**COSC428 Lab 1 – Working with Cameras**

## **Objectives**

The overall goal of this lab is to introduce basic OpenCV code for displaying, recording and replaying video. Given this background, you will then use tools to change camera attributes, edit video and finally calibrate a camera to remove radial distortion (fisheye lens distortion). You will have to read and run some small applications written in Python. During this lab, you should enhance your understanding some concepts of computer vision video processing by implementing Python programs that demonstrate this.

## **Preparation**

Download the “[Lab 1 files](http://learn.canterbury.ac.nz/mod/resource/view.php?id=2371)” zip archive from Learn for a copy of the files to be used in this laboratory. The following files should now be should now be in the current directory:

camera\_driver.py, checkerboard.png, fiducial.png, fiducial.py, fiducial2.png, filelist.txt, find\_camera\_calibration.py, load\_camera\_calibration.py, load\_video\_file.py, video\_from\_webcam.py, webcam\_to\_file.py

## **Troubleshooting**

* If you get a video I/O error, change the “0” in VideoCapture to “-1” which grabs the first available camera.
* If you still cannot access the camera at any time, try unplugging it and plugging it back in.
* BTW, you can check your camera exists on Linux with: “ls /dev/video\*”

## **Display Video from a Webcam**

One of the most basic tasks in OpenCV is to simply start retrieving video data from the camera so that we can start providing our algorithms with some data. At the moment, however, we don’t have any data, so we’ll start with showing the video to the screen for now.

The integer passed into the cv2.VideoCapture() function specifies the camera used. For a computer with a single camera, using 0 will work. Otherwise, you might need to do some testing to determine which device it is that you wish to.

**To do:**

Run video\_from\_webcam.py

Add the following line of code just before imshow() and observe the effect:

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

|  |
| --- |
| E:\UC Casual Work\COSC428\labs\lab01\image.png |

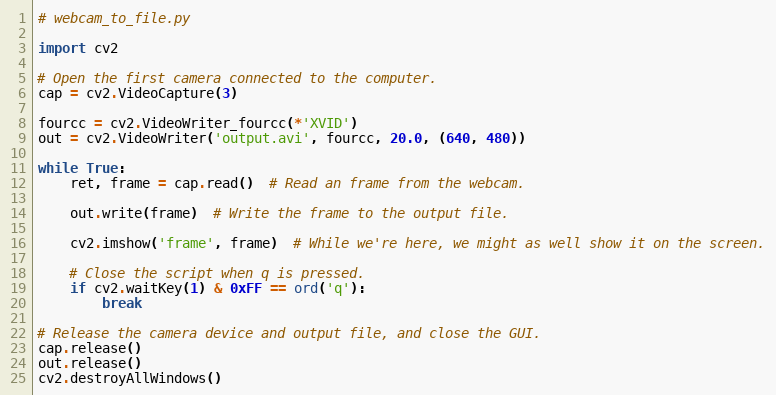
## **Save Video from a Webcam to a File**

It is convenient to save video from a webcam to file for use as a consistent test file while tweaking an algorithm. This code stores the video in an .avi file using the XVID codec, but the commands for other containers and codecs can be found in the OpenCV documentation.

**To do:**

Run webcam\_to\_file.py and check that a file *output.avi* was created in your local directory.

Play *output.avi* to check that it recorded correctly.



