Lab 6

Computer Vision

Author:

Tarik Rifai, Jonas Lussi, Franziska Ullrich, Naveen Shamsudhin, Burak Zeydan, and Prof. Bradley J. Nelson

Institute of Robotics and Intelligent Systems

Date: 2019

Version: 1.2

Summary: In this lab, you will learn how computer vision is used in the field of robotics. We will use the OpenCV library to detect and track objects by using two different tracking algorithms.

6.1 Background

We perceive the three-dimensional structure of the world around us with apparent ease and can see how vivid the world is by perceiving light rays through our eyes and analyzing information in the brain. Researchers in computer vision have been developing, in parallel, mathematical techniques for recovering the three-dimensional shape and appearance of objects in imagery. Computer vision includes event detection, object recognition, object pose estimation, scene reconstruction, video tracking, learning, indexing, motion estimation, and image restoration. Nowadays, computer vision is highly implemented in the autonomous detection and artificial intelligence (AI) areas such as HONDA's ASIMO robots with embedded camera to recognise objects and estimate the motion, NASA's Mars Exploration Rover Mission using multiple sensors to detect and reconstruct the environment of Mars, and face perception using phone camera to track and recognise human eyes (Figure 6.1).

In this week's lab, you will learn how to track objects and output their coordinates in an image frame. In many robot applications, tracking several objects allows for interacting with the world around.

6.2 Tracking algorithms

In this lab, we will introduce two different object tracking approaches, which are color tracking and edge tracking. With the two tracking algorithms, you will track objects from images or videos provided to you. You will learn that different methods are suitable in different environments, to track different objects and in different light conditions.

6.2.1 Color Tracking

First, you will implement a color tracking algorithm. The program reads an image from a video or a camera and converts it into HSV (hue-saturation-value) color space. Converting an image into HSV color space gives us the

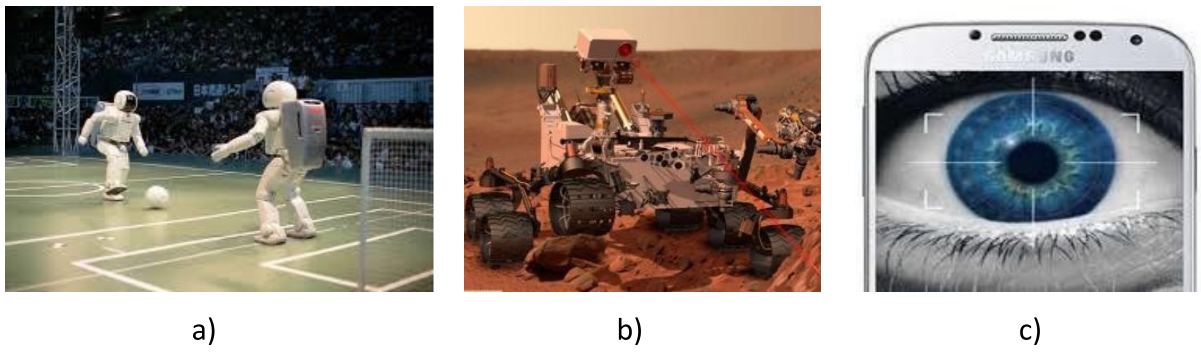

Figure 6.1: Computer vision implemented in (a) humanity robots, (b) Mars Exploration Rover Mission and (c) Biometric systems in face perception.

advantage of finding a single value for the hue, which is independent of the multiple shades of the color that is being tracked and therefore makes the algorithm less sensible to the surrounding light conditions. Subsequently, a threshold is applied to the HSV image to find a certain color range on the image. The algorithm should find the area of the detected object and derives its centroid. For implementing color tracking we are using OpenCV (see OpenCV Documentation: `https://docs.opencv.org/2.4/index.html`).

6.2.1.1 HSV Color Space

HSV is the projection of RGB (red, green and blue) color cube onto a non-linear chroma angle (Hue), a radial saturation percentage (Saturation), and a luminance-inspired value (value) as shown in Figure 6.2. In more detail, the value is defined as the height, saturation is defined as a scaled distance from the diagonal, and hue is defined as the direction around a color wheel. Such decomposition is quite natural in graphics applications such as color picking.

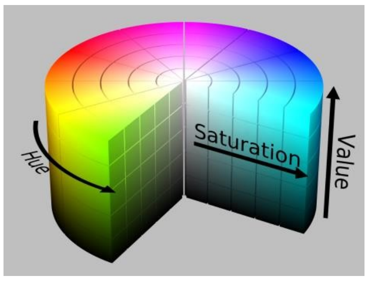

Figure 6.2: HSV color space.

