DUBLIN INSTITUTE of TECHNOLOGY

Institiúid Teicneolaíochta Bhaile Átha Cliath

Euro Coin Classification Using Image Processing & Machine Learning

Final Year Project Report

DT228

BSc in Computer Science

Yumin Chen

Cindy Liu

School of Computing

Dublin Institute of Technology

4th April 2017

Abstract

This project aims to investigate the visual features of euro coins and develop models that are able to represent different euro coin denominations and thus be used to accordingly classify coins detected from natural images. The objective of this project is to investigate the use of machine learning specifically in the context of euro coin classification.

In this project, a model-based euro coin classification system was developed to address this question. This system aims to classify euro coins according to their denomination using image processing and machine learning techniques. This system was built by collecting large datasets of euro coin images, extracting features, and data modelling. Different classification algorithms were implemented and evaluated. Results returned in this project suggest that the Normal Bayesian classification method is 28.93% more accurate than the k-Nearest Neighbors (k-NN) method over the tested euro coin datasets.

Keywords: Image Processing, Computer Vision, Object Recognition, Data Mining Classification, Artificial Intelligence, Machine Learning.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Yumin Chen

4th April, 2017

Acknowledgements

I wish to acknowledge my supervisor Cindy Liu for her guidance and support from the proposal stages of the project and throughout.

I also wish to thank Dr. Jane Courtney for her support in the Image Processing area, and Dr. Svetlana Hensman in the Artificial Intelligence and Machine Learning area. Their advice, interest in my work and encouragement has been invaluable.

Finally, I wish to thank Dr. Susan Mckeever for her tireless work coordinating our final year project module.

Table of Contents

Table of Figures

Chapter 1 Introduction

1.1 Overview

This project examines an object recognition problem in computer vision and aims to classify euro coins from natural images based on self-trained models of known euro coin denominations. This project investigates the use of machine learning techniques specifically in the context of model-based object recognition to develop an object classification system for euro coins.

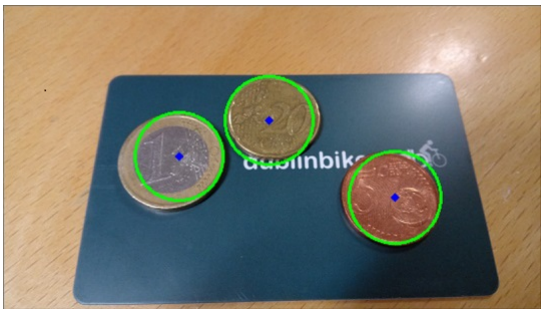


Figure 1: Euro Coin Recognition

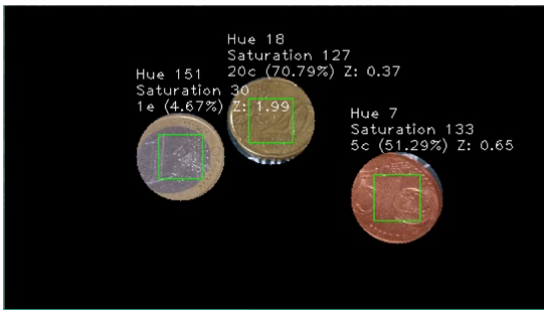


Figure 2: Euro Coin Classification

This is a model-based object recognition project. It addresses the problem of recognizing euro coins from images by means of a suitable mathematical model that is used to describe each denomination of euro coins. In this project, different models of euro coins are built to generalize their geometrical and appearance properties at the appropriate level of specificity [1]. Each model of euro coins retains key visual properties for representing a specific euro coin denomination.

The main technologies involved in this project are image processing and machine learning. For Image Processing, computer vision techniques are used to process the image-based sample dataset and extract features. Machine Learning is used for predictive data analytics to build the models of generalized euro coin denominations. These models contain features that are important in describing and recognizing the object in relation to other objects.

1.2 Objectives

This project aims to replicate in machines the way humans visually perceive and recognize euro coins in natural surroundings and the ability to instantly classify different denominations of euro coins and identify their values. The human eye and brain naturally do an exceptional job understanding and recognizing euro coins. The purpose of this project is to try and implement this task in machines using image processing and machine learning techniques.

The main objective of this project is to research and develop a cross-platform object recognition system that can classify the complete euro coin series of their obverse side (common side). The development of this euro coin classifier involves designing

and building suitable mathematical models that is used to describe all 8 denominations, from one cent up to two euro, as shown below in Figure 3.

Figure 3: The Common Obverse Side of Euro Coins

The detailed objectives are as following:

1) To investigate and research the use of machine learning techniques specifically in the context of euro coin classification.
2) To build suitable mathematical models that retain key visual properties for representing each euro coin denomination.
3) To develop a functioning euro coin recognition system that can classify euro coins according to their denomination and tell their values.
4) To provide sample demo applications, including a web-based euro coin recognition API service, and a companion sample demo mobile app that can demonstrate this euro coin recognition system on mobile devices.

Each objective has its own deliverables which are achieved and delivered separately.

1.3 Challenges

Visual object recognition is one of the most challenging computational problems in machine vision. Humans can easily recognize any euro coins effortlessly and instantaneously. However, emulating such humans' visual abilities in machines can be extremely difficult. To replicate the euro coin recognition task in computer vision requires the construction of an artificial recognition system.

The main challenge in developing this model-based euro coin recognition system is to design models that are capable of retaining key visual properties for representing a euro coin denomination at the appropriate level of abstraction. In this project, each denomination of euro coins contains very similar features in terms of shape, color and texture, etc.

It is tremendously challenging to build suitable mathematical models that are able to precisely describe the subtle differences between different euro coin denominations. The data must be analyzed with statistical methods to determine the relevance of each attribute or feature of the objects, and decisions must be made on what must be captured in these models.

The development also involves extensive research on which features are useful in terms of classifying euro coins. For some objects, geometric descriptions may be available and may also be efficient, while for another class, one may have to rely on generic or functional features. Due to the nature of the image formation process, it is difficult to determine the best approach to extract relevant features effectively and reliably. Some features are easy to compute reliably while others are not.

In this project, the representation of a euro coin class should capture all relevant information about that coin type without any redundancies. Moreover, the structure of the models should also be organized in a form that allows easy access by different components of this euro coin classification system.

Furthermore, computer vision depends on various ambient factors. Any euro coin can cast an infinite number of different images onto the retina, depending on the coin's position, orientation, lighting, etc. The algorithms for euro coin recognition have many limitations because the changes in external illumination conditions, camera, captured image quality, ambient occlusion, scales, positions as well as digitization artifacts, all produce significant variations in the euro coins' appearance. Even with computer vision techniques, such as contrast stretching, image smoothing filters, the euro coin models built in this project may still be defeated by a simple image with a higher range of variation observed in the real world. These are the challenges that can severely affect the performance of this euro coin classifier.

Chapter 2 Research

2.1 Previous Work & Existing Solutions

Euro coin recognition systems have long been in existence. Currently, the majority of coin recognition machines rely on examining the physical properties of coins. The classification of various euro coin denominations and finding the sum of the coins is a tedious process. There are coin counting machines, coin sorting machines and vending machines that can recognize coins by testing physical properties of coins such as size, weight and materials [2].

As for image-based coin recognition systems, there are also existing solutions that can effectively identify coins using different approaches. For example, a rotation-invariant neural network designed by Y. Mitsukura [3] uses a genetic algorithm and simulated annealing to classify Japanese coins.

Besides neural networks, other strategies such as decision trees and Bayesian classifiers are also useful when it comes to coin recognition. Paul Davidsson [4] compares several strategies and derives a decision tree algorithm to accept or to reject the coins.

In regards to the coins in European countries specifically, there is a euro coin recognition and sorting system name Dogobert [5] developed at the ARC Seibersdorf research center which sorted the pre-euro coins from 12 European countries and returned the face value of coins for charity organizations. By incorporating the visual features with traditional features, higher accuracy is achieved by new coin recognition models [6].

Other coin recognition system taking advantage of image processing and machine learning techniques have been developed. A color image based metallic surfaces (coins) recognition system designed by Markus Adameck [7] presented an interesting method of detecting a straight line in a coin's image. In their system, the translational invariance is achieved through segmentation, whereas rotational invariance is a result of a polar coordinate representation and correlation. Their system uses special hardware to ensure that no fraudulent coins are accepted by the system. The criteria for coin classification based on gray-level, color, texture, shape, model, etc., discussed by R.Bremananth [8], has helped to further improve the recognition accuracy and energize the research of coin recognition in recent years.

2.2 Euro Coins

According to the European Central Bank [9], the euro coin series is comprised of eight different denominations: €2, €1, and 50, 20, 10, 5, 2 and 1 cent.

Each euro coin has a common side or 'European' side obverse that is shared throughout the entire Eurozone, and a different national side reverse that is designed by each issuing country in the Eurozone [9].

The Figure 4 below shows the different reverse side designs of the €2 commemorative coins, from which we can see that each country has their own design for their coins' obverse side.

Figure 4: Different Reverse Side Designs of the €2 Commemorative Coins

For simplicity, this project only focuses on the recognition of the common side or 'European' side obverse of the euro coin. Due to the numerous different designs that each European country chooses, there are more than 300 different types of coins that have been produced since the launch of the euro coin series. Thus, attempting to recognize the national side is outside the scope of this project.

	Diameter	***Thickness***	***Weight***	***Color***	***Composition***	***Edge***
2 Euro	25.75 mm	2.20 mm	8.50 g	Silver Gold	Copper-nickel Nickel brass	Edge lettering, fine milled
1 Euro	23.25 mm	2.33 mm	7.50 g	Gold Silver	Nickel brass Copper-nickel	Interrupted milled
50 Cent	24.25 mm	2.38 mm	7.80 g	Gold	Nordic gold	Shaped edge Fine scallops
20 Cent	22.25 mm	2.14 mm	5.74 g	Gold	Nordic gold	Plain
10 Cent	19.75 mm	1.93 mm	4.10 g	Gold	Nordic gold	Shaped edge Fine scallops
5 Cent	21.25 mm	1.67 mm	3.92 g	Copper	Copper-covered steel	Smooth
2 Cent	18.75 mm	1.67 mm	3.06 g	Copper	Copper-covered steel	Smooth with a groove
1 Cent	16.25 mm	1.67 mm	2.30 g	Copper	Copper-covered steel	Smooth

Table 1: Euro Coin Denominations and Their Attributes

The Table 1 above shows the Euro Coin Denominations and Their Attributes. Different denominations of the euro coins have their unique features. The euro coins have incorporated high-security machine-readable characteristics so they can be easily distinguished from one another.

2.3 Object Recognition

2.3.1 Object Recognition Background

Recognizing objects of interest in images has always been an interesting challenge in the realm of computer vision. Humans can understand and recognize objects effortlessly. In recent years, many approaches have been developed to mimic the way our human eyes and brains perceive objects for computer programs to acquire, process, analyze and understand digital images.

Scientists have been striving to effectively mimic the human visual system on computers since the 1960s. The problem was severely underestimated at that time by Marvin Minsky, a professor at MIT, who asked his undergraduate student to "spend the summer linking a camera to a computer and getting the computer to describe what it saw" [10]. It was not until 30 years later that a computer could recognize a human face. Today, computer vision applications include automatically finding faces of your friends on Facebook, Google's robotic cars that autonomously navigate the hectic streets of San Francisco, and NASA's Curiosity rover that will explore Mars over the next decade [11]. In each of these examples, object recognition plays a vital role.

In this project, the main purpose of the research was to find out the process of categorizing the recognized objects into an appropriate class. In the rest of this section, different techniques of category classification will be discussed. There are a number of approaches to solving category recognition, from the simpler *Bag-of-Words* approach, to more complex *Parts-and-Structure* representations and recognition algorithms [12].

2.3.2 Bag-of-Words Models

One of the simplest algorithms for category recognition is the Bag-of-Words (BoW) model. The BoW approach attempts to represent objects and images as unordered collections of feature descriptors [13]. This model is basically a histogram representation based on independent features.

The BoW approach simply computes the distribution (histogram) of visual "words" (features) found in the query image and compares this distribution to those found in the training images. In the BoW model, the image can be treated a "document", and then the algorithm tries to search for "words", or features in the image to define this "document". This is usually achieved with the following 3 steps: feature detection, feature description, and codebook generation [14].

The BoW is simple, but is notoriously known for its disadvantages – It ignores the spatial relationships among the patches, which are very important in image representation. Researchers have proposed several methods to incorporate the spatial information. The following *Parts-and-Structure* models is one of them. [15]

2.3.3 Parts-and-Structure Models

The Parts-and-Structure models recognize an object by finding its constituent parts and measuring their geometric relationships [15]. This is one of the oldest approaches to object recognition.

The part-based models use various parts of the image separately to determine if and where an object of interest exists. It focuses on the representation of geometric relationships, the representation of individual parts, and algorithms for learning such descriptions and recognizing them at run time [10].

The part-based models are built on the original idea of Fischler and Elschlager [16] of using the relative position of a few template matches and evolve in complexity in the work of Perona and others [12].

Other part-based model examples such as discriminative training of part appearance model designed by Felzenszwalb [17] offers much better performance.

2.3.4 Classifier-based Methods

A classifier is the combination of a feature and its corresponding learned threshold value. A feature applied to an image sample results in a value that gets compared to a threshold. [11] Anything below that threshold is negative (i.e., no detection), and anything above it is positive, e.g., the classifier decides the object is present in the sample image.

Strong classifiers are then arranged in a cascade. [18] Commonly, they are arranged according to their number of weak classifiers, ascending in order. A sample must pass inspection by every strong classifier in the cascade before being classified as positive. A negative classification at any point in the cascade results in a final negative classification.

Modern object recognition has made large advancements as the machine learning techniques have been incorporated. The following are some of the popular features for training the cascade classifier [19].

