Auto recognizing system for Dice games

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

In

Computer Science & Engineering

Under the Supervision of

Mr. Suman Saha

By

Neeraj Singh | 111316

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled “Auto Recognition system For Dice games”, submitted by Mr. Neeraj Singh in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Mr. Suman Saha
Assistant Professor

Acknowledgement

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to **Mr.Suman Saha** for his guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project. I would like to express my gratitude towards faculty member of **Jaypee University of Information Technology** for their kind co-operation and encouragement which help me in completion of this project. I would like to express my special gratitude and thanks to staff persons for giving me such attention and time. My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

Date: Neeraj Singh

Table of Content

List of Figures

Abstract

Dice have been part of the way that humans play games for many centuries. It is probably about as long as man has been playing games at all. Six-sided dice had been developed near Rome dating back to 900 B.C.. Recently, dice are most often associated with gambling games and strategy war games; they are used dice in highly abstract ways in both. Similarly, liar dice games are a kind of interesting and popular game that can be found in almost every pub in China and Hong Kong.

Nowadays, electronic gambling machines have become increasingly popular. The first such machines employing electromechanical principles were slot machines. For increasing interest, electronic gambling machines have emerged using mechanical motion of various types to produce random numbers. Typically, these kinds of games are electronic roulette and electronic dice machines. In their structure, the important module of electronic dice machines is the detection of dice location and throwing manner. Such machines are extremely attractive to users. Simultaneously, they provide results visually giving no room for fraudulently acquired numbers.

Here, we propose an automated detection system with machine vision to execute such inspection. Machine vision is a powerful tool and is widely employed in automatic monitoring and detecting processes. The system employs image processing techniques, and the unsupervised algorithm to estimate the location of each die and identify the spot number accurately and effectively. The proposed algorithms are substituted for manual recognition. From the experimental results, it is found that this system is excellent due to its good capabilities which include flexibility, high speed, and high accuracy.

Chapter 1
Introduction

1.Introduction

1.1Introduction

Dice is a common table game in casinos, and most popular in the casinos in Asia. As automatic or computer-controlled games are becoming popular, many are interested in the technologies able to assist or even replace human bankers. A computer vision system is proposed in this paper for *dice recognition*, which refers to the automatic reading of the numbers of dots on dice, in generic table game settings.

The proposed system consists of three modules for dice detection, top surface segmentation and dots identification. To detect the dice, a scheme called modified unsupervised grey clustering algorithm is proposed to locate the dice.
The grey theory has been applied to research in industry, social system, ecological system, environmental system, education, business management, and traffic control. The object of the present study is to automatically recognize the score of dice. This study develops an iterative calculation that is suitable for dice identification, and is organized as follows: an image acquisition system, an image processing method and an auto-recognition system for dice are presented contains conclusions about the proposed system.

Our aim, which is to develop a method of dice recognition that is fast, robust, reasonably simple and accurate with a relatively simple and easy to understand algorithms and techniques. The examples provided in this thesis are real-time and taken from our own surroundings.

Chapter 2
Literature Review

2. Literature Review

The Project on dice recognition had helped detailed survey of a number of dice recognition algorithm.

There is a lot of previous work is done on Dice recognition system.

Description of related paper work is described as:

2.1Color and Illumination Invariant Dice Recognition

Published by: Gee-Sern Hsu, Hsiao-Chia Peng, Shang-Min Yeh
Artificial Vision Lab, National Taiwan University of Science and Technology Taipei, Taiwan

Abstract: A system is proposed for automatic reading of the number of dots on dice in general table game settings. Different from previous dice recognition systems which recognize dice of a specific color using a single top-view camera in an enclosure with controlled settings, the proposed one uses multiple cameras to recognize dice of various colors posed in a wide range of viewing angle and under uncontrolled conditions. It is composed of three modules. Module-1 locates the dice using the gradient-conditioned color segmentation (GCCS), proposed in this paper, to segment dice of arbitrary colors from the background. Module-2 exploits the local invariant features good for building homographies across multiple views and lighting conditions. The homographies are used to enhance coplanar features and weaken non-coplanar features, giving a solution to segment the top faces of the dice and make up the features ruined by possible specular reflection. To identify the dots on the segmented top faces, an MSER (Maximally Stable Extreme Region) detector is embedded in its consistency in locating the dot regions regardless of illumination and viewpoint variations. Experiments show that the proposed system performs satisfactorily in various test conditions.

2.2Image Edge Detection Based On Opencv

Published by: Guobo Xie and Wen Lu
School of computer, Guangdong University of technology, Guang zhou, China

Abstract: Image processing is one of most growing research area these days and now it is very much integrated with the industrial production. Generally speaking, It is very difficult for us to distinguish the exact number of the copper core in the tiny wire, However, in order to ensure that the wire meets the requirements of production, we have to know the accurate number of copper core in the wire. Here the paper will introduce a method of image edge detection to determine the exact number of the copper core in the tiny wire based on OpenCV with rich computer vision and image processing algorithms and functions. Firstly, we use high-resolution camera to take picture of the internal structure of the wire. Secondly, use

OpenCV image processing functions to implement image pre-processing. Thirdly we use morphological opening and closing operations to segment image because of their blur image edges. Finally the exact number of copper core can be clearly distinguished through contour tracking. By using of Borland C++ Builder 6.0, experimental results show that OpenCV based image edge detection methods are simple, high code integration, and high image edge positioning accuracy.

2.3An Auto-Recognizing System for Dice Games Using a Modified Unsupervised Grey Clustering Algorithm

Published by: Kuo-Yi Huang
Department of Mechatronic Engineering, Huafan University, Taipei, Taiwan

Abstract: a novel identification method based on a machine vision system is proposed to recognize the score of dice. The system employs image processing techniques, and the modified unsupervised grey clustering algorithm (*MUGCA*) to estimate the location of each die and identify the spot number accurately and effectively. The proposed algorithms are substituted for manual recognition. From the experimental results, it is found that this system is excellent due to its good capabilities which include flexibility, high speed, and high accuracy.

2.4Image Identification Scheme for Dice Game

Published by: Chin-Ho Chung, Wen-Yuan Chen, and Bor-Liang Lin

Department of Electronic Engineering,

Ta Hwa Institute of Technology, Hsin-Chu, Taiwan

Abstract: In this paper, based on the features of dice images and the least distance criterion (LDC) method, developed an auto recognition scheme for dice games. First, use R plane to replace gray-scale image to speed up processing. Next, use O'stu method to convert the R plane into binary. Furthermore, image pre-processing steps are used to filter out the noise, and a non-circular object discard stage eliminates the bigger noise blocks. Finally, the LDC, the dice score calculation, and the score exception processing is used to accomplish the automatic identification scheme of the dice.

Chapter 3
OpenCV

3. OpenCV

3.1Introduction

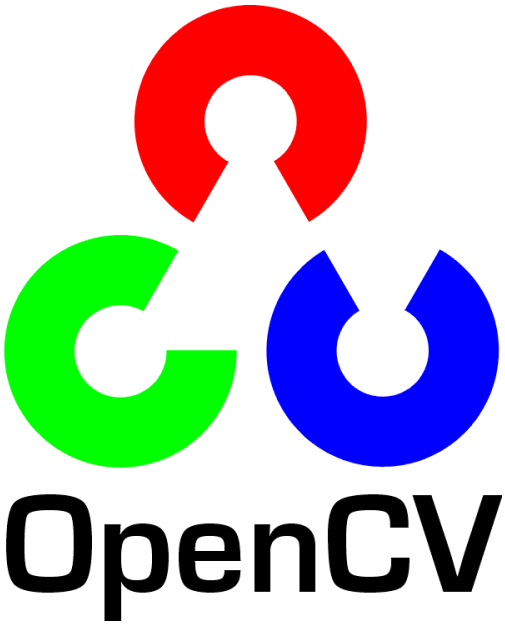

FIGURE 1:OPENCV

OpenCV (Open Source Computer Vision Library) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. The document describes the so-called OpenCV 2.x API, which is essentially a C++ API, as opposite to the C-based OpenCV 1.x API. It is developed by Intel Russia research center in Nizhny Novgorod, and now supported by Willow Garage and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

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 to be open or free themselves.