6.2.2 Edge Tracking

The second tracking algorithm you will implement in this lab is edge tracking. Edge tracking is using mathematical methods that are aimed at detecting regions in a digital image that differs in brightness compared to the surrounding areas. The program reads the image from the video, converts it into grey level scale and blurs the image, which smoothens the grey image. Using an edge detection method finds the edges of objects and finally draws the contours of the tracked object (Figure 6.3).

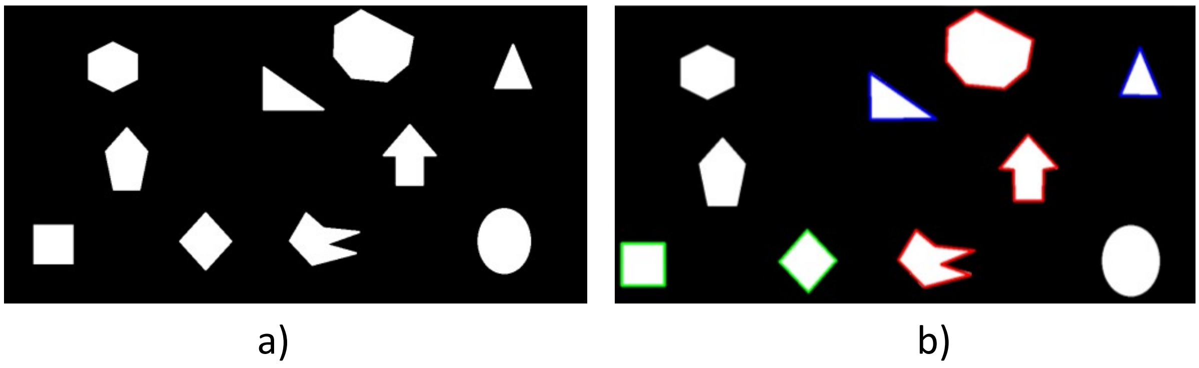

Figure 6.3: Shape detection by using edge tracking method (a) original image (b) tracked image with contour.

6.2.3 OpenCV

OpenCV is an abbreviation of *Open Source Computer Vision* and is library of programming functions for real time computer vision. It has C++, C, Python and Java interfaces running under Windows, Linux, Android and Mac (refer to www.opencv.org).

NOTE: you will need to include the OpenCV library in your code.

6.3 Prelab Procedure

Note: Prelab assignments must be done before reporting to the lab and must be turned in to the lab instructor at the beginning of your lab session. Additionally, you also have to upload your solution as a single PDF-file to moodle. Make sure to upload the file before your lab session starts, late submissions will not be corrected. The prelab needs to be handed in as a group. All the prelab tasks are marked with **PreLab Qx**.

6.3.1 Color Tracking with Matlab

In the lecture you have seen the basic idea of color tracking. The RGB (red-green-blue) or HSV channels of an image are filtered and a threshold is used to identify areas of interest in the image. In the prelab, you are provided with a single image and a MATLAB skeleton `color_tracker.m`. Write a MATLAB script that reads the image, plots the three hsv channels seperately, uses threshhold on hue and saturation to find the red, green and blue shapes independently and determines the centroid of the object. You can find the details in your skeleton. Your output should look similar to Figure. 6.4. Print and upload your matlab code along with the figures. **(PreLab Q1)**

6.3.2 Color Tracking with OpenCV

Here we provide you an easy color tracking approach by reading an image from file. Please study it and make sure you are familiar with the corresponding OpenCV functions.

```
#include "cv.h"
#include "highgui.h"
#include "ctype.h"

int hmax = 0, hmin = 0, vmin = 0, vmax = 0, smin = 0, smax = 0;
int main(int argc, char **argv)
{
        If(argc<=1)`
        {
                Std::cout<<"Error:please load a picture!"<<std::endl;
```