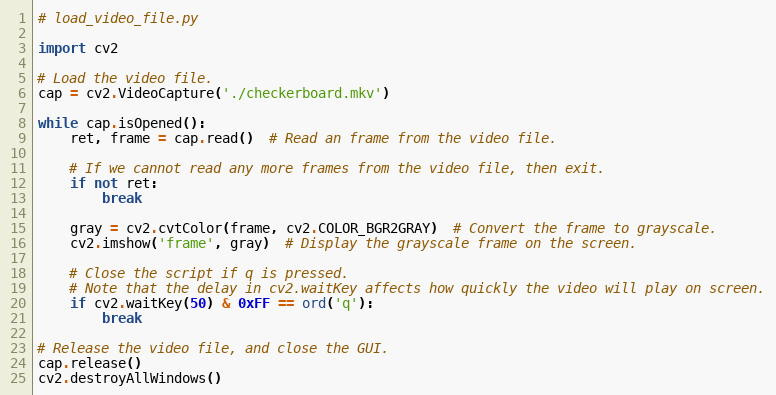
## **Load a Video from File**

Now that we have stored our video, we need a way of opening it again. As it happens, the method for capturing the video from a webcam is the same for opening a video file. Instead of specifying a number for the camera, we pass in a file name instead. In this example, the frame is converted to grayscale before being displayed, but in practice that is where you would perform your algorithm.

**To do:**

Run load\_video\_file.py

Comment out the cvtColour() line and observe the difference



# **Other Tools**

## **Webcam Driver – Guvcview**

This application is useful for directly controlling many parameters for your webcam, as well as allowing you to easily capture images and videos. These files can then be later loaded by OpenCV for testing your algorithms with a consistent data source. **Note that any changes you make to these settings will persist until the next time you reboot your computer, since you’re making changes inside the camera, rather than editing images that have already been received.**

It included by default in the Ubuntu and Mint repositories for you to install on your own machines at home.

sudo apt-get install guvcview

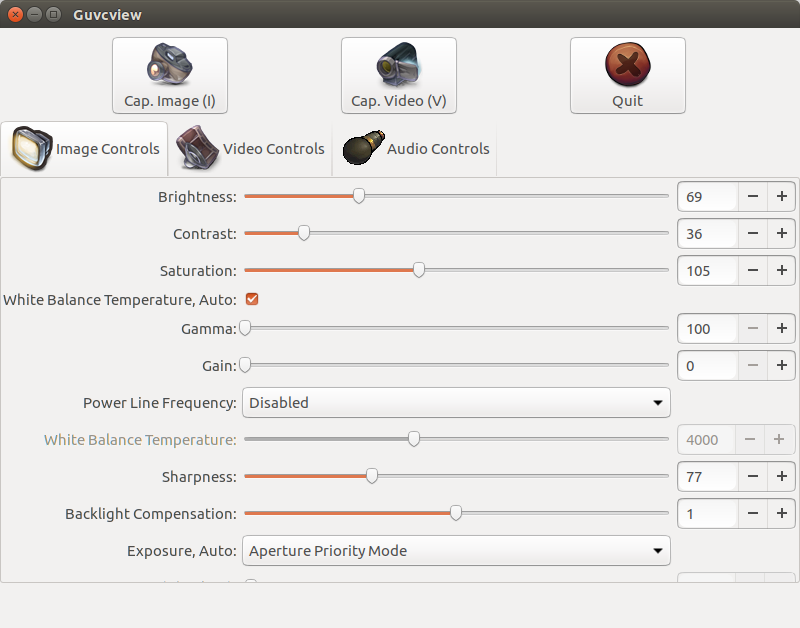


Figure 1 - Some of the settings available through guvcview

Note that if you are running guvcview inside a virtual machine, some of the options may not be available, in which case you will need to use on of the lab machines or on another device running Linux.

Also, guvcview outputs files as .mkv video files instead of the .avi files that you might be used to. .avi and .mkv are both video container formats, and OpenCV is happy to deal with both, so there’s no need to be worried about this.

### Updating webcam settings from OpenCV and Python

Changing webcam settings such as brightness, contrast and saturation can be done using Python. An example of this is in camera\_driver.py in the lab directory. All you need is a line (placed before the while loop), like

cap.set(cv2.CAP\_PROP\_BRIGHTNESS, brightness)

Where “brightness” is a value between 0 and 1. There are a range of other values that can be modified. These can be found online if you want to look for them. Alternatively, guvcview should also work fine.

**To do**: Run *guvcview* above and observe the impact of changing settings.

Add the above line of code (*cap.set()*) to video\_from\_webcam.py (and remove your added *cv2.cvtColor()* line).

Is the change persistent or does brightness revert to the original value every time you restart your program?