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. In mid-2008, OpenCV obtained corporate support from Willow Garage, and is now again under active development. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was on 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 months 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.

3.2 Application

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Ego motion 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 (meta-algorithm)
- 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)

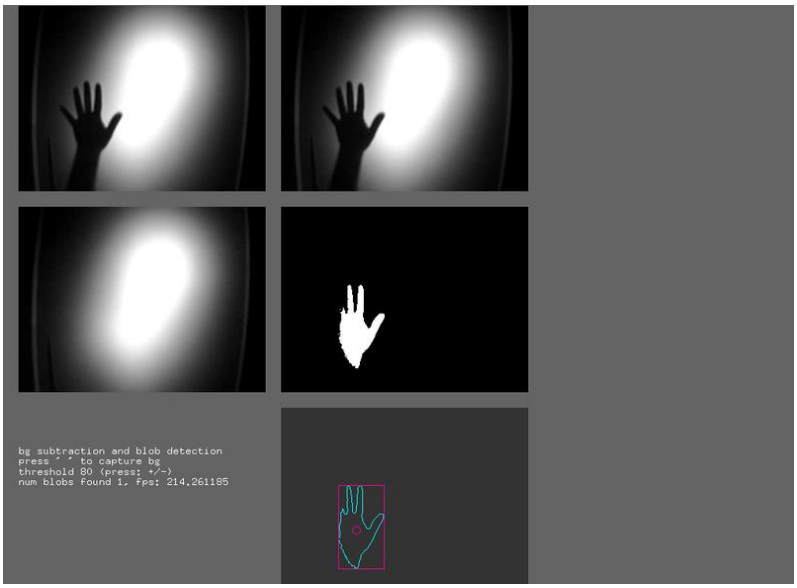

FIGURE 2:OPENFRAMEWORKS RUNNING THE OPENCV ADD-ON EXAMPLE

3.3 Modular Structure

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- **core** - a compact module defining basic data structures, including the dense multi-dimensional arrayMat and basic functions used by all other modules.
- **imgproc** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **video** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **features2d** - salient feature detectors, descriptors, and descriptor matchers.
- **objdetect** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- **highgui** - an easy-to-use interface to simple UI capabilities.
- **videoio** - an easy-to-use interface to video capturing and video codecs.
- **gpu** - GPU-accelerated algorithms from different OpenCV modules.
- ... some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

Automatic Memory Management

First of all, std::vector, Mat, and other data structures used by the functions and methods have destructors that deallocate the underlying memory buffers when needed. This means that the destructors do not always deallocate the buffers as in case of Mat. They take into account possible data sharing. A destructor decrements the reference counter associated with the matrix data buffer. The buffer is deallocated if and only if the reference counter reaches zero, that is, when no other structures refer to the same buffer. Similarly, when a Mat instance is copied, no actual data is really copied. Instead, the reference counter is incremented to memorize that there is another owner of the same data. There is also theMat::clone method that creates a full copy of the matrix data.

Automatic Allocation of Output Data

OpenCV deallocates the memory automatically, as well as automatically allocates the memory for output function parameters most of the time. So, if a function has one or more input arrays (cv::Mat instances) and some output arrays, the output arrays are automatically allocated or reallocated. The size and type of the output arrays are determined from the size and type of input arrays. If needed, the functions take extra parameters that help to figure out the output array properties.

InputArray and OutputArray

Many OpenCV functions process dense 2-dimensional or multi-dimensional numerical arrays. Usually, such functions take cpp:class:*Mat* as parameters, but in some cases it's more convenient to use std::vector<> (for a point set, for example) or Matx<> (for 3x3 homography matrix and such). To avoid many duplicates in the API, special "proxy" classes have been introduced. The base "proxy" class is InputArray. It is used for passing read-only arrays on a function input. The derived from InputArray class OutputArray is used to specify an output array for a function. Normally, you should not care of those intermediate types (and you should not declare variables of those types explicitly) - it will all just work automatically. You can assume that instead of InputArray/OutputArray you can always use Mat, std::vector<>, Matx<>, Vec<> or Scalar. When a function has an optional input or output array, and you do not have or do not want one, pass cv::noArray().

Fixed Pixel Types. Limited Use of Templates

Templates is a great feature of C++ that enables implementation of very powerful, efficient and yet safe data structures and algorithms. However, the extensive use of templates may dramatically increase compilation time and code size. Besides, it is difficult to separate an interface and implementation when templates are used exclusively. This could be fine for basic algorithms but not good for computer vision libraries where a single algorithm may span thousands lines of code. Because of this and also to simplify development of bindings for other languages, like Python, Java, Matlab that do not have templates at all or have limited template capabilities, the current OpenCV implementation is based on polymorphism and runtime dispatching over templates. In those places where runtime dispatching would be too slow (like pixel access operators), impossible or just very inconvenient the current implementation introduces small template classes, methods, and functions. Anywhere else in the current OpenCV version the use of templates is limited.

Consequently, there is a limited fixed set of primitive data types the library can operate on. That is, array elements should have one of the following types:

- 8-bit unsigned integer (uchar)
- 8-bit signed integer (schar)
- 16-bit unsigned integer (ushort)
- 16-bit signed integer (short)
- 32-bit signed integer (int)
- 32-bit floating-point number (float)
- 64-bit floating-point number (double)
- a tuple of several elements where all elements have the same type (one of the above). An array whose elements are such tuples, are called multi-channel arrays, as opposite to the single-channel arrays, whose elements are scalar values. The maximum possible number of channels is defined by theCV_CN_MAX constant, which is currently set to 512.

Error Handling

OpenCV uses exceptions to signal critical errors. When the input data has a correct format and belongs to the specified value range, but the algorithm cannot succeed for some reason (for example, the optimization algorithm did not converge), it returns a special error code (typically, just a boolean variable).

The exceptions can be instances of the cv::Exception class or its derivatives. In its turn, cv::Exception is a derivative of std::exception. So it can be gracefully handled in the code using other standard C++ library components. The exception is typically thrown either using the CV_Error(errcode, description) macro, or its printf-likeCV_Error_(errcode, printf-spec, (printf-args)) variant, or using the CV_Assert(condition) macro that checks the condition and throws an exception when it is not satisfied. For performance-critical code, there is CV_DbgAssert(condition) that is only retained in the Debug configuration. Due to the automatic memory management, all the intermediate buffers are automatically deallocated in case of a sudden error.

Multi-threading and Re-enterability

The current OpenCV implementation is fully re-enterable. That is, the same function, the same *constant* method of a class instance, or the same *non-constant* method of different class instances can be called from different threads. Also, the same cv::Mat can be used in different threads because the reference-counting operations use the architecture-specific atomic instructions.

Algorithm

class **Algorithm**

This is a base class for all more or less complex algorithms in OpenCV, especially for classes of algorithms, for which there can be multiple implementations. The examples are stereo correspondence (for which there are algorithms like block matching, semi-global block matching, graph-cut etc.), background subtraction (which can be done using mixture-of-gaussians models, codebook-based algorithm etc.), optical flow (block matching, Lucas-Kanade, Horn-Schunck etc.).

The class provides the following features for all derived classes:

- So called "virtual constructor". That is, each Algorithm derivative is registered at program start and you can get the list of registered algorithms and create instance of a particular algorithm by its name (see Algorithm::create). If you plan to add your own algorithms, it is good practice to add a unique prefix to your algorithms to distinguish them from other algorithms.
- Setting/retrieving algorithm parameters by name. If you used video capturing functionality from OpenCV highgui module, you are probably familiar SetCaptureProperty(), cvGetCaptureProperty(), VideoCapture::set) andVideoCapture::get(). Algorithm provides similar method where instead of integer id's you specify the parameter names as text strings. See Algorithm::set and Algorithm::get for details.
- Reading and writing parameters from/to XML or YAML files. Every Algorithm derivative can store all its parameters and then read them back. There is no need to re-implement it each time.

Creating own algorithm

If you want to make your own algorithm, derived from Algorithm, you should basically follow a few conventions and add a little semi-standard piece of code to your class:

- Make a class and specify Algorithm as its base class.
- The algorithm parameters should be the class members. See Algorithm::get() for the list of possible types of the parameters.
- Add public virtual method AlgorithmInfo* info() const; to your class.
- Add constructor function, AlgorithmInfo instance and implement the info() method.

