

Attendance system based on the face recognition of webcam's image of the classroom

**Submitted in partial fulfillment of the requirements
for the award of**

**Bachelor of Engineering degree in
Computer Science and Engineering**

By

**Abbireddy Mahesh Bhupathi Rayudu
(Reg. No. 38110002)
Areti charan sri ram sai
(Reg. No. 38110045)**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC**

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

MARCH - 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with “A” grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Abbireddy Mahesh Bhupathi rayudu(38110002)** and **Areti charan sri ram sai (38110045)** who carried out the project entitled “**Attendance system based on the face recognition of webcam’s image of the classroom**” under my supervision from **October 2021** to **March2022**.

Internal Guide

Mrs.Ramya G Franklin, M.E.,

Head of the Department

DR. VIGNESHWARI S, M.E., Ph. D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, **Abbireddy Mahesh Bhupathi rayudu** (38110002) and **areti charan sri ram sai**(38110002) hereby declare that the Project Report entitled “**Attendance system based on the face recognition of webcam’s image of the classroom**” done by me under the guidance of Mrs. RAMYA G FRANKLIN(Internal)is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering / Technology degree in **Computer Science and Engineering**.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of SATHYABAMA for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to Dr.T.Sasikala M.E., Ph. D, Dean, School of Computing, Dr.S.Vigneshwari M.E., Ph.D., and Dr.L.Lakshmanan M.E., Ph.D., Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Mrs.RAMYA G FRANKLIN M.E., for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways for the completion of the project.

ABSTRACT:

The system should be able to detect students' frontal faces in a classroom within 30% accuracy. The system should be able to automatically reveal the number of students present on a GUI. Recognise student stored on a database of faces by matching them to images on a database with an accuracy within 30%. The system should be able to match detected students faces cropped from an image to those on a database on the system. The system should be able to process an image within 10 minutes to be able to achieve the objective of recognition by the end of a lecture. i.e. 5 names per hour per lecture. The algorithm implemented for the system's functionality will achieve system accuracy within 20%. The positive prediction should be within 20%. The system designed will be user friendly with a Graphical User Interphase that will serve as an access to the functionalities of the system.

TABLE OF CONTENTS

chapter	TITLE	PAG E NO
	ABSTRACT	5
1	Introduction	9
2	Literature survey	18
2.1	Accurate and Robust Facial Capture Using a Single RGBD Camera	18
2.2	Face Detection with a 3D Model	19
2.3	Real-time 3-D face tracking and modeling from a webcam	20
2.4	FACECEPT: REAL TIME 3D FACE TRACKING AND ANALYSIS	20
2.5	DEVELOPMENT OF REAL TIME FACE RECOGNITION SYSTEM USING OPENCV	21
2.6	REALTIME DATA ACQUISITION BASED ON OPENCV FOR CLOSE-RANGE PHOTOGRAMMETRY APPLICATIONS	21
2.7	Creating a real model from a video by a single camera	22
3	SYSTEM ANALYSIS	24
3.1	EXISTING SYSTEM	24
3.2	DISADVANTAGES	24
3.3	PROPOSED SYSTEM	24
3.4	Advantages	25
4	DIAGRAMATIC REPRESENTATION	25
4.1	BLOCK DIAGRAM	25
4.2	UML DIAGRAM	26
4.3	DATA FLOW DIAGRAM	27
4.4	ARCHITECTURE DIAGRAM	28
5	SOFTWARE MODULES	28
5.1	DIFFERENT APPROACHES OF FACE RECOGNITION	28
5.2	FACE DETECTION	29
5.2.1	Pre-Processing	30
5.2.2	Classification	30

5.2.3	Localization	30
6	DIGITAL IMAGE PROCESSING	39
7	OPEN CV	67
7.1	PROGRAMMING LANGUAGE	69
7.2	HARDWARE ACCELERATION	70
7.3	OS SUPPORT	70
8	CONCLUSION	71
9	RESULTS	87
10	REFERENCES	88

LIST OF FIGURES

SI NO	TITLE	PAGE NO
1	BLOCK DIAGRAM	25
2	UML DIARAM	26
3	DATA FLOW DIAGRAM	27
4	ARCHITECTURE DIAGRAM	28

CHAPTER 1

INTRODUCTION

We address the problem of 3-D face tracking, which is a fundamental problem in the context of human computer interaction. The input consists in a video stream acquired by a low-cost webcam pointing to the user's face. Our efforts focus on real-time and robust full 3-D face tracking (6 DOF), in the presence of facial expression changes, wide pose variations, and partial occlusions. To this end, we make very few assumptions compared to most of the state-of-the-art approaches. The distance to the camera should comply with the requirements of a facial interactive application, but can vary according to the focal length used. Also, we require a near frontal pose at initialization. The proposed approach includes the use of a generic 3-D face model to track in real-time the 3-D head motion of users and produce accurate and textured face models. As illustrated in Fig. 1, our application is composed of three main modules: initial 3-D model fitting, 3-D head tracking, and re-acquisition. The applications of the 3-D tracker and 3-D model are not discussed here. We need a 3-D model, which can either be a generic model, or a specific one retrieved from a database. In the initial 3-D modeling step, the model is warped orthogonally to the focal axis in order to fit to the user's face in an input image, by matching 2-D facial landmarks extracted at run time. The initial 3-D reconstruction of the face model is a critical step.

Face detection is a computer technology that is being applied for many different applications that require the identification of human faces in digital images or video. It can be regarded as a specific case of object-class detection, where the task is to find the locations and sizes of all objects in an image that belongs to a given class. The technology is able to detect frontal or near-frontal faces in a photo, regardless of orientation, lighting conditions or skin colour.

Using a certain algorithm to detecting human faces within an Image (Detect Human Faces on Farm full with that's when you use Face Detection. On the flip side, Face recognition describes a biometric technology that goes way beyond recognizing when a human face is present. It actually attempts to establish whose face it is. In this article, I'm not going deep into recognizing. I'll keep that

for a future blog article and for the time being, I'm going to explain how to run a simple Face Detection program using your WebCam with Python.

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

I recommend you to save that Pdf and read it when you have a chance if you plan to stay in this face detection/recognition plus machine learning rock star! In this paper, they have introduced the concept of **Cascade of Classifiers**. Haar-cascade Detection in OpenCV

OpenCV comes with a trainer as well as a detector. If you want to train your own classifier for any object like a car, planes, etc. you can use OpenCV to create one. Its full details are given here: [Cascade Classifier Training](#).

Here we will deal with detection. OpenCV already contains many pre-trained classifiers for face, eyes, smile, etc. Those XML files are stored in ***opencv/data/haarcascades/*** Folder. Let's create a face detector with OpenCV.

The growing interest in computer vision of the past decade. Fueled by the steady doubling rate of computing power every 13 months, face detection and recognition has transcended from an esoteric to a popular area of research in computer vision and one of the better and successful applications of image analysis and algorithm based understanding. Because of the intrinsic nature of the problem, computer vision is not only a computer science area of research, but also the object of neuro-scientific and psychological studies, mainly because of the general opinion that advances in computer image processing and understanding research will provide insights into how our brain work and vice versa. Because of general curiosity and interest in the matter, the author has proposed to create an application that would allow user access to a particular machine based on an in-depth analysis of a person's facial features. This application will be developed using Intel's open source computer vision project, OpenCV

Face recognition is an easy task for humans. Experiments in [6] have shown, that even one to three day old babies are able to distinguish between known faces. So how hard could it be for a computer? It turns out we know little about human recognition to date. Are inner features (eyes, nose, mouth) or outer features (head shape, hairline) used for a successful face recognition? How do we analyze an image and how does the brain encode it? It was shown by David Hubel and Torsten Wiesel, that our brain has specialized nerve cells responding to specific local features of a scene, such as lines, edges, angles or movement. Since we don't see the world as scattered pieces, our visual cortex must somehow combine the different sources of information into useful patterns. Automatic face recognition is all about extracting those meaningful features from an image, putting them into a useful representation and performing some kind of classification on them. Face recognition based on the geometric features of a face is probably the most intuitive approach to face recognition. One of the first automated face recognition systems was described in marker points (position of eyes, ears, nose,) were used to build a feature vector (distance between the points, angle between them, ...). The recognition was performed by calculating the Euclidean distance between feature vectors of a probe and reference image. Such a method is robust against changes in illumination by its nature, but has a huge drawback: the accurate registration of the marker points is complicated, even with state of the art algorithms. Some of the latest work on geometric face recognition was carried out in [7]. A 22-dimensional feature vector was used and experiments on large datasets have shown, that geometrical features alone don't carry enough information for face recognition. The Eigen faces method described in [8] took a holistic approach to face recognition: A facial image is a point from a high-dimensional image space and a lower-dimensional representation is found, where classification becomes easy. The lower-dimensional subspace is found with Principal Component Analysis, which identifies the axes with maximum variance. While this kind of transformation is optimal from a reconstruction standpoint, it doesn't take any class labels into account. Imagine situation where the variance is generated from external sources, let it be light. The axes with maximum variance do not necessarily contain any discriminative information at all, hence a classification becomes impossible. So a class-specific projection with a Linear Discriminant Analysis was applied to face recognition in [9].

. The basic idea is to minimize the variance within a class, while maximizing the variance between the classes at the same time

1). Recently various methods for a local feature extraction emerged. To avoid the high-dimensionality of the input data only local regions of an image are described, the extracted features are (hopefully) more robust against partial occlusion, illumination and small sample size. Algorithms used for a local feature extraction are Gabor Wavelets, Discrete Cosines Transform and Local Binary Patterns. It's still an open research question how to preserve spatial information when applying a local feature extraction, because spatial information is potentially useful information.

Development of the technology in the area of cameras, computers and algorithms for 3D reconstruction of the objects from images resulted in increased popularity of the photogrammetry. With development of new, affordable cameras even the price of the photogrammetric survey has dropped. According to the measurement characteristics, cameras can be divided into two categories, metric and non-metric cameras. Although metric cameras have better measurement characteristics, their price is significantly higher. Non-metric cameras, on the other side, are much more affordable and in most cases, are sufficient for the certain application, given that determination of the Internal Orientation (IO) parameters i.e. calibration is carefully done. Algorithms for the 3D model reconstruction are so advanced that non-professionals can make a 3D model of photographed object, even though in that case accuracy of the model is questionable. That very fact, that it is so easy nowadays to create a 3D model was our motivation for this paper. With minimal expenses, we created stereo system based on the open technology. Our stereo system consists of two Xiaomi Yi (Xiaomi, 2017) cameras, mounted on a plastic frame. Currently on the market, there is considerable amount of commercial software for the 3D reconstruction e.g. Agisoft (Agisoft PhotoScan, 2017), Pix4D Mapper (Pix4D SA, 2017) based on the Structure from Motion (SfM) algorithms, so we decided to use open source alternative that is under the constant development, which in this case was the OpenCV.

In reference to the photogrammetry, rather than the Computer Vision (CV), OpenCV is mostly mentioned as a tool for camera calibration (Balletti et. al., 2014), and much rarely as a tool for the 3D reconstruction. Even though various

authors examined different application of OpenCV Development of the technology in the area of cameras, computers and algorithms for 3D reconstruction of the objects from images resulted in increased popularity of the photogrammetry. With development of new, affordable cameras even the price of the photogrammetric survey has dropped. According to the measurement characteristics, cameras can be divided into two categories, metric and non-metric cameras. Although metric cameras have better measurement characteristics, their price is significantly higher. Non-metric cameras, on the other side, are much more affordable and in most cases, are sufficient for the certain application, given that determination of the Internal Orientation (IO) parameters i.e. calibration is carefully done. Algorithms for the 3D model reconstruction are so advanced that non-professionals can make a 3D model of photographed object, even tough in that case accuracy of the model is questionable. That very fact, that it is so easy nowadays to create a 3D model was our motivation for this paper.