2.3.4.1 Haar-like Features

In 2001, Paul Viola and Michael Jones invented an efficient algorithm for face recognition [19]. In their article, they described a visual object detection framework that was faster, more accurate, and more robust than anything at the time. Their Haar feature-based cascade classifiers that showed faces being detected in real time on a webcam feed was the most stunning demonstration of computer vision and its potential at that time. This framework was duly well-received, and since then, many object detection algorithms, as well as many action detection and recognition

algorithms, have been built off of the Viola-Jones system.

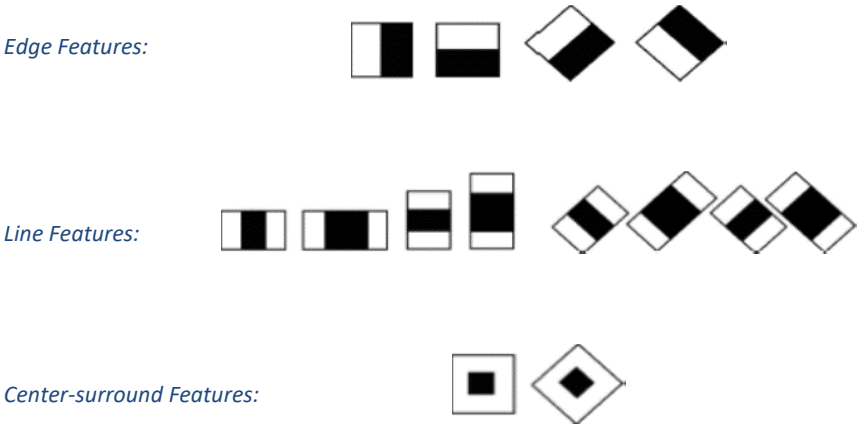


Figure 5: Haar-like Features

2.3.4.2 Local Binary Patterns Features

Local Binary Patterns (LBP) is a type of visual descriptor used for classification in computer vision. The LBP features based cascade classifier invented by Shengcai Liao [20] came to the forefront in 2007. It has since been found to be a powerful feature for texture classification. The LBP features are integer in contrast to Haar features, so both training and detection with LBP can be performed several times faster than with Haar features while providing almost the same quality as Haar-based ones.

2.3.5 Model-based Object Recognition Systems

A model-based object recognition system finds objects in the real world from an image of the world, using object models which are already known.

In model-based object recognition [1], an object model is typically defined so as to capture object's geometrical and appearance properties at the appropriate level of specificity. For instance, an object model can be designed to recognize a generic "face" as opposed to "someone's face" or vice versa. In the former case, which is often referred to as the object categorization problem, the main challenge is to design models that are capable of retaining key visual properties for representing an object category, such as a "face," at the appropriate level of abstraction.

The object recognition problem can be defined as a labeling problem based on models of known objects. Formally, given an image containing one or more objects of interest (and background) and a set of labels corresponding to a set of models known to the system, the system should assign correct labels to regions, or a set of regions, in the image [21].The object recognition problem is closely tied to the segmentation problem: without at least a partial recognition of objects, segmentation cannot be done, and without segmentation, object recognition is not possible.

2.4 Image Processing

Image processing is the analysis and manipulation of a digitized image. In imaging science, image processing is a type of signal dispensation to extract some useful information from the input data using mathematical operations. [22]

The purpose of image processing in this project is to aid the euro coin recognition process. Multiple computer vision techniques are used in this project to process the image so as to classify the euro coins.

2.4.1 Normalization

Normalization, aka contrast stretching, is an image enhancement technique that attempts to improve an image by stretching the range of intensity values it contains to make full use of possible values. [23]

The intensity variation of the original image is widened using a simple linear transform:

$$I_O = \frac{I_i - I_{min}}{I_{max} - I_{min}}$$

Equation 1: Image Normalization

Contrast Stretching increases the dynamic range of the grey level in the image being processed. [22] The idea behind the contrast stretching is stretching the histogram so that the color levels of widely populated areas be separated.

Before Normalization *After Normalization*

Figure 6: Normalization (Contrast Stretching)

In this machine learning based object recognition project, the consistency of the data range is vital, because the statistical techniques used to produce the classifier is very sensitive to noise and outliers. [24] The purpose of dynamic range expansion is to bring the image into a range that is more familiar or normal to the senses, hence the term normalization. The motivation is to achieve consistency in dynamic range for a set of data, signals, or images to avoid mental distraction or fatigue.

2.4.2 Thresholding

Thresholding is the simplest method of image segmentation. It is used to segment an image by setting all pixels whose intensity values are above a threshold to a foreground value and all the remaining pixels to a background value. [25]

Some of the most common thresholding methods are global thresholding and adaptive (dynamic) thresholding, as shown below in Figure 7: Thresholding.

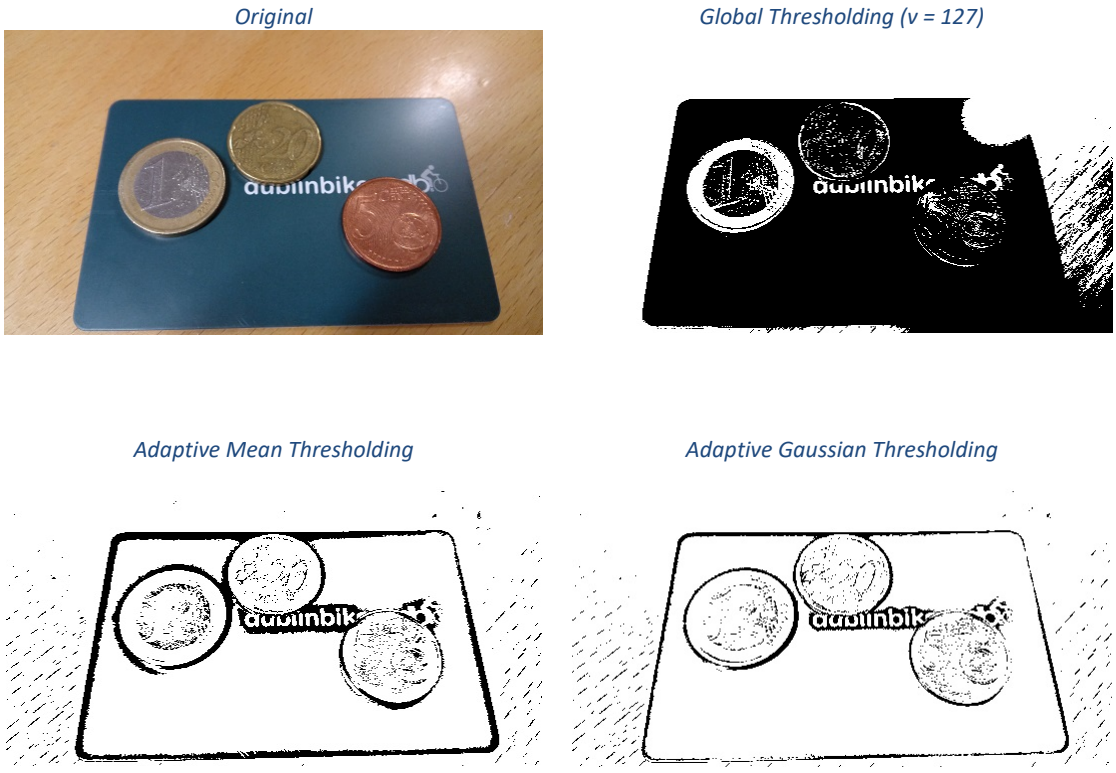


Figure 7: Thresholding

Global Thresholding replaces each pixel in an image with a black pixel if the image intensity $I_{i,j}$ is less than some fixed constant T (that is, $I_{i,j} < T$), or a white pixel if the image intensity is greater than that constant.

Adaptive thresholding, on the other hand, has an algorithm that calculates the threshold for a small region of the image and dynamically changes the threshold over the image. [26] Therefore, for different regions of the same image, this method uses different thresholds and thus produces better results for images with varying illumination.

2.4.3 Gaussian Blur

The Gaussian blur is a type of image-blurring filter that can be used to reduce image noise (or reduce detail). This can be applicable to be used in combination with thresholding to produce better results of segmentation and further reduce the background noise.

In one dimension, the Gaussian Function [27] is the probability density function of the normal distribution.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$$

Equation 2: Gaussian Function

As an example, consider selecting $\alpha = 2$ and $\sigma = 1$. Applying the Equation 2 above yields the following vector:

$$K_{Gauss} = \frac{1}{27}\begin{pmatrix} 1 & 4 & 1 \\ 4 & 7 & 4 \\ 1 & 4 & 1 \end{pmatrix}$$

Equation 3: Gaussian Kernel

Mathematically, applying a Gaussian blur to an image is the same as convolving the image with the equation above. A visual illustration of applying this Gaussian kernel to the sample image is shown below in Figure 8: Gaussian Blur.

Original Grayscale *Gaussian Blur (σ = 1)*

Figure 8: Gaussian Blur

The Gaussian smoothing filter is exceptionally useful when used in combination with thresholding. As illustrated below in Figure 9: Gaussian Blur with Thresholding, the Gaussian blur filter successfully reduces the noise in the background, and produces a much cleaner result.

Adaptive Thresholding without Gaussian Blur *Adaptive Thresholding after Gaussian Blur*

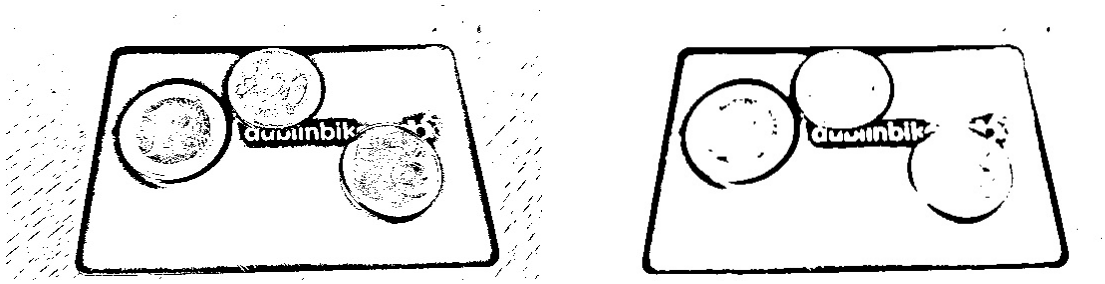


Figure 9: Gaussian Blur with Thresholding

In this project, the Gaussian blur filter can be especially effective in avoiding false detection by reducing some of the details of the image.

2.4.4 Edge Detection

The process of edge detection involves detecting sharp edges in the image and producing a binary image as the output. The edge detection of an image is implemented using localization properties. It also searches for the edge pixels. The edge image obtained prominently produces rectangular shapes in the image. There are many edge finding methods, among which the Sobel, Laplacian and Canny edge finding methods are the most widely used. [28]

The Sobel filter is composed of the following two kernels:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Equation 4: Sobel–Feldman Operator

The equation above shows the kernels of the Sobel–Feldman Operator. The kernel on the left detects horizontal edges and the kernel on the right detects vertical edges.

Sobel Horizontal *Sobel Vertical*

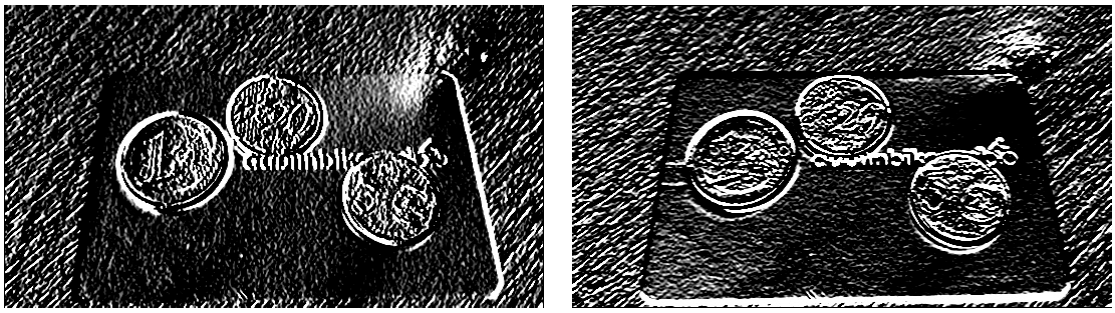


Figure 10: Sobel Edge Detection

As illustrated above in Figure 10, the Sobel filter detects edges in either a horizontal or vertical direction. However, this method doesn't provide a holistic view of all the edges.

To combine both directions, the Laplacian filter can be used. The Laplacian filter uses double derivative in both directions. This function adds up the second x and y derivatives calculated using the Sobel operator:

$$dst = \Delta src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$$

Equation 5: Laplacian Function

When the aperture size is equal to 1, the Laplacian is computed by filtering the image with the following 3x3 aperture:

$$K_{Lapl} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Equation 6: Laplacian Kernel

A visual illustration of applying this Laplacian kernel to the sample image is shown below in Figure 11: Laplacian Edge Detection.

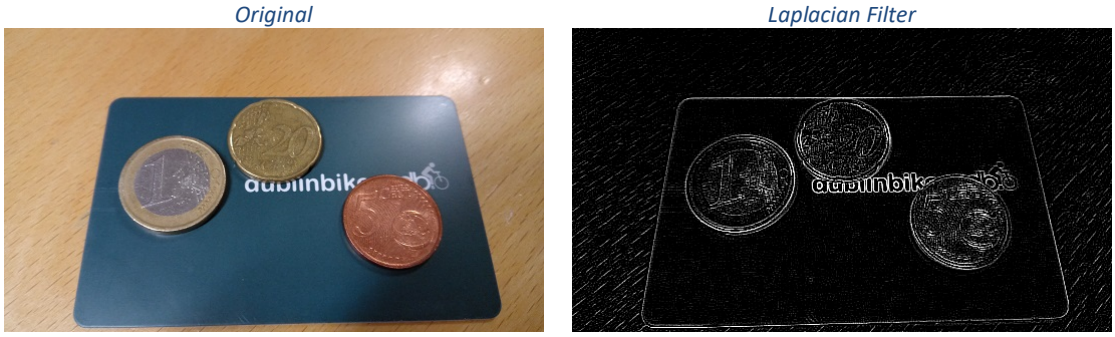


Figure 11: Laplacian Edge Detection

Even though the Laplacian kernel works great for the foreground of the euro coins, the background of this image is notoriously marked. This is due to the fact that the contrast is higher in that region, which gives rise to a lot of noise in the output.