## **Simple Video Editing – OpenShot**

OpenShot is a simple video editing tool that can be used to quickly assemble a video from a series of shorter clips and/or still images. Additional text slides can also be created inside OpenShot from within the “Title” menu on the menu bar.

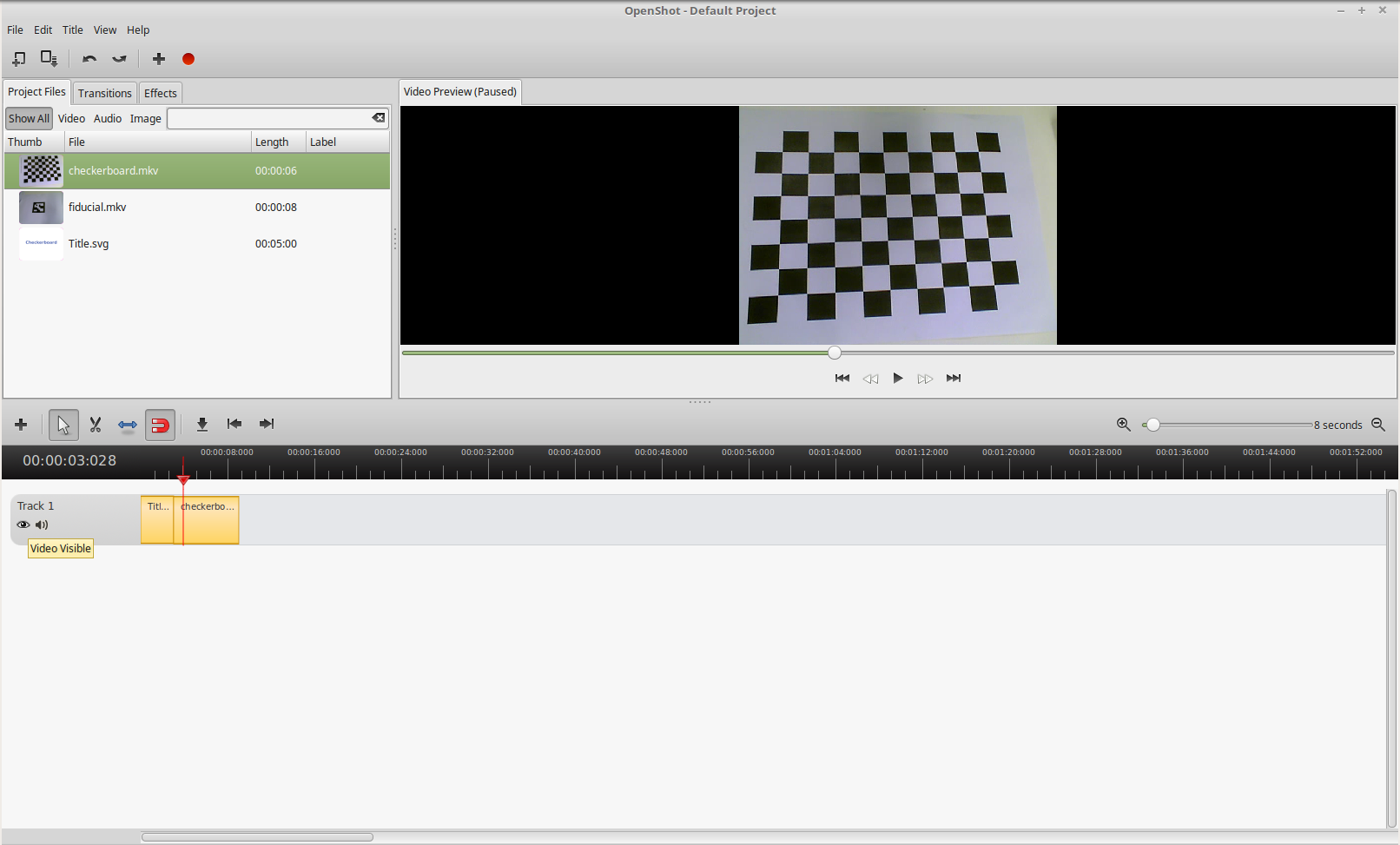


Figure 2 - The OpenShot interface

**To Do**: Do the following using the video you recorded previously.

(Note: If openshot doesn’t accept the *XVID* codec, then try *PIM1* or *MJPG* in webcam\_to\_file.py)

Videos and images are added to the project by dragging them into the “Project Files” window in the top left. From there, clips can be dragged down into the “Timeline” window at the bottom of the screen.