With minimal expenses, we created stereo system based on the open technology. Our stereo system consists of two Xiaomi Yi (Xiaomi, 2017) cameras, mounted on a plastic frame. Currently on the market, there is considerable amount of commercial software for the 3D reconstruction e.g. Agisoft (Agisoft PhotoScan, 2017), Pix4D Mapper (Pix4D SA,2017) based on the Structure from Motion (SfM) algorithms, so we decided to use open source alternative that is under the constant development, which in this case was the OpenCV. In reference to the photogrammetry, rather than the Computer Vision (CV), OpenCV is mostly mentioned as a tool for camera calibration (Balletti et. al., 2014), and much rarely as a tool for the 3D reconstruction. Even tough various authors examined different application of OpenCV

Development of the technology in the area of cameras, computers and algorithms for 3D reconstruction of the objects from images resulted in increased popularity of the photogrammetry. With development of new, affordable cameras even the price of the photogrammetric survey has dropped. According to the measurement characteristics, cameras can be divided into two categories, metric and non-metric cameras. Although metric cameras have better measurement characteristics, their price is significantly higher. Non-metric cameras, on the other side, are much more affordable and in most cases,

are sufficient for the certain application, given that determination of the Internal Orientation (IO) parameters i.e. calibration is carefully done. Algorithms for the 3D model reconstruction are so advanced that non-professionals can make a 3D model of photographed object, even though in that case accuracy of the model is questionable. That very fact, that it is so easy nowadays to create a 3D model was our motivation for this paper. With minimal expenses, we created stereo system based on the open technology. Our stereo system consists of two Xiaomi Yi (Xiaomi, 2017) cameras, mounted on a plastic frame. Currently on the market, there is considerable amount of commercial software for the 3D reconstruction e.g. Agisoft (Agisoft PhotoScan, 2017), Pix4D Mapper (Pix4D SA, 2017) based on the Structure from Motion (SfM) algorithms, so we decided to use open source alternative that is under the constant development, which in this case was the OpenCV. In reference to the photogrammetry, rather than the Computer Vision (CV), OpenCV is mostly mentioned as a tool for camera calibration (Balletti et. al., 2014), and much rarely as a tool for the 3D reconstruction. Augmented reality (Schmidt et. al., 2002) and various other sciences where near real-time or post processing position is needed. The main goal of this paper is to examine the possibility of obtaining 3D data for the purpose of the close-range photogrammetry, based on the open source technologies. This goal is chosen mainly for the sake of archeology in Croatia, where the archaeological artifacts are mostly documented descriptively or with the 2D images (Gajski et. al., 2016). In this paper application of open technology for purposes of the easier obtaining of 3D models is examined

The system we used is consisted of two Xiaomi Yi cameras. Xiaomi Yi is the low cost action camera with respectable performance, and is adequate for certain photogrammetric purposes (Gašparović & Gajski, 2016). Cameras were mounted on the solid plastic frame in a way that their optical axes are approximately parallel and fixed, simulating normal case in photogrammetry. The baseline of the stereo system is roughly set to be 10 cm, but real value of the baseline is determined with procedure of so-called stereo calibration. Special attention was given to the procedure of individual calibration of cameras, because of the nature of the stereo calibration. After the individual camera calibration, the stereo calibration with predetermined and fixed IO parameters was performed.

A baseline of the stereo systems is in direct correlation with optimal distance of cameras from object of interest. Optimal distance from object of interest is five to ten times of the baseline distance, which in this case means that with baseline of 10cm, this stereo system is optimized to capture and reconstruct objects on distance of 50 cm to 1 m (McGlone, 2013). With the short baseline, the system maintains physical robustness even in the dynamic applications

With the development of the sensors in the last decade, quality of the cameras on the market improved, as well. Camera calibration is mandatory step in photogrammetric 3D reconstruction. In this paper, a large significance is given to the camera calibration. Xiaomi Yi is the camera with the fisheye ultra wide lens of 155° Field Of View (FOV), and initial tests confirmed that we should use the two step calibration procedure (Gajski & Gašparović, 2015) in order to get sufficient quality of the calibration. The calibration procedure was done according to the model implemented in OpenCV, and presented in the paper A Flexible New Technique for Camera Calibration (Zhang, 2000). This method uses 2D chessboard as a Ground Control Point (GCP) field. Totally 13 images were used for calibration of each camera. During process of the calibration we encountered the problem of OpenCV's module inability to detect chessboard on image. This problem was solved with the radiometric manipulation on the images in the open source software GIMP (version 2.8.14) (GIMP, 2017), after tests proved that such manipulation doesn't influence the values of the chessboard detection. OpenCV's camera calibration module uses pinhole model of the camera and the Brown's lens distortion model (Brown, 1966). Since the two-step camera calibration method is used, this camera and the distortion model are sufficient, even though this camera has a fisheye lens. The equation (1) represents the image coordinates relieved of the distortion after Brown's model, where the x and the y are reduced image coordinates, and the r is square radial distance.

For the calibration purposes, we determined all parameters of distortion, 3 parameters (k_1 , k_2 , k_3) of radial distortion and the both parameters (p_1 , p_2) of tangential distortion. Reprojection errors for both cameras were 1.2 pixels. Such a large error is due to photogrammetrical approach to the calibration problem. Calibration chessboard was as close as possible to the edge of the image in order

to eliminate the distortion through the whole cadre, and since the distortion has greatest influence on the edges of the image, it increases average reprojection error, which may lead to the misleading conclusions on quality of the calibration. To ensure that the distortion parameters are well determined, visual inspection and another calibration in the Agisoft Lens (version 0.4.2) is done. Visual inspection is done in term of the inspection of the straight lines in the certain images.

Term “stereo calibration” in the classic photogrammetry doesn’t exist, but in CV it represents procedure of obtaining IO elements for both cameras and their Relative Orientation (RO). The stereo calibration is calculated based on the image pairs of the calibration field, the spatial resection is calculated and the External Orientation (EO) elements i.e. spatial and rotational elements of both cameras in reference to the chessboard are obtained. With that data, their position in respect to one another is calculated. Task of the stereo calibration is approached with the photogrammetrical point of view. The calibration parameters obtained during the individual camera calibration are used to calculate the undistorted image points of the points on images acquired for the purpose of stereo calibration. The IO elements entered iterative algorithm of stereo calibration as a fixed value, and images used for the stereo calibration covered only middle section of the cadre where distortion model preforms well, in order to provide the best geometry the for the resection, unlike images for the individual camera calibration. The OpenCV’s method for the stereo calibration returns the translation vector and the rotation matrix of the left camera in respect to the right and the fundamental matrix. Rotation and translation vectors are later used for the image rectification, and the fundamental matrix is used for the calculation of epipolar lines

After the stereo calibration, the RO elements of the cameras are obtained. Considering that the RO elements are known, with the procedure of triangulation it is possible to compute the coordinates of each point in the object space that is visible on both images. The calculated points in the object space would be referenced to the coordinate system of the stereo system. That approach is possible but slow and unpractical for large amount of the points that are required for most uses in close-range photogrammetry. “Normal case” in the stereo photogrammetry is the special case of orientation of the stereo cameras,

where only difference in the orientation of stereo cameras is translation in direction of the X axis. Axis Z is directed perfectly to the nadir, which implies that the stereo system is perfectly horizontal. The Y axis is forming right coordinate system. In this special case, every point in the object space has the same y coordinate in both image coordinate systems. Based on the difference of x coordinate i.e. x parallax, distance to the point is calculated. In this special case of the RO, problem of the point correspondence is simplified. This is the ideal case and in practical applications it is not achievable. However, with usage of epipolar geometry and the image transformations it is possible to simulate that special case. Generally, the problem of the identification of the corresponding points is time-consuming process and often leads to the identification of wrong corresponding points. That is a consequence of the uniform pattern objects appearance. In the stereo photogrammetry, that problem is approached with the introduction of the epipolar constraint. Epipolar geometry is projective geometry between two views (Hartley and Zisserman, 2003). In epipolar geometry, every image point on the first image is manifested as a line on the second image (Figure 2), and somewhere on that line lies the corresponding point of the first point. That is so-called, epipolar constraint.

The developed algorithm for 3D data acquisition was tested by calculating the spatial difference between the 3D point cloud produced in the OpenCV and 3D point cloud acquired based on the same stereo pair, in the software Agisoft Photoscan (version 1.2.6). Accuracy of the 3D point cloud produced with the OpenCV was tested in reference to the full 3D model produced by the images taken with the camera Canon Rebel XsI with the lens of focal length 55 mm. Cannon Rebel XsI was also calibrated with Agisoft Lens which also uses the Brown's model of lens distortion (1). Self-calibration was not employed in this case, since the most of the detected points were in center of the image due to masking, which does not favor modeling distortion elements of higher-order. This research was carried over the stone replica of archeological artifact with dimensions 16x13x10 cm. Name of the artifact is "Head of the goddess" and it dates from the 3rd century (Cambi, 1993). The imaged artifact can be seen on figure 5. In order to confirm the research goal, imaging of the front side of the artifact was carried out. During the data acquiring, stereo system was stabilized on the tripod on various distances from the model. Distances varied from full cadre distance to 50cm, which should be optimal distance for this application.

However, due to the resolution and the ground sample distance, detailed analysis is carried only on the full cadre images. The artifact was stabilized on the metal pillar and the metal bars with the markers were set in the immediate vicinity of the artifact. Markers are set in order to acquire better orientation for the full 3D model, also distances between the markers were precisely measured with the standard deviation of 10 μ m, and are used as a scale for the model. Software used for the comparison of models is open source software CloudCompare

CHAPTER 2

LITERATURE SURVEY

1. Accurate and Robust Facial Capture Using a Single RGBD Camera

Author: Yen-Lin Chen, Hsiang-Tao Wu, Fuhao Shi

This paper presents an automatic and robust approach that accurately captures high-quality 3D facial performances using a single RGBD camera. The key of our approach is to combine the power of automatic facial feature detection and image-based 3D non rigid registration techniques for 3D facial reconstruction. In particular, we develop a robust and accurate image-based non rigid registration algorithm that incrementally deforms a 3D template mesh model to best match observed depth image data and important facial features detected from single RGBD images. The whole process is fully automatic and robust because it is based on single frame facial registration frame-work. The ability to accurately capture 3D facial performances has many applications including animation, gaming, human-computer interaction, security, and tele presence. This problem has been partially solved by commercially available marker-based motion capture equipment, but this solution is far too expensive for common use. It is also cumbersome, requiring the user to wear more than 60 carefully positioned retro-reflective markers on the face. This paper presents an alternative to solving this problem: reconstructing the user's 3D facial performances using a single RGBD camera. The main contribution of this paper is a novel 3D facial modeling process that accurately reconstructs 3D facial expression models from single RGBD images. We focus on single frame facial reconstruction because it ensures the process is fully automatic and does not suffer from drifting errors. At the core of our system lies a 3D facial deformation registration process that incrementally deforms a template face model to best match observed depth data. We model 3D facial deformation in a reduced subspace through embedded deformation [16] and extend model-based optical flow formulation to depth image data. This allows us to formulate the 3D nonrigid registration process in the Lucas-Kanade registration framework [1] and use linear system solvers to incrementally deform the template face model to match observed depth images.

2. Face Detection with a 3D Model

Author: Adrian Barbu, Nathan Lay, Gary Gramajo

This paper presents a part-based face detection approach where the spatial relationship between the face parts is represented by a hidden 3D model with six parameters. The computational complexity of the search in the six dimensional pose space is addressed by proposing meaningful 3D pose candidates by image-based regression from detected face key point locations. The 3D pose candidates are evaluated using a parameter sensitive classifier based on difference features relative to the 3D pose. A compatible subset of candidates is then obtained by non-maximal suppression. Experiments on two standard face detection datasets show that the proposed 3D model based approach obtains results comparable to or better than state of the art. Face recognition has been a hot research area for its wide range of applications . In human identification scenarios, facial metrics are more naturally accessible than many other biometrics, such as iris, fingerprint, and palm print . Face recognition is also highly valuable in human computer interaction, access control, video surveillance, and many other applications.

Although 2D face recognition research made significant progresses in recent years, its accuracy is still highly depended on light conditions and human poses . When the light is dim or the face poses are not properly aligned in the camera view, the recognition accuracy will suffer.

3. Real-time 3-D face tracking and modeling from a webcam

Author: Jongmoo Choi, Yann Dumortier, Muhammad Bilal Ahmad, Sang-Il Choi

We first infer a 3-D face model from a single frontal image using automatically extracted 2-D landmarks and deforming a generic 3-D model. Then, for any input image, we extract feature points and track them in 2-D. Given these correspondences, sometimes noisy and incorrect, we robustly estimate the 3-D head pose using PnP and a RANSAC process. As the head moves, we dynamically add new feature points to handle a large range of poses.

This article objective is to implement an eye-blink detection-based face likeness detection algorithm to thwart photo attacks. The algorithm works in real time through a webcam and displays the person's name only if they blinked. In layman's terms, the program runs as follows:

1. Detect faces in each frame generated by the webcam.
2. For each detected face, detect eyes.
3. For each detected eyes, detect if eyes are open or closed.
4. If at some point it was detected that the eyes were open then closed then open, we conclude the person has blinked and the program displays its name (in the case of a facial recognition door opener, we would authorize the person to enter).