To overcome this problem and remove the noisy output that is not exactly useful, this is where the Canny edge detector comes in handy.

The Canny Edge detector was developed in 1986 by John F. Canny [29], also known to many as the optimal detector. Canny algorithm aims to satisfy three main criteria: low error rate, good localization and minimal response.

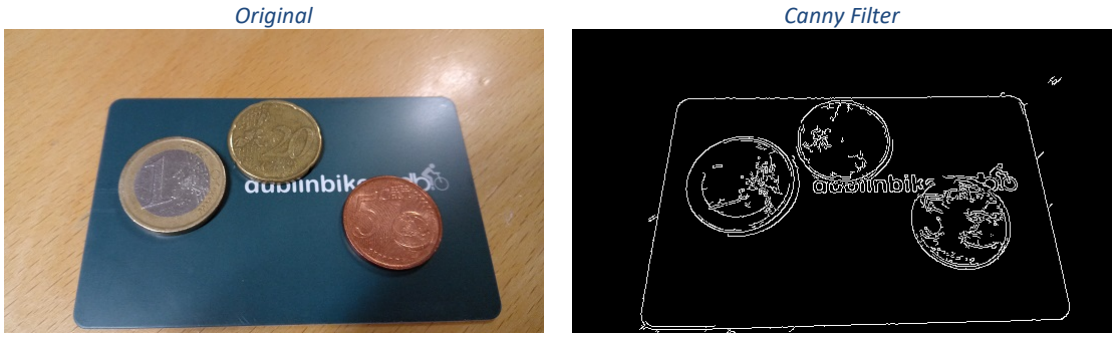


Figure 12: Canny Edge Detection

The Canny edge detector uses a low threshold value and a high threshold value to determine the edges. If the gradient value is above the high threshold value, it is marked as a strong edge. The Canny Edge Detector starts tracking the edge from this point and continues the process until the gradient value falls below the low threshold value. As these thresholds increase, the weaker edges will be ignored. The output image can be significant cleaner and sparser as illustrated above in Figure 12: Canny Edge Detection.

2.4.5 Hough Transform

The Hough Transform is a very useful feature extraction technique to find a pattern in an image such as lines, curves or circles.

The Hough Transform begins with Parametric Modelling [30] that involves moving from the x-y coordinate plane into a parameter plane. By transforming a point into a parameter space, it recognizes local patterns easily, especially for noisy and sparsely digitized images.

In this project, the circle Hough Transform (CHT) [31] is used for detecting euro coins, as they are circular objects that can be described using the following model:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

Equation 7: Circle Hough Transform Model

The circle has 3 parameters describing it: x_0, y_0 and r. This means that the accumulator in this case would be 3D and the computational complexity is increased significantly. To improve the efficiency, the Hough Gradient Method is used to reduce the complexity by using the gradient information of edges, considering not just the edge locations but also their directions.

Original *Hough Circle Transform Detection*

Figure 13: Hough Circle Transform

The Circle Hough Transform can be exceptionally useful in this project for segmenting euro coins from natural images, as illustrated above in Figure 13: Hough Circle Transform.

2.5 Machine Learning

Machine learning is the science of getting computers to act without being explicitly programmed. The idea of machine learning is that the same generic algorithms can be reused with different data to solve different problems. The following machine learning algorithms are widely used for classification problems.

2.5.1 K-Nearest Neighbors Classification

The K-Nearest Neighbors (k-NN) method is one of the most fundamental and the simplest of all machine learning algorithms. This non-parametric method is widely used for classification problems in pattern recognition. [32]

In this classification algorithm, the input consists of the k closest training examples in the feature space, and the output is the class membership. An object is classified by

computing its distance from the cluster of points that represent each class in the feature space.

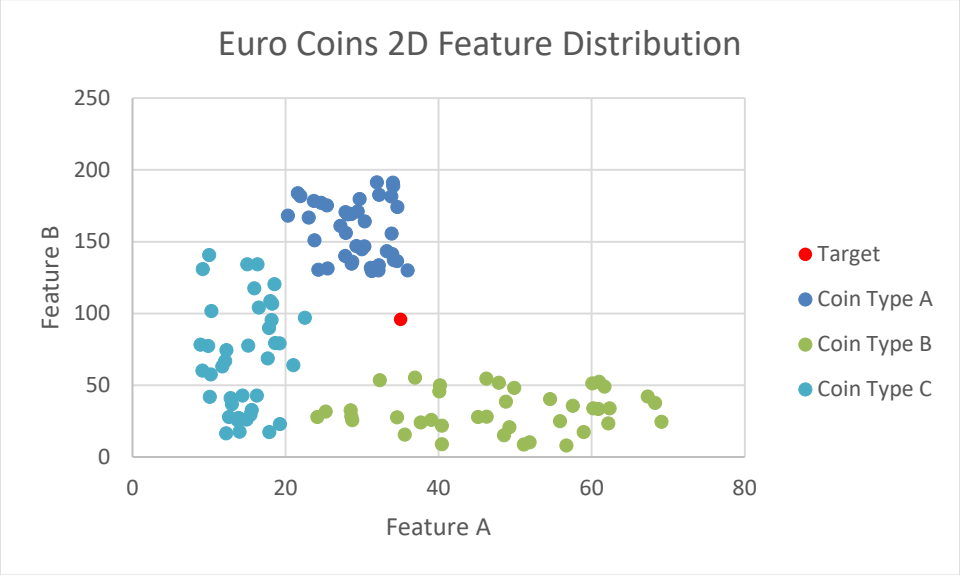


Figure 14: Example of k-NN Classification

An example of this algorithm being potentially applied to this euro coin classification project is shown above in Figure 14: Example of k-NN Classification.

In this hue-saturation distribution chart, the trained known euro coin denominations are represented as a cluster of points in the feature space, assuming the relevant features are the hue and saturation values. The centroid of the cluster represents the class. The target object can be classified by measuring the similarity it has with each known class. This is determined by computing its distance from the points representing each class in the feature space. The distance can be calculated by using either Euclidean metric $d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$ in this simple 2-dimensional example. This can be generalized as the following formula:

$$d_i = \left[\sum_{j=1}^{N} (u_j - f_{ij})^2\right]^{\frac{1}{2}}$$

Equation 8: Generalized N-dimensional Euclidean Distance

The above Equation 8 is the distance d_j of the unknown object from class j. When $k = 1$, this is the most intuitive nearest neighbor type classifier that will simply assign the single nearest class to the target object x:

$$d_R = \min_{i=1}(d_i)$$

Equation 9: One Nearest Neighbor Classifier

As the size of training data set approaches infinity, the 1 nearest neighbor (1-NN) classifier guarantees an error rate of no worse than twice the Bayes error rate [33].

2.5.2 Bayesian Classification

In machine learning, Bayesian classifiers are a family of simple probabilistic classifiers based on Bayes' theorem. [34] Bayesian classifiers can be used to predict class membership probabilities such as the probability that an unknown object belongs to a particular known class. [35]

A Bayesian approach has been used for recognizing objects when the distribution of objects is not as straightforward as the 1-NN classification example. In general, there is a significant overlap in feature values of different objects. [36] An illustration of cases like this is shown below in Figure 15: Example of Bayesian Classification.

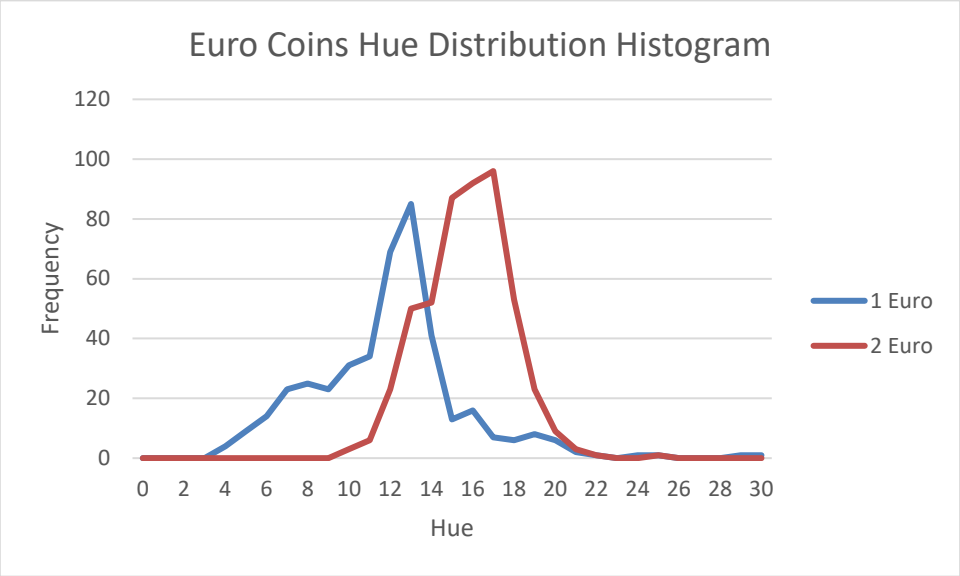


Figure 15: Example of Bayesian Classification

The graph above shows the probability of the *Hue* feature values for 1 euro and 2 euro denominations. It is clear from this graph that sometimes the euro coins of different denomination can have same feature value. This makes it hard to make a decision on which class the unknown coin would fall into, and the Bayesian classifier would come in handy in cases like this.

There are many different practical implementations of Bayesian model selection in object recognition [37]. In this project, the simple Bayesian classifier implemented by K. Fukunaga can be useful. [33] This implementation assumes feature vectors from each class are normally distributed, so the whole data distribution function is assumed to be a Gaussian mixture, one component per class. Using the training data, the algorithm estimates mean vectors and covariance matrices for every class, and then it uses them for prediction.

Chapter 3 Design

3.1 Development Methodology

The Rapid Application Development (RAD) model approach is a software development methodology that is characterized by small-scale projects of short duration. [38] RAD is especially well suited for small development teams where the developers themselves are also the design decision makers.

James Martin coined the term Rapid Application Development (RAD) in the early 1990s to distinguish the methodology from the traditional waterfall model for systems development, as illustrated below in Figure 16: Phases in the James Martin approach to RAD. “RAD refers to a development life cycle designed to give much faster development and higher quality results than the traditional life cycle. It is designed to take maximum advantage of powerful development software that has evolved recently.” [39]

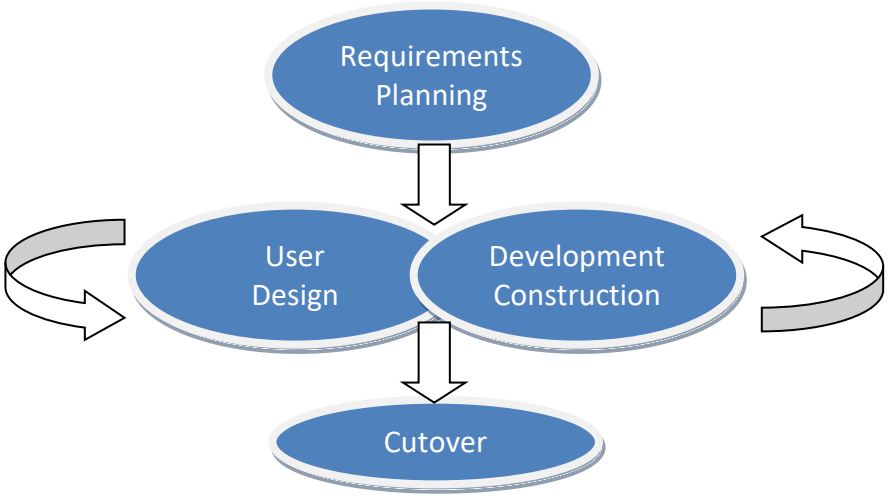


Figure 16: Phases in the James Martin approach to RAD

In general, RAD approaches to software development put less emphasis on planning and more on speedy application delivery. Compared with traditional methods, the planning (design) and cutover (testing, changeover) phases are compressed. Therefore, the software is built, delivered, and placed in operation much sooner.

The choice of using RAD (Rapid Application Development), an exceptionally simple software development methodology, was in an attempt to reduce unnecessary overhead. This is a project that is to be completed entirely by a sole developer. Most other design methodologies are geared around managing larger teams. When the developers are also responsible for project planning and decision making, it can be unnecessary to over plan and thus waste a huge amount of time setting up structures to no real benefit.

RAD is chosen for this project for the following reasons:

1) Small Team

   RAD is usually used in small development teams of typically less than 8 persons. Such teams are made up of both developers and users who are empowered to

make design decisions. [38] In this project, the size of the development team is the smallest. The simplicity of RAD can reduce overhead.

2) Short Duration

   The duration of the development of this project is relatively short. Typically, the length of a RAD project seems to be shorter than six months. The main rationale being that any project taking more than six months to complete is likely to be overtaken by business developments. [38] This project suits those criteria.

3) Strict Deadline

   This project has a very strict deadline. To ensure speedy software delivery, the rapidity of the development is vital.

The goal of following Rapid Application Development (RAD) for this project is to focus more on producing software in a rapid and timely manner and deliver this project in time.

3.2 System Components

As a typical object recognition and classification system, this project involves necessary stages for recognizing, classifying, or otherwise comparing image information about objects with constructed models of certain features.