To preview the video, drag the red arrow in the “Timeline” window to the desired point in the video and select play in the preview window in the top right.

Once you are satisfied with your video, it can be exported to a range of formats. The export interface can be accessed under the “File” menu, or by pressing “Ctrl+E”. There are some presets available that target a range of web services, devices, etc., but additional control is available under the “Advanced” settings tab in the export window.

Once your video has been exported, it is possible to upload a video directly to YouTube through the “Upload Video” under the “File” menu.

OpenShot has a range of other features (including animations and scene transitions) that are documented on the [website](http://www.openshot.org/static/files/user-guide/index.html).

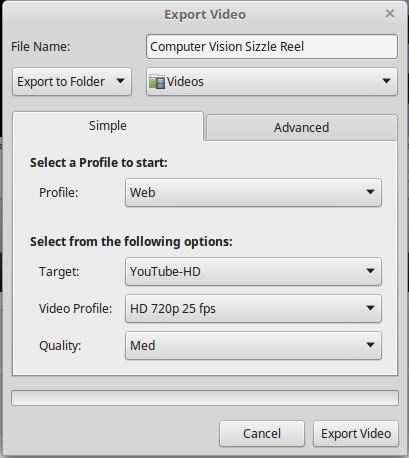


Figure 3 - The OpenShot export interface

# **Camera Calibration**

Perfection is hard, and isn’t typically cheap either. As a result, the lens and image sensor of a webcam differ slightly from device to device. Camera calibration allows one to correct distortions caused by these irregularities. The calibration parameters generated by this process can then be saved and loaded when the camera is used for a computer vision algorithm.

While modern cameras are relatively consistent, and can seem to be free from most distortion effects, even subtle changes can cause problems. For example, if an algorithm uses straight edges to detect a given feature, if there is a radial distortion introduced into the image as a result of a lens imperfection, then the algorithm may not perform correctly.

To calibrate the camera, run the “find\_camera\_calibration.py” script, and hold a checkerboard image in front of the camera and **ensure that the entire checkerboard is visible at all times**. To get a good calibration result, it is important to rotate the checkerboard, and put it at angles to the webcam, while also making sure that a range of poses are recorded all over the webcam’s field of view. (Tip: Lie checkerboard flat on desk and move webcam.)

When calibrating your camera, it is very important that the checkerboard pattern image is perfectly flat, otherwise you can have some subtle distortion effects. Using a checkerboard image on your monitor will work for camera calibration. Also, make sure to get calibration readings from the corners of the screen.

Figure 4 is an extreme case of what can occur if your checkerboard is not flat when used for calibration. Figure 5 shows how the checkerboard should look when the “load\_camera\_calibration.py” script is run.



Figure 4 - A frame from a poorly calibrated camera



Figure 5 - A frame from a well calibrated camera

The 6x9 checkerboard used for this implementation is available [here](https://docs.opencv.org/2.4/_downloads/pattern.png) and on the last page of this handout.

## **Generating the Calibration Files**

Here the camera calibration is performed with the checkerboard pattern. There is a short delay after every frame where the checkerboard is detected to limit the number of frames we need to process later.