4. FACECEPT: REAL TIME 3D FACE TRACKING AND ANALYSIS

Author: Sergey Tulyakov, Radu L. Vieri, Nicu Sebe, Enver Sangineto

We present an open source cross platform technology for 3D face tracking and analysis. It contains a full stack of components for complete face understanding: detection, head pose tracking, facial expression and action unit's recognition. Given a depth sensor, one can combine FaceCept3D modules to fulfill a specific application scenario. Key advantages of the technology include real time processing speed and ability to handle extreme head pose variations. There is one important constraint shared by all these scenarios when solving the above-

mentioned tasks: non-invasiveness, i.e. the solution must not hinder the naturalness of the subject's behavior. Consequently, the vision sensors are typically placed out of the direct sight of the subject. FaceCept3D is motivated by challenges arising from these types of scenarios and is able to successfully address them in a unified, open source and cross-platform solution. Additionally, our system can be deployed in a much broader spectrum of applications (e.g. Those cases for which the face is not fully visible to the sensor), being able to maintain state-of-the-art performance,

5. DEVELOPMENT OF REAL TIME FACE RECOGNITION SYSTEM USING OPENCV

AUTHOR: N. Bayramoglu, G. Zhao, and M. Pietikainen

A real-time, GUI based automatic Face detection and recognition system is developed in this project. It can be used as access control system by registering the staff or students of an organization with their faces, and later it will recognize the people by capturing their images with faces, when they are entering or leaving the premises. The system is implemented on a desktop with a Graphical User Interface, Initially it detects the faces in the images that are grabbed from a web camera. All the tools and operating, used to develop this system like Ubuntu, open Face, Python ..., are open source tools. This real time GUIbasedface recognition system is developed using Open source tool Open face. Open Face is the face recognition tool developed by Carnegie Mellon University, using OpenCV. Open Face, consists in a broader Prospective, three phases: Detection, Feature extraction, and Recognition. The dimensionality of face image is reduced by the Histogram of Oriented Gradients (HOG) and this algorithm is developed to detect frontal views of faces. After detecting the face part of image, extract the 128 face features for the given image by using a Deep Neural Network algorithm and the recognition is done by the Support Vector machine (SVM) classifier. HOG is one of the most popular representation methods for a face image. It not only reduce the dimensionality of the image, but also extracting the facial features of the given images, and retains some of the variations in the image data. So dimensionality of face image reduced by HOG using deep learning algorithm and recognition is done by SVM approach

6. REALTIME DATA ACQUISITION BASED ON OPENCV FOR CLOSE-RANGE PHOTOGRAMMETRY APPLICATIONS

AUTHOR: L. Jurjević a, *, M. Gašparović b

Development of the technology in the area of the cameras, computers and algorithms for 3D the reconstruction of the objects from the images resulted in the increased popularity of the photogrammetry. Algorithms for the 3D model reconstruction are so advanced that almost anyone can make a 3D model of photographed object. The main goal of this paper is to examine the possibility of obtaining 3D data for the purposes of the close-range photogrammetry applications, based on the open source technologies. All steps of obtaining 3D point cloud are covered in this paper. Special attention is given to the camera calibration, for which two-step process of calibration is used. Both, presented algorithm and accuracy of the point cloud are tested by calculating the spatial difference between referent and produced point clouds. During algorithm testing, robustness and swiftness of obtaining 3D data is noted, and certainly usage of this and similar algorithms has a lot of potential in the real-time application. That is the reason why this research can find its application in the architecture, spatial planning, protection of cultural heritage, forensic, mechanical engineering, traffic management, medicine and other sciences.

7. Creating a real model from a video by a single camera

AUTHOR: Kristjan Krips Timo Petmanson

It has been shown by that a 3D model can be constructed from a video made by a stereo camera. However, stereo cameras are yet not widely available and thus in this paper we show how a 3D model could be constructed from a video created by a single camera. As creating a 3D model with a single camera is more difficult than with a stereo camera we assume that the videos used for modelling are taken in strict conditions. We propose two different approaches for solving this problem. The results of this paper show that these methods can be used to create a 3D model. However, the results are visually not comparable to real 3D models, both of these methods have a drawback which is described in the following paragraphs. Thus, the results are a proof on concept which shows that these methods can be used for creating 3D models in certain conditions.

CHAPTER 3

SYSTEM ANALYSIS

EXISTING SYSTEM:

- In existing system it will not only improve classroom control during lectures, it will also possibly detect faces for student attendance purposes.
- In existing system use MATLAB to build and implement this system.

DISADVANTAGES:

- Time consumption high
- Accuracy level is low

PROPOSED SYSTEM:

- Face recognition can be defined as the method of identifying an individual based on biometrics by way of comparing a digital captured image or video with the stored record of the person in question.
- In the early 90s numerous algorithms were developed for face recognition and increase in the need for face detection. Systems were designed to deal with video streaming. Hence the reason why there is still on-going research on this system.
- With the most recent technology of facial recognition and detection techniques, commercial products have emerged on the markets. Despite the commercial success a few issues are still to be explored. Face Recognition in two primary tasks.
- Verification; a one-to-one matching of an unknown face alongside a claim of identity, to ascertain the face of the individual claiming to be the one on the image. Identification which is also a one-to-one matching, given an input image of a face for an individual (unknown), to determine their identity by comparing the image against a database of images with known individuals.

- However, Face Recognition can also be used in numerous applications such as Security, Surveillance, General Identity Verification (electoral registration, national ID cards, passports, driving licenses, student IDs), Criminal Justice systems, Image Database Investigations, Smart Card, Multi-media Environments, Video Indexing and Witness face reconstruction

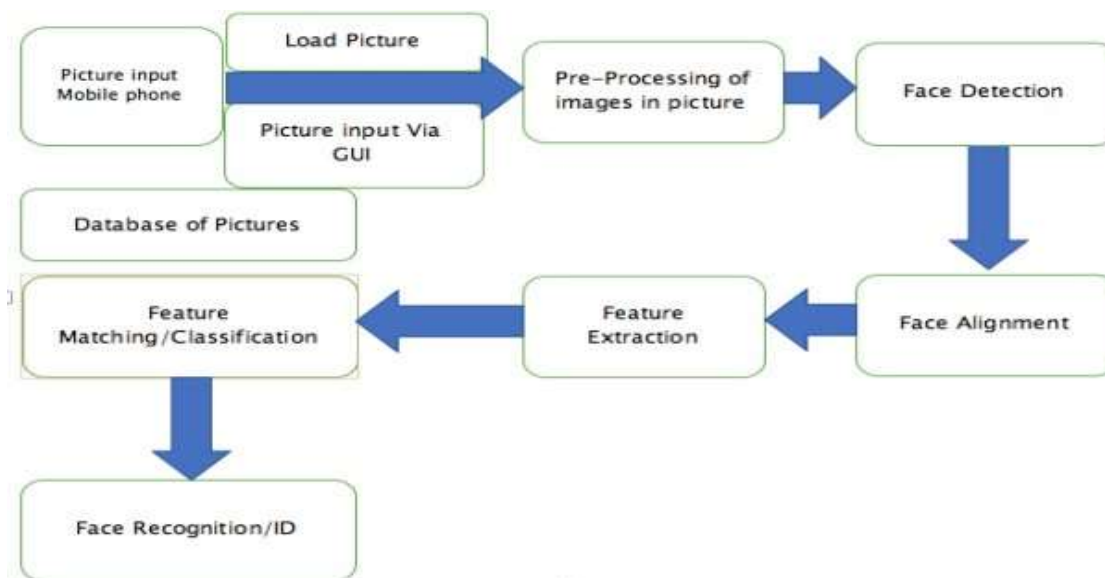
Advantages:

- High accuracy compared to existing system

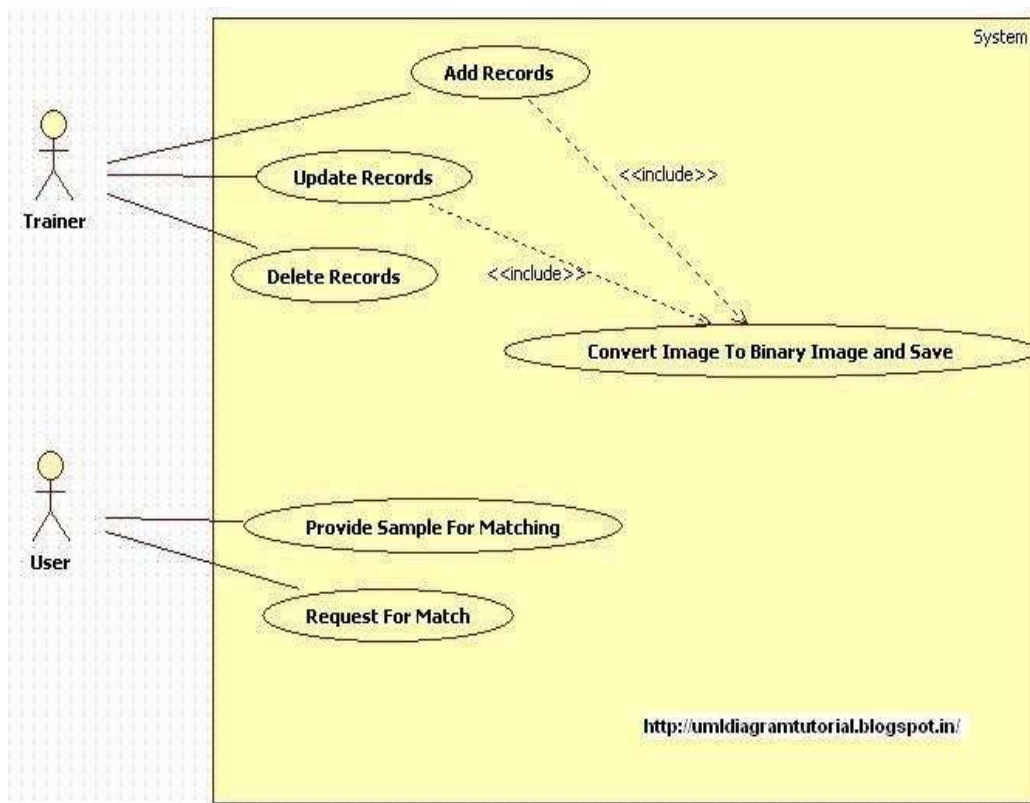
CHAPTER 4

DIAGRAMATIC REPRESENTATION

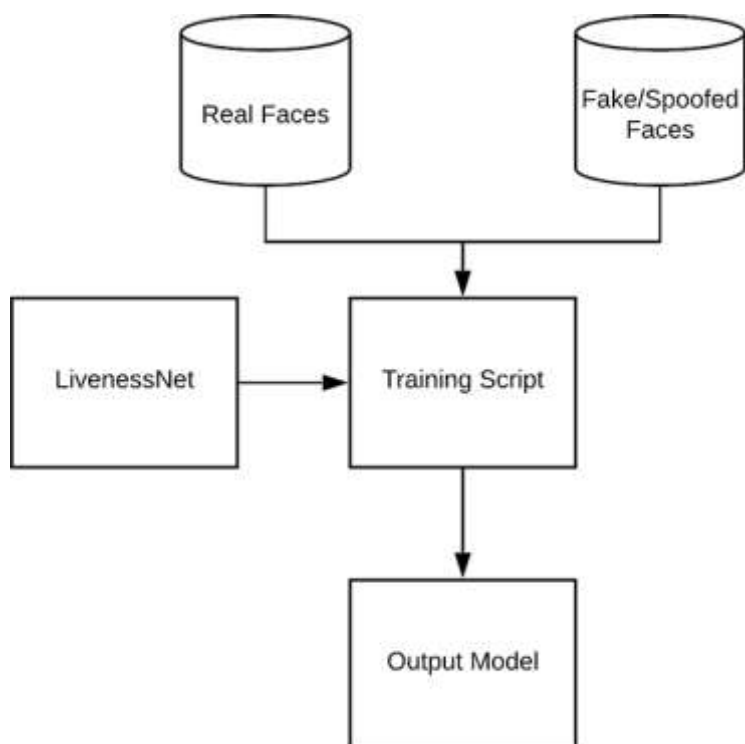
BLOCK DIAGRAM:



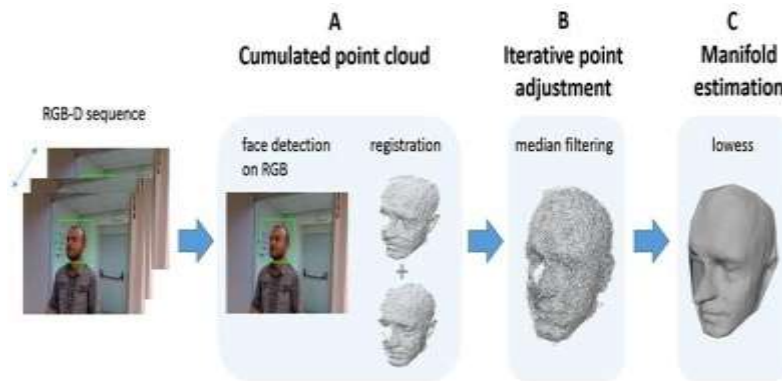
UML DIAGRAM:



DATA FLOW DIAGRAM:



ARCHITECTURE DIAGRAM:



CHAPTER 5

SOFTWARE MODULES

DIFFERENT APPROACHES OF FACE RECOGNITION:

There are two predominant approaches to the face recognition problem: Geometric (feature based) and photometric (view based). As researcher interest in face recognition continued, many different algorithms were developed, three of which have been well studied in face recognition literature.

RECOGNITION ALGORITHMS CAN BE DIVIDED INTO TWO MAIN APPROACHES:

1. Geometric:

Is based on geometrical relationship between facial landmarks, or in other words the spatial configuration of facial features. That means that the main geometrical features of the face such as the eyes, nose and mouth are first located and then faces are classified on the basis of various geometrical distances and angles between features.

2. Photometric stereo:

Used to recover the shape of an object from a number of images taken under different lighting conditions. The shape of the recovered object is defined by a gradient map, which is made up of an array of surface normals (Zhao and Chellappa, 2006)

POPULAR RECOGNITION ALGORITHMS INCLUDE:

1. Principal Component Analysis using Eigenfaces, (PCA)
2. Linear Discriminate Analysis,
3. Elastic Bunch Graph Matching using the Fisherface algorithm,

FACE DETECTION:

Face detection involves separating image windows into two classes; one containing faces (tarning the background (clutter)). It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin colour and facial expression. The problem is further complicated by differing lighting conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise. An ideal face detector would therefore be able to detect the presence of any face under any set of lighting conditions, upon any background. The face detection task can be broken down into two steps.

The first step is a classification task that takes some arbitrary image as input and outputs a binary value of yes or no, indicating whether there are any faces present in the image. The second step is the face localization task that aims to take an image as input and output the location of any face or faces within that image as some bounding box with (x, y, width, height).