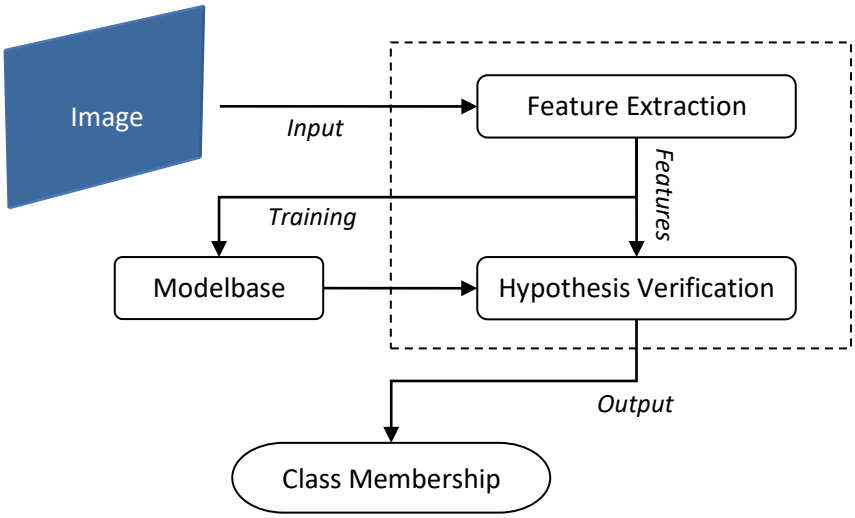


Figure 17: Schematic Diagram of the Euro Coin Classification System

As diagrammed in Figure 17: Schematic Diagram of the Euro Coin Classification System, this project is comprised of the following components:

1) Modelbase
2) Feature Extractor
3) Hypothesis Verifier

Normally, an object recognition system would have a hypothesizer that forms hypotheses as a heuristic to reduce the size of the search space. This euro coin classification system, on the other hand, bypasses the hypothesis formation stage

entirely, because the hypotheses can be easily defined as all possible denominations of euro coins, since the models are limited to only 8 templates.

3.2.1 Modelbase

The Modelbase is a set of models that can represent specific objects or their classes with key visual properties or geometric attributes.

The modelbase is developed using machine learning techniques based on large datasets. In this project, the models are built using many images of the euro coin series of different denominations to describe each euro coin type's attributes, as illustrated below in Figure 18: Classifier Training Dataset.

Figure 18: Classifier Training Dataset

Descriptive statistics and other data mining techniques are essential in this stage to produce the abstract feature vectors that contain precise geometric surface information about the specific class of the objects, which in this case, are required to be able to precisely describe the features of different denominations of euro coins.

The development of the modelbase requires designing a proper mathematical presentation of each model. The data must be analyzed with statistical methods to determine the relevance of each attribute or feature of the objects, and decisions must be made on what must be captured in these models.

In this project, the representation of a euro coin class should capture all relevant information about that coin type without any redundancies. For some objects, geometric descriptions may be available and may also be efficient, while for another class, one may have to rely on generic or functional features. A euro coin model should be a lean presentation of all relevant features. Moreover, the structure of the models should be organized in a form that allows easy access by different components of this euro coin classification system.

3.2.2 Feature Extractor

The Feature Extractor is developed using image processing techniques to identify features that help in forming object hypotheses.

This feature extractor processes the image input and produces the numeric geometric data that mathematically defines the objects. A feature is some attributes of the object that is considered important in describing and recognizing the object in relation to other objects. [21] In this project, size, color, shape and patterns are some of the features that are important in the recognition of euro coins.

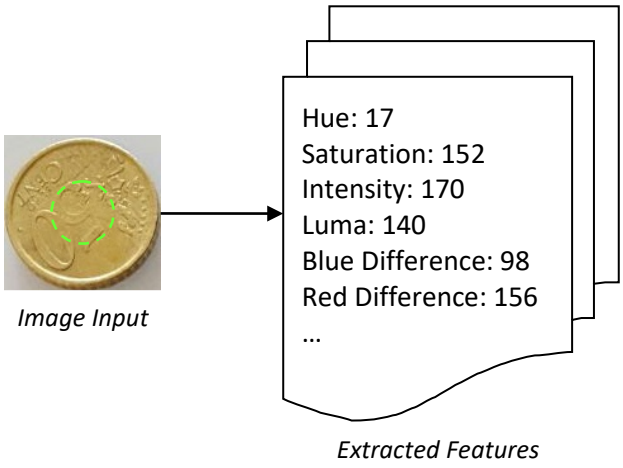


Figure 19: Feature Extraction

As illustrated above in Figure 19: Feature Extraction, the purpose of feature extraction is to produce palpable and quantifiable data that helps in template matching to the models. Effectiveness of features and efficiency of a matching technique must be considered in this step.

The features that are extracted depend on the types of objects to be recognized and the organization of the model database. Due to the nature of the image formation process, some features are easy to compute reliably while others are very difficult. In this step, it is a challenge to determine which features should be detected, and how can they be detected reliably.

3.2.4 Hypotheses Verifier

The Hypotheses Verifier is the most complex module of the euro coin classification system. Unlike other object recognition systems that have a hypothesizer where the likelihoods are assigned to the proposed hypotheses in the hypotheses formation stage, this project skips the hypothesizer completely and relies entirely on the verification phase to achieve the classification of euro coins.

The hypotheses verifier uses the models built in previous phases to verify the hypotheses and determine the likelihood of objects using the extracted features in the image, as illustrated below in Figure 20: Hypotheses Verification Stage. Different

machine learning algorithms have been evaluated and implemented in this stage to compute the probabilities of each hypothesis.

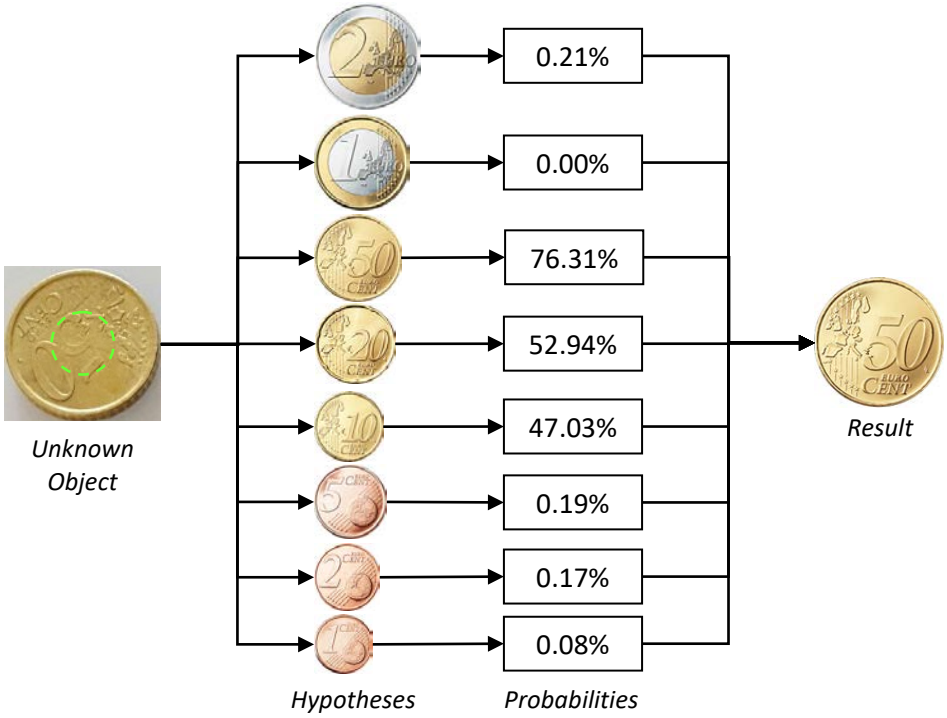


Figure 20: Hypotheses Verification Stage

The system then selects the object with the highest likelihood, based on all the evidence, as the correct object.

Chapter 4 Development

4.1 Overview

This project is implemented using OpenCV (Open Source Computer Vision) library with Python. OpenCV is a cross-platform open-source library of programming functions mainly aimed at real-time computer vision. Any code snippets in this paper would require OpenCV 2.4 environment to run properly.

The implementation of this project involves 1) Recognition, and 2) Classification. The Recognition of the euro coins is mostly achieved using image processing techniques. The classification, on the other hand, is more of a machine learning process that involves data mining techniques.

Besides the euro coin classification system itself, a demo app has also been developed using Ionic frameworks. This Cordova-based cross-platform mobile app aims to demonstrate how this system works on mobile devices. This app communicates with a web-based API service that is developed for the demonstration of this euro coin classification system. The demo app is not part of the euro coin classification system. It is for demonstration purpose only.

4.2 Recognition

The recognition of the euro coins is comprised of various stages where different image processing operations are performed, such as resizing, converting color space, blurring, adaptive thresholding, Hough circles transform, etc., as illustrated below in Figure 21: Various Stages of Euro Coin Recognition.

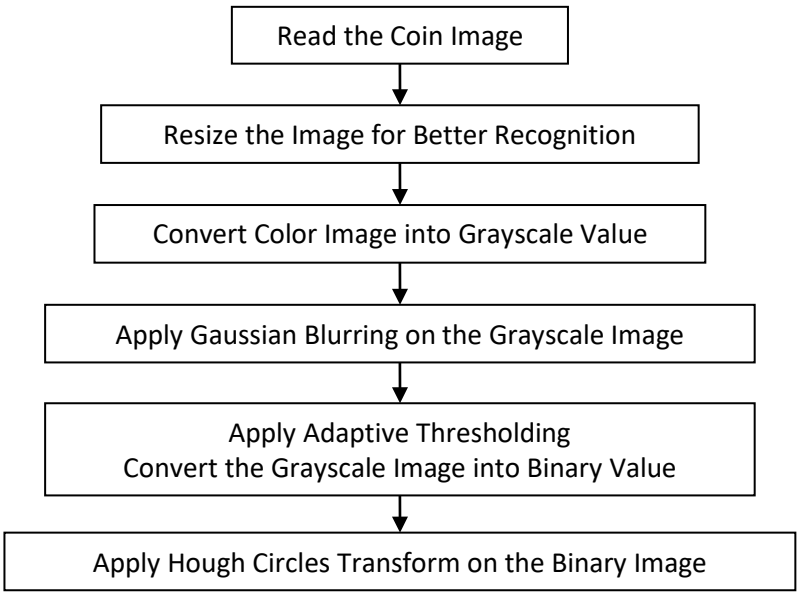


Figure 21: Various Stages of Euro Coin Recognition

The full source code for euro coin recognition can be found in Appendix (9.2.1 Euro Coin Recognition).

4.2.1 Grayscale

After reading the image, it is necessary to convert the original color image into grayscale before any further operations could be applied. The conversion is performed using the color space converting function *cvtColor* in OpenCV.

```
Coin.gray = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
```

This operation results in a 1-dimensional vector leaving only luminance (or intensity). Luminance is by far more important in distinguishing visual features, and this conversion to grayscale could make other image operations easier.

Figure 22: Recognition Step 1 – Convert to Grayscale

The grayscale image is stored as a class variable in the *Coin* class for future reference.

4.2.2 Smoothing

The smoothing filter can effectively reduce the level of noise in the image, which can drastically help to improve the detection accuracy when applied correctly.

The smoothing filter is applied using the *GaussianBlur* function in OpenCV.

```
gray_blur = cv2.GaussianBlur(Coin.gray, (15, 15), 0)
```

This function convolves the source image with the specified Gaussian kernel. In this system, a 15×15 kernel is applied on the previously converted grayscale image.

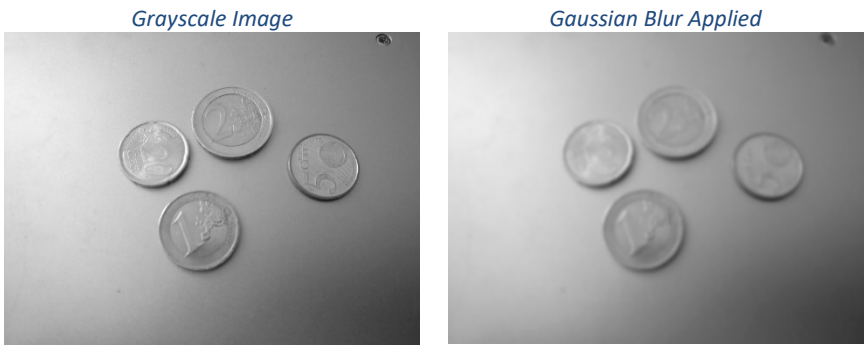


Figure 23: Recognition Step 2 – Gaussian Blur

The Gaussian Blur filter is applied before the detection algorithm, reducing the noises for better detection results.

4.2.3 Foreground-Background Segmentation

Segmenting between foreground and background can be exceedingly helpful when it comes to object detection. Thresholding is an important step to convert the image into binary values that represent the segmentation.

In this system, adaptive thresholding is applied using the *adaptiveThreshold* function in OpenCV.

```
thresh = cv2.adaptiveThreshold(gray_blur, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
```

This function transforms a grayscale image to a binary image. In this system, the Gaussian method is used to apply the threshold value $T(x, y)$, which is a weighted sum (cross-correlation with a Gaussian window) of the 11×11 neighborhood of $(x, y) - 2$.