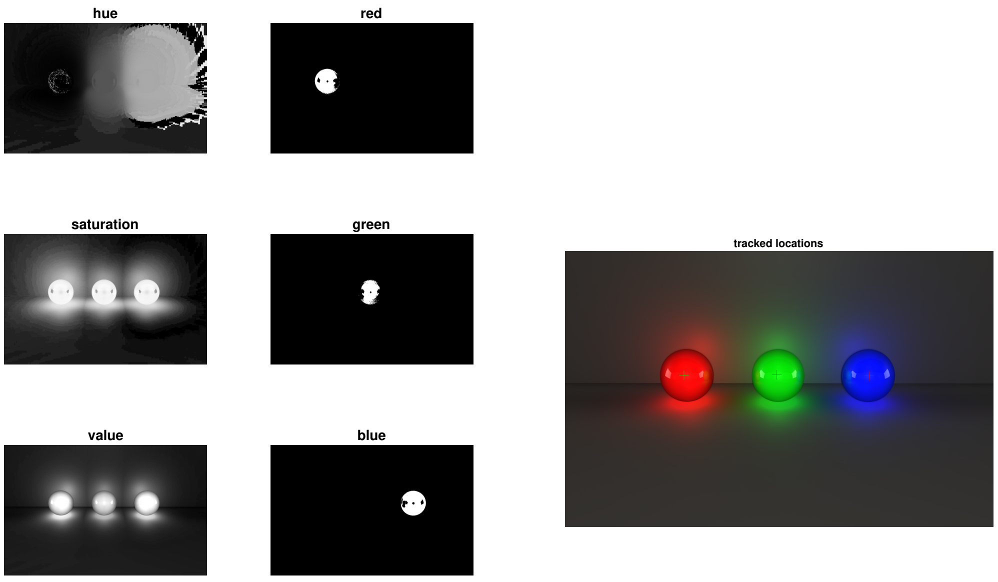

Figure 6.4: Color tracker implemented with MATLAB.

```
        return 0;
}
IplImage *image, *hsv, *mask;

//create windows
cvNamedWindow("image", CV_WINDOW_AUTOSIZE);
cvNamedWindow("hsv", CV_WINDOW_AUTOSIZE);
cvNamedWindow("mask", CV_WINDOW_AUTOSIZE);
cvNamedWindow("Track", CV_WINDOW_AUTOSIZE);

//create a threshold trackbar for tuning the threshold range
cvCreateTrackbar("Hmin", "Track", &hmin, 256, 0);
cvCreateTrackbar("Hmax", "Track", &hmax, 256, 0);
cvCreateTrackbar("Smin", "Track", &smin, 256, 0);
cvCreateTrackbar("Smax", "Track", &smax, 256, 0);
cvCreateTrackbar("Vmin", "Track", &vmin, 256, 0);
cvCreateTrackbar("Vmax", "Track", &vmax, 256, 0);

// read the image and arrange the new images image = cvLoadImage(argv[1]);
hsv = cvCreateImage(cvGetSize(image), 8, 3);
mask = cvCreateImage(cvGetSize(image), 8, 1);

//convert RGB to HSV
cvCvtColor(image, hsv, CV_RGB2BGR);

//display the images cvShowImage("image", image); cvShowImage("hsv", hsv);

int _hmax = 0, _hmin = 0, _vmin = 0, _vmax = 0, _smin = 0, _smax = 0, flag = 0;
while (1)
{
        _hmax = hmax, _hmin = hmin, _vmin = vmin, _vmax = vmax, _smin = smin, _smax = smax;

        //set the threshold range
        cvInRangeS(hsv, cvScalar(MIN(_hmax, _hmin), MIN(_smax, _smin), MIN(_vmax, _vmin), 0),
                cvScalar(MAX(_hmax, _hmin), MAX(_smax, _smin), MAX(_vmax, _vmin), 0), mask);

        //show the tracking images
         cvShowImage("mask", mask);

}
```

```
        //release all the used functions
        cvDestroyAllWindows();
        cvReleaseImage(&image);
        cvReleaseImage(&hsv);
        cvReleaseImage(&mask); system("pause");
        return 0;
}
```

6.3.3 Edge Detection with OpenCV

In this lab, an edge tracking method is compared with the color tracking algorithm. Finalize the C function `EdgeDetect()` that implements an edge tracker. Write your code in the provided `tracking.h` (declaration) and a `tracking.cpp` (definition) file (don't mind the .cpp ending, it's a workaround to compile with OpenCV). You do not need to test the code - it can be cumbersome to get OpenCV running on certain operating systems. OpenCV is already installed for you on the UDOO's. Print and upload all the code. **(PreLab Q2)**

6.4 Lab procedure

Note: Questions marked with **Postlab Qx** have to be answered as part of **PostLab 06**.

