Assignment 8

Hendrik Werner s4549775

May 28, 2017

1 Abstract

In this paper I present an easy to understand and follow approach to detect an object's position, and the direction it is facing, from visual input using OpenCV. This is done as a case study of an autonomous football playing robot.

2 Introduction

For a university project I wanted to detect the position of a robot and which way it was facing, as well as the position of a ball, using visual input. After looking into different computer vision frameworks I eventually chose OpenCV. I expected there to be some prebuilt functionality for something as mundane as detecting an arrow but was disappointed.

In this article I want to present the approach I took to detect the features I needed; and how this was applied in a robotics project. This article is positioned as an introduction into image detection and does not assume familiarity with the subject. Concepts and terminology are introduced as needed; it serves as a quick overview over image detection and OpenCV in particular.

3 Problem

As input for my robot's decision making algorithm I need the position of the ball, as well as the position of the robot, and the direction it faces. The ball is round, so direction does not really apply to it.

For this purpose I mounted a camera above the playing field facing straight down. The ball is a bright yellow, and on the robot I mounted a bright red arrow, pointing forward.

In summary the problem we need to solve is the following: Given visual input with unique, and brightly colored objects, how can the positions and directions of the objects be extracted?

4 Overview of Computer Vision

Computer Vision is the field of study that deals with the extraction of high level features from digital visual data. In our case this data is the video stream from the camera, and the features are the positions and directions of the ball and car.

There are many applications for Computer Vision, most of which are concerned with automation of tasks previously done by humans. In our case playing football, though there are of course more practical applications as well. Tesla heavily invests in Computer Vision for its application in self driving vehicle technology [13] [14].

5 Overview of OpenCV

OpenCV (Open Computer Vision) is a cross-platform, open source computer vision library. Development was started by Intel in 1999 as a research project. It is released under the BSD license which allows for commercial and non-commercial free use [5].

Today it is one of the industry standards for Computer Vision, used and sponsored by companies such as Google, Microsoft, and Intel, among many

others. It has interfaces for C, C++, Python, Java, and MATLAB [2].

OpenCV is highly optimized and geared towards real time systems [2]. This is important for many applications in automation. Your car must be able to process visual information in real time to avoid collisions, for example. For our purposes this is not of great importance but it is nice to be able to visualize the algorithms working. Quicker response times are also generally nice.

The OpenCV API provides facilities for image (pre)processing, persistence, clustering, hight-level GUI, video analysis, camera calibration, 2d-feature extraction, object detection, Machine Learning, GPU acceleration, computational photography, image stitching, OpenCL acceleration, super resolution, 3d-visualization and even some non-free components [11].

In addition to that there are some deprecated modules and experimental / contributed functionality. [10]

6 Solution

6.1 Preprocessing

OpenCV reads the image in as BGR (blue, green, red), where a color's hue is split across 3 different channels. We want to filter on the hue, so it is the easiest solution to convert the color to HSV (hue, saturation, value), which packs hoe information into a single channel.

```
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

The images we capture from the camera are pretty noisy, so we apply a median blur. A bilateral filter is even better because it reduces noise while retaining edges. For out purposes a median blur is sufficient, so we use it because it's performance is better. [12]

```
image_hsv = cv2.medianBlur(image_hsv, 3)
```

I made a generic object finder class, that does the following things:

1. Make a mask. This is a black and white image of the same dimensions as the input image, in which all pixels falling within a certain color

range are white, and the rest is black. It basically maps all pixels to whether or not they are in the color range.

We use OpenCV's *inRange* function for this:

```
mask = cv2.inRange(
        image_hsv,
        threshold["lower"],
        threshold["upper"],
)
```

2. Find the contours. The contours are the outlines of all the shapes in our mask.

   We use *findContours*:

   ```
   contours = cv2.findContours(
           mask,
           cv2.RETR_LIST,
           cv2.CHAIN_APPROX_SIMPLE,
   )
   ```

3. Filter out all contours with an area below some threshold:

   ```
   contours = [
           c for c in contours
           if cv2.contourArea(c) >= minimum_area
   ]
   ```

6.2 Finding the ball's position

This is the easy part of our project. To find the ball we just need to find the center of the minimum enclosing circle of the biggest blob of pixels that have the right color. We assume that the general steps described above have been done already.

We can use *contourArea* and *minEnclosingCircle*:

```
contour = max(self.contours, key=cv2.contourArea)
pos, radius = cv2.minEnclosingCircle(contour)
```

6.3 Finding the robot's position and direction

One could use the ball finding technique for the robot too, but that would only tell one it's position. Since we also need the direction it is facing we strapped an arrow to the robot. We look for the center of this arrow, and for the pointy end. With these two points we have our robot vector.

After applying the general steps described above, we take all contours and approximate them using *approxPolyDP*:

```
contour = cv2.approxPolyDP(contour, 10, closed=True).
    squeeze()
```

Next we extract all the lines and turn them into vector combinations:

```
def angle_heuristic(
        desired_angle: float,
        v1: np.ndarray,
        v2: np.ndarray,
        min_angle: int = 10,
) -> float:
    min_angle = as_rad(min_angle)
    inner_angle = angle(v1, v2, True)
    if inner_angle < min_angle:
        return np.pi * 2
    return min(
        abs(desired_angle - inner_angle),
        abs(desired_angle - np.pi + inner_angle),
    )

vectors = [
    (i, [x2 - x1, y2 - y1])
    for i, (x1, y1, x2, y2) in enumerate(lines)
]
combinations = [
    (angle_heuristic(arrow_angle, c[0][1], c[1][1]), c)
    for c in itertools.combinations(vectors, 2)
]
```

Then we add the best combination we found to our candidates list. After we have done this for all contours, we select the best of all of our candidates,

which is a pair of lines representing the best arrow we could find.

The intersection of these two lines is the tip of the arrow, while the end of the shaft is the center of all the points that make up the two lines:

```
arr_pos = cv2.minEnclosingCircle(
    np.append(arr_lines[0], arr_lines[1]).reshape((4,
        2))
)[0]
```

7 Application

We use the information we extracted as input for an algorithm that decides on a plan of action, that describes what our robot is supposed to do. This information is sent to an executor, which controls the robot via Bluetooth.

Even though the techniques were basic, and we extracted very limited information from the camera input, this was enough to make an arbitrarily placed robot find an arbitrarily placed ball, as long as they are visible to the camera.

This demonstrates the ease with which Computer Vision can be deployed even by amateurs, with the help of frameworks such as OpenCV.

8 Conclusion

OpenCV is a great tool for Computer Vision, which provides a basis to build sophisticated

The documentation isn't very comprehensive, especially for non-native interfaces of OpenCV. This can make in unnecessarily hard for beginners to get into the matter. With the right reading material however, it can be easy to quickly achieve great results, even for beginners; by building on the solid foundation of the OpenCV API.

The application for Computer Vision are diverse and it can be used from little hobby or school projects, to building the next generation of self driving vehicles.

9 Further Reading

- The full source code, the planner, the executor, the specifications for building the robot, and additional documentation can be found on the project's GitHub page[7].
- The OpenCV 3 Cookbook [9] is a complete introduction into the third version of the OpenCV library. "You will be presented with a variety of computer vision algorithms and exposed to important concepts in image and video analysis that will enable you to build your own computer vision applications." [4]
- If you have Python experience you might want to look at OpenCV Computer Vision with Python [8], which focuses on Python.

  "This book has practical, project-based tutorials for Python developers and hobbyists who want to get started with computer vision with OpenCV and Python. It is a hands-on guide that covers the fundamental tasks of computer vision, capturing, filtering, and analyzing images, with step-by-step instructions for writing both an application and reusable library classes." [1]
- If you are more of a C++ fan, or you want to use OpenCV through its native interface, maybe you are interested in Learning OpenCV 3 Application Development [6].

  "This book provides the steps to build and deploy an end-to-end application in the domain of computer vision using OpenCV/C++.". Topics covered are: how images are stored and processed by OpenCV, OpenCV-specific jargon, image traversal and pixel-wise operations, filtering, thresholding, edge detection, face detection, and much more. [3]

Keep in mind that the C++ interface has much better documentation because of being the native language of OpenCV. However, generally speaking, Python is used for a lot of automation and Machine Learning, and provides a wealth of other supporting libraries for a wide variety of tasks. Python is also considered to be more concise than C++, which is payed for in reduced performance.

Do not focus too much on the choice of languages though, not even the choice of frameworks. The general concepts, algorithms, and approaches are much

more generic. They will serve you well no matter the language or framework you happen to be working in.

References

[1] *About Computer Vision with Python*. URL: https://www.packtpub.com/application-development/opencv-computer-vision-python.

[2] *About OpenCV*. URL: http://opencv.org/about.html.

[3] *About OpenCV 3 Application Development*. URL: https://www.packtpub.com/application-development/learning-opencv-3-application-development.

[4] *About OpenCV 3 Cookbook - Third Edition*. URL: https://www.packtpub.com/application-development/opencv-3-computer-vision-application-programming-cookbook-third-edition.

[5] Gary Bradski and Adrian Keahler. *Learning OpenCV*. O'Reilly, 2008.

[6] Samyak Datta. *Learning OpenCV 3 Application Development*. Packt, 2016.

[7] Timo Schrijvers Hendrik Werner. *Football Project - SoccerBot*. URL: https://github.com/NewDevices/football_project/tree/subprocess.

[8] Joseph Howse. *OpenCV Computer Vision with Python*. Packt, 2013.

[9] Robert Laganiere. *OpenCV 3 Computer Vision Application Programming Cookbook - Third Edition*. Packt, 2017.

[10] *OpenCV API Reference*. URL: http://docs.opencv.org/2.4/modules/refman.html.

[11] *OpenCV nonfree - Feature Detection and Description*. URL: http://docs.opencv.org/2.4/modules/nonfree/doc/feature_detection.html.

[12] *Smoothing Images*. URL: http://docs.opencv.org/3.0-beta/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html.

[13] *Tesla Autopilot*. URL: https://www.tesla.com/autopilot.

[14] *Tesla is about to increase its lead in semi-autonomous driving w/ 'Tesla Vision': computer vision based on NVIDIA's parallel computing*. URL: https://electrek.co/2016/10/10/tesla-vision-autopilot-autonomous-driving-nvidia-cuda/.

10 Problems during writing the article

The main problems I had when writing this article were that

- I never wrote a similar article positioned as introductory before, so it was a bit difficult to strike the right balance between being technical and interesting.

  It may also be the case that I used incomprehensible language to somebody new to the topic, because it is difficult to judge which terms are hard to grasp, if you are very familiar with them.

- I wasn't sure of how much code to provide because when following tutorials I was always annoyed if they left out important pieces of code that were needed to make something work. Too much code could be boring too though, and possibly turn readers away.

  I went with putting little code snippets directly into the article, and provising a link to the full GitHub project.