Blurred grayscale Image *Adaptive Thresholding Applied*

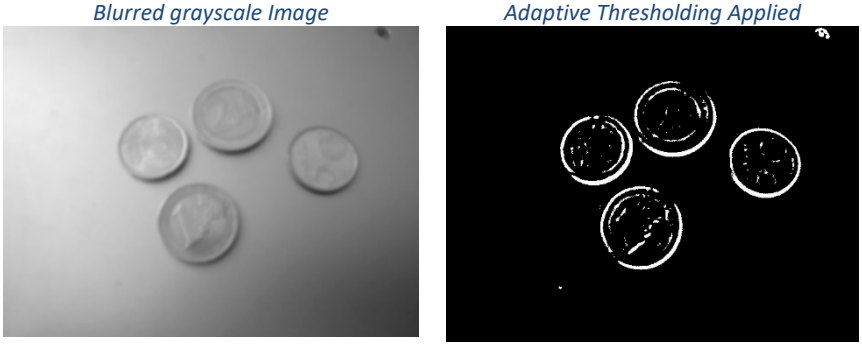


Figure 24: Recognition Step 3 – Adaptive Thresholding

The thresholding aims to produce an image with the objects segmented from the background. This is to help the circle detection in the next step.

4.2.4 Circles Detection

To recognize euro coins from the previously segmented foreground, the circular shape of the euro coins can be used to locate the euro coins using Hough Transform.

In this system, Hough Circle Transform is used to find the circle in the image by using the *HoughCircles* function in OpenCV.

```
circles = cv2.HoughCircles(gray_blur, cv2.HOUGH_GRADIENT, 1, 64,
                           param1=20, param2=40, minRadius=24,
                           maxRadius=96)
```

This function finds circles in a grayscale image using the Hough transform. To refine the results, the parameters have been selected through a series of trial-and-error experiments.

```
coins = map(lambda c: Coin(c[0], c[1], c[2]), circles[0])
```

The detected circles are then used to initialize an array of coin objects. Then the detected circles are drawn to the image using cv2.circle function. The reconstructed image is illustrated below in Figure 25: Recognition Step 4 – Hough Circle Transform.

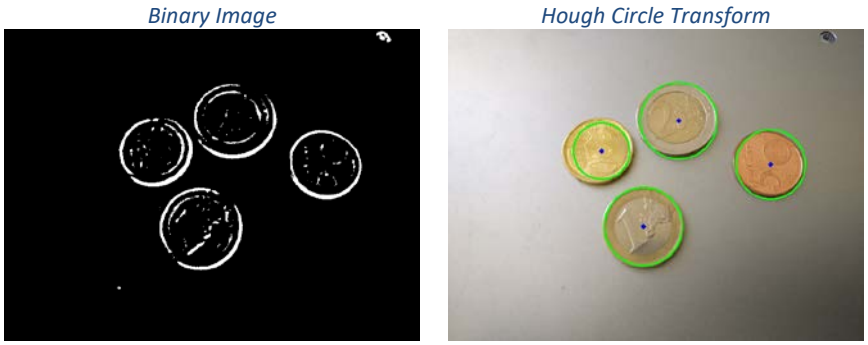


Figure 25: Recognition Step 4 – Hough Circle Transform

The full source code for the circle detection and re-constructing the image with detected circles can be found in Appendix (9.2.2 Reconstructing Image for Demonstration).

4.3 Classification

The classification of the euro coins is heavily involved with machine learning and data mining techniques. The goal of classification is to accurately predict the target class for each case in the data, in this case, the euro coin denominations.

4.3.1 Data Preparation

The training of the classifier requires a large number of sample images as datasets. However, taking pictures of euro coins in real life with a digital camera can be time-consuming.

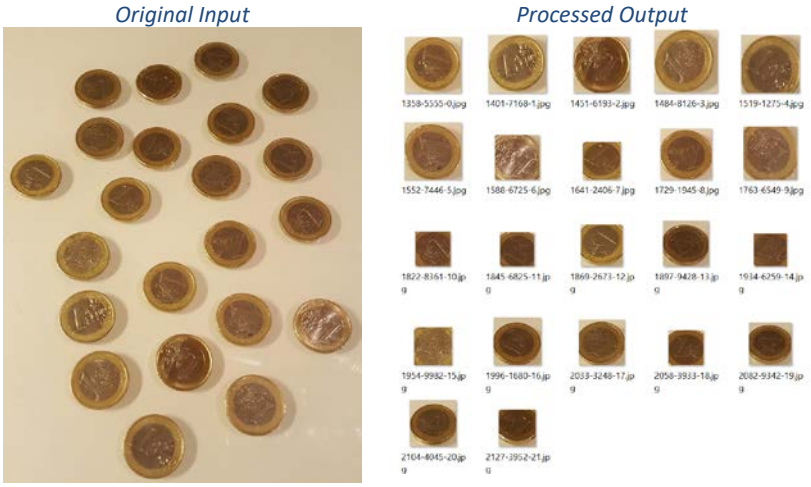


Figure 26: Data Collector Script

In this project, a script is written to prepare the dataset for the training of the classifier. This *dataset collector* script uses computer vision techniques to recognize and segment euro coins from natural images, and output the segmented coins into separate files.

The source code of this dataset collector script can be found in Appendix (9.2.3 Dataset Collector Script).

This dataset collector script takes natural images of euro coins as input, and process them into segmented image dataset that is ready for the classifier training process.

```
$ python dataset_collector.py [image_files...]
```

This script supports processing multiple image at once. Just pass all the file names to this script and they will all be processed.

Figure 27: Pictures of Euro Coins Captured from Digital Camera

To acquire enough sample images for the dataset, more than hundreds of digital images were taken with different angles, positions, lighting, illumination conditions, etc., as illustrated above in Figure 27: Pictures of Euro Coins Captured from Digital Camera. This is to make sure the various datasets would function in all different scenarios regardless of ambient factors, or the type of camera you're using, etc.

After all the coins being processed by the *dataset collector* script, the coins from the original pictures were segmented into thousands of separate files. Then, they were grouped according to their denomination into different corresponding folders.

4.3.2 Extracting Features

The datasets generated in the previous step were all still digital images. Each and every image contains a magnitude of information that is hard to effectively measure and compare.

Before any data analysis could be performed, it is necessary to process the images and convert them to numeric geometric data that mathematically defines the euro coins, so the data could be analyzed.

In this step, different features are extracted from the raw images using image processing techniques. Size, color, shape and patterns are some of the features that are important in the recognition of euro coins.

The purpose of feature extraction is to produce palpable and quantifiable data that helps in template matching to the models.

filename	**hue**	**saturation**	**lightness**	**luma**	**blue_diff**	**red_diff**	**(more)…**
0589-4691	17	120	157	134	106	148	…
5556-3209	17	113	170	145	106	150	…
8368-9629	13	156	170	130	100	162	…
2733-9382	13	209	113	76	100	160	…
(more)…	…	…	…	…	…	…	…

Table 2: Extracted Features

To make sure the data is consistent and avoid outlier, some image processing operations are also performed in this step, such as normalization (contrast stretching) and color space conversion.

The source code for feature extraction and dataset processing could be found in Appendix (9.2.4 Feature Extraction).

4.3.3 Model Training

To build the models of each euro coin denomination, data mining techniques such as descriptive statistics, data cleaning (outlier elimination) and relevance alanysis, are used in this project to help understand the datasets.

The goal of this step is to produce the abstract feature vectors that contain precise geometric surface information about the features of different denominations of euro coins.

The source code for the model training be found in Appendix (9.2.7 Euro Coin Models Training).

4.3.3.1 Data Exploration

In this project, the descriptive features that can be easily extracted are: hue, saturation, intensity, luma, etc… Descriptive statistics are used to describe the basic features of the data in this project. They provide simple summaries about the sample

and the measures, as shown in the Table 3: Descriptive Statistics of 1-Euro Coins below from 1794 samples of 1 euro coins.

	Minimum	1st Quartile	Mean	Median	3rd Quartile	Maximum	Standard Deviation
Hue	3	10	23.17	13	15	156	30.28
Saturation	3	53	98.21	97	135	228	53.09
Intensity	52	94	118.85	115	141	218	31.94
Luma	48	78	102.55	98	123	209	31.75
Blue Difference	93	112	117.80	118	124	151	7.54
Red Difference	77	134	141.26	142	149	164	9.28
(More)…	…	…	…	…	…	…	…

Table 3: Descriptive Statistics of 1-Euro Coins

The source code for generating the above outcome can be found in Appendix (9.2.5 Data Exploration – Continuous Features).

For each feature, its minimum, first quartile, mean, median, third quartile, maximum and standard deviation are computed. This data exploration process is exceptionally important, as this helps to condense large amounts of information into more manageable chunks.

Data visualization can be useful for gaining insights to the data as well. The following is the 1-euro coin HSI (Hue, Saturation, and Intensity) histogram from 1794 samples.

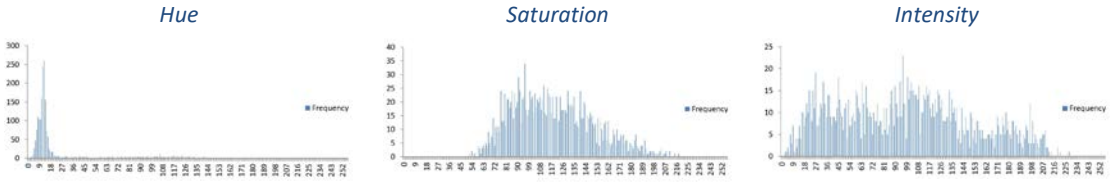


Figure 28: Histograms of 1 Euro Coins Color Attributes

To be able to model these euro coins, it is necessary to understand the relationships between these features, and try to build a model that generalizes beyond the dataset and that isn't influenced by the noise in the dataset.

4.3.3.2 Data Cleaning

Data cleaning is a technique that is applied to remove the noisy data and correct the inconsistencies in data. Data cleaning involves transformations to correct the wrong data.

The raw datasets of extracted features are noisy, as illustrated below in Figure 29: Hue-Saturation Scatter Plot of Raw Euro Coins Data.

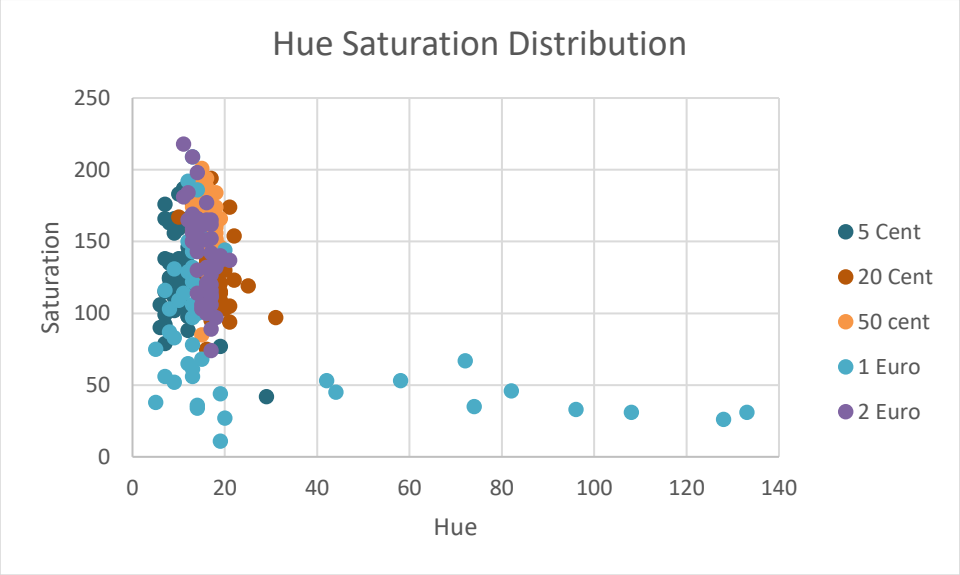


Figure 29: Hue-Saturation Scatter Plot of Raw Euro Coins Data

The above scatter plot shows that the raw data of extracted features contains a significant number of outliers. It is hard to predict and model the euro coins when the patterns in the data are not consistent.

Figure 30: Identified Outliers

Because the models built using statistical techniques are very sensitive to noises and outliers in the data, which may lead to poor quality clusters, the identified outliers should be removed from the datasets to improve data quality and thus boost prediction accuracy.

The outliers are removed using a clamp transformation that clamps all values above an upper threshold of $\mu + \sigma$ and below a lower threshold of $\mu - \sigma$ to these threshold values using the 68 – 95 – 99.7 rule (three-sigma rule) [40], thus removing the offending outliers.

$$a_i = \begin{cases} \mu + \sigma, & a_i > \mu + \sigma \\ \mu - \sigma, & a_i < \mu - \sigma \\ a_i, & otherwise \end{cases}$$

Equation 10: Clamp Transformation to Remove Outliers Using 3-Sigma Rule

In the above formula, a_i is a specific value of feature a, and μ and σ are the median and the standard deviation of this feature respectively.

The source code of the implementatoin of this Clamp Transformation can be found in Appendix (9.2.6 Outlier Elimination – Clamp Transformation).

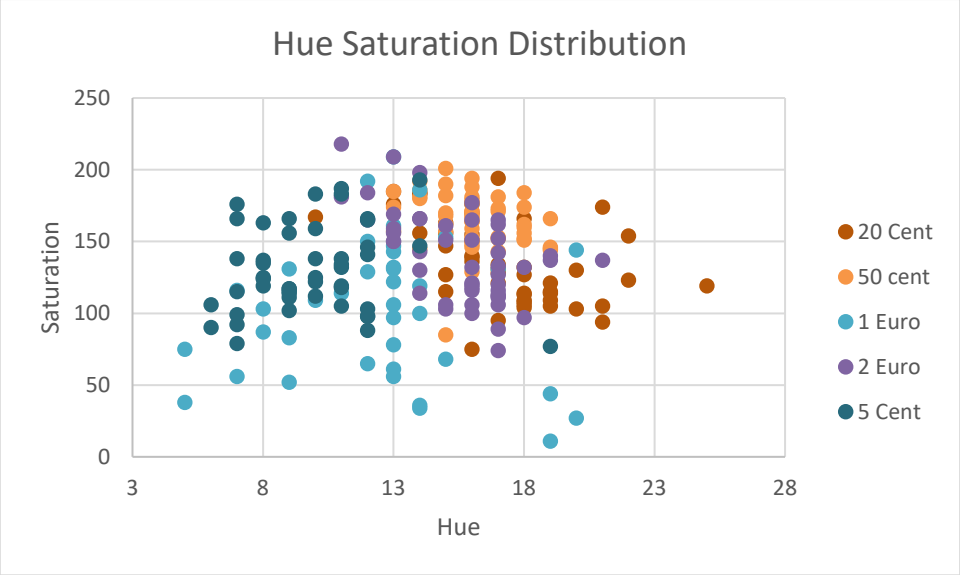


Figure 31: Hue-Saturation Scatter Plot of Euro Coins Data After Data Cleaning

As illustrated above in Figure 31: Hue-Saturation Scatter Plot of Euro Coins Data After Data Cleaning, the distribution plot after outlier elimination is much more consistent and thus easier to find patterns and build models on.