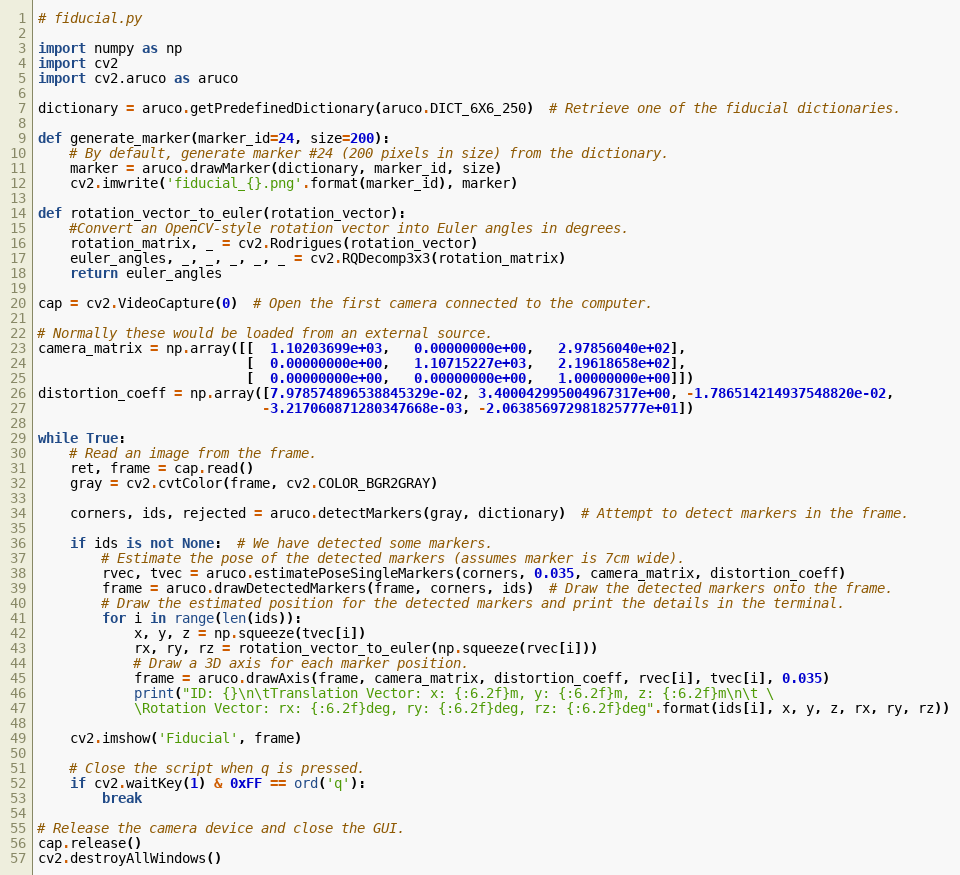
The two calibration files “*camera\_matrix.npy*” and “*distortion\_coeff.npy*” are saved for later use.