6.4.1 Color Tracking

1. On moodle you will find a video and skeletons for various color tracking functions.
2. Write a function `ColorTrackingSetColors()` to create color trackbars for tuning the threshold ranges of the HSV image to obtain a visually clear tracked object and save these threshold range. The source code in section 6.3.2 is your reference for creating a trackbar with OpenCV. Follow the instructions in the skeleton code. In the main() function, open the provided .avi file and use the `ColorTrackingSetColors()` function you just wrote to set the parameters. Follow the directions in the skeleton code. **(PostLab Q1)**
3. Implement the color tracking function `ColorTracking()` with the help of the comments in the skeleton code. In the main() function, call the `ColorTracking()` function you just wrote on each frame to track the red marker in the .avi file. Use the provided `CrossTarget()` function to target the objects with a cross on the calculated coordinates. Record the trajectory of the tracked objects by saving their coordinates in a txt-file. **(PostLab Q2)**

   You could use the following function to write the data to a text file:

   ```
   // Open .txt file to store all the coordinates
           FILE *coordinate;
           coordinate = fopen("Coordinates.txt", "w+");

           fprintf(coordinate, "%d,%d\t",(int)positionX,(int)positionZ);

           fclose(coordinate);
   ```
4. Open the txt-file in Matlab and plot the tracked trajectory. Print out and hand in the labelled plot. **(PostLab Q3)**

6.4.2 Edge Tracking

1. Use the `EdgeDetect()` function from your prelab to track the red dot and save the tracked edges in an image. Integrate the `EdgeDetect()` function in the main(). Make sure you are not running two trackers at the same time. **(PostLab Q4)**

2. Try different value of "thresh" (double and half of original value of "edgeparam") to see how results differ with different ranges of Canny edge detection. Save the results and print them. **(PostLab Q5)**

6.5 Postlab and lab report

1. Upload a single PDF-file with your solution to moodle. The file should contain your answers (including plots, images and code) to the postlab questions Q1-Q5 (this includes your complete `tracking.cpp` and `tracking.h` file). Also, make sure to include the plot from PostLab Q3 (Matlab code is not required) and the results and comparison from PostLab Q5. Please upload your solution in time (before next lab session), late submissions will not be corrected.

2. Print the PDF-file and hand it in at the beginning of the next lab session.

3. Show your working code to your assistant in the next lab session. The assistant will ask you to show the following:

   - You should be able to manually select the tracking parameters using ColorTrackingSetColors ()
   - You should be able to demonstrate the Color tracking algorithm by continuously tracking and displaying the red dot (with cross)
   - You should be able to demonstrate the Edge tracking algorithm by continuously displaying the contours of the red dot

4. Come prepared with the Prelab procedure for the next lab.

6.6 Functions

- `int ColorTracking(IplImage* img, int* positionX, int* positionY, CvScalar min, CvScalar max)`
- `int EdgeDetect(IplImage* img, int thresh)`
- `IplImage* CrossTarget(IplImage* inImg, int x, int y, int size, int line_thickness)`
- `int ColorTrackingSetColors (IplImage* img, int* hmax, int* hmin, int* smax, int* smin, int* vmax, int* vmin)`
- `SvImage (IplImage* img, char* filename)`

6.6.1 int ColorTracking(IplImage* img, int* positionX, int* positionY, CvScalar min, CvScalar max)

These are the color tracking parameters. This function converts the original image to HSV and threshold images and computes the center positions of tracked object.

- `img`: This is where the original image is stored
- `positionX`: This is the x-coordinate of the tracked object
- `positionY`: This is the y-coordinate of the tracked object
- `min`: This is the minimum (hue, saturation, value, 0)
- `max`: This is the maximal (hue, saturation, value, 0)

6.6.2 int ColorTrackingSetColors (IplImage* img, int* hmax, int* hmin, int* smax, int* smin, int* vmax, int* vmin)

This function sets the threshold for color tracking

- `img`: This is where the original image is stored
- `hmax`: This is a pointer to the max hue for thresholding
- `hmin`: This is a pointer to the min hue for thresholding
- `smax`: This is a pointer to the max saturation for thresholding
- `smin`: This is a pointer to the min saturation for thresholding
- `vmax`: This is a pointer to the max value for thresholding
- `vmin`: This is a pointer to the min value for thresholding

6.6.3 int EdgeDetect(IplImage* img, int thresh)

These are the blob tracking parameters. This function converts the original image to grey scale image and uses canny detection to find the edges of tracked objects and finally draw contours of the tracked objects.

- `img`: This is where the original image is stored
- `thresh`: This is the threshold for canny edge detection

6.6.4 IplImage CrossTarget(IplImage* inImg, int x, int y, int size, int line_thickness)

These are the crosstarget parameters. This function draws a cross on the centroid of the tracked objects with given coordinates.

- `inImg`: This is where the input image is stored
- `x`: This is the x-coordinate
- `y`: This is the y-coordinate
- `size`: This is the size of the cross.
- `line_thickness`: This is the value determining the thickness of drawing line.

6.6.5 SvImage (IplImage* img, char* filename)

These are the SvImage parameters. This function saves an image to a file.

- `img`: This is where the input image is stored
- `filename`: name of the file to be saved