camcapture Library
A Short Introduction

Florian Schmitt
0schmitt@informatik.uni-hamburg.de

October 18, 2013

Abstract

"camcapture" is a lightweight library that helps you to get images from a Linux video input and processing simple analysis algorithms. It is specialized to run on Linux powered humanoid robots without causing too heavy system load.
This document gives you a short introduction about how to use it.

Contents

1 Introduction

This is a tutorial for the usage of the camcapture library. The library is designed to run on Linux systems with minimal system requirements. It provides image capturing from standard video devices via video4linux, simple image manipulation, analysis and visualisation tools.

1.1 Needed Components

Before compiling the camcapture library ensure you have installed the following components:

- video4linux (comes with most linux distributions)
- cmake
- makefile
- gcc, g++

To ensure that you have all components installed on your machine you can type the following command into your console (on Debian/Ubuntu):

```
sudo apt-get install gcc g++ make cmake
```

1.2 Installation: From Installer

Use the provided Debian installer. In most Linux distributions you can install just by clicking on it. Otherwise, on the command line you can install .deb files using:

```
sudo dpkg --install name.deb
```

1.3 Installation: From Source

To install the camcapture library follow the installation instructions:

1. Get the camcapture source and unpack the compressed archive
2. Toogle into the source directory
3. Type the following commands:
    - mkdir build

- cd build
- cmake ../source
- make
- sudo make install

4. Now the library should have been installed in your systems standard directory

1.4 Link the Library

With the gcc compiler you can simply link to the library using the '-lcamcapture' option. A simple command to build your first program could look like this:

```
gcc -Wall -O2 test.c -lcamcapture
```

Otherwise, if you want to use cmake you could add the following lines to your CMakeLists.txt:

```
find_library(CAMCAPTURE_LIBRARY
             LIBRARY
             NAMES camcapture libcamcapture
             REQUIRED)
[...]
target_link_libraries(yourExecuteable ${CAMCAPTURE_LIBRARY})
```

1.5 First Hello World Program

Your first hello world programm could look like this:

```
#include <camcapture.h>

/* set video device name */
char *devname = "/dev/video0";


int main(int argc, char *argv[])
{
  /* initiate - set input and image sizes */
  camcapture_init(devname, 640, 480);
  /* this is a struct to store the information */
  image_data_t imgdata;

  /* Lock the device:
```

```
    * We have to keep the cam from writing
    * new images into our memory while we
    * are working with it.
    * This function also waits for an image
    * to be read.
    */
  camcapture_lock();

  /* now get the image information into
   * our imagedata struct - be careful:
   * this struct does not contain a copy
   * of the image, it only gets a pointer
   * to the drivers buffer!
   */
  camcapture_get_frame(&imgdata);

  /* save the picture to the file frame.pnm */
  camcapture_save_pnm(&imgdata, "frame.pnm");

  /* now we can unlock the cam */
  camcapture_unlock();

  /* clean up */
  camcapture_destroy();

  return 0;
}
```

The program can be built using:

```
gcc -Wall -O2 example01.c -o example01 -lcamcapture
```

Running the program should create a pnm file named 'frame.pnm' with a fresh picture from your video device. Please take notice, that the image size depends on your cameras available modes.

2 Tutorial: Filters and Drawing into Images

2.1 Your first Filter and Series of Pictures

In most cases you don't want to take just a single picture. To take more than one picture, you have to loop around camcapture_lock() and camcapture_unlock() function, because the camera does not take pictures while it is locked. So you have to toggle between locked and unlocked inside your loop. To apply a filter you first select your filter color by creating a color_rgb_t object and then you can use the camcapture_filter_color(image_data_t*, color_rgb_t*, int) function. The first parameter represents the image object, the second the color, which you want to filter and the third option is the sensibility value to apply the filter.

