Exam Monitoring System based on Stereoscopic Computer Vision

Project Report

Tony Nguyen Tuan
nguyentu@ad.helsinki.fi

Jonas De Paolis
depaolis@ad.helsinki.fi

Phillip Schneider
pschneid@ad.helsinki.fi

UNIVERSITY OF HELSINKI

General Responsibilities

Project Description	Tony Nguyen Tuan, Jonas De Paolis, Phillip Schneider
Project Pitch	Jonas De Paolis
Project Report	Jonas De Paolis

Technicalities

Stereo Camera Calibration	Tony Nguyen Tuan, Jonas De Paolis
Data Generation	Tony Nguyen Tuan, Jonas De Paolis
Depth Estimation	Tony Ngyuen Tuan, Jonas De Paolis
Object Detection	Tony Nguyen Tuan, Jonas De Paolis
Incident Matching	Tony Nguyen Tuan, Jonas De Paolis
Headpose Estimation	Tony Nguyen Tuan
Web Application	Jonas De Paolis

1 Introduction

In higher education, many tasks related to fraud detection have been automated, yet exam monitoring is still conducted by teaching staff and therefore tends to be prone to human error and misestimation. With this project we would like to propose a system based on stereoscopic computer vision that provides a scalable and low-cost approach to automated exam monitoring.
As we use visual information to detect cheating attempts, our preferred way to solve this problem is to geometrically evaluate potential intersections between a person's viewing direction and any object of interest. Hence, we need to consider two points in $\mathbb{R}^3$ or, to be precise, the calculated centroids describing the position of their face and said object respectively. Additionally, we need to estimate a vector describing their viewing direction. In order to verify that the person is facing towards and potentially looking at the object, we evaluate whether the object is located on or within close proximity to the vector originating from their face.
The following sections give an in-depth explanation of both approach and technical implementation as well as the results we were able to achieve from a data scientific point of view.

2 Data Generation

Since stereoscopic computer vision requires exact knowledge about camera intrinsics as well as positioning, we had to calibrate the setup according to our specific needs. Due to the fact that all objects in our testing environment were located within 5m distance from the origin, we placed two identical Logitech C920 cameras in parallel with a comparatively short baseline of 140mm (see fig. 1), so the disparity between the recorded frames would not become too large.
Regarding our calibration workflow, we made use of the Matlab *Stereo Camera Calibrator App*[1] which generates a set of adequate stereo parameters based on chessboard detection in image pairs. In order to achieve a sufficiently low mean reprojection error, we recorded 100 image pairs and eliminated those that exceeded our cut-off limit at 0.1px per image pair. Using this approach, we were able to achieve a mean reprojection error of 0.07px. Stereo parameters were then generated automatically and stored in an object that is necessary for further processing. Fig. 2 shows the calibration interface in detail.

[1] More information on Matlab *Stereo Camera Calibrator App* at https://www.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html

For lack of processing power, our test video sequences were recorded with a low resolution of 640x480. Especially critical is their constant framerate of 30fps to ensure that each pair of frames is synchronous in itself. It is important to note that these video files can only be processed using the stereo parameters as created in the previous step.
The recorded scene depicts a single student in an exam situation who is repeatedly looking at various objects like a bottle and a calculator, both located on top of the table. Using this setting, we want to give a demonstration of our system's basic applicability.

3 Data Processing

3.1 Depth Estimation

In order to estimate depth from a visual input, we need an image pair as well as its underlying stereo parameters. Since we already had performed camera calibration in Matlab, we decided to make use of other beneficial functions from the *Computer Vision System Toolbox*[2].
We started off performing rectification of the image pair with regard to the stereo parameters so both images would be projected in parallel, i.e. corresponding points in both images would share the same y-coordinates. Based on the rectified image pair, we were then able to calculate the disparities between corresponding points by creating a disparity map. Using the latter in combination with our stereo parameters, it is possible to reconstruct the scene by projecting all points occurring in both images onto 3-dimensional space. As depicted in fig. 3, those points can be visualised in a so-called point cloud that gives an intuitive representation of the 3-dimensional world surroundings.
When computing disparity, a monotonous scene can lead to a substantial amount of missings in our point cloud, as the disparity estimation algorithm is dependent on distinctive points for good results. Although we prevented this problem using an adequate scene, we tried filling the few remaining gaps using interpolation, so we would not run into errors when estimating the depth of concrete points. The disparity estimation algorithm we used is called SemiGlobal and considers disparity ranges of up to 128px distance.