With Clamp Transformation, the datasets have become more consistent. The difference can be seen in the table below Table 4: Clamp Transformation Before and After on Mean of Hue.

Hue Values	**2 Euro**	**1 Euro**	**50 Cent**	**20 Cent**	**10 Cent**	**5 Cent**	**2 Cent**	**1 Cent**
Before	15.62	23.17	15.81	17.44	18.38	9.86	9.74	10.10
After	15.61	16.28	15.84	17.01	17.77	9.73	9.75	10.11

Table 4: Clamp Transformation Before and After on Mean of Hue

For most other coin denominations, the clamp transformation has very little impact. However, for outlier-ridden denominations such as 1 euro, the difference is huge.

4.3.3.3 Modelling

The final models of each euro coin denomination contain all the key visual properties or geometric attributes that are selected from previous steps. The models are created using the median values of all the extracted features.

A visual illustration of the some of the features that the models have is below in Figure 32: Final Models of Euro Coin Denominations (Hue-Saturation). This chart is a good indication of where each model is. As we can see, most of the euro coin denominations have their own distinct position in the chart. The actual models contain much more information than just hue-saturation.

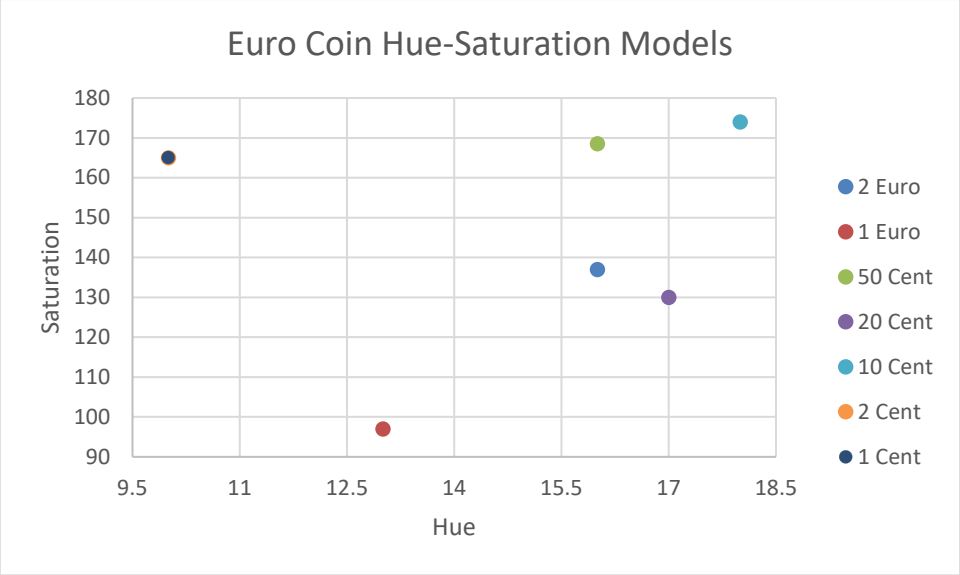


Figure 32: Final Models of Euro Coin Denominations (Hue-Saturation)

The models are generated using the previously processed data, outputting a JSON file that contains all the relevant features is created, which can be found in Appendix (9.1 Euro Coin Models).

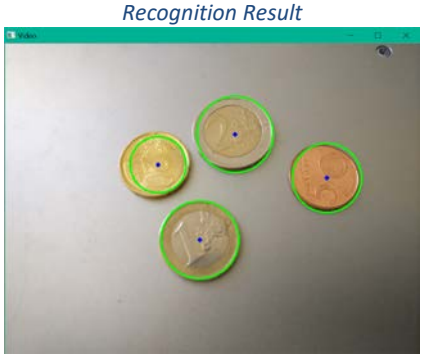
The source code for generating the models can also be found in Appendix (9.2.7 Euro Coin Models Training).

4.3.4 Probability Prediction

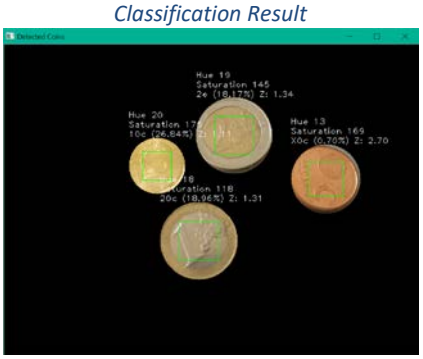
This project aims to provide a probabilistic classification solution that not only outputting the most likely class that the sample should belong to but also provide classification with a degree of certainty.

This system uses the models built in previous phases to verify each hypothesis and determine the likelihood of objects using the extracted features in the image.

Recognition Result



Classification Result



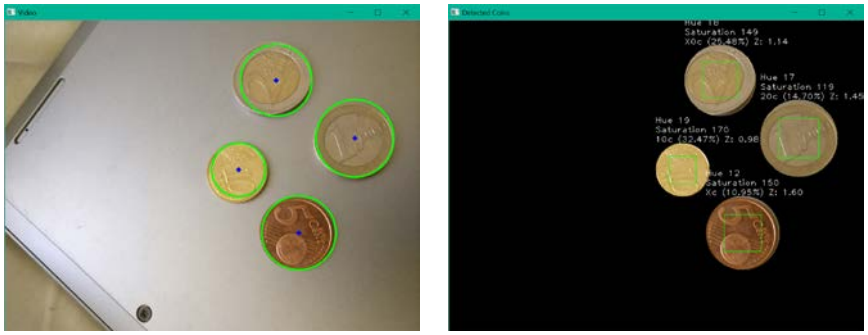


Figure 33: Results on Desktop Machine

Different machine learning algorithms have been evaluated and implemented in this stage to compute the probabilities of each hypothesis.

In early stages of this project, the classification is performed using the k-Nearest Neighbors algorithm. However, the k-NN classification is a discriminative classification and it is not naturally probabilistic.

The normal Bayesian classification approach has been used in this system to compute the probability of an unknown object belongs to a certain euro coin denomination. As we can see from the histogram below in Figure 34: Hue Histogram of Euro Coins, the feature vectors from each class can be safely assumed to be normally distributed.

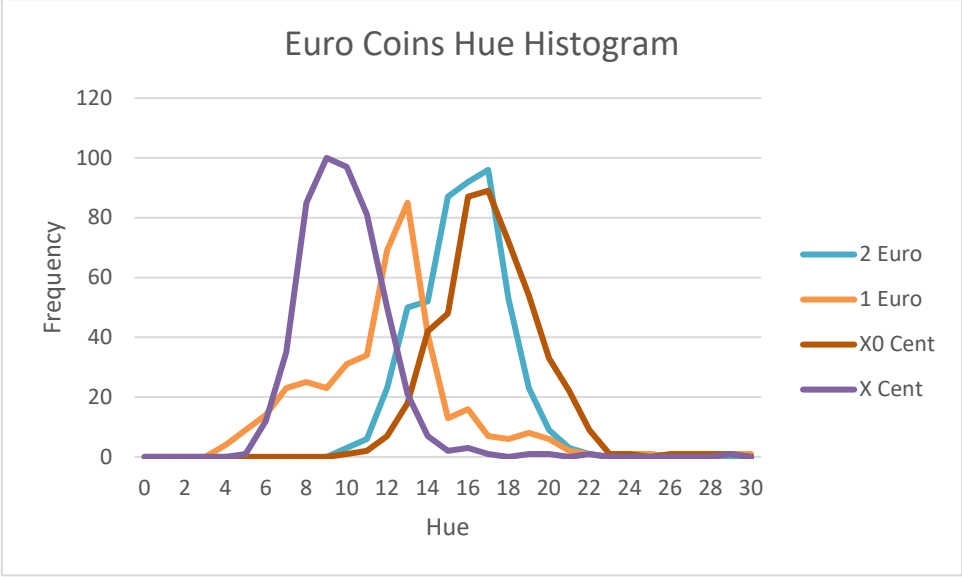


Figure 34: Hue Histogram of Euro Coins

For each feature, it firstly computes the z-score using the median and standard deviation.

```
z[feature] = (x - float(f['median'])) / float(f['std_deviation'])
```

Then, all the z-scores are combined using the standard Euclidean distance.

```
z = sum(x**2 for x in z.itervalues()) ** 0.5
```

And the z-score is then converted to the p-value, which is the probability of the unknown coin belonging to a certain denomination.

```
p = 1.0 + math.erf(-z / math.sqrt(2.0))
```

The system then selects the object with the highest likelihood, based on all the evidence, as the correct object.

The full source code for the probability prediction can be found in Appendix (9.2.8 Euro Coin Classification).

4.4 Demo App

Due to the nature of this project being more of a research-based machine learning system, it is hard to demonstrate this project for practical purposes. Thus, a simple demo app was developed using Ionic Frameworks to demonstrate this euro coin classification system. Users can easily snap a picture of euro coins using the phone's camera and the app will report the classification results and count the sum of the euro coins in the picture.

Figure 35: Mobile App for Demonstration

This Cordova-based cross-platform mobile app aims to demonstrate how this system would work on mobile devices. The demo app is not part of the euro coin classification system. It is for demonstration purpose only. The source code for this Ionic app can be found in Appendix (9.2.A Demo App – Client).

This app communicates with a web-based API service that is developed for the demonstration of this euro coin classification system. The web API service is developed using a lightweight open-source Python web framework Flask. The source code for the API web service can be found in Appendix (9.2.9 Demo App – Server).

Chapter 5 System Validation

5.1 Overview

Throughout the lifetime of the project, there were many iterations of prototyping, testing and evaluation that was required to locate the best performing classifiers and to examine if they were functioning correctly.

In this project, system validation is done to ensure this system satisfies its intended functional requirements. The system validation stage focuses on the different methods of testing that has been done in order to ensure that the system executes the way it's supposed to perform. The various methods of testing include unit testing and integration testing.

5.2 Unit Testing

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.

5.2.1 Coin Segmentation Unit

A unit was created to test the euro coin segmentation module which is one of the core functional components of the system that can locate and segment the euro coins in an image. This was tested in every increment of the module. The following table shows the results of these tests.

No.	Feature	Expectation	Result
1	Read image from file	Return an array of BGR stream representing the image	PASS
2	Convert BGR stream to grayscale	Return an array of Intensity stream	PASS
3	Apply Gaussian Blur filter on grayscale image	Return a convolved array of Intensity stream representing the blurred image	PASS
4	Apply adaptive threshold on the blurred image	Return an array of binary values	PASS
5	Apply Hough Circle Transform	Return a 3-element floating-point vector of found circles	PASS

Table 5: Coin Segmentation Unit Test Results

The results above in Table 5: Coin Segmentation Unit Test Results indicate the coin segmentation unit is functioning as expected.

5.2.2 Feature Extraction Unit

The feature extractor is one of the main components of the euro coin classification system. The feature extractor is required to be able to convert the digital image files to numeric geometric data. The table below shows the results of these tests.

No.	Feature	Expectation	Result
1	Read image from file	Return an array of BGR stream representing the image	PASS
2	Convert BGR stream to YUV color space	Return an array of YUV stream representing the image	PASS
3	Normalize, equalize the Y channel	Return a contrast stretched normalized image	PASS
4	Convert to HSV color space	Return an array of HSV stream representing the image	PASS
5	Sum the color values in the center of the image	Return an integer that is a feature extracted from the image	PASS

Table 6: Feature Extraction Unit Test Results

The feature extractor is all functioning as reported from the unit test.

5.2.3 Data Exploration Unit

The data exploration is one of the key steps for training the models of euro coins. The unit testing results for this module is shown below in Table 7: Data Exploration Unit Test Results.

No.	Feature	Expectation	Result
1	Count the dataset size	Return an integer that is the size of the dataset	PASS
2	Compute the minimum of a feature	Return a value that is the minimum value of a feature	PASS
3	Compute the maximum of a feature	Return a value that is the maximum value of a feature	PASS
4	Compute the mean of a feature	Return a value that is the average value of a feature	PASS
5	Compute the first quartile of a feature	Return a value that is the first quartile of a feature	PASS
6	Compute the second quartile of a feature	Return a value that is the median of a feature	PASS
7	Compute the third quartile of a feature	Return a value that is the third quartile of a feature	PASS
8	Compute the standard deviation of a feature	Return a value that is the standard deviation of a feature	PASS
9	Compute the histogram	Return an array that represents the distribution of the values	PASS

Table 7: Data Exploration Unit Test Results

5.2.4 Model Training Unit

This model training unit is to evaluate the functionality of the model training module that process the data and output to a JSON file. The table below shows the results of these tests.

No.	Feature	Expectation	Result
1	Eliminate outliers in the raw data	Return cleaned data without outliers	PASS
2	Generate data reports	Output a .csv file containing the data reports	PASS
3	Generate models to a JSON file	Output a .json file containing all the classes and their features	PASS

Table 8: Model Training Unit Test Results

This model training unit involves a significant amount of data analysis that is not programmed in this module. For example, the test result does not reflect the quality of the data, or the accuracy of the models. The model training unit is functional as shown in the reported results.