- Add some public function (e.g. initModule_<mymodule>()) that calls info() of your algorithm and put it into the same source file as info() implementation. This is to force C++ linker to include this object file into the target application. SeeAlgorithm::create() for details.

Data type

Template “trait” class for OpenCV primitive data types. A primitive OpenCV data type is one of unsigned char, bool, signed char, unsignedshort, signed short, int, float, double, or a tuple of values of one of these types, where all the values in the tuple have the same type. Any primitive type from the list can be defined by an identifier in the form CV_<bit-depth>{U|S|F}C(<number_of_channels>), for example: uchar ~CV_8UC1, 3-element floating-point tuple ~ CV_32FC3, and so on. A universal OpenCV structure that is able to store a single instance of such a primitive data type is **Vec**. Multiple instances of such a type can be stored in a std::vector, Mat, Mat_, SparseMat, SparseMat_, or any other container that is able to store Vec instances.

Ptr

Class Ptr

Template class for smart reference-counting pointers. This class provides the following options:

- Default constructor, copy constructor, and assignment operator for an arbitrary C++ class or a C structure. For some objects, like files, windows, mutexes, sockets, and others, a copy constructor or an assignment operator are difficult to define. For some other objects, like complex classifiers in OpenCV, copy constructors are absent and not easy to implement. Finally, some of complex OpenCV and your own data structures may be written in C. However, copy constructors and default constructors can simplify programming a lot.Besides, they are often required (for example, by STL containers). By wrapping a pointer to such a complex object TObj to Ptr<TObj>, you automatically get all of the necessary constructors and the assignment operator.
- *O(1)* complexity of the above-mentioned operations. While some structures, like std::vector, provide a copy constructor and an assignment operator, the operations may take a considerable amount of time if the data structures are large. But if the structures are put intoPtr<>, the overhead is small and independent of the data size.

- Automatic destruction, even for C structures. See the example below with FILE*.
- Heterogeneous collections of objects. The standard STL and most other C++ and OpenCV containers can store only objects of the same type and the same size. The classical solution to store objects of different types in the same container is to store pointers to the base classbase_class_t* instead but then you loose the automatic memory management. Again, by using Ptr<base_class_t>() instead of the raw pointers, you can solve the problem.

The Ptr class treats the wrapped object as a black box. The reference counter is allocated and managed separately. The only thing the pointer class needs to know about the object is how to deallocate it. This knowledge is encapsulated in the Ptr::delete_obj() method that is called when the reference counter becomes 0. If the object is a C++ class instance, no additional coding is needed, because the default implementation of this method calls delete obj;. However, if the object is deallocated in a different way, the specialized method should be created.

Mat

OpenCV n-dimensional dense array class. The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout in Mat is fully compatible with CvMat, IplImage, and CvMatND types from OpenCV 1.x. It is also compatible with the majority of dense array types from the standard toolkits and SDKs, such as Numpy (ndarray), Win32 (independent device bitmaps), and others, that is, with any array that uses *steps* (or *strides*) to compute the position of a pixel. Due to this compatibility, it is possible to make a Mat header for user-allocated data and process it in-place using OpenCV functions.

Matrix Expressions

This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)):

- Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A

- Scaling: A*alpha
- Per-element multiplication and division: A.mul(B), A/B, alpha/A
- Matrix multiplication: A*B
- Transposition: A.t() (means A^T)
- Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems:

  A.inv([method]) (~ A^{-1}) , A.inv([method])*B (~ X: AX=B)
- Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of : >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0.
- Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of: &, |, ^.
- Element-wise absolute value: abs(A)
- Cross-product, dot-product: A.cross(B) A.dot(B)
- Any function of matrix or matrices and scalars that returns a matrix or a scalar, such as norm, mean, sum, countNonZero, trace,determinant, repeat, and others.
- Matrix initializers (Mat::eye(), Mat::zeros(), Mat::ones()), matrix comma-separated initializers, matrix constructors and operators that extract sub-matrices (see **Mat** description).
- Mat_<destination_type> () constructors to cast the result to the proper type.

Input Array

class **InputArray**

This is the proxy class for passing read-only input arrays into OpenCV functions.

Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind:

- When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc.
- Optional input arguments: If some of the input arrays may be empty, pass cv::noArray().
- The class is designed solely for passing parameters. That is, normally you *should not* declare class members, local and global variables of this type.
- If you want to design your own function or a class method that can operate of arrays of multiple types, you can use InputArray(or OutputArray) for the respective parameters. Inside a function you should use _InputArray::getMat() method to construct a matrix header for the array (without copying data). _InputArray::kind() can be used to distinguish Mat from vector<> etc., but normally it is not needed.

Output Array

class **OutputArray**

This type is very similar to InputArray except that it is used for input/output and output function parameters. Just like with InputArray, OpenCV users should not care about OutputArray, they just pass Mat, vector<T> etc. to the functions. The same limitation as for InputArray: Do not explicitly create OutputArray instances applies here too. If you want to make your function polymorphic (i.e. accept different arrays as output parameters),it is also not very difficult. Note that _OutputArray::create() needs to be called before _OutputArray::getMat(). This way you guarantee that the output array is properly allocated. Optional output parameters. If you do not need certain output array to be computed and returned to you, pass cv::noArray(), just like you would in the case of optional input array. At the implementation level, use _OutputArray::needed() to check if certain output array needs to be computed or not.

NAryMatIterator

class **NAryMatIterator**

Use the class to implement unary, binary, and, generally, n-ary element-wise operations on multi-dimensional arrays. Some of the arguments of an n-ary function may be continuous arrays, some may be not. It is possible to use conventional MatIterator‘s for each array but incrementing all of the iterators after each small operations may be a big overhead. In this case consider using NAryMatIterator to iterate through several matrices simultaneously as

long as they have the same geometry (dimensionality and all the dimension sizes are the same). It iterates through the slices (or planes), not the elements, where “slice” is a continuous part of the arrays. On each iteration it.planes[0], it.planes[1] , ... will be the slices of the corresponding matrices.

SparceMat

class **SparseMat**

The class SparseMat represents multi-dimensional sparse numerical arrays. Such a sparse array can store elements of any type that Mat can store. Sparse means that only non-zero elements are stored (though, as a result of operations on a sparse matrix, some of its stored elements can actually become 0. It is up to you to detect such elements and delete them using SparseMat::erase). The non-zero elements are stored in a hash table that grows when it is filled so that the search time is O(1) in average (regardless of whether element is there or not).

3.4 Platform

- **Linux**

FIGURE 3:LINUX

Compatibility: > OpenCV 2.0

FIGURE 4:GCC

Using OpenCV with gcc and CMake

Compatibility: > OpenCV 2.0

FIGURE 5:CDT

Using OpenCV with Eclipse (plugin CDT)

Compatibility: > OpenCV 2.0

- **Windows**

FIGURE 6:WINDOWS

FIGURE 7:VISUAL STUDIO

B*uild applications with OpenCV inside the Microsoft Visual Studio*

Compatibility: > OpenCV 2.0

- **Desktop Java**

FIGURE 8:JAVA

Compatibility: > OpenCV 2.4.4

FIGURE 9:ECLIPSE

Using OpenCV Java with Eclipse

Compatibility: > OpenCV 2.4.4

- **Android**

FIGURE 10:ANDROID

Compatibility: > OpenCV 2.4.2

- **iOS**

FIGURE 11:IOS

Compatibility: > OpenCV 2.4.2

FIGURE 12:GCC

Cross compilation for ARM based Linux systems

Compatibility: > OpenCV 2.4.4

3.5 Setup Development Environment

Using OpenCV Java with Eclipse

Since version 2.4.4 OpenCV supports Java. We will define OpenCV as a user library in Eclipse, so we can reuse the configuration for any project. Launch Eclipse and select *Window –> Preferences* from the menu.