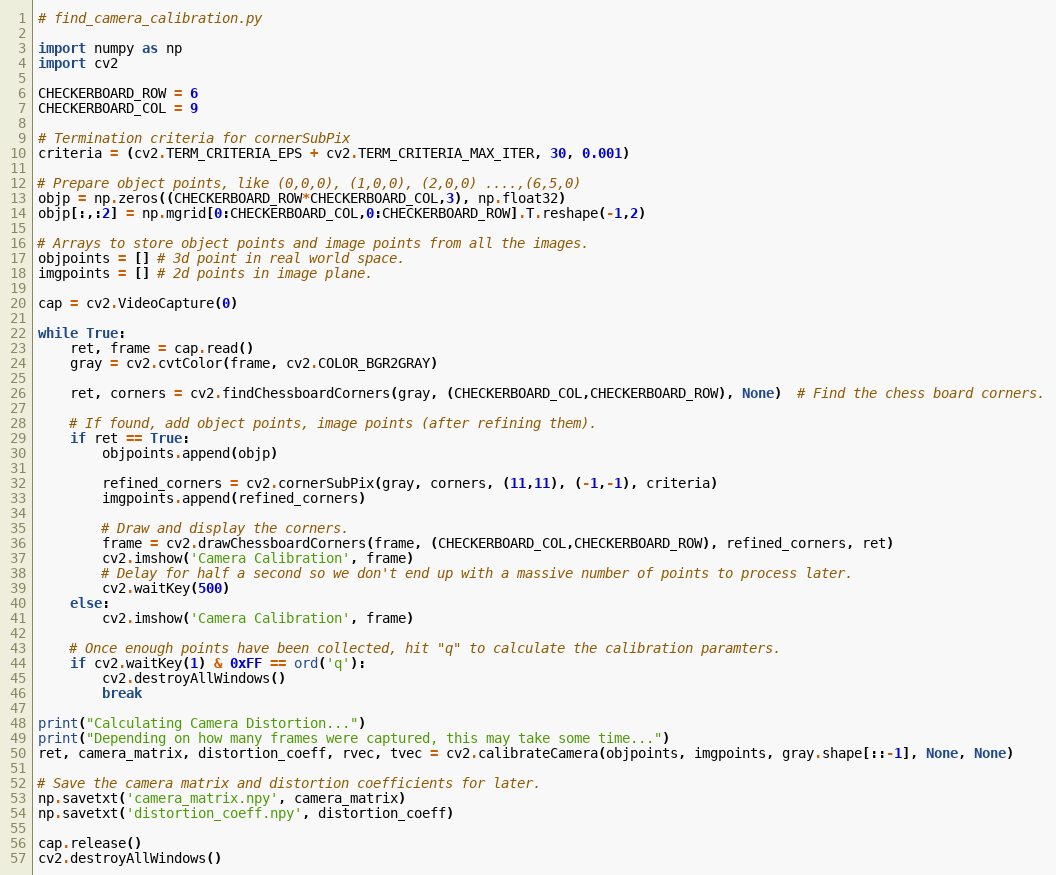
**To do:**

Run *find\_camera\_calibration.py* and collect about six images at completely different angles and positions in the image,and **then hold down “*q*” until “Calculating Camera Distortion…” is displayed**.

(Note: If you have changed brightness, contrast or saturation, change them all back to default or 50%.)

Compare your values in your two calibration files “*camera\_matrix.npy*” and “*distortion\_coeff.npy*” with those I got for my own webcam below:



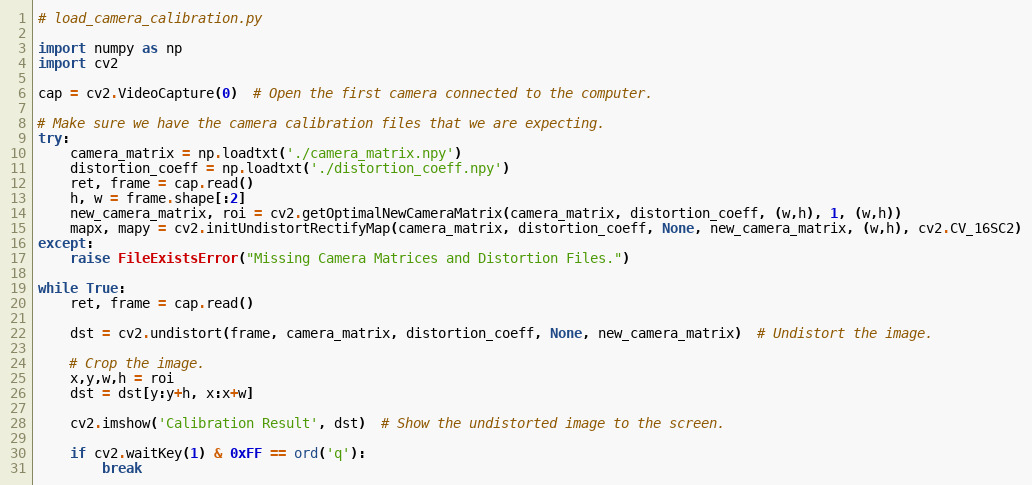


## **Loading the Camera Calibration**

This shows loading the calibration data for a camera and using it to produce an image that has been corrected for defects in the camera.

**To do:**

Replace the *dst* parameter in *imshow()* with *frame* and look for the distortion difference in straight lines in the scene/image near the edge of the camera with and without using *cv2.undistort* (i.e. using *dst* or *frame* in *imshow*).