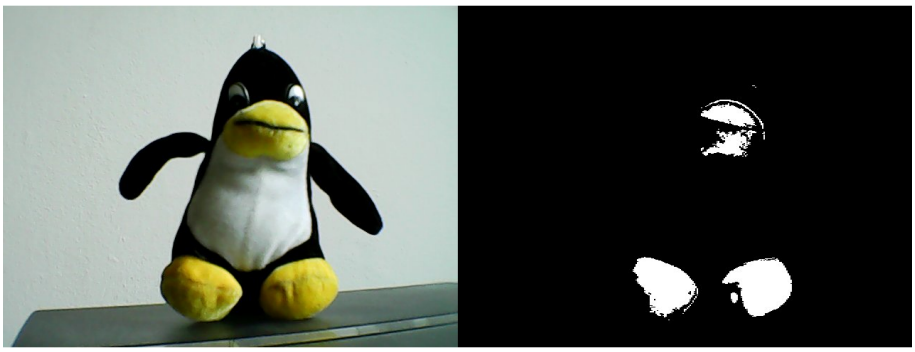

Figure 1: A yellow filtered image - left: before, right: after

The following code example shows you how to catch series of images and filter a color before.

```
#include <camcapture.h>

/* set video device name */
char *devname = "/dev/video0";


int main(int argc, char *argv[])
{
  int n;
  /* initiate */
  camcapture_init(devname, 640, 480);
  /* image data struct */
  image_data_t imgdata;
```

```
  /* the color to be filtered */
  color_rgb_t filtercolor;
  /* set the color to a warm yellow */
  filtercolor.r = 220;
  filtercolor.g = 200;
  filtercolor.b = 0;

  /* lets take n frames */
  n = 20;
  while( (n--) > 0 )
  {
    /* Lock the device
     *  - Keeps cam from writing into buffer
     *  - Waits for an image     */
    camcapture_lock();

    /* Get the frame */
    camcapture_get_frame(&imgdata);

    /* now filter the color filtercolor with a
     * sensibility of 50  */
    camcapture_filter_color(&imgdata, &filtercolor, 50);

    /* save the picture to the file frame<cnt>.pnm
     * the _inc extended version automaticaly
     * appends the frame number and '.pnm' to the name */
    camcapture_save_pnm_inc(&imgdata, "frame");

    /* now we can unlock the cam */
    camcapture_unlock();
  }

  /* clean up */
  camcapture_destroy();

  return 0;
}
```

Please take notice: The camcapture_filter_color(...) function converts the RGB-color to YCbCr[1] before applying the filter and only uses the CbCr values for color selection. Y represents the luminance (brightness) of an image

[1]See http://en.wikipedia.org/wiki/YCbCr for further information on YCbCr

point - so it is left out from the calculation to handle different lighting situations.

If you want to filter areas exceeding a given luminance use:
camcapture_filter_lumi(image_data_t* image, int threshold)

2.2 Drawing Functions

The following code example shows you how you can draw into a taken image:

```
#include <camcapture.h>

/* set video device name */
char *devname = "/dev/video0";


int main(int argc, char *argv[])
{
  /* initiate */
  camcapture_init(devname, 640, 480);
  /* image data struct */
  image_data_t imgdata;

  /* color to draw with (red) */
  color_rgb_t c;
  c.r = 255;
  c.g = 0;
  c.b = 0;

  /* Lock the device from writing to buffer */
  camcapture_lock();

  /* get a picture */
  camcapture_get_frame(&imgdata);

  /* sets a single pixel */
  camcapture_set_pixel(&imgdata, 15, 15, &c);

  /* draw a simple line thrugh the middle of the image */
  camcapture_draw_line(&imgdata, 10, 480/2, 630, 480/2, &c);

  /* draw a circle (center=image center, radius=50 */
  camcapture_draw_circle(&imgdata, 640/2, 480/2, 50, &c);
```

```
  /* draw a rectangle from p1=(10,10) to p2=(630,470) */
  camcapture_draw_rect(&imgdata, 10, 10, 630, 470, &c);

  /* draw a cross marker at p=(50, 100) */
  camcapture_draw_cross(&imgdata, 50, 100, &c);

  /* save the picture to the file frame.pnm */
  camcapture_save_pnm(&imgdata, "frame.pnm");

  /* unlock the device */
  camcapture_unlock();

  /* clean up */
  camcapture_destroy();

  return 0;
}
```