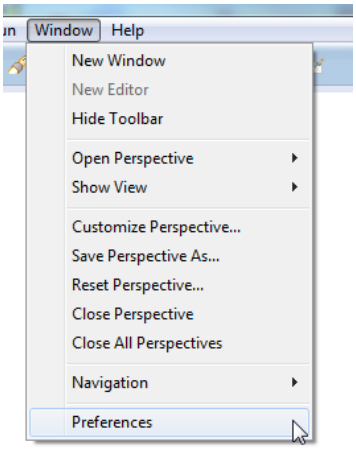

FIGURE 13:OPEN WINDOW

Navigate under *Java –> Build Path –> User Libraries* and click *New...*

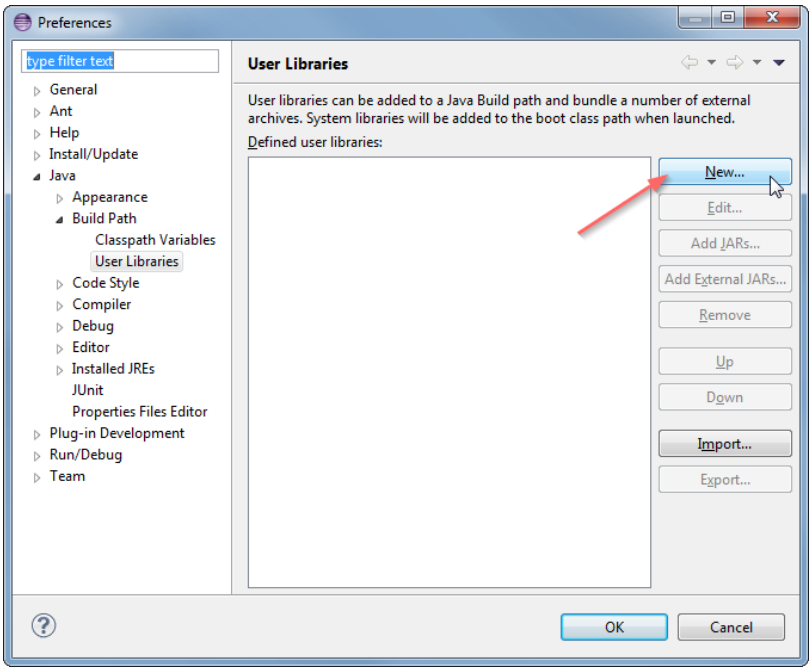

FIGURE 14:GO TO NEW

Enter a name, e.g. OpenCV-2.4.6, for your new library.

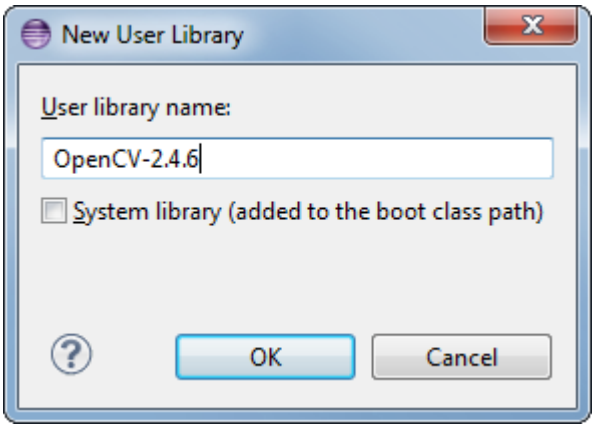

FIGURE 15:ENTER THE NAME OF LIBRARY

Now select your new user library and click *Add External JARs...*.

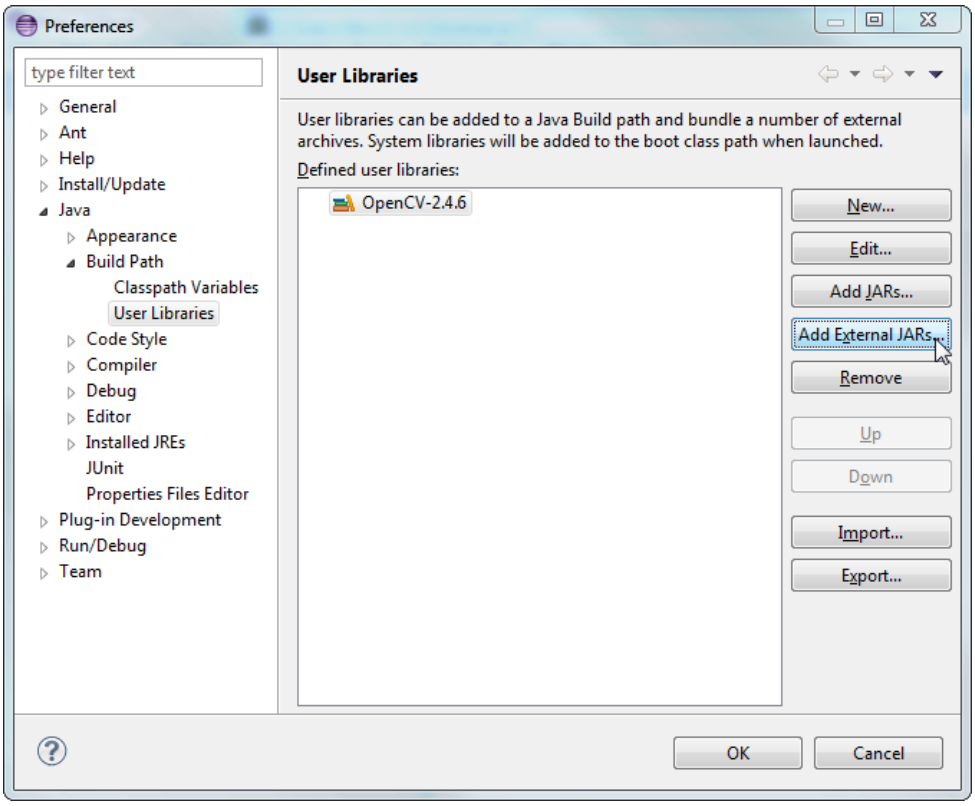

FIGURE 16:CLICK ON EXTERNAL JAR

Browse through C:\OpenCV-2.4.6\build\java\ and select opencv-246.jar. After adding the jar, extend the *opencv-246.jar* and select *Native library location* and press *Edit...*.

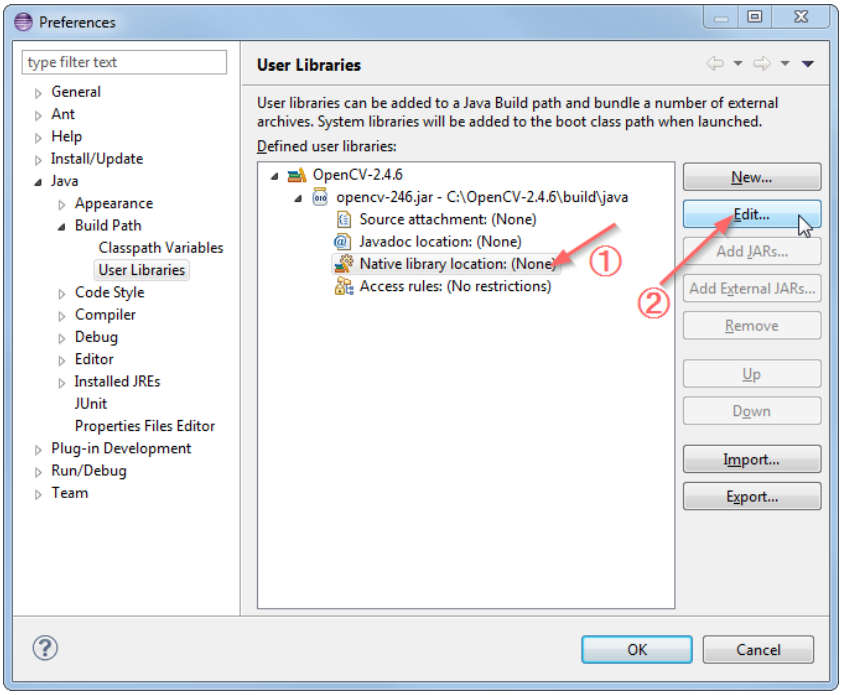

FIGURE 17:CLICK ON EDIT

Select *External Folder*... and browse to select the folder C:\OpenCV-2.4.6\build\java\x64. If you have a 32-bit system you need to select the x86 folder instead of x64.