5.2.5 Hypotheses Verification Unit

This hypotheses verification unit loads the previously trained models and make a prediction based on the models and produce the probability that an unknown coin being a specific denomination.

No.	Feature	Expectation	Result
1	Load the generated models from a JSON file	Return a dictionary that represents the JSON structure	PASS
2	Process the features of the image	Return a list of features that are extracted from the image	PASS
3	Compare the features with the models	Return an integer that is the difference between the object and the model	PASS
4	Compute the probability	Return a value that is the probability of this object belongs to that model	PASS
5	Select the result with highest probability	Return a class membership	PASS

Table 9: Hypotheses Verification Unit Test Results

This test only focuses on whether this hypotheses verification unit is functional. The reliability and the accuracy of the predication, however, is not the subject of this test. As the report indicates, the hypotheses verification unit has passed all the tests.

5.3 Integration Testing

An object recognition system is not just made up of one single component but multiple separate components that are closely related. Integration testing is a software development process which program units are combined and tested as groups in multiple ways.

The bottom up testing approach is used for this euro coin classification system, where the lowest level components are tested first, they are then used as the base to test higher level components. The process will be repeated until the components at the top of the hierarchy are tested.

No.	Feature	Expectation	Result
1	Integrate the Coin class with the Dataset Collector	An instance of Coin that is initialized from the collected coins	PASS
2	Integrate the Coin class with the Feature Extractor	An instance of Coin that contains the extracted features	PASS
3	Integrate the Coin class with the Model Trainer	A list of instances of Coin that can be used for data analysis	PASS
4	Integrate the Coin class with the Prediction Module	An instance of Coin that can be used to predict the probability	PASS

Table 10: Euro Coin Classification System Integration Testing Results

The test results indicate that the completed euro coin classification system is integrated for its compliance with the requirements that are specified to it within the project scope. All components that have passed the integration testing are put together to combine the full implementation of the project.

Chapter 6 Project Plan

6.1 Project Evolution

In the proposal stage, this project initially only aimed to achieve the recognition of euro coins. The actual classification between euro denominations was originally not part of the main objectives, but rather a nice-to-have feature.

The project did not exactly follow the original plan set out in the proposal stage. The slight shift in its topic from recognition to classification has caused the project to change its direction from building an actual piece of software application that detects coins, to a research-based thesis-centered study on the euro coin classification using image processing and machine learning.

6.2 Issues and Regrets

Since the start of the development of the project, there has always been issues that have affected the overall project plan.

There were many iterations of design, prototyping and user feedback that pushed the development of this project's functionalities out much later than originally scheduled. There were many times in the project that the actual implementation had to undergo large shifts with more technologies being researched and new approaches being introduced. Had a more ridged progression plan been in place either a lot of time would have been wasted correcting this plan or the freedom of movement of the project would have been stifled.

Following Rapid Application Development (RAD) methodology, even though the prototype was delivered rapidly, the problem of immediately jumping into coding and not managing oneself at all has proven to be disastrous. This project initially adopted some of the existing pattern-based classifiers. At the beginning of this project, a lot of time was wasted in a series of blind trial-and-error experiments with the ineffective classification methods. Had proper research and designing been conducted before blindly jumping into coding, this project could have moved along more smoothly.

Chapter 7 Conclusion

7.1 Summary

A model-based euro coin classification system has been developed in this project by collecting large datasets of images of euro coins, extracting their features and data modelling.

Different approaches of building models for object recognition have been investigated and some models of euro coin denominations have been successfully developed using the visual features that are extracted from the datasets. Different classification algorithms have been implemented and evaluated for model-based object recognition systems.

Results returned in this project suggest that the Normal Bayesian classification method is 28.93% more accurate than the k-Nearest Neighbors (k-NN) method over the tested datasets.

7.2 Learning Outcomes

This project has thoroughly investigated the use of machine learning specifically in the context of euro coin classification. Extensive researches have been conducted in this project in search for reliable visual features of euro coins that could be possibly used to develop this euro coin classification system.

Besides the data mining techniques and statistical skills required to construct the mathematical models of euro coins in this project, relevant computer vision techniques have also been extensively studied to further help this object recognition system move forward.

7.2 Future Work

In considering the evolution of this project, there are several design choices that could be improved. The process of making bad choices and reflecting on them is part of the learning curve, as is the ability to recognize a less than perfect approach and the reasons why. There follows an account of some of these realizations and their outcomes.

7.2.1 Relevance Analysis

There are so many features that could be extracted from euro coins. It is hard to understand the patterns and the relationship between the features and their corresponding denomination label.

In this project, all the extracted attributes and features have been assumed to be relevant to determining the denomination of the euro coins. However, in actual fact, some of the features might be irrelevant, or weakly relevant. Using these features to

construct the models of euro coin denominations may affect the accuracy of the classification.

For future work, proper relevance analysis should be conducted. The irrelevant or weakly relevant features of the euro coins should be filtered out, and the strongly relevant features are retained for modelling. This could be done by sorting and selecting the most relevant dimensions and levels. The entropy and information gain measure of each attribute could be calculated to determine the relevance of these attribute.

7.7.2 Prediction Accuracy

The Bayesian classification method was introduced into the scope at quite a later stage. Prior to that, it was assumed that the extracted features would simply follow Gaussian distribution. The Bayesian classifier was not considered at an early enough stage for implementation.

There are other classification algorithms that could be researched, such as Artificial Neural Network (ANN), Support Vector Machines (SVM), that could further improve the accuracy of the current implementation of the euro coin classification.

Chapter 8 Bibliography

[1] M. Sun and S. Savarese, "Model-Based Object Recognition," in *Computer Vision: A Reference Guide*, Springer US, 2014, pp. 488-492.

[2] B. Chetan and P. Vijaya, "A Robust Method of Image Based Coin Recognition," 2013.

[3] Y. Mitsukura, M. Fukumi and N. Akamatsu, "Design and evaluation of neural networks for coin recognition by using GA and SA," IEEE, 2000.

[4] P. Davidsson, "Coin Classification Using A Novel Technique For Learning Characteristic Decision Trees By Controlling The Degree Of Generalization," Lund University, Lund, 1998.

[5] M. Nölle, H. Penz, M. Rubik, K. Mayer, I. Holländer and R. Granec, "Dagobert – A New Coin Recognition and Sorting System," ARC Seibersdorf, Syndney, Australia, 2003.

[6] C. H. Chen, "Computer Recognition and Evaluation of Coins," in *Handbook of Pattern Recognition and Computer Vision*, World Scientific, 2015, p. 142.

[7] M. Adameck, M. Hossfeld and M. Eich, "Three Color Selective Stereo Gradient Method for Fast Topography Recognition of Metallic Surfaces," Machine Vision Applications in Industrial Inspection XI, 2003.

[8] R. Bremananth, B. Balaji, M. Sankari and A. Chitra, "A New Approach to Coin Recognition using Neural Pattern Analysis," IEEE Indicon 2005 Conference, Chennai, 2006.

[9] European Central Bank, "Coins," 2013. [Online]. Available: https://www.ecb.europa.eu/euro/coins/html/index.en.html.

[10] R. Szeliski, in *Computer Vision: Algorithms and Applications*, 2009, p. 11.

[11] M. J. Laielli, "Multi-Frame Object Detection," Naval Postgraduate School, Monterey, California, 2012.

[12] R. Fergus, P. Perona and A. Zisserman, "Weakly Supervised Scale-Invariant Learning of Models for Visual Recognition," in *International Journal of Computer Vision*, 2007, p. 273–303.

[13] J. Zhang, M. Marszałek, S. Lazebnik and C. Schmid, "Local Features and Kernels for Classification of Texture and Object Categories: A Comprehensive Study," 2007.

[14] L. Fei-Fei and P. Perona, "A Bayesian Hierarchical Model for Learning Natural Scene Categories," in *Computer Society Conference on Computer Vision and Pattern Recognition*, California, 2005.

[15] L. Fei-Fei, R. Fergus and A. Torralba, "Recognizing and Learning Object Categories," ICCV, 2009.

[16] M. Fischler and R. Elschlager, "The Representation and Matching of Pictorial Structures," in *Transactions on Computers*, IEEE, 1973.

[17] P. Felzenszwalb, R. Girshick, D. McAllester and D. Ramanan, "Object Detection with Discriminatively Trained Part-based Models," in *Transactions on Pattern*

Analysis and Machine Intelligence, IEEE, 2010, pp. 1627-1645.

[18] P. Viola and M. Jones, "Robust Real-Time Object Detection," in *International Journal of Computer Vision*, 2 ed., vol. 57, 2001, pp. 137-154.

[19] P. Viola and M. J. Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, Cambridge: Conference on Computer Vision and Pattern Recognition (CVPR), 2001, pp. 511-518.

[20] S. Liao, X. Zhu, Z. Lei, L. Zhang and S. Z. Li, "Learning Multi-scale Block Local Binary Patterns for Face Recognition," Beijing, China, International Conference on Biometrics (ICB), 2007, pp. 828-837.

[21] R. Jain, R. Kasturi and B. G. Schunck, "Object Recognition," in *Machine Vision*, McGraw-Hill, Inc., 1995, pp. 459-491.

[22] R. C. Gonzalez and R. E. Woods, in *Digital Image Processing*, Prentice Hall, 2008.

[23] R. Fisher, "Contrast Stretching," 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm.

[24] P.-N. Tan, "Cluster Analysis: Basic Concepts and Algorithms," in *Introduction to Data Mining*, University of Minnesota, 2006, pp. 518-520.

[25] Shapir, L. G. &. S. and G. C, "Computer Vision," Prentice Hall, 2002.

[26] R. Fisher, "Adaptive Thresholding," 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm.

[27] W. EW, "Gaussian Function," 4 November 2001. [Online]. Available: http://mathworld.wolfram.com/GaussianFunction.html.

[28] M. Sonka, V. Hlavac and R. Boyle, Image Processing Analysis and Machine Vision, 2nd ed., Boston: PWS Publishing Company, 2001.

[29] J. Canny, "A Computational Approach to Edge Detection," in *Pattern Analysis and Machine Intelligence*, IEEE Trans, 1986, pp. 679-698.

[30] D. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes," in *Pattern Recognition*, 2 ed., vol. 13, Rochester, New York: University of Rochester, 1981, pp. 111-122.

[31] H. Rhody and C. F. Carlson, "Hough Circle Transform," 2015.

[32] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," in *The American Statistician*, 1992, pp. 175-185.

[33] K. Fukunaga, "Introduction to Statistical Pattern Recognition," 1990, p. 3 and 97.

[34] D. J. Hand and K. Yu, "Idiot's Bayes — not so stupid after all?," in *International Statistical Review*, 2001, pp. 385-399.

[35] D. MacKay, "Bayesian Inference and Sampling Theory," in *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003, pp. 457-466.

[36] L. Wasserman, "Bayesian Model Selection and Model Averaging," Carnegie Mellon University, 2000.

[37] H. Chipman, E. I. George and R. E. McCulloch, "The Practical Implementation of Bayesian Model Selection," The University of Waterloo, The University of Pennsylvania and The University of Chicago, 2001.

[38] P. Beynon-Davies, C. Carne, H. Mackay and D. Tudhope, "Rapid application development (RAD): an empirical review," in *European Journal of Information Systems*, Palgrave Macmillan UK, 1999, pp. 211-223.

[39] J. Martin, Rapid Application Development, New York: Macmillan, 1991.

[40] E. W. Grafarend, "Three-Sigma Rule," in *Linear and Nonlinear Models: Fixed Effects, Random Effects, and Mixed Models*, Walter de Gruyter, 2006, p. 553.

[41] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, ICCV, 1999, p. 1150–1157.

[42] Y. Amit and P. Felzenszwalb, "Object Detection," University of Chicago.

Chapter 9 Appendix

9.1 Euro Coin Models

The euro coin models are generated to a JSON (JavaScript Object Notation) file for better accessibility with other components of the system.

The following is an example of the euro coin models that are generated in this project. Only a short portion of the models is excerpted from the actual output file. Most of the features, and denominations are omitted to conserve space. This is only to demonstrate the structure of the actual models. For the full complete models, please visit the Github repository given in section 9.2.


```
{
    "name": "Euro Coin Detector Classifier",
    "author": "Chen Yumin",
    "website": "http://chenyumin.com",
    "version": "1.0.0",
    "__comment": [
        "    Euro Coin Detector Classifier",
        "    Created by Chen Yumin (http://chenyumin.com)",
"////////////////////////////////////////////////////////////////////////////",
" This software is provided by the copyright holders and contributors 'as is' and",
" any express or implied warranties, including, but not limited to, the implied",
" warranties of merchantability and fitness for a particular purpose are disclaimed.",
" In no event shall the Intel Corporation or contributors be liable for any direct,",
" indirect, incidental, special, exemplary, or consequential damages",
" (including, but not limited to, procurement of substitute goods or services;",
" loss of use, data, or profits; or business interruption) however caused",
" and on any theory of liability, whether in contract, strict liability,",
" or tort (including negligence or otherwise) arising in any way out of",
" the use of this software, even if advised of the possibility of such damage.",
"////////////////////////////////////////////////////////////////////////////"
    ],
    "classification": {
        "10c": {
            "hue": {
                "count": 988,
                "f_quartile": 16.0,
                "t_quartile": 19.0,
                "median": 18.0,
                "maximum": 93,
                "std_deviation": 6.9160388586533017,
                "minimum": 11,
                "mean": 18.383603238866396
            },
            "saturation": {
                "count": 988,
                "f_quartile": 160.0,
                "t_quartile": 186.0,
                "median": 174.0,
                "maximum": 226,
                "std_deviation": 23.686367488029557,
                "minimum": 43,
                "mean": 170.9402834008097
            },
            …
            … (Other features are omitted to conserve space.)
            …
```

```
            "lightness": {
                "count": 988,
                "f_quartile": 135.0,
                "t_quartile": 165.0,
                "median": 151.0,
                "maximum": 205,
                "std_deviation": 21.975253516542811,
                "minimum": 80,
                "mean": 149.36437246963564
            }

        },
         …
         … (Other denominations are omitted to conserve space.)
         …
        "5c": {
            "hue": {
                "count": 1389,
                "f_quartile": 8.0,
                "t_quartile": 11.0,
                "median": 10.0,
                "maximum": 38,
                "std_deviation": 2.6427839325594071,
                "minimum": 3,
                "mean": 9.8560115190784732
            },
            "saturation": {
                "count": 1389,
                "f_quartile": 112.0,
                "t_quartile": 147.0,
                "median": 128.0,
                "maximum": 214,
                "std_deviation": 27.334895199262309,
                "minimum": 20,
                "mean": 129.79553635709144
            },
            …
            … (Other features are omitted to conserve space.)
            …

            "lightness": {
                "count": 1389,
                "f_quartile": 134.0,
                "t_quartile": 166.0,
                "median": 151.0,
                "maximum": 224,
                "std_deviation": 22.957861218591709,
                "minimum": 87,
                "mean": 151.0460763138949
            }
        }
    }
}
```


9.2 Code Snippets

This section does not contain the full complete source code of the entire project. Only a short portion of the actual implementation was excerpted from the source code to conserve space.