THE FACE DETECTION SYSTEM CAN BE DIVIDED INTO THE FOLLOWING STEPS:-

1. Pre-Processing:

To reduce the variability in the faces, the images are processed before they are fed into the network. All positive examples that is the face images are obtained by cropping images with frontal faces to include only the front view. All the cropped images are then corrected for lighting through standard algorithms.

2. Classification:

Neural networks are implemented to classify the images as faces or non faces by training on these examples. We use both our implementation of the neural network and the neural network toolbox for this task. Different network configurations are experimented with to optimize the results.

3. Localization:

The trained neural network is then used to search for faces in an image and if present localize them in a bounding box. Various Feature of Face on which the work has done on:- Position Scale Orientation Illumination

Face detection is a computer technology that determines the location and size of human face in arbitrary (digital) image. The facial features are detected and any other objects like trees, buildings and bodies etc are ignored from the digital image. It can be regarded as a ‘specific’ case of object-class detection, where the task is finding the location and sizes of all objects in an image that belong to a given class. Face detection, can be regarded as a more ‘general’ case of face

localization. In face localization, the task is to find the locations and sizes of a known number of faces (usually one). Basically there are two types of approaches to detect facial part in the given image i.e. feature base and image base approach. Feature base approach tries to extract features of the image and match it against the knowledge of the face features. While image base approach tries to get best match between training and testing images.

FEATURE BASE APPROCH:

Active Shape Model Active shape models focus on complex non-rigid features like actual physical and higher level appearance of features Means that Active Shape Models (ASMs) are aimed at automatically locating landmark points that define the shape of any statistically modelled object in an image. When of facial features such as the eyes, lips, nose, mouth and eyebrows. The training stage of an ASM involves the building of a statistical

a) facial model from a training set containing images with manually annotated landmarks. ASMs is classified into three groups i.e. snakes, PDM, Deformable templates

b) 1.1)Snakes:The first type uses a generic active contour called snakes, first introduced by Kass et al. in 1987 Snakes are used to identify head boundaries [8,9,10,11,12]. In order to achieve the task, a snake is first initialized at the proximity around a head boundary. It then locks onto nearby edges and subsequently assume the shape of the head. The evolution of a snake is achieved by minimizing an energy function, E_{snake} (analogy with physical systems), denoted as $E_{snake} = E_{internal} + E_{External}$ Where $E_{internal}$ and $E_{External}$ are internal and external energy functions. Internal energy is the part that depends on the intrinsic properties of the snake and defines its natural evolution

The typical natural evolution in snakes is shrinking or expanding. The external energy counteracts the internal energy and enables the contours to deviate from the natural evolution and eventually assume the shape of nearby features—the head boundary at a state of equilibria. Two main consideration for

forming snakes i.e. selection of energy terms and energy minimization. Elastic energy is used commonly as internal energy. Internal energy is vary with the distance between control points on the snake, through which we get contour an elastic-band characteristic that causes it to shrink or expand. On other side external energy relay on image features. Energy minimization process is done by optimization techniques such as the steepest gradient descent. Which needs highest computations. Huang and Chen and Lam and Yan both employ fast iteration methods by greedy algorithms. Snakes have some demerits like contour often becomes trapped onto false image features and another one is that snakes are not suitable in extracting non convex features.

DEFORMABLE TEMPLATES:

Deformable templates were then introduced by Yuille et al. to take into account the a priori of facial features and to better the performance of snakes. Locating a facial feature boundary is not an easy task because the local evidence of facial edges is difficult to organize into a sensible global entity using generic contours. The low brightness contrast around some of these features also makes the edge detection process. Yuille et al. took the concept of snakes a step further by incorporating global information of the eye to improve the reliability of the extraction process Deformable templates approaches are developed to solve this problem. Deformation is based on local valley, edge, peak, and brightness .Other than face boundary, salient feature (eyes, nose, mouth and eyebrows) extraction is a great challenge of face recognition. $E = E_v + E_e + E_p + E_i + E_{\text{internal}}$; where E_v , E_e , E_p , E_i , E_{internal} are external energy due to valley, edges, peak and image brightness and internal energy

PDM (Point Distribution Model):

Independently of computerized image analysis, and before ASMs were developed, researchers developed statistical models of shape . The idea is that once you represent shapes as vectors, you can apply standard statistical methods to them just like any other multivariate object. These models learn allowable constellations of shape points from training example sand use principal components to build what is called a Point Distribution Model. These

have been used in diverse ways, for example for categorizing Iron Age broaches. Ideal Point Distribution Models can only deform in ways that are characteristic of the object. Cootes and his colleagues were seeking models which do exactly that so if a beard, say, covers the chin, the shape model can "override the image" to approximate the position of the chin under the beard. It was therefore natural (but perhaps only in retrospect) to adopt Point Distribution Models. This synthesis of ideas from image processing and statistical shape modelling led to the Active Shape Model. The first parametric statistical shape model for image analysis based on principal components of inter-landmark distances was presented by Cootes and Taylor in. On this approach, Cootes, Taylor, and their colleagues, then released a series of papers that cumulated in what we call the classical Active Shape Model.

LOW LEVEL ANALYSIS:

Based on low level visual features like colour, intensity, edges, motion etc. Skin Colour BaseColor is vital feature of human faces. Using skin-color as a feature for tracking a face has several advantages. Colour processing is much faster than processing other facial features. Under certain lighting conditions, colour is orientation invariant. This property makes motion estimation much easier because only a translation model is needed for motion estimation. Tracking human faces using colour as a feature has several problems like the colour representation of a face obtained by a camera is influenced by many factors (ambient light, object movement, etc

Majorly three different face detection algorithms are available based on RGB, YCbCr, and HIS colour space models. In the implementation of the algorithms there are three main steps viz.

- (1) Classify the skin region in the colour space,**
- (2) Apply threshold to mask the skin region and**
- (3) Draw bounding box to extract the face image.**

Crowley and Coutaz suggested simplest skin colour algorithms for detecting skin pixels. The perceived human colour varies as a function of the relative direction to the illumination. The pixels for skin region can be detected using a normalized colour histogram, and can be normalized for changes in intensity on dividing by luminance. Converted an $[R, G, B]$ vector is converted into an $[r, g]$ vector of normalized colour which provides a fast means of skin detection. This algorithm fails when there are some more skin region like legs, arms, etc. Cahi and Ngan suggested skin colour classification algorithm with YCbCr colour space. Research found that pixels belonging to skin region having similar Cb and Cr values. So that the thresholds be chosen as $[Cr1, Cr2]$ and $[Cb1, Cb2]$, a pixel is classified to have skin tone if the values $[Cr, Cb]$ fall within the thresholds. The skin colour distribution gives the face portion in the color image algorithm is also having the constraint that the image should be having only face as the skin region. Kjeldson and Kinder defined a color predicate in HSV color space to separate skin regions from background. Skin color classification in HSI colour space is the same as YCbCr colour space but here the responsible values are hue (H) and saturation (S). Similar to above the threshold be chosen as $[H1, S1]$ and $[H2, S2]$, and a pixel is classified to have skin tone if the values $[H, S]$ fall within the threshold and this distribution gives the localized face image. Similar to above two algorithm this algorithm is also having the same constraint

MOTION BASE:

When use of video sequence is available, motion information can be used to locate moving objects. Moving silhouettes like face and body parts can be extracted by simply thresholding accumulated frame differences. Besides face regions, facial features can be located by frame differences.

GRAY SCALE BASE:

Gray information within a face can also be treated as important features. Facial features such as eyebrows, pupils, and lips appear generally darker than their surrounding facial regions. Various recent feature extraction algorithms search for local gray minima within segmented facial regions. In these algorithms, the input images are first enhanced by contrast-stretching and gray-scale morphological routines to improve the quality of local dark patches and thereby make detection easier. The extraction of dark patches is achieved by low-level gray-scale thresholding. Based method and consist three levels. Yang and huang presented new approach i.e. faces gray scale behaviour in pyramid (mosaic) images. This system utilizes hierarchical Face location consist three levels. Higher two level based on mosaic images at different resolution. In the lower level, edge detection method is proposed. Moreover this algorithms gives fine response in complex background where size of the face is unknown

EDGE BASE:

Face detection based on edges was introduced by Sakai et al. This work was based on analysing line drawings of the faces from photographs, aiming to locate facial features. Then later Craw et al. proposed a hierarchical framework based on Sakai et al.'s work to trace a human head outline. Then after remarkable works were carried out by many researchers in this specific area. Method was very simple and fast. They proposed frame work which consist three stepwise. Initially the images are enhanced by applying median filter for noise removal and histogram equalization for contrast adjustment. In the second step the edge images constructed from the enhanced image by applying sober operator. Then a novel edge tracking algorithm is applied to extract the sub windows from the enhanced image based on edges. Further they used Back propagation Neural Network (BPN) algorithm to classify the sub-window as either face or non-face.

FEATURE ANALYSIS

These algorithms aim to find structural features that exist even when the pose, viewpoint, or lighting conditions vary, and then use these to locate faces. These methods are designed mainly for face localization

FEATURE SEARCHING

VIOLA JONES METHOD:

Paul Viola and Michael Jones presented an approach for object detection which minimizes computation time while achieving high detection accuracy. Paul Viola and Michael Jones proposed a fast and robust method for face detection which is 15 times quicker than any technique at the time of release with 95% accuracy at around 17 fps. The technique relies on the use of simple Haar-like features that are evaluated quickly through the use of a new image representation. Based on the concept of an —Integral Image|| it generates a large set of features and uses the boosting algorithm AdaBoost to reduce the overcomplete set and the introduction of a degenerative tree of the boosted classifiers provides for robust and fast interferences. The detector is applied in a scanning fashion and used on gray-scale images, the scanned window that is applied can also be scaled, as well as the features evaluated.

GABOR FEATURE METHOD:

Sharif et al proposed an Elastic Bunch Graph Map (EBGM) algorithm that successfully implements face detection using Gabor filters. The proposed system applies 40 different Gabor filters on an image. As a result of which 40 images with different angles and orientation are received. Next, maximum intensity points in each filtered image are calculated and mark them as fiducially points. The system reduces these points in accordance to distance between them. The next step is calculating the distances between the reduced points

Using distance formula. At last, the distances are compared with database. If match occurs, it means that the faces in the image are detected

CONSTELLATION METHOD

All methods discussed so far are able to track faces but still some issue like locating faces of various poses in complex background is truly difficult. To reduce this difficulty investigator form a group of facial features in face-like constellations using more robust modelling approaches such as statistical analysis. Various types of face constellations have been proposed by Burl et al. . They establish use of statistical shape theory on the features detected from a multiscale Gaussian derivative filter. Huang et al. also apply a Gaussian filter for pre-processing in a framework based on image feature analysis. Image Base Approach

Neural Network

Neural networks gaining much more attention in many pattern recognition problems, such as OCR, object recognition, and autonomous robot driving. Since face detection can be treated as a two class pattern recognition problem, various neural network algorithms have been proposed. The advantage of using neural networks for face detection is the feasibility of training a system to capture the complex class conditional density of face patterns. However, one demerit is that the network architecture has to be extensively tuned (number of layers, number of nodes, learning rates, etc.) to get exceptional performance. In early days most hierarchical neural network was proposed by Agui et al. [43]. The first stage having two parallel sub networks in which the inputs are filtered intensity values from an original image. The inputs to the second stage network consist of the outputs from the sub networks and extracted feature values. An output at the second stage shows the presence of a face in the input region. Propp and Samal developed one of the earliest neural networks for face detection . Their network consists of four layers with 1,024 input units, 256 units in the first hidden layer, eight units in the second hidden layer, and two output units. Feraud and Bernier presented a detection method using auto associative neural networks . The idea is based on which shows an auto associative network with five layers is able to perform a nonlinear principal component analysis. One auto associative network is used to detect

frontal-view faces and another one is used to detect faces turned up to 60 degrees to the left and right of the frontal view. After that Lin et al. presented a face detection system using probabilistic decision-based neural network (PDBNN) . The architecture of PDBNN is similar to a radial basis function (RBF) network with modified learning rules and probabilistic interpretation

LINEAR SUB SPACE METHOD

Eigen faces Method:

An early example of employing eigen vectors in face recognition was done by Kohonen in which a simple neural network is demonstrated to perform face recognition for aligned and normalized face images. Kirby and Sirovich suggested that images of faces can be linearly encoded using a modest number of basis images. The idea is arguably proposed first by Pearson in 1901 and then by HOTELLING in 1933 . Given a collection of n by m pixel training.

Images represented as a vector of size $m \times n$, basis vectors spanning an optimal subspace are determined such that the mean square error between the projection of the training images onto this subspace and the original images is minimized. They call the set of optimal basis vectors Eigen pictures since these are simply the eigen vectors of the covariance matrix computed from the vectorized face images in the training set. Experiments with a set of 100 images show that a face image of 91×50 pixels can be effectively encoded using only 50 Eigen pictures.