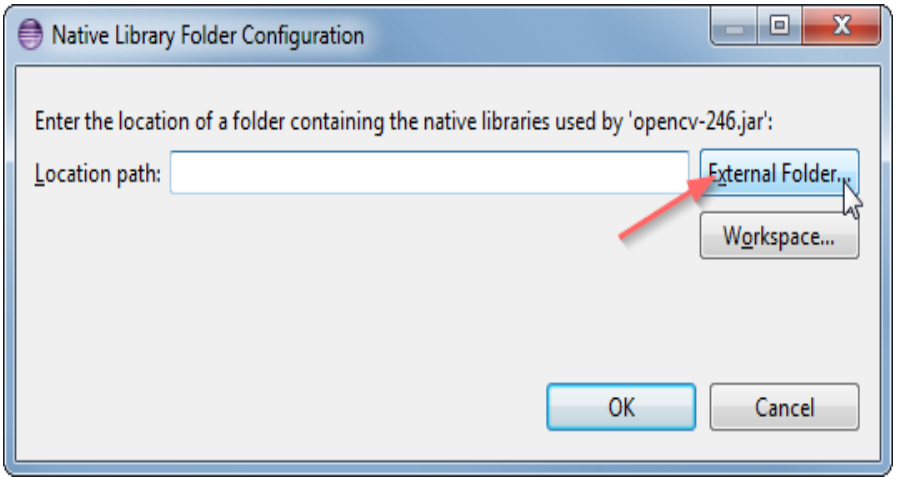

FIGURE 18:SET EXTERNAL PATH

Your user library configuration should look like this:

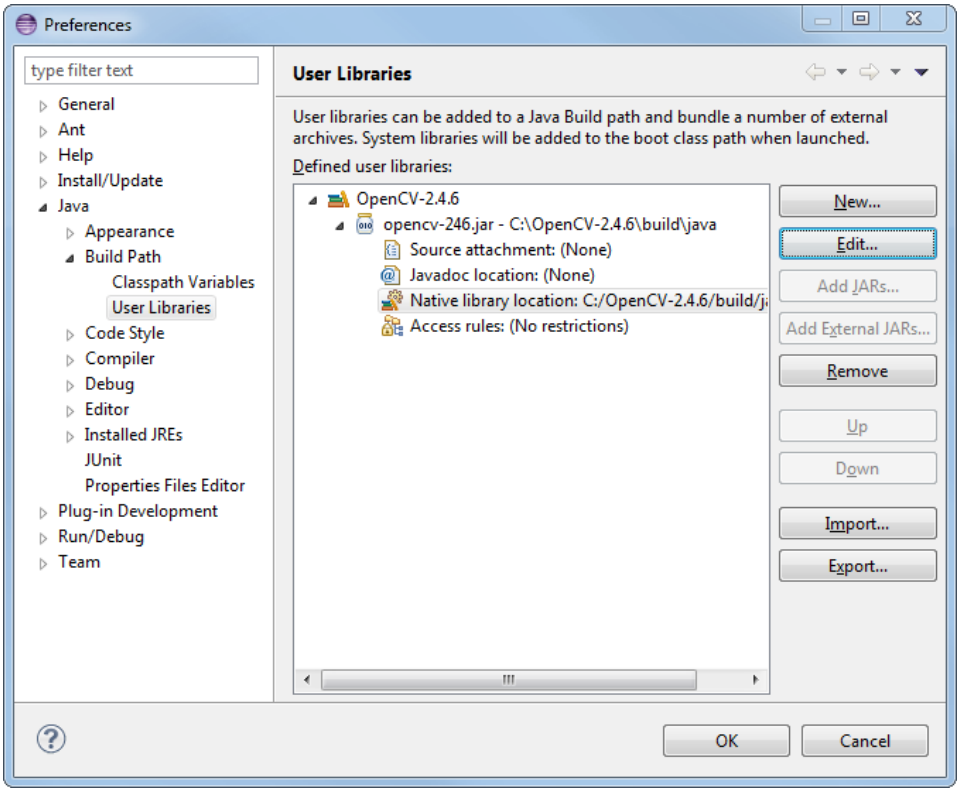

FIGURE 19:USER LIBRARY

Now start creating a new Java project.

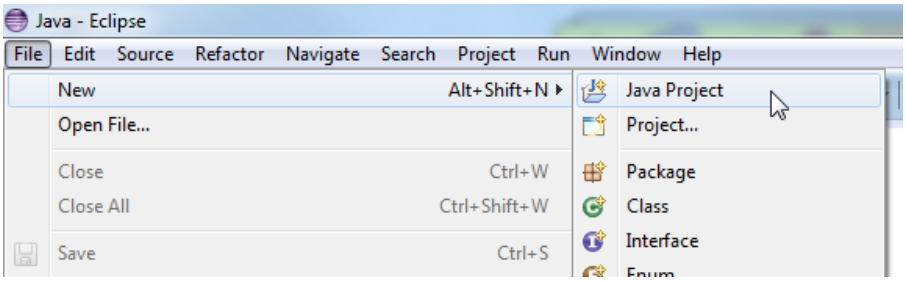

FIGURE 20:OPEN JAVA PROJECT

On the *Java Settings* step, under *Libraries* tab, select *Add Library...* and select *OpenCV-2.4.6*, then click *Finish*.

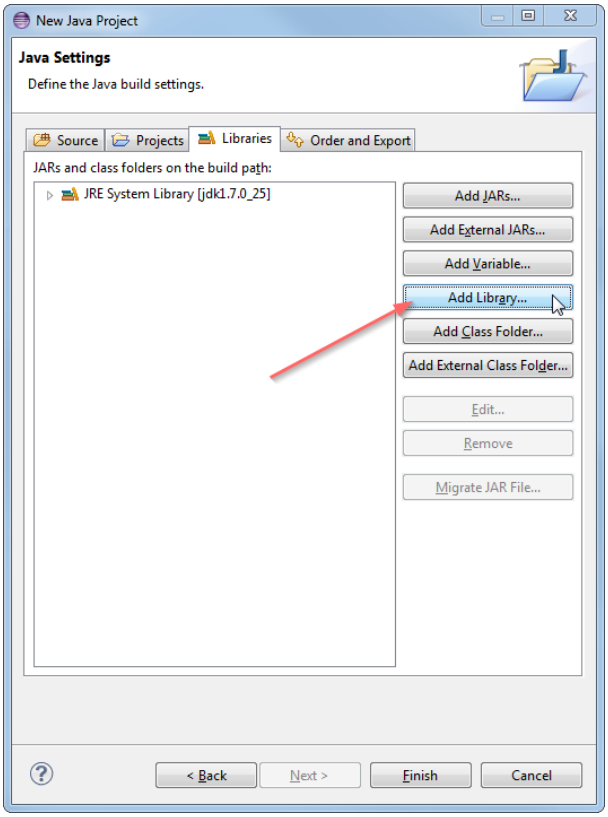

FIGURE 21:ADD EXTERNAL LIBRARY

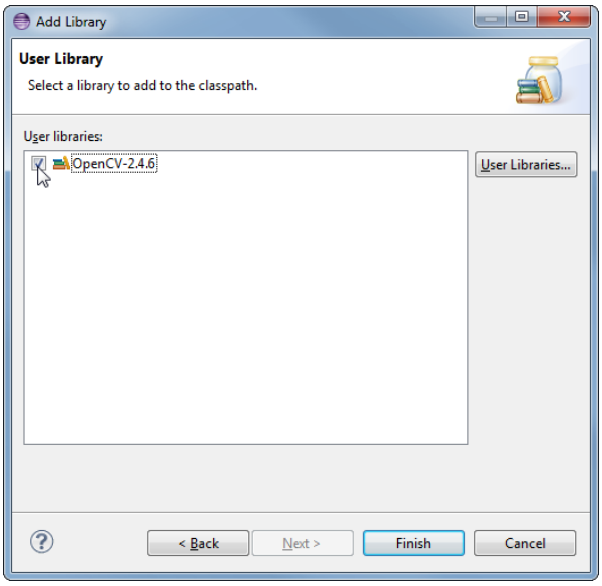

FIGURE 22ADD OPENCV LIBRARY

Libraries should look like this:

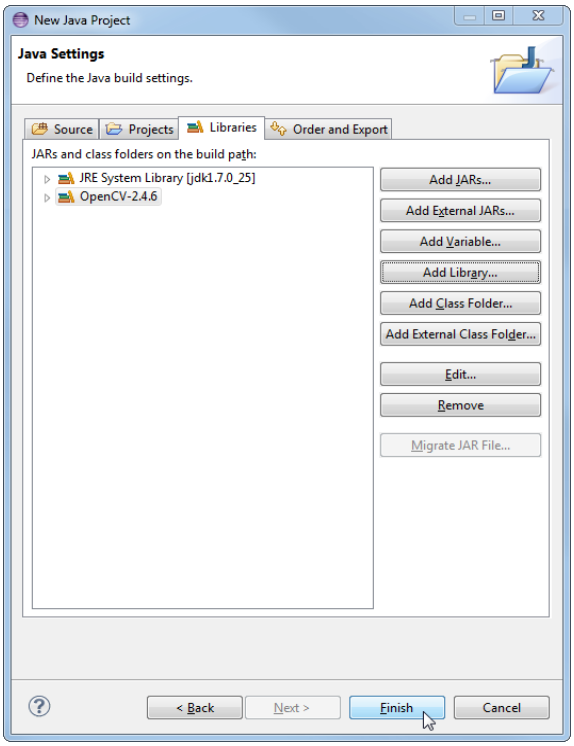

FIGURE 23

Now have created and configured a new Java project it is time to test it. Create a new java file.

import org.opencv.core.Core;

import org.opencv.core.CvType;

import org.opencv.core.Mat;

public class Hello

{

  public static void main(String[] args)

  {

    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

    Mat mat = Mat.eye(3, 3, CvType.CV_8UC1);

    System.out.println("mat = " + mat.dump());

  }

}

When you run the code you should see 3x3 identity matrix as output

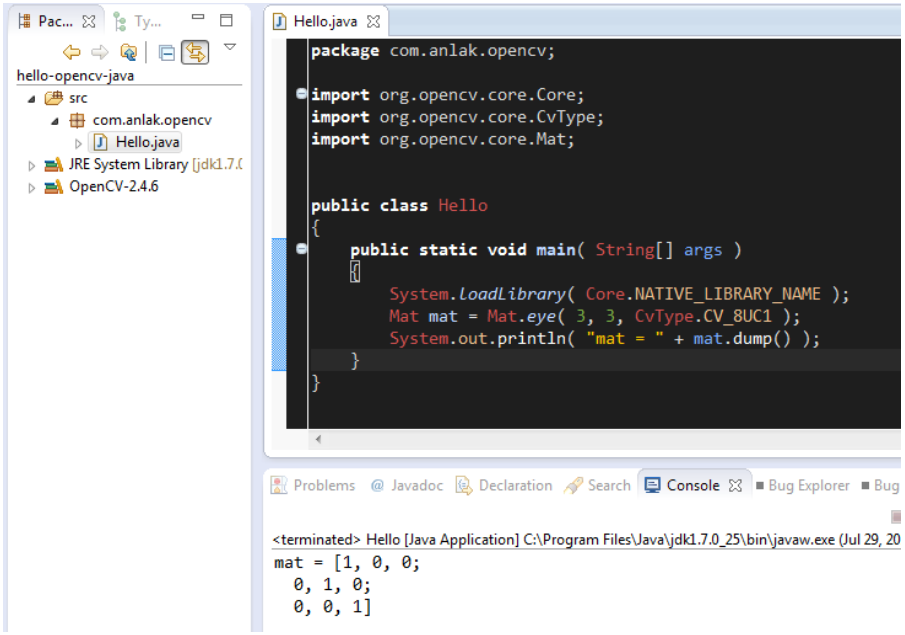

FIGURE 24:OUPUT OF CODE

Chapter 4 Image Edge Detection based on OpenCV

4. Image Edge Detection Based on OpenCV

4.1 Introduction

The rapid development of computer industry production and computer intelligence, as well as the corresponding developments in computer-aided image analysis, has made industrial image processing to be a very important branch of scientific image processing. Image processing plays a very important role in industrial production, which can visualize the anatomical structure of the product, therefore, we can check and judge the merits of the product in time and to some extent reduce the unnecessary losses, so it is necessary for us to avoid the traditional off-line manual detection methods which may easily result in error detection.

4.2Working on Image

4.2.1Capture an image

In OpenCV, the data type of image is usual IplImage. IplImage comes from Intel Image Processing Library. Image Processing Library is inheritance from the actual OpenCV versions which may require IplImage data type is defined in CXCORE . IplImage is the main image structure used in OpenCV. IplImage has been in OpenCV since the very beginning. It is a part of the C interface for OpenCV. You need to allocate and deallocate memory for Iplmage structures yourself. OpenCV has many powerful image processing functions. In order to capture and show an image we should use cvLoadImage cvNamedWindow, and cvShowImage functions. The function cvLoadImage loads the specified image file and return IplImage pointer to the file. CvNameWindown defines a window for displaying. The function cvShowImage display the image in the specified window. By using the above three functions, we can successfully display the image that we will deal with.

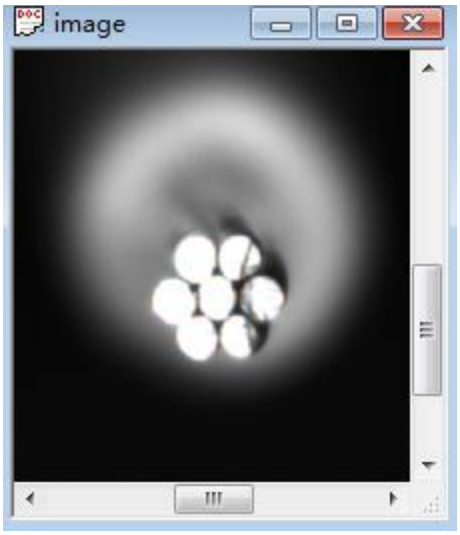

FIGURE 25:SOURCE IMAGE EDGE IS NOT CLEAR

4.2.2Color to Gray

Opencv offers us cvCvtColor function to convert the color image to grayscale image. The effect of RGB to GrayScale image is well demonstrated in the Fig.

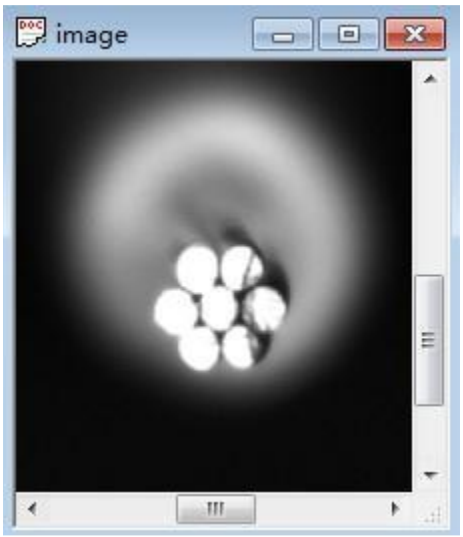

FIGURE 26:RGB TO GRAYSCALE IMAGE

4.2.3Median filter

At the same time, because of the noise interfering to the detection accuracy, it is necessary for us to do denoising for the captured images. OpenCV includes a Median filter that can be applied to an image by calling the cvSmooth function. The image after smooth processing is as Fig.

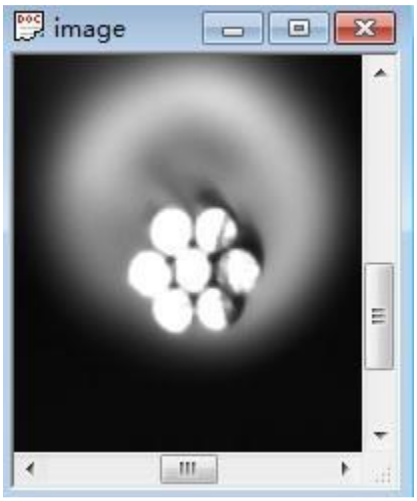

FIGURE 27:AFTER MEDIAN FILTER

4.2.4 Thresholding

Thresholding or binarization is a conversion from a color image to a bi-level one. This is the first step in several image processing applications. This process can be understood as a classification between objects and background in an image. we use Opencv's cvThreshold function.