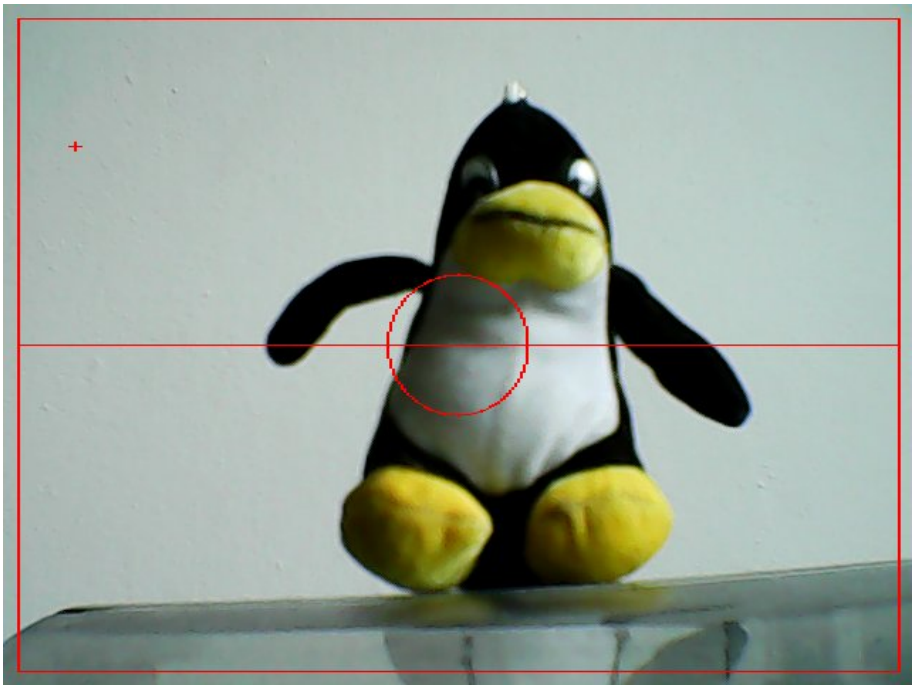

Figure 2: An image created with the given code example

An overview above all drawing functions can be found in the reference documentation (imgdraw.c). Some further functions to draw lists of points

are introduced in the edge detection tutorial.

3 Edge Detection

3.1 Introduction to Edge Detection

In many cases you try to find objects with given attributes in your image. If you want to find a specific object with the camcapture library, it is a good way to apply a filter with the target objects color and then do an edge detection. After a successful edge detection you can sort out edges that don't belong to the object for shure and use the remaining points for calculations that tell you more about the found object (e.g. object radius, roundness, median point, distance).

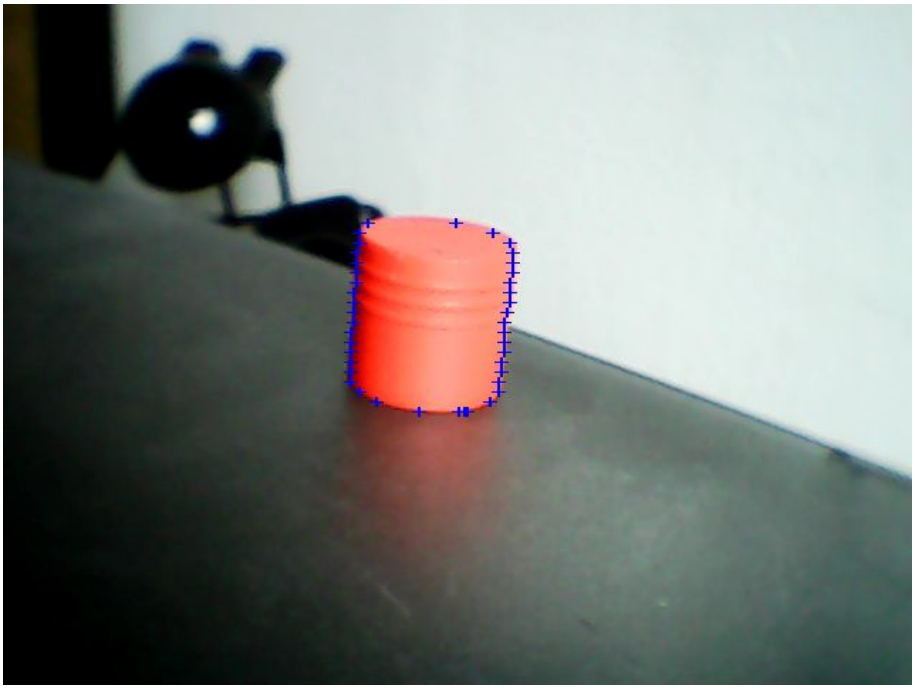

Figure 3: An image after edge detection and drawing the edges (created with the following code example)

3.2 Lists of Image Points

This section may be hard to understand if you are not used to pointers in C. If you come from other programming languages like Java you should read the introduction to pointers in the appendix first.
camcapture's edge detection does not behave as an image effect: The image will not be modified. In many applications it is better to have the edges as

lists of points (x, y). From a list of edge points it is easy to calculate their median and do analysis of the found objects. So, to perform an edge detection you first have to create a list to collect image points (image_point_t), which will later contain all found edge points. An image_point_list_t can be created by camcapture_create_point_list() and destroyed by camcapture_destroy_point_list(image_point_list_t*).