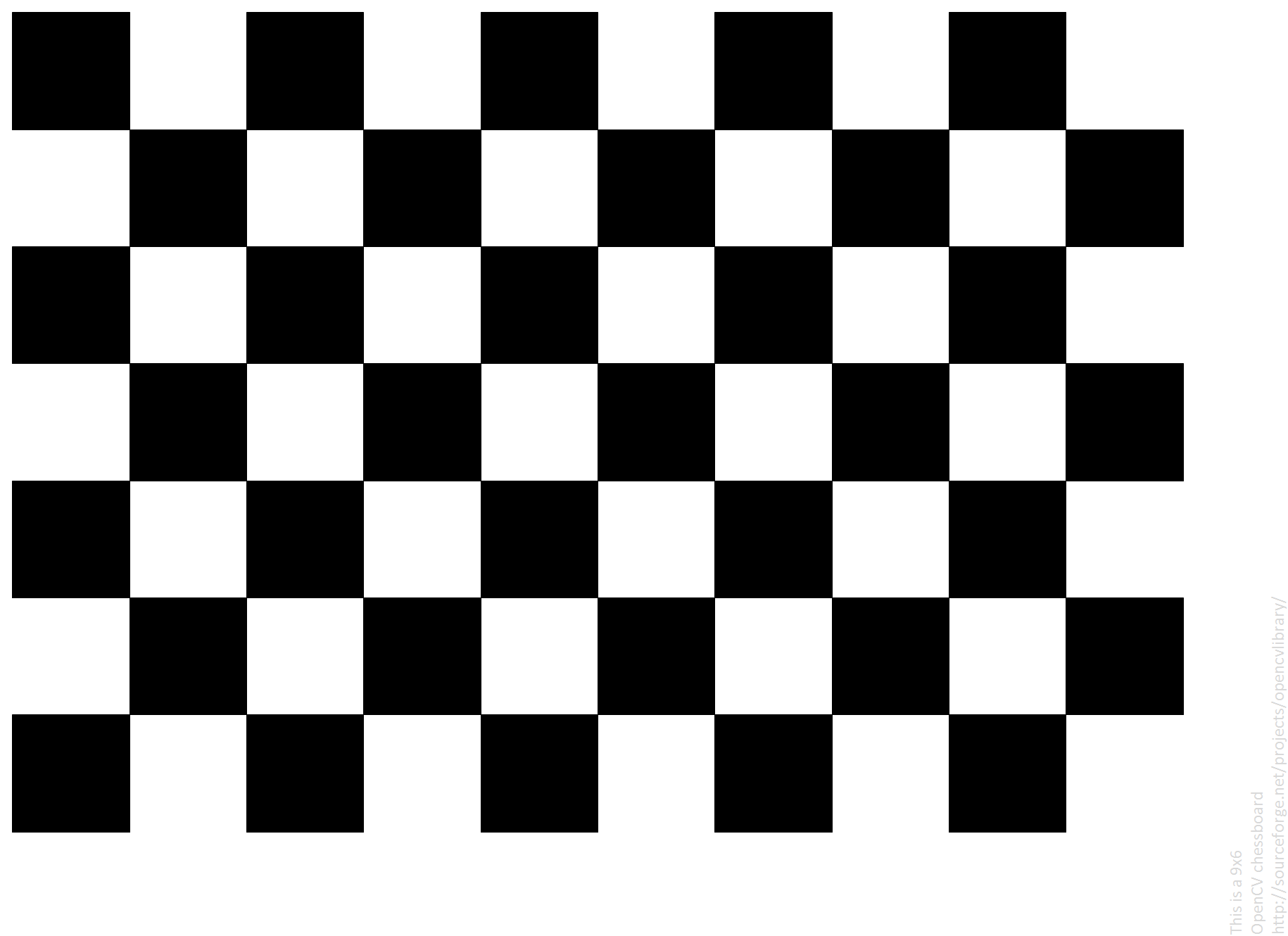


Figure 6 - 6x9 checkerboard for calibration