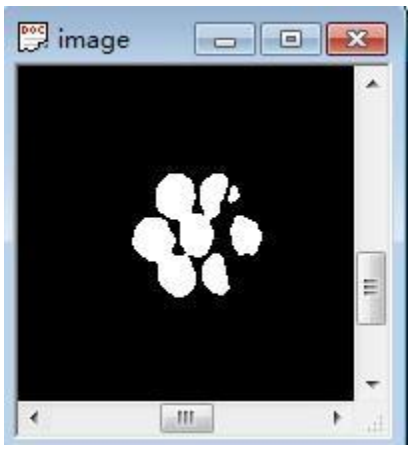

FIGURE 28:THE IMAGE AFTER THRESHOLDING

4.3 MORPHOLOGICAL PROCESSING

Mathematical morphology examines the geometrical structure of an image by probing it with small patters, called ‘structuring elements’ of varying size and shape. This procedure results in nonlinear image operators which are well-suited to exploring geometrical and topological structures. A succession of such operators is applied to an image in order to make certain features apparent, and distinguish meaningful information from irrelevant distortions, by reducing it to a sort of caricature.

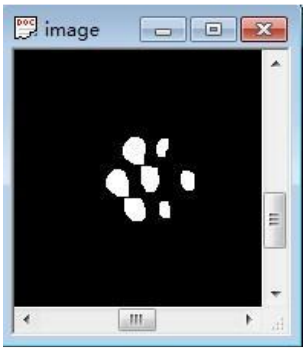

FIGURE 29:MORPHOLOGICAL PROCESSING IMAGE

4.4 CONTOUR TRACKING

The idea of contour tracking, also known as boundary following or edge tracking, is to traverse the border of a region completely and without repetition.The result of the contour tracking is to obtain a boundary points sequence. Opencv offers cvFindContours function to realize contour tracking, which retrives contour from the binary image, and return the number of contour.

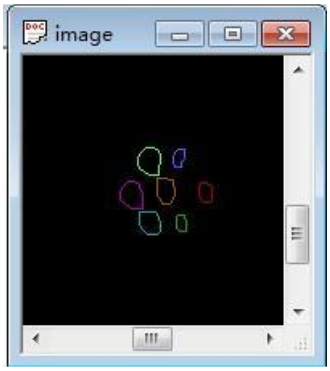

FIGURE 30:THE EFFECT OF CONTOUR TRACKING

Chapter 5
DICE DETECTION

5.DICE DETECTION

5.1 Introduction

We propose an automated detection system with machine vision. Machine vision is a powerful tool and is widely employed in automatic monitoring and detecting processes. Many applications for dice gambling machines using machine vision have been proposed. The Chroma-key principle and the smoothing vectors were used to estimate the location of each die, and a template matching technique was proposed for fine-tuning and detecting the number of spots. However, estimated results depended on the image contrast. Another system which automated the tasks of detection and classification of the dice scores on the playing tables in casinos was based on the online analysis of images captured by a monochrome CCD camera and the spots of dice were extracted. This system includes the diameter of each spots and the distance between each spots on the die surface had to be known before the detection process, a sort of template matching was repeated to classify dice and all spots associated to each die. In addition, an electromechanical dice gambling machine was established to detect dice location based on contactless electronic ID keys and a scanner. As machine vision was not provided during the recognition process, the suspicion of cheating cannot be eliminated.
Because of the above-mentioned issues, a novel auto-recognition system is proposed to estimate the location of each die and the score of dice accurately and effectively in the games using machine vision techniques.

5.2 Materials and Methods

5.2.1 Image acquisition

In order to obtain online images for auto-recognition, a machine vision system was being developed. This system includes a CCD (coupled-charge device) monochrome camera with a zoom lens, a frame grabber, and a personal computer (Intel Pentium 4 processor 2.4 GHz). Basic programming is linked to grab the monochrome images and perform the image processing. The CCD camera was employed for image acquisition with a light intensity of

5.2.2 Image processing and apply algorithm for finding spot of dices

Segmenting the spot images of dice is an important procedure in the classification process. The spot images are extracted by using the global thresholding, hole-filling, closing, and opening operators the location of each dice can be obtained according to the images of spots. This geometric characteristic would be treated as the reference data to recognize the spot number.

FIGURE 31: ORIGINAL IMAGE.

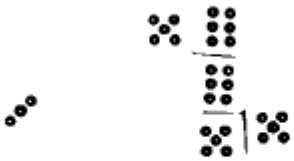

FIGURE 32: BINARY IMAGE.

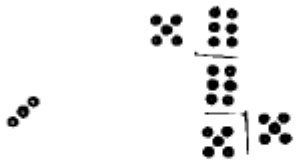

FIGURE 33: AFTER HOLE-FILLING.

FIGURE 34: AFTER CLOSING AND OPENING.

Steps for recognizing the dice are:

1. Auto-acquire Image
2. Extract the spots of dices
3. Compute the coordinates of spots of dices
4. Apply supervised algorithm for detecting the number of spot on dices
5. Obtain score of dices.

5.3 Supervised Algorithm for detecting the spot on Dice

Step for finding the Dice spots are:

1. Find all the squares type of objects on the image using canny method. The threshold is being changed multiple times to make sure all quadrangles are found.

2. Delete all objects that bound similar area. This is made because during step 1 usually each quadrangle is replicated few time.

3. Delete all the pixels from the outside of the chosen color ranges. Chosen color range depend on the color of spot on the dice. Color of spot of dice is entered manually by user. So pixel other than spot are deleted.

4. Make a list of blobs that were not rejected.

5. Check if a blob shape is similar to circle. Also their square area must be between preset boundaries.

6. Count the number of blobs which are present in Square Boundary - that's the number on the dice.

Step 1: Find all the squares using canny method.

1. First you split the image to R, G, and B planes.

2. Then for each plane perform edge detection, and in addition to that, **threshold** for different values like 50, 100, Etc.

3. And in all these binary images, find contours in the images.

4. After finding contours, remove some small unwanted noises by filtering according to area.

5. For a rectangle, it will give you the four corners. For others, corresponding corners will be given. So filter these contours with respect to number of elements in approximated contour that should be four, which is same as number of corners. **First property of rectangle.**

6. Next, there may be some shapes with four corners but not rectangles. So we take second property of rectangles, i.e. all inner angles are 90.

Canny Function for Edge Detection:

Void Canny(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3).

Parameters:

image – single-channel 8-bit input image.
edges – output edge map; it has the same size and type as image .
threshold1 – first threshold for the hysteresis procedure.
threshold2 – second threshold for the hysteresis procedure.
apertureSize – aperture size .

Step 2: Filter the image according to the color of the spot of dice.

For this we use Core.inRange ( ) function.

In this,

public static void inRange(Mat src,
Scalar lowerb,
Scalar upperb,
Mat dst)

The function checks the range as follows:

- For every element of a single-channel input array:

dst(I)= lowerb(I)_0 <= src(I)_0 <= upperb(I)_0

- For two-channel arrays:

dst(I)= lowerb(I)_0 <= src(I)_0 <= upperb(I)_0 land lowerb(I)_1 <= src(I)_1 <= upperb(I)_1

- and so forth.

That is, dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D,... box and 0 otherwise.
When the lower and/or upper boundary parameters are scalars, the indexes (I) at lowerb and upperb in the above formulas should be omitted.

Parameters:

src - first input array.

lowerb - inclusive lower boundary array or a scalar.

upperb - inclusive upper boundary array or a scalar.

dst - output array of the same size as src and CV_8U type.

Step 3: Make a list of Blob and check if blob is similar to circle.

For this we Hough Circle Transform function.

HoughCircles(InputArray **image**, OutputArray **circles**, int **method**, double **dp**, double **minDist**, double **param1**, double**param2**, int **minRadius**, int **maxRadius**)

Parameters:

Image – 8-bit, single-channel, grayscale input image.

Circles – Output vector of found circles. Each vector is encoded as a 3-element floating-point vector .