[2] Relevant functions are rectifyStereoImages(), disparity() and reconstructScene(). More information on Matlab *Computer Vision System Toolbox* at https://www.mathworks.com/help/vision/index.html

3.2 Object Detection

Before we can retrieve the 3-dimensional world coordinates of an object, we first need to find its position in a 2-dimensional image. Object Detection itself therefore only needs a single image instead of an image pair, usually deriving from the left camera channel. As we intend to locate a face as well as multiple objects, we implemented different pre-trained models for either task. In view of face detection, we decided to use the Matlab *cascadeObjectDetector* system object[3] that implements the Viola-Jones Algorithm to detect facial features in images. In particular, we estimate the face position recognizing a person's nose or, if the face is turned sideways, either one of their eyes. Although it is virtually impossible to narrow down the set of objects potentially used for cheating, we decided to train our own model using multiple Matlab *acfObjectDetector* system objects[4] which are capable of differentiating between multiple classes of objects and can easily be improved by feeding them further training data. Fig. 4 shows the results of both face and object detection.

Having detected face and objects, we then calculate all of their centroids in 2-dimensional space as a means of simplification. Knowing both x- and y-coordinates, we can easily infer the z-coordinate by reading it from our previously generated point cloud. This way, we get an accurate position of each centroid in 3-dimensional space that can be used for further computation.

3.3 Headpose Estimation

To put the centroids of both face and objects into spatial perspective, we need to retrieve information about the person's viewing direction. Since this is a rather complex task, we decided to build upon a pre-existent solution called *head-pose-estimation*[5] that is implemented in Python and based on OpenCV and dlib. Said solution uses cascaded regression trees to predict the head position, or viewing direction respectively, with regard to 68 unique facial landmarks on a single image. As is the case with object detection, only 2-dimensional input is needed in order to estimate 3-dimensional information.

---

[3] More information on Matlab *CascadeObjectDetector* system object at https://www.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html

[4] More information on Matlab *acfObjectDetector* system object at https://www.mathworks.com/help/vision/ref/acfobjectdetector.html

[5] More information on *head-pose-estimation* at https://github.com/lincolnhard/head-pose-estimation

Instead of a video feed, we modified the code to take in a single image as we will need to use its output for further computation one frame at a time. The function returns a precise 3-dimensional vector of angles in degrees: the x-coordinate (pitch) denotes the vertical position, the y-coordinate (yaw) denotes the horizontal position and the z-coordinate (roll) describes how the head is tilted sideways. These values are then applied to parametrise the line describing the center of a person's visual field. Said visual field, as produced by the algorithm, is depicted in fig. 5.

3.4 Incident Matching

The main function simultaneously loops through both video files frame by frame and with every step calls three functions that return the 3-dimensional position of the detected face, the 3-dimensional positions of all detected objects as well as the 3-dimensional vector describing the face's orientation. It then computes the distances between all object positions and the line resulting from the headpose vector, selects the closest object and checks if it lies within a pre-defined proximity range. If it does, the frame resulting from the left video channel gets visually marked, otherwise it stays unchanged. With each frame belonging to an incident, an iterator runs to examine compliance with pre-defined minimum length requirements. A frame belongs to an incident if it either got marked or occurred within a pre-defined timeout period after the last marked frame. Whenever timeout is reached, a new incident is created.
Primary output of our incident matching algorithm is a modified version of our left channel video file. Additionally, we save incident ID, starting point as well as all marked frames per incident to later display them in our web application.

4 Prototype

To demonstrate how we envision our solution to work in practice, we built a web application that covers some basic functionality using HTML, CSS, JavaScript and jQuery. At this point, we decided to omit upload and export options since currently there is no way for users to test them. As stated before, this solution only works with our provided video footage, as otherwise one would need to generate their own stereo video files including corresponding stereo parameters. Because Matlab is not an open-source project, it is prohibited for non-licensed public use. For all the above reasons, our web application is to be considered a rather static output because we did not yet establish a database to go along with our data. All images and information is saved on the local machine running the application.

According to the approach described in section 3.4, our incident matching algorithm processes the input video and returns the marked output video as well as detailed information on each incident including incident ID, starting point and sample frames for manual verification. Using the interface it is possible to review given information, replay the video footage and verify each incident. Fig. 6 depicts a screenshot of the web application.

