 Sine Sweep Generation

3.2 RIR recording

3.3 Analysis

Calibration It is required to differentiate the delay induced by the physical distance between microphone, walls and the speakers from the delay induced by the audio system itself. Therefore an analysis of the latency of the audio system is performed.

For this purpose, the speaker is placed at a well known position with respect to the microphone, so that the physical delay Δt_ph can be precisely calculated. It is then sufficient to get the total delay of the signal, Δt_{tot} which is composed of the physical delay and the latency ($\Delta t_{tot} = \Delta t_{ph} + \Delta t_l$). The total delay is found by sending a reference signal u_{N1} (in this case, a white noise) and recording the response of the microphone, y_{N2}.

The response is a scaled and delayed noisy version of the input. Finding the delay comes back to laying the output signal over the input signal with different phase lags until we get a maximum similarity. The lag corresponding to this maximum is the total delay between the input and the output. The tool that performs these steps is the cross-correlation, which, for discrete signals can be written as:

$$r_{uy}[k] = \sum_{n=-\infty}^{\infty} u[n]y^*[n-k] \qquad k = 0, \pm 1, \pm 2, ... \tag{1}$$

As the cross-correlation is quite computationally expensive, only the first N_{max} samples of both input and output signals are correlated, which is sufficient if we choose N_{max} much bigger than the sample index of the expected delay (for example, $N_{max} = 2\text{ s} \times F_s = 88200$)

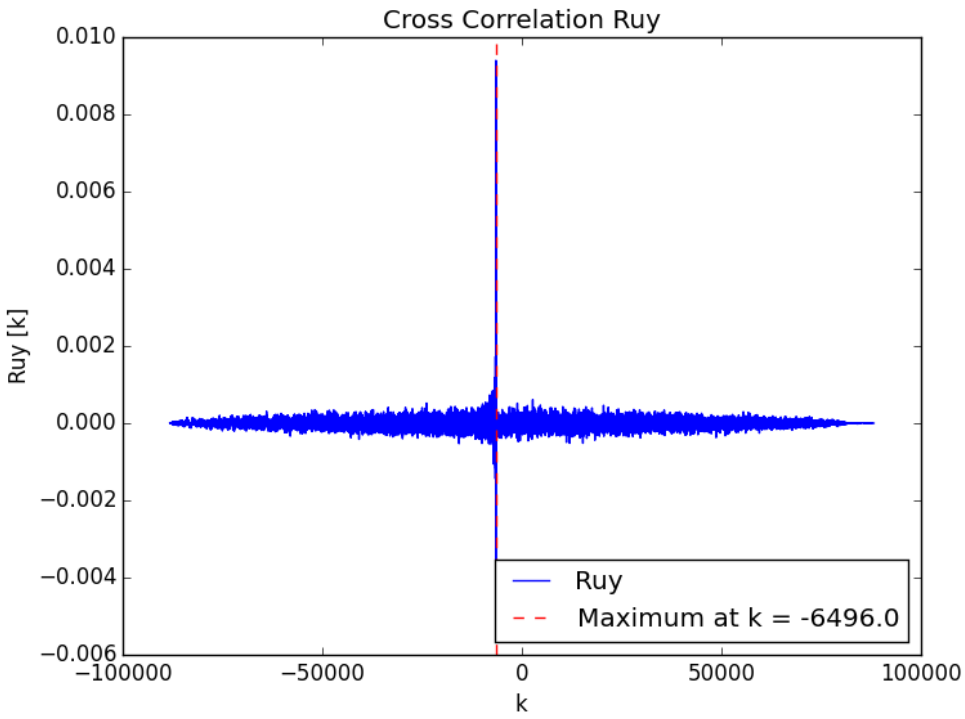

Figure 1: Cross correlation of white noise for latency estimation

From Figure 1, we can see that the maximum occurs at sample $k_{max} = -6496$, which corresponds to a delay of $\Delta t_{tot} = k_{max}/F_s = 147.3$ ms. The distance between microphone and speaker being

$d = 660mm$, we find $\Delta t_{ph} = d/c = 1.9$ ms, so the estimated latency of the sound system is about $\Delta t_l = 145.4$ ms.

4 Robot movement

4.1 Control

How robot is controlled.

4.2 Odometry

Odometry algorithm used

5 Experimental Setup

5.1 Router

The Router used is a TP_LINK TL_MR3020 up to 150 Mbps

computer to router: (when connected to internet upload: 45, download: 50) sending 11MB, 11536384 (test10.zip) raspberry to router: 1.57 MB/s, (in 7.0s) computer to router: 1.73 MB/s (in 6.4s)

5.2 Camera Setup

Hardware The camera is made of a Raspberry Pi 2 Model B with the corresponding Raspi-Camera module. The OS loaded to the camera is the common Raspian WHEEZY system with a few scripts running on startup in the background. The camera connected is the Raspi-Camera module.