Circle storage –This is a memory storage that will contain the output sequence of found circles.

Method – Detection method to use. Currently, the only implemented method is CV_HOUGH_GRADIENT.

dp – Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height.

minDist – Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.

param1 – First method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the higher threshold of the two passed to the Canny() edge detector (the lower one is twice smaller).

param2 – Second method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.

minRadius – Minimum circle radius.

maxRadius – Maximum circle radius.

For finding number of Spot in dices we use findContour() fuction.

void findContours(Mat **image**,
contours,
Mat **hierarchy**,
int **mode**,
int **method**)

image - Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary. You can use "compare", "inRange", "threshold", "adaptiveThreshold", "Canny", and others to create a binary image out of a grayscale or color one. The function modifies the image while extracting the contours. If mode equals to CV_RETR_CCOMP or CV_RETR_FLOODFILL, the input can also be a 32-bit integer image of labels (CV_32SC1).

contours - Detected contours. Each contour is stored as a vector of points.

hierarchy - Optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0], hiearchy[i][1], hiearchy[i][2], and hiearchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements ofhierarchy[i] will be negative.

mode - Contour retrieval mode

- CV_RETR_EXTERNAL retrieves only the extreme outer contours. It sets hierarchy[i][2]=hierarchy[i][3]=-1 for all the contours.
- CV_RETR_LIST retrieves all of the contours without establishing any hierarchical relationships.
- CV_RETR_CCOMP retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.

- CV_RETR_TREE retrieves all of the contours and reconstructs a full hierarchy of nested contours. This full hierarchy is built and shown in the OpenCV contours.c demo.
  method - Contour approximation method (if you use Python see also a note below).
- CV_CHAIN_APPROX_NONE stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, max(abs(x1-x2),abs(y2-y1))==1.
- CV_CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
- CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS applies one of the flavors of the Teh-Chin chain approximation algorithm.

Kmeans function is used for detecting groups for spots of dices.

Kmeans function is given as:

kmeans(InputArray **data**,

int **K**,

InputOutputArray **bestLabels**,

TermCriteria **criteria**,

int **attempts**,

int **flags**,

OutputArray**centers**=noArray())

Parameters:

samples – Floating-point matrix of input samples, one row per sample.

data – Data for clustering.

cluster_count – Number of clusters to split the set by.

K – Number of clusters to split the set by.

labels – Input/output integer array that stores the cluster indices for every sample.

criteria – The algorithm termination criteria, that is, the maximum number of iterations and/or the desired accuracy. The accuracy is specified as criteria.epsilon. As soon as each of the cluster centers moves by less thancriteria.epsilon on some iteration, the algorithm stops.

termcrit – The algorithm termination criteria, that is, the maximum number of iterations and/or the desired accuracy.

attempts – Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness (see the last function parameter).

rng – CvRNG state initialized by RNG().

flags –Flag that can take the following values:

KMEANS_RANDOM_CENTERS Select random initial centers in each attempt.

KMEANS_PP_CENTERS Use kmeans++ center initialization by Arthur and Vassilvitskii .

KMEANS_USE_INITIAL_LABELS During the first (and possibly the only) attempt, use the user-supplied labels instead of computing them from the initial centers. For the second and further attempts, use the random or semi-random centers. Use one of KMEANS_*_CENTERS flag to specify the exact method.

centers – Output matrix of the cluster centers, one row per each cluster center.

_centers – Output matrix of the cluster centers, one row per each cluster center.

compactness – The returned value that is described below.

The function kmeans implements a k-means algorithm that finds the centers of cluster_count clusters and groups the input samples around the clusters. As an output, $lables_i$ contains a 0-based cluster index for the sample stored in the i^{th} row of the samples matrix.

The function returns the compactness measure that is computed as

$$\sum_i ||samplesample_i - centers_{labels_i}||^2$$

After every attempt, the best (minimum) value is chosen and the corresponding labels and the compactness value are returned by the function. Basically, you can use only the core of the function, set the number of attempts to 1, initialize labels each time using a custom algorithm, pass them with the (flags = KMEANS_USE_INITIAL_LABELS) flag, and then choose the best (most-compact) clustering.

6. Conclusion and Future Work

6.1 Conclusion and Future Work

Recognition technology has come a long way in the last twenty years. Today, machines are able to automatically verify identity information for secure transactions, for surveillance and security tasks, and for access control to buildings. These applications usually work in controlled environments and recognition algorithms that can take advantage of the environmental constraints to obtain high recognition accuracy. However, next generation recognition systems are going to have widespread application in smart environments, where computers and machines are more like helpful assistants. A major factor of that evolution is the use of neural networks in face recognition. A different field of science that also is very fast becoming more and more efficient, popular and helpful to other applications.

Firstly, we obtain an image of dice using an auto-acquisition method with machine vision. Secondly, the locations of spots on the dice are estimated by the image processing techniques. Finally, an unsupervised grey clustering algorithm is proposed to identify the spot number of each die. In traditional methods, manual detection with human eyes is used to recognize dice, thus we propose a methodology which can estimate the score of dice in dice games accurately and effectively.

In future, we can use this in Statistical analysis. We are going to use the recognizing system in making automated Weldon dice system.

In 1894, Weldon rolled a set of 12 dice 26,306 times. He collected the data in part, to judge whether the differences between a series of group frequencies and a theoretical law, taken as a whole, were or were not more than might be attributed to the chance fluctuations of random sampling.

So by the use automated recognizing system we can decrease the processing time.

We can do more counting in less time.

7. References

7.1 References

[1] Kuo-Yi Huang, "An Auto-Recognizing System for Dice Games Using a Modified Unsupervised Grey Clustering Algorithm", Feb 2008, pp-1212-1221.

[2] Chin-Ho Chung, Wen-Yuan Chen, and Bor-Liang Lin, "Image Identification Scheme for Dice Game", 2009.

[3] *Wen-Yuan Chen, "*Dice Image Recognition Scheme Using Pattern Comparison and Affine Transform Techniques".

[4] John SUM, Jan CHAN, "On **A** Liar Dice Game – BLUFF", Proceedings of the Second International Conference on Machine Learning and Cybernetics, Wan, 2-5 November 2003.

[5] Gee-Sern Hsu, Hsiao-Chia Peng, Shang-Min Yeh, "Color and Illumination Invariant Dice Recognition", IEEE International Conference on Systems, Man, and Cybernetics, COEX, Seoul, Korea, 14-17October, 2012.

[6] http://docs.opencv.org

[7] http://en.wikipedia.org/wiki/OpenCV

[8] http://docs.opencv.org/doc/tutorials/introduction

[9] http://en.wikipedia.org/wiki/Walter_Frank_Raphael_Weldon

[10] http://en.wikipedia.org/wiki/K-means_clustering

[11] https://galton.uchicago.edu/about/docs/2009/2009_**dice**_zac_labby.pdf

[12] https://www.youtube.com/watch?v=8tYbGweDBCM

[13] http://stackoverflow.com/questions/8667818/opencv-c-obj-c-detecting-a-sheet-of-paper-square-detection

[14] http://answers.opencv.org/question/16665/android-opencv-find-square-edges-in-contour-then-getperspectivetransform-and-warpperspective/

[15] http://stackoverflow.com/questions/22295247/using-k-means-clustering-pixel-in-opencv-using-java

[16] http://mnemstudio.org/clustering-k-means-example-1.htm

[17] https://github.com/badlogic/opencv-fun/blob/master/src/pool/tests/Cluster.java

[18]http://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Machine_Learning_Clustering_K-Means_Clustering_Vector_Quantization.php

[19]http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/bounding_rects_circles/bounding_rects_circles.html

[20] http://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/

[21] http://stackoverflow.com/questions/13446374/android-opencv-find-contours

[22] https://eclipse.org/

[23] http://en.wikipedia.org/wiki/Eclipse_(software)

[24] https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html

[25] http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html

[26] http://www.oracle.com/technetwork/java/javase/downloads/index.html

[27] http://www.tutorialspoint.com/eclipse/eclipse_create_java_interface.htm

[28] http://answers.opencv.org/question/6127/core-inrange/

[29] http://stackoverflow.com/questions/13031357/opencv-mat-processing-time