5 Conclusion

The aim of this project was to build a computer vision system capable of detecting students' cheating attempts in exam situations. Simplifying the testing environment by monitoring only one person at a time, we were able to achieve excellent results detecting most instances of deception in our video footage. Although all incidents were chosen to be quite obvious, this goes to show that by further working on our solution we could potentially scale this solution up to monitor whole lecture halls with high accuracy using only two cameras. Next steps include a refinded model for dealing with incidents involving objects located out of camera view and the introduction of eye-tracking as well as reinforcement learning for better accuracy.

For now, we are able to extract substantial knowledge from simple images. As it was not possible to obtain actual exam video footage, this project did not use real-world data. However, it solves a real-world problem and serves as a proof of concept for future analyses. From a data scientific perspective, the presented solution gives plenty of opportunity to build upon for further statistical insight. Summarizing and aggregating collected data may lead to more beneficial information that could potentially be used by universities not only for fraud detection but also to prevent people from cheating in the first place.

Appendix

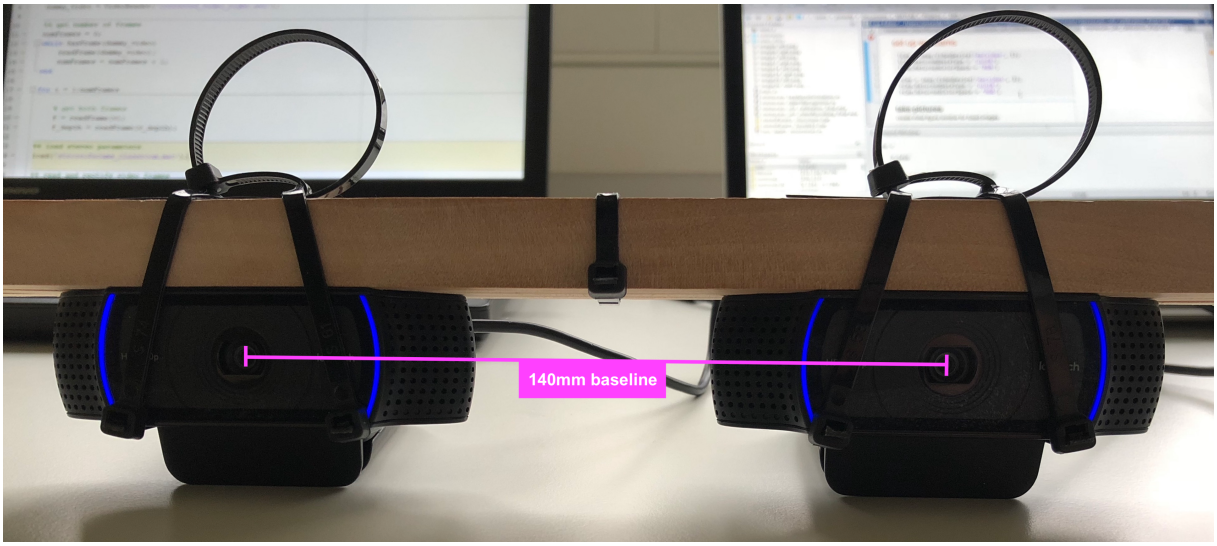

Fig. 1: The stereo camera setup involves two Logitech C920 webcams positioned in parallel with a 140mm baseline between both lenses.