Support Vector Machine (SVM):

SVMs were first introduced by Osuna et al. for face detection. SVMs work as a new paradigm to train polynomial function, neural networks, or radial basis function (RBF) classifiers. SVMs work on induction principle, called structural risk minimization, which targets to minimize an upper bound on the expected generalization error. An SVM classifier is a linear classifier where the separating hyper plane is chosen to minimize the expected classification error of the

unseen test patterns. In Osuna et al. developed an efficient method to train an SVM for large scale problems, and applied it to face detection. Based on two test sets of 10,000,000 test patterns of 19 X 19 pixels, their system has slightly lower error rates and runs approximately 30 times faster than the system by Sung and Poggio . SVMs have also been used to detect faces and pedestrians in the wavelet domain.

CHAPTER 6

DIGITAL IMAGE PROCESSING

INTEREST IN DIGITAL IMAGE PROCESSING METHODS STEMS FROM TWO PRINCIPAL APPLICATION AREAS:

1. Improvement of pictorial information for human interpretation
2. Processing of scene data for autonomous machine perception

In this second application area, interest focuses on procedures for extracting image information in a form suitable for computer processing.

Examples include automatic character recognition, industrial machine vision for product assembly and inspection, military reconnaissance, automatic processing of fingerprints etc.

IMAGE:

An image refers to a 2D light intensity function $f(x, y)$, where (x, y) denotes spatial coordinates and the value of f at any point (x, y) is proportional to the brightness or gray levels of the image at that point. A digital image is an image $f(x, y)$ that has been discretized both in spatial coordinates and brightness. The elements of such a digital array are called image elements or pixels.

A SIMPLE IMAGE MODEL:

To be suitable for computer processing, an image $f(x, y)$ must be digitalized both spatially and in amplitude. Digitization of the spatial coordinates (x, y) is called image sampling. Amplitude digitization is called gray-level quantization.

The storage and processing requirements increase rapidly with the spatial resolution and the number of gray levels.

Example: A 256 gray-level image of size 256x256 occupies 64k bytes of memory

Low level processing means performing basic operations on images such as reading an image, resize, image rotate, RGB to grey level conversion, histogram equalization etc..., The output image obtained after low level processing is raw image. Medium level processing means extracting regions of interest from output of low level processed image. Medium level processing deals with identification of boundaries i.e. edges. This process is called segmentation. High level processing deals with adding of artificial intelligence to medium level processed signal.

FUNDAMENTAL STEPS IN IMAGE PROCESSING

Fundamental steps in image processing are

1. **Image acquisition:** to acquire a digital image
2. **Image pre-processing:** to improve the image in ways that increases the chances for success of the other processes.
3. **Image segmentation:** to partitions an input image into its constituent parts of objects.
4. **Image segmentation:** to convert the input data to a form suitable for computer processing.
5. **Image description:** to extract the features that result in some quantitative information of interest of features that are basic for differentiating one class of objects from another.
6. **Image recognition:** to assign a label to an object based on the information provided by its description.

ELEMENTS OF DIGITAL IMAGE PROCESSING SYSTEMS

The basic operations performed in a digital image processing system include

1. Acquisition

2. Storage
3. Processing
4. Communication
5. Display

A SIMPLE IMAGE FORMATION MODEL

Image are denoted by two-dimensional function $f(x, y)$. $f(x, y)$ may be characterized by 2 components:

1. The amount of source illumination $i(x, y)$ incident on the scene
2. The amount of illumination reflected $r(x, y)$ by the objects of the scene

$$4. f(x, y) = i(x, y)r(x, y), \text{ where } 0 < i(x, y) < \infty \text{ and } 0 < r(x, y) < 1$$

TYPICAL VALUES OF REFLECTANCE $R(X, Y)$:

- • Sun on a clear day: $\sim 90,000 \text{ lm/m}^2$, down to $10,000 \text{ lm/m}^2$ on a cloudy day
- • Full moon on a clear evening : $\sim 0.1 \text{ lm/m}^2$
- • Typical illumination level in a commercial office. $\sim 1000 \text{ lm/m}^2$

Format name	Full name	Description	Recognized extensions
TIFF	Tagged Image File Format	A flexible file format supporting a variety image compression standards including JPEG	.tif, .tiff
JPEG	Joint Photographic Experts Group	A standard for compression of images of photographic quality	.jpg, .jpeg
GIF	Graphics Interchange Format	Frequently used to make small animations on the internet	.gif
BMP	Windows Bitmap	Format used mainly for simple uncompressed images	.bmp
PNG	Portable Network Graphics	Compresses full color images with transparency(up to 48bits/p	.png

TABLE 3.3.1: Popular Image File Formats

FACE DETECTION

The problem of face recognition is all about face detection. This is a fact that seems quite bizarre to new researchers in this area. However, before face recognition is possible, one must be able to reliably find a face and its landmarks. This is essentially a segmentation problem and in practical systems, most of the effort goes into solving this task. In fact the actual recognition based on features extracted from these facial landmarks is only a minor last step.

There are two types of face detection problems:

- 1) Face detection in images and
- 2) Real-time face detection

FACE DETECTION IN IMAGES

Most face detection systems attempt to extract a fraction of the whole face, thereby eliminating most of the background and other areas of an individual's head such as hair that are not necessary for the face recognition task. With static images, this is often done by running a across the image. The face detection system then judges if a face is present inside the window (Brunelli and Poggio, 1993). Unfortunately, with static images there is a very large search space of possible locations of a face in an image

Most face detection systems use an example based learning approach to decide whether or not a face is present in the *window* at that given instant (Sung and Poggio,1994 and Sung,1995). A neural network or some other classifier is trained using supervised learning with 'face' and 'non-face' examples, thereby enabling it to classify an image (*window* in face detection system) as a 'face' or 'non-face'.. Unfortunately, while it is relatively easy to find face examples, how would one find a representative sample of images which represent non-faces (Rowley et al., 1996)? Therefore, face detection systems using example based learning need thousands of 'face' and 'non-face' images for effective training. Rowley, Baluja, and Kanade (Rowley et al.,1996) used 1025 face images and 8000 non-face images (generated from 146,212,178 sub-images) for their training set!

There is another technique for determining whether there is a face inside the face detection system's *window* - using Template Matching. The difference between a fixed target pattern (face) and the window is computed and thresholded. If the window contains a pattern which is close to the target pattern(face) then the window is judged as containing a face. An implementation of template matching called Correlation Templates uses a whole bank of fixed sized templates to detect facial features in an image (Bichsel, 1991 & Brunelli and Poggio, 1993). By using several templates of different (fixed) sizes, faces of different scales (sizes) are detected. The other implementation of template matching is using a deformable template (Yuille,

1992). Instead of using several fixed size templates, we use a deformable template (which is non-rigid) and there by change the size of the template hoping to detect a face in an image.

A face detection scheme that is related to template matching is image invariants. Here the fact that the local ordinal structure of brightness distribution of a face remains largely unchanged under different illumination conditions (Sinha, 1994) is used to construct a spatial template of the face which closely corresponds to facial features. In other words, the average grey-scale intensities in human faces are used as a basis for face detection. For example, almost always an individuals eye region is darker than his forehead or nose. Therefore an image will match the template if it satisfies the 'darker than' and 'brighter than' relationships (Sung and Poggio, 1994).

REAL-TIME FACE DETECTION

Real-time face detection involves detection of a face from a series of frames from a video-capturing device. While the hardware requirements for such a system are far more stringent, from a computer vision stand point, real-time face detection is actually a far simpler process than detecting a face in a static image. This is because unlike most of our surrounding environment, people are continually moving. We walk around, blink, fidget, wave our hands about, etc. Since in real-time face detection, the system is presented with a series of frames in which to detect a face, by using spatio-temporal filtering (finding the difference between subsequent frames), the area of the frame that has changed can be identified and the individual detected (Wang and Adelson, 1994 and Adelson and Bergen 1986). Further more as seen in Figure exact face locations can be easily identified by using a few simple rules, such as, 1) the head is the small blob above a larger blob -the body 2) head motion must be reasonably slow and contiguous -heads won't jump around erratically (Turk and Pentland 1991a, 1991b).

Real-time face detection has therefore become a relatively simple problem and is possible even in unstructured and uncontrolled environments using these very simple image processing techniques and reasoning rules.

FACE DETECTION PROCESS

It is process of identifying different parts of human faces like eyes, nose, mouth, etc... this process can be achieved by using MATLAB code. In this project the author will attempt to detect faces in still images by using image invariants. To do this it would be useful to study the grey-scale intensity distribution of an average human face. The following 'average human face' was constructed from a sample of 30 frontal view human faces, of which 12 were from females and 18 from males. A suitably scaled colormap has been used to highlight grey-scale intensity differences.

The grey-scale differences, which are invariant across all the sample faces are strikingly apparent. The eye-eyebrow area seem to always contain dark intensity (low) gray-levels while nose forehead and cheeks contain bright intensity (high) grey-levels. After a great deal of experimentation, the researcher found that the following areas of the human face were suitable for a face detection system based on image invariants and a deformable template.

The above facial area performs well as a basis for a face template, probably because of the clear divisions of the bright intensity invariant area by the dark intensity invariant regions. Once this pixel area is located by the face detection system, any particular area required can be segmented based on the proportions of the average human face. After studying the above images it was subjectively decided by the author to use the following as a basis for dark intensity sensitive and bright intensity sensitive templates. Once these are located in a subject's face, a pixel area 33.3% (of the width of the square window) below this

Note the slight differences which were made to the bright intensity invariant sensitive template which were needed because of the pre-processing done by the system to overcome irregular lighting (chapter six). Now that a suitable dark and bright intensity invariant templates have been decided on, it is necessary to find a way of using these to make 2 A-units for a perceptron, i.e. a computational model is needed to assign neurons to the distributions displayed

FACE RECOGNITION

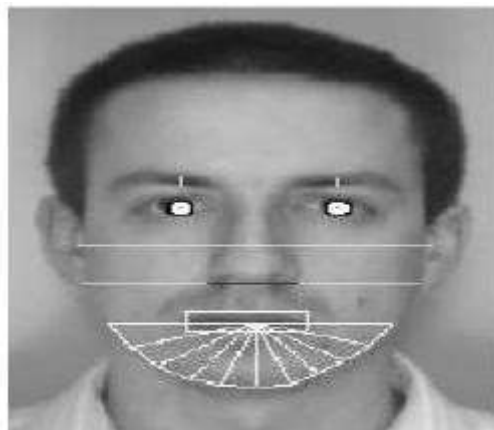
Over the last few decades many techniques have been proposed for face recognition. Many of the techniques proposed during the early stages of computer vision cannot be considered successful, but almost all of the recent

approaches to the face recognition problem have been creditable. According to the research by Brunelli and Poggio (1993) all approaches to human face recognition can be divided into two strategies:

- (1) Geometrical features and
- (2) Template matching.

FACE RECOGNITION USING GEOMETRICAL FEATURES

This technique involves computation of a set of geometrical features such as nose width and length, mouth position and chin shape, etc. from the picture of the face we want to recognize. This set of features is then matched with the features of known individuals. A suitable metric such as Euclidean distance (finding the closest vector) can be used to find the closest match. Most pioneering work in face recognition was done using geometric features (Kanade, 1973), although Craw et al. (1987) did relatively recent work in this area.



The advantage of using geometrical features as a basis for face recognition is that recognition is possible even at very low resolutions and with noisy images (images with many disorderly pixel intensities). Although the face cannot be viewed in detail its overall geometrical configuration can be extracted for face recognition. The technique's main disadvantage is that automated extraction of the facial geometrical features is very hard. Automated geometrical feature extraction based recognition is also very sensitive to the scaling and rotation of a face in the image plane (Brunelli and Poggio, 1993). This is apparent when we examine Kanade's(1973) results where he reported a recognition rate of between 45-75 % with a database of only 20 people. However if these features

are extracted manually as in Goldstein et al. (1971), and Kaya and Kobayashi (1972) satisfactory results may be obtained.

FACE RECOGNITION USING TEMPLATE MATCHING

This is similar the template matching technique used in face detection, except here we are not trying to classify an image as a 'face' or 'non-face' but are trying to recognize a face.



Whole face, eyes, nose and mouth regions which could be used in a template matching strategy. The basis of the template matching strategy is to extract whole facial regions (matrix of pixels) and compare these with the stored images of known individuals. Once again Euclidean distance can be used to find the closest match. The simple technique of comparing grey-scale intensity values for face recognition was used by Baron (1981). However there are far more sophisticated methods of template matching for face recognition. These involve extensive pre-processing and transformation of the extracted grey-level intensity values. For example, Turk and Pentland (1991a) used Principal Component Analysis, sometimes known as the eigenfaces approach, to pre-process the gray-levels and Wiskott et al. (1997) used Elastic Graphs encoded using Gabor filters to pre-process the extracted regions. An investigation of geometrical features versus template matching for face recognition by Brunelli and Poggio (1993) came to the conclusion that although a feature based strategy may offer higher recognition speed and smaller memory requirements, template based techniques offer superior recognition accuracy.

PROBLEM SCOP AND SYSTEM SPECIFICATION

The following problem scope for this project was arrived at after reviewing the literature on face detection and face recognition, and determining possible real-world situations where such systems would be of use. The following system(s) requirements were identified

- 1 A system to detect frontal view faces in static images
- 2 A system to recognize a given frontal view face
- 3 Only expressionless, frontal view faces will be presented to the face detection recognition
- 4 All implemented systems must display a high degree of lighting invariance.
- 5 All systems must possess near real-time performance.
- 6 Both fully automated and manual face detection must be supported
- 7 Frontal view face recognition will be realised using only a single known image
- 8 Automated face detection and recognition systems should be combined into a fully automated face detection and recognition system. The face recognition sub-system must display a slight degree of invariance to scaling and rotation errors in the segmented image extracted by the face detection sub-system.
- 9 The frontal view face recognition system should be extended to a pose invariant face recognition system.