The creation of an image_point_list_t looks like this:

```
image_point_list_t *edgelist;
edgelist = camcapture_create_point_list();
[...]
camcapture_destroy_point_list(edgelist);
```

You can simply loop through an image_point_list_t and e.g. draw the points using code like this:

```
image_point_list_t *tmp = edgelist;
while( (tmp = tmp->next) != 0)
  camcapture_draw_cross_obj(&imgdata, tmp->data, &c1);
```

Adding points by hand is easy, too:

```
image_point_t *mypoint = malloc(sizeof(image_point_t));
mypoint->x = 10;
mypoint->y = 200;
camcapture_add_point_list(edgelist, &mypoint);
```

Notice: After adding points to a point list you won't have to clean them up, because the list destructor does the clean-up automatically.
The add function inserts at the beginning of the list. If you want to insert at the end of the list you could use camcapture_append_point_list(image_point_list_t*, image_point_t*) - but be careful, image lists don't have a reference to the last inserted list member, so appending is really inefficient! Try to use add instead of append in any case it is possible!

3.3 Edge Detection Methods

Edge detection is performed by camcapture_edge_detection_x(image, threshold, raster, maximum, list). This function goes through the image lines with a given raster (all other lines are ignored) and checks if the delta value between two neighboured points exceeds a given threshold. If the number of found points exceeds the given maximum value, the function the function is cancelled earlier.

Finally in most cases it is much more efficient to combine edge detection with image filtering. So there is an extended function for edge detection and applying a filter at once: camcapture_edge_detection_x_col(image, color, sensibility, raster, max, list). This extended version lets the image untouched, the filter is just used for the edge detection and not written into the image data. So the image remains viewable and you can apply other filters on it later.

3.4 Edge Detection Example

```
1  #include <camcapture.h>
2
3  /* set video device name */
4  char *devname = "/dev/video0";
5
6
7  int main(int argc, char *argv[])
8  {
9    int n;
10   /* initiate */
11   camcapture_init(devname, 640, 480);
12   /* image data struct */
13   image_data_t imgdata;
14   /* the edgelist */
15   image_point_list_t *edgelist;
16
17   /* the color to be filtered */
18   color_rgb_t filtercolor;
19   /* set the filters color to red */
20   filtercolor.r = 200;
21   filtercolor.g = 0;
22   filtercolor.b = 0;
23
24   /* the color to draw the edges */
25   color_rgb_t blue;
26   /* set the color to blue */
27   blue.r = 0;
28   blue.g = 0;
29   blue.b = 255;
30
31   /* lets take n frames */
32   n = 20;
```

```
33    while( (n--) > 0 )
34    {
35      /* Lock video device */
36      camcapture_lock();
37
38      /* Get frame */
39      camcapture_get_frame(&imgdata);
40
41      /* create an empty list */
42      edgelist = camcapture_create_point_list();
43
44      /* find edges with a filter sensibility of 40,
45       * a raster of 5 and break when found 512 points */
46      camcapture_edge_detection_x_col(&imgdata,
47                                      &filtercolor,
48                                      40,
49                                      5,
50                                      512,
51                                      edgelist);
52
53      /* now we use a simple loop through the list to
54       * draw all detected edge points into the image */
55      image_point_list_t *tmp = edgelist;
56      while( (tmp = tmp->next) != 0)
57        camcapture_draw_cross_obj(&imgdata, tmp->data, &blue);
58
59      /* the list has to be cleaned up now
60       * all points are deleted, too */
61      camcapture_destroy_point_list(edgelist);
62
63      /* store frame in a file */
64      camcapture_save_pnm_inc(&imgdata, "frame");
65
66      /* unlock */
67      camcapture_unlock();
68    }
69
70    /* clean up */
71    camcapture_destroy();
72
73    return 0;
74  }
```

4 Working with Point Lists

4.1 Median Point

If you have created your edge point list, it can be used to make some useful calculations. There is a simple function to calculate the lists median point as an image_point_t object:

```
1 image_point_t camcapture_point_list_median(
2   image_point_list_t*);
```

4.2 Object Radius

After calculating the median, you can use it to do further calculations. If you expect to detect something that is visible as a circle (a ball e.g.) the next step you could do is calculate the radius. There are basically two ways to do this:

- Calculate the medium distance to the median
- Calculate the maximum distance to the median

There are two functions given to do that:

```
1 /* calculate the points median delta */
2 int camcapture_point_list_median_delta(image_point_list_t*,
3                                        image_point_t);
4
5 /* calculate the points maximum delta */
6 int camcapture_point_list_max_delta(image_point_list_t*,
7                                     image_point_t);
```

4.3 Elimination of unwanted Points

In many cases you will not just recognize the wanted object, but also some small objects with the same colour or singe pixels that have the same colour. To eliminate those points there are two methods you can use on their own or combined:

- Analysis of the surrounding colours
- Eliminate overshoots that are far over the medium distance to the median (radius)