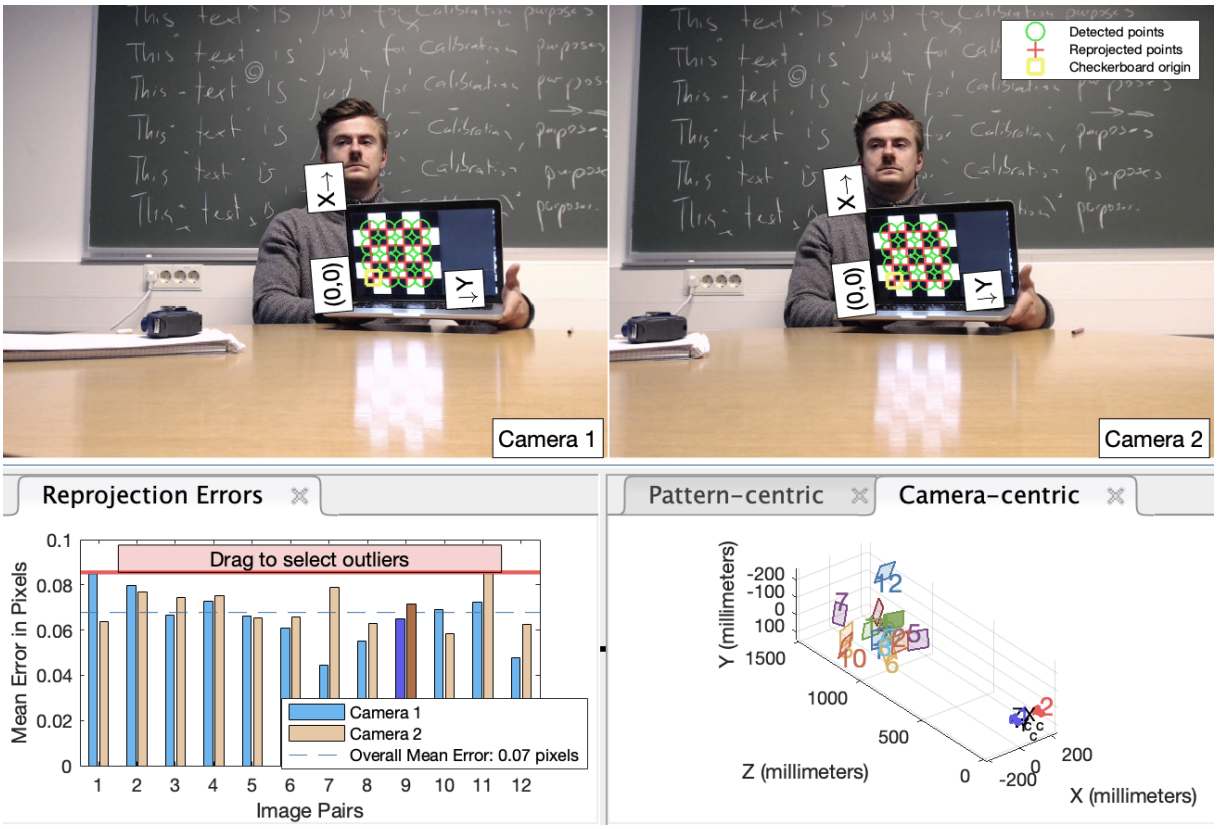

Fig. 2: The Stereo Camera Calibrator App detects an arbitrary chessboard in image pairs. Here, a chessboard is detected on the laptop screen. By providing the application with the real-world size of 35mm per chessboard square, its interface computes multiple metrics to evaluate calibration accuracy such as reprojection errors (bottom left) and camera extrinsics (bottom right).

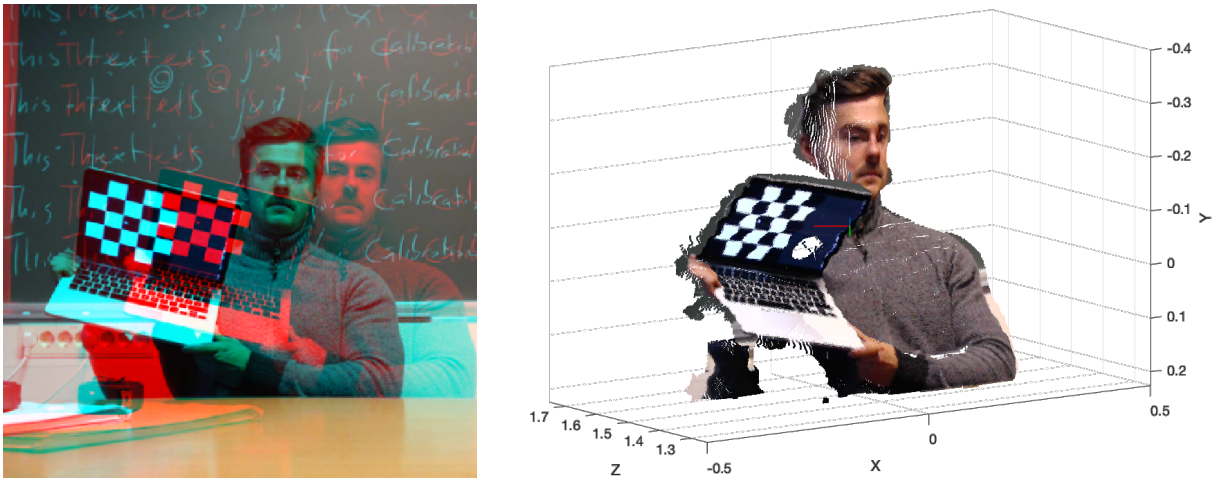

Fig. 3: The rectified stereo image pair (left) shows how both images are parallel in that the corresponding points share the same y-coordinate. Based on the disparity between those points and our stereo parameters, we can create a point cloud (right) to reconstruct the 3-dimensional scene and determine depth for each pixel.