Unfortunately although we may specify constricting conditions to our problem domain, it may not be possible to strictly adhere to these conditions when implementing a system in the real-world.

FACE RECOGNITION DIFFICULTIES

1. Identify similar faces (inter-class similarity)
2. Accommodate intra-class variability due to
 - 2.1 head pose
 - 2.2 illumination conditions

- 2.3 expressions
- 2.4 facial accessories
- 2.5 aging effects
- 3. Cartoon faces

Face recognition and detection system is a pattern recognition approach for personal identification purposes in addition to other biometric approaches such as fingerprint recognition, signature, retina and so forth. The variability in the faces, the images are processed before they are fed into the network. All positive examples that is the face images are obtained by cropping images with frontal faces to include only the front view. All the cropped images are then corrected for lighting through standard algorithms.

INTER – CLASS VARIABILITY

Faces with intra-subject variations in pose, illumination, expression, accessories, color, occlusions, and brightness

PRINCIPAL COMPONENT ANALYSIS (PCA)

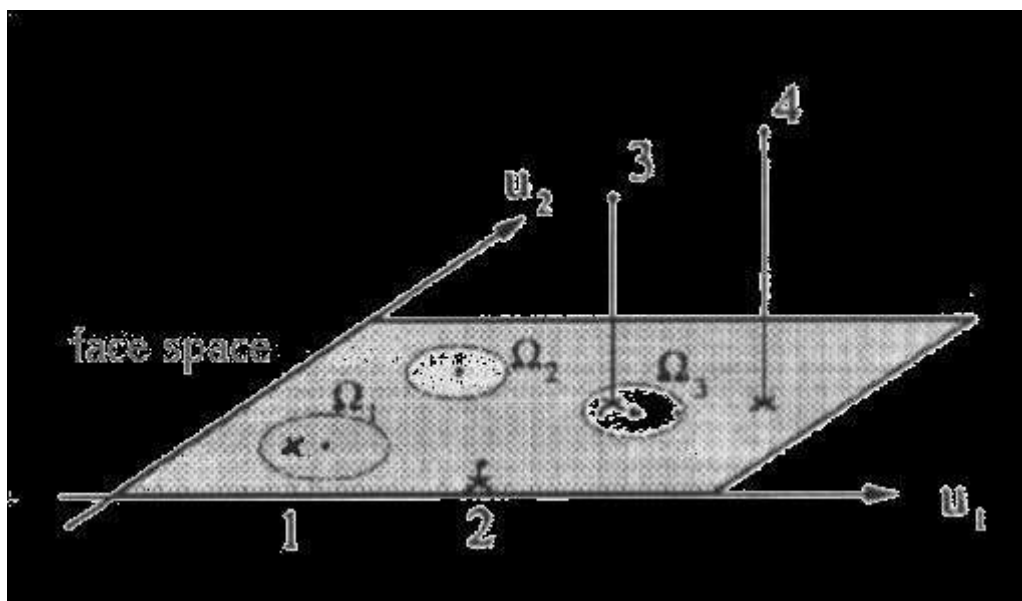
Principal Component Analysis (or Karhunen-Loeve expansion) is a suitable strategy for face recognition because it identifies variability between human faces, which may not be immediately obvious. Principal Component Analysis (hereafter PCA) does not attempt to categorise faces using familiar geometrical differences, such as nose length or eyebrow width. Instead, a set of human faces is analysed using PCA to determine which 'variables' account for the variance of faces. In face recognition, these variables are called eigen faces because when plotted they display an eerie resemblance to human faces. Although PCA is used extensively in statistical analysis, the pattern recognition community started to use PCA for classification only relatively recently. As described by Johnson and Wichern (1992), 'principal component analysis is concerned with explaining the variance- covariance structure through a few linear combinations of the original variables.' Perhaps PCA's greatest strengths are in its ability for data reduction and interpretation. For example a 100x100 pixel area containing a face can be very accurately represented by just 40 eigen values. Each eigen value describes

the magnitude of each eigen face in each image. Furthermore, all interpretation (i.e. recognition) operations can now be done using just the 40 eigen values to represent a face instead of the manipulating the 10000 values contained in a 100x100 image. Not only is this computationally less demanding but the fact that the recognition information of several thousand.

UNDERSTANDING EIGENFACES

Any grey scale face image $I(x,y)$ consisting of a $N \times N$ array of intensity values may also be consider as a vector of N^2 . For example, a typical 100x100 image used in this thesis will have to be transformed into a 10000 dimension vector!

This vector can also be regarded as a point in 10000 dimension space. Therefore, all the images of subjects' whose faces are to be recognized can be regarded as points in 10000 dimension space. Face recognition using these images is doomed to failure because all human face images are quite similar to one another so all associated vectors are very close to each other in the 10000-dimension space.



The transformation of a face from image space (I) to face space (f) involves just a simple matrix multiplication. If the average face image is A and U contains the (previously calculated) eigenfaces This is done to all the face images in the face database (database with known faces) and to the image (face of the subject)

which must be recognized. The possible results when projecting a face into face space

THERE ARE FOUR POSSIBILITIES:

1. Projected image is a face and is transformed near a face in the face database
2. Projected image is a face and is not transformed near a face in the face database
3. Projected image is not a face and is transformed near a face in the face database
4. Projected image is not a face and is not transformed near a face in the face database

While it is possible to find the closest known face to the transformed image face by calculating the Euclidean distance to the other vectors, how does one know whether the image that is being transformed actually contains a face? Since PCA is a many-to-one transform, several vectors in the image space (images) will map to a point in face space (the problem is that even non-face images may transform near a known face image's faces space vector). Turk and Pentland (1991a), described a simple way of checking whether an image is actually of a face. This is by transforming an image into face space and then transforming it back (reconstructing) into image space. Using the previous notation

With these calculations it is possible to verify that an image is of a face and recognise that face. O'Toole et al. (1993) did some interesting work on the importance of eigen faces with large and small eigenvalues. They showed that the eigen vectors with larger eigenvalues convey information relative to the basic shape and structure of the faces. This kind of information is most useful in categorising faces according to sex, race etc. Eigen vectors with smaller eigenvalues tend to capture information that is specific to single or small subsets of learned faces and are useful for distinguishing a particular face from any other face. Turk and Pentland (1991a) showed that about 40 eigen faces were

sufficient for a very good description of human faces since the reconstructed image have only about 2% RMS. pixel-by-pixel errors.

IMPROVING FACE DETECTION USING RECONSTRUCTING

Reconstruction cannot be used as a means of face detection in images in near real-time since it would involve resizing the face detection *window* area and large matrix multiplication, both of which are computationally expensive. However, reconstruction can be used to verify whether potential face locations identified by the deformable template algorithm actually contain a face. If the reconstructed image differs greatly from the face detection *window* then the *window* probably does not contain a face. Instead of just identifying a single potential face location, the face detection algorithm can be modified to output many high 'faceness' locations which can be verified using reconstruction. This is especially useful because occasionally the best 'faceness' location found by the deformable template algorithm may not contain the ideal frontal view face pixel area. potential face locations that have been identified by the face detection system (the best face locations it found on its search) are checked whether they contain a face. If the threshold level (maximum difference between reconstruction and original for the original to be a face) is set correctly this will be an efficient way to detect a face. The deformable template algorithm is fast and can reduce the search space of potential face locations to a handful of positions. These are then checked using reconstruction. The number of locations found by the face detection system can be changed by getting it to output, not just the best face locations it has found so far but any location, which has a 'faceness' value, which for example is, at least 0.9 times the best heuristic value that has been found so far. Then there will be many more potential face locations to be checked using reconstruction. This and similar speed versus accuracy trade-off decisions have to be made keeping in mind the platform on which the system is implemented.

Similarly, instead of using reconstruction to check the face detection system's output, the output's correlation with the average face can be checked. The segmented areas with a high correlation probably contains a face. Once again a threshold value will have to be established to classify faces from non-faces. Similar to reconstruction, resizing the segmented area and calculating its

correlation with the average face is far too expensive to be used alone for face detection but is suitable for verifying the output of the face detection system.

POSE INVARIANT FACE RECOGNITION

Extending the frontal view face recognition system to a pose-invariant recognition system is quite simple if one of the proposed specifications of the face recognition system is relaxed. Successful pose-invariant recognition will be possible if many images of a known individual are in the face database. Nine images from each known individual can be taken as shown below. Then if an image of the same individual is submitted within a 30° angle from the frontal view he or she can be identified.

Nine images in face database from a single known individual Pose invariant face recognition highlights the generalisation ability of PCA. For example, when an individual's frontal view and 30° left view known, even the individual's 15° left view can be recognised

PYTHON (PROGRAMMING LANGUAGE)

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather

than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation

HISTORY

Python was conceived in the late 1980s, and its implementation began in December 1989 [28] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the operating system Amoeba. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, benevolent dictator for life (BDFL). About the origin of Python, Van Rossum wrote in 1996: Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus). Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed. Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008 after a long period of testing. Many of its major features have been backported to the

backwards-compatible Python 2.6.x and 2.7.x version series. The End Of Life date (EOL, sunset date) for Python 2.7 was initially set at 2015, then postponed to 2020 out of concern that a large body of existing code cannot easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler, which The Register speculated was in response to Python 2.7's planned end-of-life but Google cited performance under concurrent workloads as their only motivation.

Features and philosophy Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other 6/13/2017 Python (programming language) paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing and a mix of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution. The design of Python offers some support for functional programming in the Lisp tradition. The language has `map()`, `reduce()` and `filter()` functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML. The core philosophy of the language is summarized by the document The Zen of Python (PEP 20), which includes aphorisms such as: Beautiful is better than ugly Explicit is better than implicit Simple is better than complex Complex is better than complicated Readability counts Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible. Python can also be embedded in existing applications that need a programmable interface. This design of a small core language with a large standard library and an easily extensible interpreter was intended by Van Rossum from the start because of his frustrations with ABC, which espoused the opposite mindset. While offering choice in coding methodology, the Python philosophy rejects exuberant syntax, such as in Perl, in favor of a sparser, less-cluttered grammar. As Alex Martelli put it: "To describe something as clever is not considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language

design in favor of "there should be one—and preferably only one—obvious way to do it". Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython that would offer a marginal increase in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or try using PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter. An important goal of Python's developers is making it fun to use. This is reflected in the origin of the name, which comes from Monty Python, and in an occasionally playful approach to tutorials and reference materials, such as using examples that refer to spam and eggs instead of the standard foo and bar. A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is *pythonic* is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*. Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoners. The third slice parameter, called *step* or *stride*, allows elements to be skipped and reversed. Slice indexes may be omitted, for example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy. In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example: List comprehensions vs. for-loops Conditional expressions vs. if blocks The `eval()` vs. `exec()` built-in functions (in Python 2, `exec` is a statement); the former is for expressions, the latter is for statements. 6/13/2017 Python (programming language) - Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as `a = 1` cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator `=` for an equality operator `==` in conditions: `if (c = 1) { ... }` is syntactically valid (but probably unintended) C code but `if c = 1: ...` causes a syntax error in Python.

Methods Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby). Typing Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them. Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, `SpamClass()` or `EggsClass()`), and the classes are instances of the metaclass type (itself an instance of itself), allowing meta programming and reflection. Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0. The long term plan is to support gradual typing and as of Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named `mypy` supports compile-time type checking.

Mathematics Python has the usual C arithmetic operators (+, -, *, /, %). It also has `**` for exponentiation, e.g. `5**3 == 125` and `9**0.5 == 3.0`, and a new matrix multiply `@` operator is included in version 3.5. [71] Additionally, it has a unary operator (`~`), which essentially inverts all the bytes of its one argument. For integers, this means `~x=-x-1`. Other operators include bitwise shift operators `x << y`, which shifts `x` to the left `y` places, the same as `x*(2**y)`, and `x >> y`, which shifts `x` to the right `y` places, the same as `x/(2**y)`. [73] The behavior of division has changed significantly over time: Python 2.1 and earlier use the C division behavior. The `/` operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. `7 / 3 ==`

2 and $-7 / 3 == -2$. Python 2.2 changes integer division to round towards negative infinity, e.g. $7 / 3 == 2$ and $-7 / 3 == -3$. The floor division `//` operator is introduced. So $7 // 3 == 2$, $-7 // 3 == -3$, $7.5 // 3 == 2.0$ and 6/13/2017 Python (programming language)

-

Wikipedia

[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) 8/19 $-7.5 // 3 == -3.0$. Adding from `__future__` import division causes a module to use Python 3.0 rules for division (see next). Python 3.0 changes `/` to be always floating-point division. In Python terms, the pre-3.0 `/` is classic division, the version-3.0 `/` is real division, and `//` is floor division. Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation $(a+b) // b == a // b + 1$ is always true. It also means that the equation $b * (a // b) + a \% b == a$ is valid for both positive and negative values of a . However, maintaining the validity of this equation means that while the result of $a \% b$ is, as expected, in the half-open interval $[0, b)$, where b is a positive integer, it has to lie in the interval $(b, 0]$ when b is negative. Python provides a `round` function for rounding a float to the nearest integer. For tie-breaking, versions before 3 use round-away-from-zero: `round(0.5)` is 1.0, `round(-0.5)` is -1.0. Python 3 uses round to even: `round(1.5)` is 2, `round(2.5)` is 2. Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression $a < b < c$ tests whether a is less than b and b is less than c . C-derived languages interpret this expression differently: in C, the expression would first evaluate $a < b$, resulting in 0 or 1, and that result would then be compared with c . Python has extensive built-in support for arbitrary precision arithmetic. Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type `int`, to arbitrary precision, belonging to the python type `long`, where needed. The latter have an "L" suffix in their textual representation. (In Python 3, the distinction between the `int` and `long` types was eliminated; this behavior is now entirely contained by the `int` class.) The `Decimal` type/class in module `decimal` (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes. The `Fraction` type in module `fractions` (since version 2.6) provides arbitrary precision for rational numbers. Due to Python's extensive mathematics library, and the third-party library NumPy which further extends the native capabilities, it is frequently used as a scientific scripting language to aid in

problems such as numerical data processing and manipulation. Libraries Python has a large standard library, commonly cited as one of Python's greatest strengths, providing tools suited to many tasks. This is deliberate and has been described as a "batteries included" Python philosophy

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it.

