2D Computer Vision
AI Programming for Games
COMP09041

Paul Keir

March 11, 2025

Date Performed: March 11, 2025
Instructor: Paul Keir

Laboratory Objectives

Plug in your USB camera. If your camera is built into the lid of your laptop, there is nothing else to do. You may then be able to see it via the Windows Control Panel - Hardware and Sound - Devices and Printers.

Prepare for Visual Studio

Use CMake to create a 64-bit Visual Studio project as usual; the CMakeLists.txt file is located within the `src` subdirectory. If you haven't set `VCPKG_ROOT` yourself, CMake will expect to find the OpenCV libraries within the VCPKG archive at `C:\vcpkg` that we worked with in earlier labs.

OpenCV Windows, Colours and Images

- The first example, `00_opencv_version.cpp` simply reports the version of OpenCV you have installed (within the Vcpkg repository).
- The second example, `01_show_colour.cpp` opens a window and gives it a red background. Try using a different colour. Note the use of BGR colouring.
- The third example, `02_show_image.cpp` loads a PGM file containing the image of a baboon and displays it.
- Next, `03_capture_show_image.cpp`, displays a static image captured from your webcam when a key is pressed.

- The next example, `04_capture_show_video.cpp` displays video from your camera. Compare the code to the previous example. Use OpenCV's flip function to get a mirror effect, or make the video output upside-down (modify `frame` before you pass it to `imshow`).

Adding a Trackbar

Let's now return to work on `01_show_colour.cpp` and add a trackbar GUI component to change the background colour interactively. You should be able to compile and run your program after each step below:

1. Copy the `while` loop from `4_capture_show_video.cpp`. Use it to replace the calls to `imshow` and `cv::waitKey` in `01_show_colour.cpp`. Remove the `vid_in >> frame` part. Also copy across the declaration of `fps`.

2. Rather than `255` for the redness, define a global `int` called `g_redness`; set it to `128`; and use it instead of the `255` for the frame colour.

3. Add the function definition for `on_trackbar_slide` as shown below:

```
void on_trackbar_slide(int pos, void *) {
  g_redness = pos;
}
```

4. Call `cv::createTrackbar` *after* the call to `cv::NamedWindow`. Any string literal will do for the first argument, and the second argument is `win_name`. Give the address of `g_redness` as the third argument, and `255` as the maximum value of the trackbar slider (argument four). Give `on_trackbar_slide` for trackbar callback parameter five. Argument six can be omitted.

```
int createTrackbar(
  const string &trackbarname, const string &winname,
  int *value, int count,
  TrackbarCallback onChange=0, void *userdata=0
);
```

5. The redness value is now being updated, but the `frame` variable is not yet affected. Before the call to `imshow` in the `while` loop, simply assign `cv::Scalar(0,0,g_redness)` to `frame` using `operator=`.

6. Before we use `cv::circle` we need to include OpenCV's image processing library. Add the following to the top of your code:

```
#include "opencv2/imgproc/imgproc.hpp"
```

7. Let's add a circle with the opposite redness to the background. The function declaration for `cv::circle` is below. Make a call to it in the `while` loop, between the assignment to `frame` assignment, and the `imshow` call. The first parameter is your `frame`. Use `255-g_redness` as the 3rd component of the `Scalar` colour. The radius can be `g_redness`. Use `cv::FILLED` for the thickness. Place it in the screen's centre.

```
void circle(
  Mat &img, Point center, int radius, const Scalar &color,
  int thickness=1, int lineType=8, int shift=0
);
```

OpenCV Video

Let's explore how OpenCV can help us work with video inputs.

- `05_capture_show_video_record.cpp` displays and also saves a video file of the feed captured from the webcam. Confirm that the .avi file is saved correctly and can be displayed in VLC media player or similar.
- Example program `06_capture_show_video_grab_image.cpp` displays video, but allows you to save a snapshot from it by pressing the space bar.
- Apply some of the filters described in the lecture to the outputs of one of the programs; for example, `cvtColor`, `GaussianBlur`, or `Sobel`. Look here for further filter operations. You may need the OpenCV `imgproc/imgproc.hpp` header file from the Trackbar section.
- Use the functions described in the lecture, such as `cv::circle` and `cv::line` to draw over the top of the video output of one of your programs. n.b. the origin is at the top-left.
- Create a rectangle using `cv::Rect`; call it `rect`. Draw it on the input video frame using say `cv::rectangle(frame, rect, cv::Scalar0, 0, 255)`. Then, make that section of the video gray, by calling `cv::cvtColor` *twice*: once with `cv::COLOR_BGR2GRAY`, and once with `cv::COLOR_GRAY2BGR`. Use two `cv::Mat` variables to help with this: `roi` and `grey_roi`.

Face Recognition

Modify the OpenCV face recognition program `07_simple_facerec_eigenfaces.cpp` to allow the program to also find your own face within the database. Use the following instructions to get started:

- Having run the slow `model->train`, a call is made to the faster `model->predict` method. Make another call to the `model->predict` method, providing another, arbitrary image to match against.
- Does the system need 10 images in each subdirectory of att_images? (Try moving one temporarily to find out.)
- Use the camera to collect a set of images of your own face; and place them within a new directory - say "s41"
- Ensure you crop your face to the same resolution & format as the other images

... that is: 92x112 8-bit pgm (use Gimp or XnView)

... how-to-create-a-pgm-using-gimp.md will also be helpful

- Set `rand_image_id` to the id of one of your images; you may need to examine `str` within the `for` loop. What is `images.size()` when `str` is equal to "s41"? What is `label` at that point?
- Is your face now matched?
- Modify the code to find your likeness from an image or video from the camera interactively