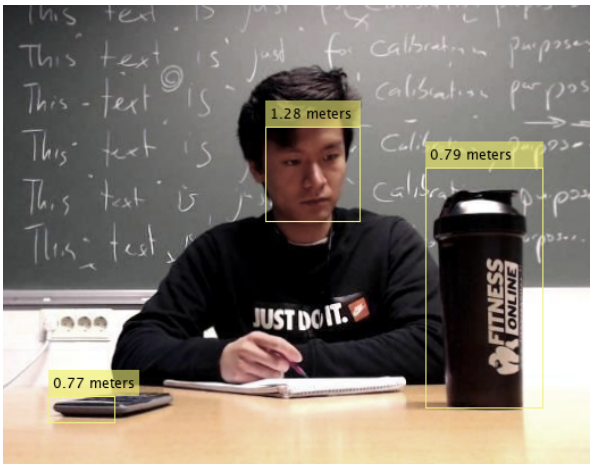

Fig. 4: Using object detection we can calculate the centroids of various objects in our 2-dimensional images. Projecting them onto the point cloud, we receive the objects' 3-dimensional locations.

Fig. 5: The headpose estimation algorithm returns pitch, yaw and roll angles. In this particular example, Tony is facing 6.81° downwards, 33.1° to the left side and also tilts his head 12.03° to the left.

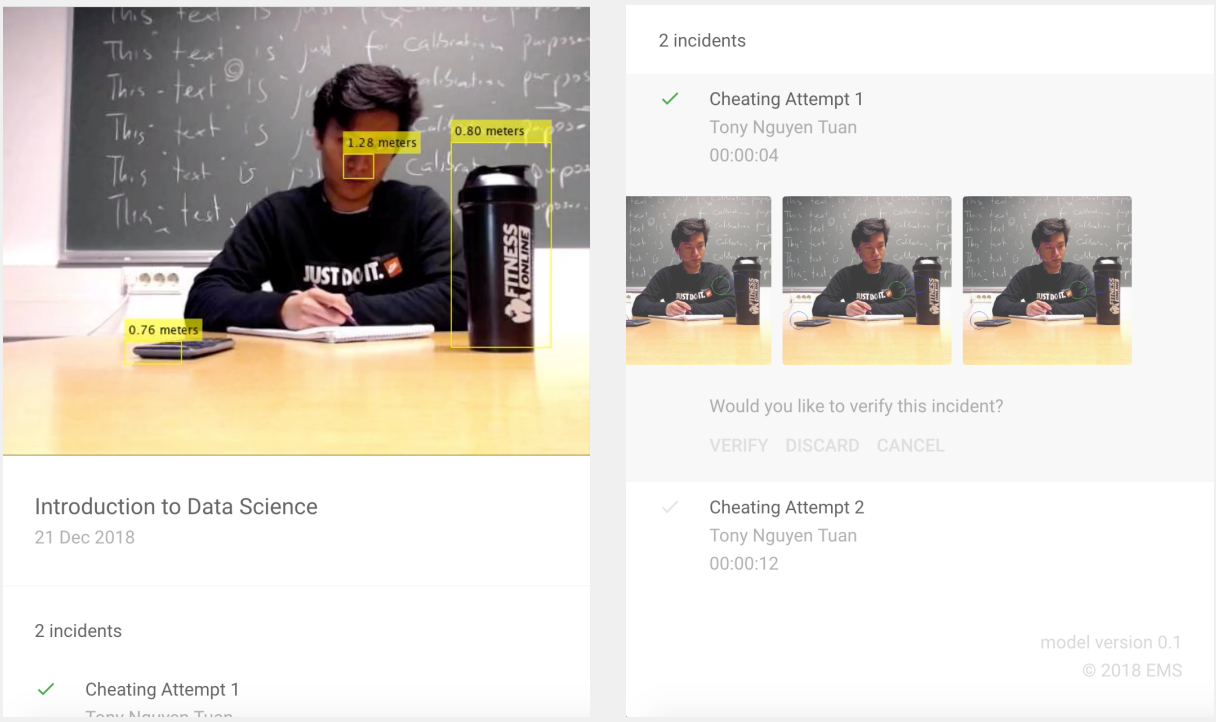

Fig. 6: Our prototype can be used to play back the processed video and to manually verify each detected incident by reviewing the affected video frames (left). It further provides aggregate information on the temporal distribution of incidents.