It is a type of signal dispensation in which input is an image, like video frame or photograph and output may be image or characteristics associated with that image.

Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

ADVANTAGES OF IMAGE PROCESSING:

The purpose of image processing is divided into 5 groups.

They are:

VISUALIZATION - Observe the objects that are not visible.

IMAGE SHARPENING AND RESTORATION - To create a better image.

IMAGE RETRIEVAL - Seek for the image of interest.

MEASUREMENT OF PATTERN – Measures various objects in an image.

IMAGE RECOGNITION – Distinguish the objects in an image.

FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING:

1. IMAGE ACQUISITION:

This is the first step or process of the fundamental steps of digital image processing. Image acquisition could be as simple as being given an image that is

already in digital form. Generally, the image acquisition stage involves pre-processing, such as scaling etc.

2. IMAGE ENHANCEMENT:

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc.

3. IMAGE RESTORATION:

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

4. COLOR IMAGE PROCESSING

Color image processing is an area that has been gaining its importance because of the significant increase in the use of digital images over the Internet. This may include color modelling and processing in a digital domain etc.

5. WAVELETS AND MULTIREOLUTION PROCESSING

Wavelets are the foundation for representing images in various degrees of resolution. Images subdivision successively into smaller regions for data compression and for pyramidal representation.

6. COMPRESSION

Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very much necessary to compress data.

7. MORPHOLOGICAL PROCESSING

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape.

8. SEGMENTATION

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.

9. REPRESENTATION AND DESCRIPTION

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. Description deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

10. OBJECT RECOGNITION

Recognition is the process that assigns a label, such as, “vehicle” to an object based on its descriptors

As CNN based learning algorithm shows better performance on the classification issues, the rich labeled data could be more useful in the training stage. 3D object classification and pose estimation is a jointed mission aiming at separate different posed apart in the descriptor form.

In the training stage, we prepare 2D training images generated from our module with their class label and pose label. We fully exploit the information lies in their labels by using a triplet and pair-wise jointed loss function in CNN training.

As CNN based learning algorithm shows better performance on the classification issues, the rich labeled data could be more useful in the training

stage. 3D object classification and pose estimation is a jointed mission aiming at separate different pose apart in the descriptor form.

In the training stage, we prepare 2D training images generated from our module with their class label and pose label. We fully exploit the information that lies in their labels by using a triplet and pair-wise jointed loss function in CNN training.

Both class and pose label are in consideration in the triplet loss. The loss score will be smaller when features from the same class and same pose is more similar and features from different classes or different poses will lead to a much larger loss score.

This loss is also jointed with a pair wise component to make sure the loss is never be zero and have a restriction on the model scale.

About the training and feature extraction process, it is a rough implementation by using OpenCV and Caffe from the idea of Paul Wohlhart. The principal purpose of this API is constructing a well labeled database from .ply models for CNN training with triplet loss and extracting features with the constructed model for prediction or other purpose of pattern recognition, algorithms into two main Class:

OpenCV is a library for real-time computer vision. It has very powerful functions that make the art of processing images and getting information about them easy. In this post, we will review some of the functions that we used for making a 3D-reconstruction from an image in order to make an autonomous robotic arm.

OpenCV uses a pinhole camera model. This model works by projecting 3D points onto the image plane using a perspective transformation.

There are some functions of OpenCV that help us accomplish our goal. These functions work with a chess board model to calibrate the model, so the first thing is to get a chess board and take some photos of it. We took several photographs in order to get a better calibration.

It has come to my attention that most 3D reconstruction tutorials out there are a bit lacking. Don't get me wrong they're great, but they're fragmented or go too deep into the theory or a combination of both.

Worse yet they use specialized datasets (like Tsukuba) and this is a bit of a problem when it comes to using the algorithms for anything outside those datasets (because of parameter tuning).

I believe that the cool thing about 3D reconstruction (and computer vision in general) is to reconstruct the world around you, not somebody else's world (or dataset). This tutorial is a humble attempt to help you recreate your own world using the power of OpenCV.

Simply put this tutorial will take you from scratch to point cloud USING YOUR OWN PHONE CAMERA and pictures. So without further ado, let's get started.

Part 1 (theory and requirements):

Covers a very very brief overview of the steps required for stereo 3D reconstruction. You are here.

Part 2 (Camera calibration):

Covers the basics on calibrating your own camera with code.

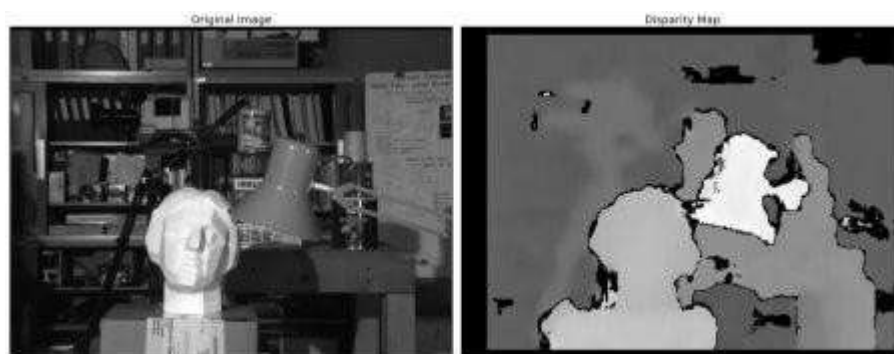
Part 3(Disparity map and point cloud):

Covers the basics on reconstructing pictures taken with the camera previously calibrated with code.

THE STEPS REQUIRED FOR 3D RECONSTRUCTION.

There are many ways to reconstruct the world around but it all reduces down to getting an actual depth map.

A depth map is a picture where every pixel has depth information (instead of color information). It is normally represented like a grayscale picture.



As mentioned before there are different ways to obtain a depth map and these depend on the sensor being used. A type of sensor could be a simple camera (from now on called RGB camera in this text) but it is possible to use others like LiDAR or infrared or a combination.

The type of sensor will determine the accuracy of the depth map. In terms of accuracy it normally goes like this: LiDAR > Infrared > Cameras. Depth maps can also be colorized to better visualize depth.



Depending on the kind of sensor used, there's more or less steps required to actually get the depth map. The Kinect camera for example uses infrared sensors combined with RGB cameras and as such you get a depth map right away (because it is the information processed by the infrared sensor).

But what if you don't have anything else but your phone camera? In this case you need to do stereo reconstruction. Stereo reconstruction uses the same principle your brain and eyes use to actually understand depth.

The gist of it consists in looking at the same picture from two different angles, look for the same thing in both pictures and infer depth from the difference in position. This is called stereo matching.

In order to do stereo matching it is important to have both pictures have the exact same characteristics. Put differently, both pictures shouldn't have any distortion. This is a problem because the lens in most cameras causes distortion. This means that in order to accurately do stereo matching one needs to know the optical centers and focal length of the camera.

In most cases this information will be unknown (especially for your phone camera) and this is why stereo 3D reconstruction requires the following steps:

1. **Camera calibration:** Use a bunch of images to infer the focal length and optical centers of your camera
2. **Undistort images:** Get rid of lens distortion in the pictures used for reconstruction
3. **Feature matching:** Look for similar features between both pictures and build a depth map
4. **Reproject points:** Use depth map to reproject pixels into 3D space.
5. **Build point cloud:** Generate a new file that contains points in 3D space for visualization.
6. Build mesh to get an actual 3D model (outside of the scope of this tutorial, but coming soon in different tutorial)

Step 1 only needs to be executed once unless you change cameras. Steps 2–5 are required every time you take a new pair of pictures...and that is pretty much it.

The actual mathematical theory (the why) is much more complicated but it will be easier to tackle after this tutorial since you will have a working example that you can experiment with by the end of it.

Development of the technology in the area of the cameras, computers and algorithms for 3D the reconstruction of the objects from the images resulted in the increased popularity of the photogrammetry. Algorithms for the 3D model reconstruction are so advanced that almost anyone can make a 3D model of photographed object. The main goal of this paper is to examine the possibility of obtaining 3D data for the purposes of the close-range photogrammetry applications, based on the open source technologies. All steps of obtaining 3D point cloud are covered in this paper. Special attention is given to the camera calibration, for which two-step process of calibration is used. Both, presented algorithm and accuracy of the point cloud are tested by calculating the spatial difference between referent and produced point clouds. During algorithm testing, robustness and swiftness of obtaining 3D data is noted, and certainly usage of this and similar algorithms has a lot of potential in the real-time application. That is the reason why this research can find its application in the architecture, spatial planning, protection of cultural heritage, forensic, mechanical engineering, traffic management, medicine and other sciences.

CHAPTER 7

OPENCV

OpenCV (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe

HISTORY:

Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects

including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months¹ and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

On May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV

APPLICATIONS:

OpenCV's application areas include:

- 2D and 3D feature toolkits

- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

PROGRAMMING LANGUAGE

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Perl, Ch, Haskell, and Ruby have been developed to encourage adoption by a wider audience.

Since version 3.4, **OpenCV.js** is a JavaScript binding for selected subset of OpenCV functions for the web platform.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface

HARDWARE ACCELERATION

If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

A CUDA-based GPU interface has been in progress since September 2010.

An OpenCL-based GPU interface has been in progress since October 2012, documentation for version 2.4.13.3 can be found at docs.opencv.org.

OS SUPPORT

OpenCV runs on the following desktop operating systems: Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD. OpenCV runs on the following mobile operating systems: Android, iOS, Maemo, BlackBerry 10. The user can get official releases from SourceForge or take the latest sources from GitHub. OpenCV uses CMake.

CHAPTER 8

CONCLUSION

The purpose of reducing the errors that occur in the traditional attendance taking system has been achieved by implementing this automated attendance system. In this paper, face recognition system have been presented using deep learning which exhibits robustness towards recognition of the users with accuracy of 98.3% . The result shows the capability of the system to cope with the change in posing and projection of faces. From face recognition with deep learning, it has been determined that during face detection, the problem of illumination is solved as the original image is turned into a HOG representation that captures the major features of the image regardless of image brightness. In the face recognition method, local facial landmarks are considered for further processing. After which faces are encoded which generates 128 measurements of the captured face and the optimal face recognition is done by finding the person's name from the encoding. The result is then used to generate an excel sheet, the pdf of which is sent to the students and professors on weekly interval. This system is convenient to the user and it gives better security.

Algorithm testing:

Algorithm was tested by comparison of the 3D point cloud acquired with an open source solution, OpenCV and point cloud acquired with the proven software Agisoft PhotoScan. RO in OpenCV was calculated based on the 152 image points in each pair of the images used for the orientation i.e. stereo calibration. Totally 10 pairs of the images were used for the orientation. On the other hand, Agisoft's algorithm detected 2247 tie points visible on the both images, creating that way a strong geometry. Ground sample distance was 0.22mm. Precalibrated camera was used for the point cloud generation and the further optimization of camera is not done. Agisoft point cloud was created with the maximal offered quality, from alignment to point cloud generation, and with minimal further processing i.e. depth filtering and interpolation, to fit OpenCV point cloud that is minimally processed.

Accuracy of the point cloud Accuracy of the point cloud was tested by the comparison of produced point cloud to the model acquired with better camera and in larger scale, which is considered as an reference in this paper. Reference i.e. etalon model is produced based on the 73 images of the resolution 4272x2848 with the ground sample distance of 0.07mm. In order to examine accuracy of the stereo point cloud, point cloud created with the OpenCV and point cloud created in the Agisoft PhotoScan with the same images were compared with the reference model.