Webcam setup In order to connect to the LAN on startup, a few lines need to be added in */etc/network/interfaces*:

Code 1: /etc/network/interfaces

```
1 auto lo
2 iface lo inet loopback
3
4 iface eth0 inet static
5 address 172.16.156.138
6 netmask 255.255.255.0
7 gateway 172.16.156.1
8
9 auto wlan0
10 allow-hotplug wlan0
11 iface wlan0 inet static
12 address 172.16.156.139
13 netmask 255.255.255.0
14 gateway 172.16.156.1
15 wireless-essid korebot
```

The camera module comes with an own library with modules for taking single images (*raspistill*) or videos (*raspivid*). The following lines of code are added in a file *~/start_camera.sh* in order to take pictures at regular intervals and save them to */tmp/stream*. This is where they will be found by the $mjpg_streamer$ module which sends the images to the corresponding server.

Code 2: ~/start_camera.sh

```
1 #!/bin/bash
2 echo "Start camera stream"
3
4 sudo mkdir -p /tmp/stream
5 sudo chmod 777 /tmp/stream
6 raspistill -w 1280 -h 720 -q 50 -ex backlight -mm backlit -o /tmp/stream/pic.jpg
      ↪  -tl 100 -t 2147483647 &
7 LD_LIBRARY_PATH=/usr/local/lib mjpg_streamer -i "input_file.so -f /tmp/stream -n
      ↪  pic.jpg" -o "output_http.so -w /usr/local/www"
```

Line 6 tells the camera to take a picture of resolution $w \times h = 1280 \times 720$ of quality 50%, 100% meaning no compression at all, at a time interval of $tl = 100ms$ until the maximum time of $t = 2147483647ms$ is reached, which is the maximum for a 32-bit signed integer. Other parameters like

the exposure (set to backlit, so that background is enhanced: TRY FIXEDFPS for constant lighting?), and the metering mode (what is this???) can be set. With these settings, the resulting picture size is not more than $N_{pic} = 803kB$ (779412). (Apparent size: 786kB) As this size is much too high, the quality is set down to 10, which is equivalent to a quality of around 85 for most applications. This leads to a size of around 100kB. In addition to that, the thumbnail, which is a bitmap that also takes up unnecessary storage space, is deleted and a final image size of 80kB is obtained.

$N_{pic} = 803kB$ qual size 100 533 kB 50 455 kB 10 74 5 37 png 1549kB bmp 2773kB The network speed that needs to be provided to avoid idle times or image distortion is therefore:

Line 7 starts the mjpg streamer from where the picture has been stored and sends it to the local network.

$$S_{network} \geq \frac{N_{pic}}{tl} = \frac{1.1MB}{0.1s} = \frac{1.05Mbits}{0.1s} = 10.5Mbps \tag{2}$$

The router capacity is ??? which would be

Button A button is added to the camera such that it can be restarted or turned off without logging into the camera. This allows the Raspberry to be turned off correctly even when the network has not be set up properly or the ethernet is not working. The button is a pull-down resistor, thus the signal at the output goes from 0 to 1 when the button is pressed. An interrupt is triggered when the button is pressed in which it counts during 6 seconds weather the button is pressed or not at a rate of 1s. For robustness, a total of 3 signals is sufficient for the system to shut down. If less than 3 signals are registered during the 6 seconds, the system reboots.

Code 3: ~/switchoff.sh

```
1 #-*- coding: utf8 -*-
2 import RPi.GPIO as GPIO
3 import os
4 import time
5 #set up GPIO using BCM numbering
6 GPIO.setmode(GPIO.BCM)
7 GPIO.setup(10,GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
8
9 #function called on pin interrupt
10 def button_triggered(channel):
11         counter = 0
12         counter_on = 0
13         while (counter <= 6):
14                 time.sleep(1)
15                 counter+=1
16                 if (GPIO.input(10)):
17                         counter_on+=1
18                 if (counter_on >= 3):
19                         break
20
21         if (counter_on >= 3):
22                 print("switchoff.py: Raspberry shutting down now")
23                 os.system("sudo halt")
24         elif (counter_on < 3):
25                 print("switchoff.py: Rapsberry is going to reboot now")
26                 os.system("sudo reboot")
27 #setup pin interrupt
28 GPIO.add_event_detect(10,GPIO.RISING,callback=button_triggered,bouncetime=300)
29
30 #wait forever
31 while True:
32         time.sleep(0.001)
33
34 GPIO.cleanup()
```

Setup Both scripts above need to be started on startup of the camera. Therefore, the following lines are put in the file /etc/rc.local:

Code 4: /etc/rc.local

```
1 #Auto start camera
2 sudo /home/pi/start_camera.sh &
3 #Auto start shutdown
4 sudo python /home/pi/switchoff.py &
```

picture quality and compression: www.raspberrypi.org/forum/viewtopic.php?f=43&t=73174&p=527300#p527300 wget: https://www.maketecheasier.com/test-internet-connection-speed-from-termina speedtest: http://www.speedtest.net/

References