CODING:

```
import cv2

import sys

import numpy as np

cascPath = sys.argv[1]

faceCascade = cv2.CascadeClassifier(cascPath)

cap = cv2.VideoCapture(0)      #640,480

w = 640.0

h = 480.0

while True:

    # Capture frame-by-frame

    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
```

```

    gray,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    #flags=cv2.HAAR_SCALE_IMAGE
    flags = 0
)

# Draw a rectangle around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
#out.write(frame)

# Display the resulting frame
gray1 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
# hsv1 = cv2.cvtColor(frame, cv2.COLOR_BGR2CAB)

# define range of red color in HSV
lower_red = np.array([36, 25, 25])
upper_red = np.array([70, 255, 255])

# create a red HSV colour boundary and
# threshold HSV image

```

```
mask = cv2.inRange(hsv, lower_red, upper_red)
edges = cv2.Canny(frame,100,200)
```

```
# Bitwise-AND mask and original image
res = cv2.bitwise_and(frame,frame, mask= mask)
cv2.imshow('Video', frame)
cv2.imshow('Video1', gray1)
cv2.imshow('video3', hsv)
cv2.imshow('video4', edges)
#cv2.imshow('video5', hsv1)
```

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

```
# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

```
from .skin_detector import *
from .scripts import display
from .scripts import find_images
```

```
import os
```

```
import cv2
```

```
import numpy
```

```
def find_images(path, recursive=False, ignore=True):
```

```
    if os.path.isfile(path):
```

```
        yield path
```

```
    elif os.path.isdir(path):
```

```
        assert os.path.isdir(path), 'FileIO - get_images: Directory does not exist'
```

```
        assert isinstance(recursive, bool), 'FileIO - get_images: recursive must be a  
boolean variable'
```

```
        ext, result = ['png', 'jpg', 'jpeg'], []
```

```
        for path_a in os.listdir(path):
```

```
            path_a = path + '/' + path_a
```

```
            if os.path.isdir(path_a) and recursive:
```

```
                for path_b in find_images(path_a):
```

```
                    yield path_b
```

```
            check_a = path_a.split('.')[-1] in ext
```

```
            check_b = ignore or ('-' not in path_a.split('/')[-1])
```

```
            if check_a and check_b:
```

```
                yield path_a
```

```
    else:
```

```
raise ValueError('error! path is not a valid path or directory')
```

```
def display(title, img, max_size=200000):  
    assert isinstance(img, numpy.ndarray), 'img must be a numpy array'  
    assert isinstance(title, str), 'title must be a string'  
    scale = numpy.sqrt(min(1.0, float(max_size) / (img.shape[0] *  
img.shape[1])))  
    shape = (int(scale * img.shape[1]), int(scale * img.shape[0]))  
    img = cv2.resize(img, shape)  
    cv2.imshow(title, img)
```

```
import logging
```

```
import cv2
```

```
import numpy
```

```
from . import scripts
```

```
logger = logging.getLogger('main')
```

```
def get_hsv_mask(img, debug=False):  
    assert isinstance(img, numpy.ndarray), 'image must be a numpy array'  
    assert img.ndim == 3, 'skin detection can only work on color images'
```

```
logger.debug('getting hsv mask')
```

```
lower_thresh = numpy.array([0, 50, 0], dtype=numpy.uint8)
```

```
upper_thresh = numpy.array([120, 150, 255], dtype=numpy.uint8)
```

```
img_hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
```

```
msk_hsv = cv2.inRange(img_hsv, lower_thresh, upper_thresh)
```

```
msk_hsv[msk_hsv < 128] = 0
```

```
msk_hsv[msk_hsv >= 128] = 1
```

```
if debug:
```

```
    scripts.display('input', img)
```

```
    scripts.display('mask_hsv', msk_hsv)
```

```
return msk_hsv.astype(float)
```

```
def get_rgb_mask(img, debug=False):
```

```
    assert isinstance(img, numpy.ndarray), 'image must be a numpy array'
```

```
    assert img.ndim == 3, 'skin detection can only work on color images'
```

```
    logger.debug('getting rgb mask')
```

```
    lower_thresh = numpy.array([45, 52, 108], dtype=numpy.uint8)
```

```

upper_thresh = numpy.array([255, 255, 255], dtype=numpy.uint8)

import argparse

import logging

import cv2

import skin_detector

logger = logging.getLogger('main')

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description=__doc__)

    parser.add_argument('image_paths', type=str, nargs='+', help="paths to one
or more images or image directories")

    parser.add_argument('-b', '--debug', dest='debug', action='store_true',
help='enable debug logging')

    parser.add_argument('-q', '--quite', dest='quite', action='store_true',
help='disable all logging')

    parser.add_argument('-d', '--display', dest='display', action='store_true',
help="display result")

    parser.add_argument('-s', '--save', dest='save', action='store_true',
help="save result to file")

    parser.add_argument('-t', '--thresh', dest='thresh', default=0.5, type=float,
help='threshold for skin mask')

```

```

args = parser.parse_args()

if args.debug:
    logging.basicConfig(level=logging.DEBUG)
else:
    logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("main")

for image_arg in args.image_paths:
    for image_path in skin_detector.find_images(image_arg):
        logging.info("loading image from {}".format(image_path))
        img_col = cv2.imread(image_path, 1)

        img_msk = skin_detector.process(img_col)

        if args.display:
            skin_detector.scripts.display('img_col', img_col)
            skin_detector.scripts.display('img_msk', img_msk)
            skin_detector.scripts.display('img_skn', cv2.bitwise_and(img_col,
img_col, mask=img_msk))
            cv2.waitKey(0)

import ez_setup

```

```

ez_setup.use_setuptools()

from setuptools import setup

setup(
    name="Skin Detector",
    version="1.0a Prototype",
    url="",
    author='Will Brennan',
    author_email='william.brennan@skytales.com',
    license='GPL',
    install_requires=["numpy"], )

from setuptools import setup, find_packages

with open('README.rst') as f:
    readme = f.read()

with open('LICENSE') as f:
    license = f.read()

with open('requirements.txt') as f:
    required = f.read().splitlines()

setup(

```

```

name='Skin Detector',
version='prototype',
description='A speedy skin-detector based upon colour thresholding',
long_description=readme,
author='WillBrennan',
author_email='WillBrennan@users.noreply.github.com',
url='https://github.com/WillBrennan/SkinDetector',
license=license,
install_requires=required,
packages=find_packages(exclude=('tests', 'docs'))

```

```
import argparse
```

```
import logging
```

```
import cv2
```

```
import skin_detector
```

```
if __name__ == '__main__':
```

```
    parser = argparse.ArgumentParser(description=__doc__)
```

```
    parser.add_argument('-b', '--debug', dest='debug', action='store_true',
help='enable debug logging')
```

```
    parser.add_argument('-t', '--thresh', dest='thresh', default=0.5, type=float,
help='threshold for skin mask')
```

```
    args = parser.parse_args()
```

```
if args.debug:
    logging.basicConfig(level=logging.DEBUG)
else:
    logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("main")

cam = cv2.VideoCapture(0)
logging.info("press any key to exit")

while True:
    ret, img_col = cam.read()
    img_msk = skin_detector.process(img_col)

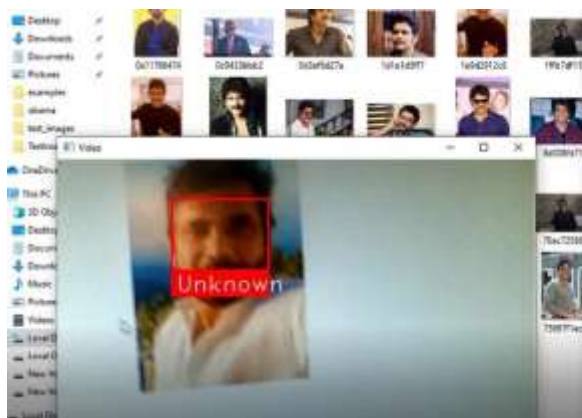
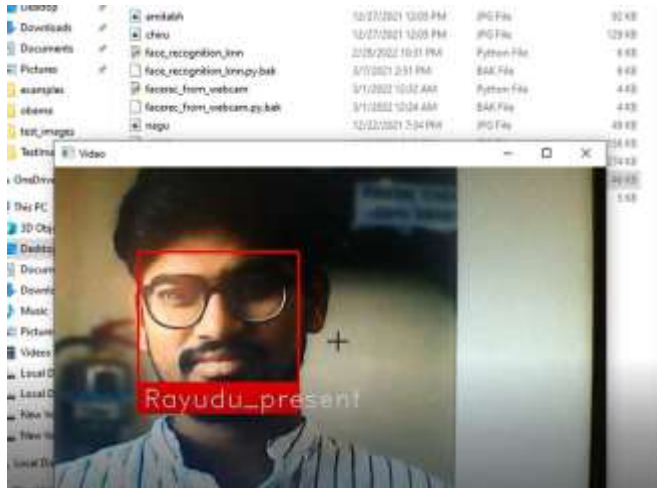
    skin_detector.scripts.display('img_col', img_col)
    skin_detector.scripts.display('img_msk', img_msk)
    skin_detector.scripts.display('img_skn', cv2.bitwise_and(img_col, img_col,
mask=img_msk))

    waitkey = cv2.waitKey(5)
    if waitkey != -1:
        break
```

CHAPTER 9

RESULTS

OUTPUT SCREENSHOT



CHAPTER 10

REFERENCES

- [1] V. Blanz and T. Vetter. Face recognition based on fitting a 3d morphable model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1063–1074, 2003.2
- [2] P. Breuer, K.-I. Kim, W. Kienzle, B. Scholkopf, and V. Blanz. Automatic 3d face reconstruction from single images or video. In *Proc. of the 8th IEEE International Conference on Automatic Face Gesture Recognition (FG '08)*, pages 1–8, 2008.2
- [3] Chaoyang, L. Wang, Y. Wang, F. Matsushita, K., and Soong. Binocular photometric stereo acquisition and reconstruction for 3d talking head applications. In *Interspeech 2013 sub-mission*, 2013.3
- [4] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. Rgb-d mapping: Using depth cameras for dense 2d modeling of indoor environments. *Proc. of International Symposium on Experimental Robotics*, 2010.1
- [5] M. Hernandez, J. Choi, and G. Medioni. Laser scan quality 3-d face modeling using a low-cost depth camera. *Proc. of the 20th European Signal Processing Conference (EU-SIPCO)*, pages 1995–1999, 2012.1,2,3,4,5,7
- [6] C. Mayer, M. Wimmer, M. Eggers, and B. Radig, “Facial Expression Recognition with 3D Deformable Models.” *Second International Conferences on Advances in Computer-Human Interactions*, pp 26-31, 2009.
- [7] V. Blanz, and T. Vetter, “Face Recognition Based on Fitting a 3D Morphable Model”, *IEEE Trans. PAMI*, vol. 25, no. 9, pp. 1063-1074, 2003.
- [8] X. Lu, and A. Jain, “Deformation Modeling for Robust 3D Face Matching.” *Transactions on Pattern Analysis and Machine Intelligence*, Vol. 30, No. 8, pp. 1347-1356, 2008.
- [9] B. Moghaddam, J. Lee, F. Fister, and R. Machiraju, “Model-Based 3D Face Capture with Shape-from-Silhouettes.” *Proceedings of the IEEE International Workshop on Analysis and Modeling of Faces and Gestures*, 2003.
- [10] Z. Riaz, C. Mayer, M. Wimmer, M. Beetz, and B. Radig, “A Model Based Approach for Expression Invariant Face Recognition.” *ICB*, pp 289-298, 2009.

- [11]B. Weyrauch, and J. Huang, "Component-based Face Recognition with 3D Morphable Models." Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2004.
- [12]J. Harguess, S. Gupta, and J. Aggarwal, "3D Face Recognition with the Average-Half-Face." IEEE Explore, 2008.
- [13]S. Ramanathan, A. Kassim, Y. Venkatesh, and W. Wah, "Human Facial Expression Recognition Using 3D Morphable Model." ICIP, pp 661-664, 2006.
- [14]J. Lee, B. Moghaddam, F. Fister, and R. Machiraju, "Finding Optimal Views for 3D Face Shape Modeling." Mitsubishi Electric Research Laboratories, Inc, Massachusetts, USA, 2004.
- [15] Agisoft. Agisoft photoscan. <http://www.agisoft.com/>, [Online; accessed 12-April-2017]. Balletti, C., Guerra, F., Tsioukas, V., Vernier, P., 2014. Calibration of Action Cameras for Photogrammetric Purposes. *Sensors*, 14(9), pp. 17471-17490. Brown, D., 1966.
- [16] Decentering distortion of lenses. *Photometric Engineering* 32(3), pp. 444-462. Cambi, N., 1993. Bilješke uz kipove Kibele (Magna Mater) iz Senja. *Senjski zbornik*, 20(1), pp. 33-43. Carrillo, L., López, A., Lozano, R., Pégard, C., 2012.
- [17] Combining stereo vision and inertial navigation system for a quad-rotor UAV. *Journal of Intelligent & Robotic Systems*, 65(1-4), pp. 373-387. CloudCompare.
- [18] Cloud Compare. <http://www.danielgm.net/cc/>, [Online; accessed 12-April-2017]. Ducrot, A., Dumortier, Y., Herlin, I., Ducrot, V., 2011.
- [19] Realtime quasi dense two-frames depth map for autonomous guided vehicles. In *Intelligent Vehicles Symposium (IV)*, IEEE, pp. 497-503. Gašparović, M., Gajski, D., 2016a.
- [20] Two-step camera calibration method developed for micro UAV's. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Prague, Czech Republic, Vol. XLI, Part B1, pp. 829-833. Gašparović, M., Gajski, D., 2016b.

[21] Testing of Image Quality Parameters of Digital Cameras for Photogrammetric Surveying with Unmanned Aircrafts. Geodetski list, 70(3), pp. 253-266. GIMP. GNU Image Manipulation Program. <https://www.gimp.org/>, [Online; accessed 12-April-2017].