The following code snippets are only for demonstration purposes. To obtain the full complete source code, please follow the link on each snippet.

9.2.1 Euro Coin Recognition

```
#! /usr/bin/python
"""
------------------------
    Euro Coin Recognition
------------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier
------------------------
"""
import cv2

def euro_detect(rgb):
    """Detect the euro coins from a RGB stream of a colored image and returns
    the detection results."""

    # Assign Coin class variables
    Coin.rgb = rgb
    Coin.gray = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
    Coin.hsv = cv2.cvtColor(rgb, cv2.COLOR_RGB2HSV)

    # Adaptive Thresholding
    gray_blur = cv2.GaussianBlur(Coin.gray, (15, 15), 0)
    thresh = cv2.adaptiveThreshold(gray_blur, 255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)


    # Circle detection
    circles = cv2.HoughCircles(thresh, cv2.HOUGH_GRADIENT, 1, 64,
                               param1=20, param2=40, minRadius=24,
                               maxRadius=96)

    coins = None
    if circles is not None:
        # Initialize coins using those detected circles
        coins = map(lambda c: Coin(c[0], c[1], c[2]), circles[0])


    return coins
```

9.2.2 Reconstructing Image for Demonstration

```
#! /usr/bin/python
"""
------------------------
    Reconstructing Image for Demonstration
------------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier
------------------------
"""
import numpy as np
import cv2
```

```
def demo(roi):
    coins = euro_detect(roi)
    height, width, depth = roi.shape
    circle_mask = np.zeros((height, width), np.uint8)

    if coins is not None:
        for c in coins:
            # Draw into the circle mask
            cv2.circle(circle_mask, (int(c.x), int(c.y)), int(c.r),
                1, thickness=-1)

    masked_data = cv2.bitwise_and(roi, roi, mask=circle_mask)

    if coins is not None:
        for c in coins:
            # Draw the outer circle
            cv2.circle(roi, (int(c.x), int(c.y)), int(c.r),
                (0, 255, 0), 2)
            # Draw the center of the circle
            cv2.circle(roi, (int(c.x), int(c.y)), 2, (0, 0, 255), 3)

            # Draw the hue sampling area
            cv2.rectangle(masked_data, (int(c.x - c.r*0.5), int(c.y - c.r*0.5)),
                (int(c.x + c.r*0.5), int(c.y + c.r*0.5)), (0, 255, 0), 1)
            # Draw the descriptive text
            cv2.putText(masked_data, "Hue " + str(c.feature['hue']),
                (int(c.x - c.r), int(c.y - c.r - 32)),
                cv2.FONT_HERSHEY_PLAIN, 1,(255,255,255), 1)
            cv2.putText(masked_data, "Saturation " +
                str(c.feature['saturation']),
                (int(c.x - c.r), int(c.y - c.r - 16)),
                cv2.FONT_HERSHEY_PLAIN, 1,(255,255,255), 1)
            for w in sorted(c.result['p'], key=c.result['p'].get, reverse=True):
                cv2.putText(masked_data, w + " (" +
                    "{:.2f}".format(c.result['p'][w] * 100) + "%) " +
                    "Z: {:.2f}".format(c.result['z'][w]),
                    (int(c.x - c.r), int(c.y - c.r)),
                    cv2.FONT_HERSHEY_PLAIN, 1,(255,255,255), 1)
                break

    cv2.imshow('Detected Coins', cv2.cvtColor(masked_data, cv2.COLOR_RGB2BGR))
    cv2.imshow('Video', cv2.cvtColor(roi, cv2.COLOR_RGB2BGR))
```

9.2.3 Dataset Collector Script

```
#! /usr/bin/python
"""
------------------------
    Dataset Collector
------------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier-training
------------------------
This script can be used to prepare the dataset for the training of the
classifier. This dataset collector script uses computer vision techniques to
recognize and segment euro coins from natural images, and output the
segmented coins into separate files.
This script takes natural images of euro coins as input, and process them into
segmented image dataset that is ready for the classifier training process.
------------------------
"""
```

```
import numpy as np
import cv2
import time
from sys import argv


FRAME_WIDTH = 747

def collect_euro_coin(img):
    """Collect the euro coins from an image."""

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

    # Adaptive Thresholding
    gray_blur = cv2.GaussianBlur(gray, (15, 15), 0)
    thresh = cv2.adaptiveThreshold(gray_blur, 255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)

    # Circle detection
    circles = cv2.HoughCircles(gray_blur, cv2.HOUGH_GRADIENT, 1, 64,
                               param1=20, param2=40, minRadius=24,
                               maxRadius=96)

    if circles is not None:
        i = 0
        for c in circles[0]:
            name = (str(int((time.time() - 1400000000) * 1000))[-4:] + '-'
                + str(int(time.clock() * 100000000000))[-4:] + '-' + str(i))
            print 'Writing to "' + 'output/'+ name +'.jpg"'
            print c
            cv2.imwrite('output/'+ name +'.jpg',
                img[int(c[1]-c[2]):int(c[1]+c[2]),
                int(c[0]-c[2]):int(c[0]+c[2])])
            i += 1


if __name__ == "__main__":
    # If this script is running as a standalone program, ask for data input
    if len(argv) > 1:
        for file_name in argv[1:]:
            print file_name
            img = cv2.imread(file_name)
            height, width, depth = img.shape
            roi = cv2.resize(img, (FRAME_WIDTH, FRAME_WIDTH * height / width))
            collect_euro_coin(roi)
        quit()
    option = raw_input('(V)ideo cam, or (L)oad an image file?')
    option = option.lower()
    if option == 'l':
        # Read from a file
        file_name = raw_input('Enter an image file name: ')

        img = cv2.imread(file_name)
        height, width, depth = img.shape
        roi = cv2.resize(img, (FRAME_WIDTH, FRAME_WIDTH * height / width))
        collect_euro_coin(img)


    elif option == 'v':
        # Start the video camera and collect results in real-time.
        cap = cv2.VideoCapture(0)

        while(True):
            ret, frame = cap.read()
            height, width, depth = frame.shape
```

```
            roi = cv2.resize(frame, (FRAME_WIDTH, FRAME_WIDTH * height / width))
            collect_euro_coin(roi)
            cv2.imshow('Video', roi)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        cap.release()
        cv2.destroyAllWindows()
```

9.2.4 Feature Extraction

```
#! /usr/bin/python
"""
-----------------------
    Feature Extraction
-----------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier-training
-----------------------
"""

import cv2

class Coin:
    """Coin class for a euro coin."""

    def __init__(self, src):
        """Initializes the euro coin object."""

        # If source is a dictionary, creating from cache
        if isinstance(src, dict):
            self.hue = int(src['hue'])
            self.saturation = int(src['saturation'])
            self.lightness = int(src['lightness'])
            self.luma = int(src['luma'])
            self.blue_diff = int(src['blue_diff'])
            self.red_diff = int(src['red_diff'])
            return

        # Else source is a bgr stream image
        bgr = src

        # Convert to YUV color space
        self.yuv = cv2.cvtColor(bgr, cv2.COLOR_BGR2YUV)
        # Contrast stretching, equalize the histogram of the Y channel
        self.yuv[:,:,0] = cv2.equalizeHist(self.yuv[:,:,0])
        # convert the YUV image back to RGB format
        self.bgr = cv2.cvtColor(self.yuv, cv2.COLOR_YUV2BGR)
        self.gray = cv2.cvtColor(self.bgr, cv2.COLOR_BGR2GRAY)
        self.hsv = cv2.cvtColor(self.bgr, cv2.COLOR_BGR2HSV)
        width, height, depth = img.shape
        self.r = width / 2
        self.__extract_features()

    def __extract_features(self):
        """Extract the features of this coin."""
        # Only use half of the coin area to determine it's center color
        r = self.r * 0.5

        # Hue, Saturation and Lightness
        roi = self.hsv[int(self.r-r):int(self.r+r), int(self.r-r):int(self.r+r)]
        if len(roi) > 0:
```

```
            self.hue = (sum([pixel[0] for rows in roi for pixel in rows])
                / len(roi) / len(roi[0]))
            self.saturation = (sum([pixel[1] for rows in roi for pixel in rows])
                / len(roi) / len(roi[0]))
            self.lightness = (sum([pixel[2] for rows in roi for pixel in rows])
                / len(roi) / len(roi[0]))

        # Luma and Chrominance
        roi = self.yuv[int(self.r-r):int(self.r+r), int(self.r-r):int(self.r+r)]
        if len(roi) > 0:
            self.luma = (sum([pixel[0] for rows in roi for pixel in rows])
                / len(roi) / len(roi[0]))
            self.blue_diff = (sum([pixel[1] for rows in roi for pixel in rows])
                / len(roi) / len(roi[0]))
            self.red_diff = (sum([pixel[2] for rows in roi for pixel in rows])
                / len(roi) / len(roi[0]))
```

… *(Other features are omitted to conserve space.)*

9.2.5 Data Exploration – Continuous Features

```
#! /usr/bin/python
"""
------------------------
    Continuous Features Data Exploration
------------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier-training
------------------------
"""
import numpy as np

class ContinuousFeature:
    """This class helps to analyze a continous feature"""

    columns = ['Count', 'Min', '1st Quart', 'Mean', 'Median', '3rd Quart',
        'Max', 'Std Dev']

    def __init__(self, data):
        self.count = len(data)
        self.minimum = (min(data))
        self.maximum = (max(data))
        self.mean = (sum(data) / float(len(data)))
        self.f_quartile = (np.percentile(data, 25))
        self.t_quartile = (np.percentile(data, 75))
        self.median = (np.median(np.array(data)))
        self.std_deviation = (np.std(data))
        self.hist, bin_edges = np.histogram(data, bins=range(256))
        self.hist = self.hist.tolist()

    def to_list(self):
        return [self.count, self.minimum, self.f_quartile, self.mean,
            self.median, self.t_quartile, self.maximum, self.std_deviation]

    def to_dict(self):
        d = {}
        d.update(vars(self))
        return d
```

… *(Other methods are omitted to conserve space.)*

9.2.6 Outlier Elimination – Clamp Transformation

```
#! /usr/bin/python
"""
------------------------
    Outlier Elimination - Clamp Transformation
------------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier-training
------------------------
"""
import numpy as np

class ContinuousFeature:
    """This class helps to analyze a continous feature"""
```

… (Other methods are omitted to conserve space.)

```
   def clean(self, coins, attr):
        """Clean the data using 3-sigma rules by applying clamp
        transformation."""
        return ContinuousFeature([
            self.median + self.std_deviation
            if getattr(c, attr) > (self.median + self.std_deviation) else
            self.median - self.std_deviation
            if getattr(c, attr) < (self.median - self.std_deviation) else
            getattr(c, attr)
            for c in coins.itervalues()])
```

… (Other methods are omitted to conserve space.)

9.2.7 Euro Coin Models Training

```
#! /usr/bin/python
"""
------------------------
    Euro Coin Models Training
------------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier-training
------------------------
"""
CLASSIFIER_COMMENT = """
    Euro Coin Detector Classifier
    Created by Chen Yumin (http://chenyumin.com)
////////////////////////////////////////////////////////////////////////////
 This software is provided by the copyright holders and contributors 'as is' and
 any express or implied warranties, including, but not limited to, the implied
 warranties of merchantability and fitness for a particular purpose are disclaimed.
 In no event shall the Intel Corporation or contributors be liable for any direct,
 indirect, incidental, special, exemplary, or consequential damages
 (including, but not limited to, procurement of substitute goods or services;
 loss of use, data, or profits; or business interruption) however caused
 and on any theory of liability, whether in contract, strict liability,
 or tort (including negligence or otherwise) arising in any way out of
 the use of this software, even if advised of the possibility of such damage.
////////////////////////////////////////////////////////////////////////////
```

```
"""

import numpy as np
import cv2
import os
import pandas
import json
import csv

classifier = {
    '__comment': CLASSIFIER_COMMENT.split("\n"),
    'author': 'Chen Yumin',
    'website': 'http://chenyumin.com',
    'name': 'Euro Coin Detector Classifier',
    'version': '1.0.0',
    'classification': {}
}

denominations = {'1c':'Xc/1c', '2c':'Xc/2c', '5c':'Xc/5c', 'Xc':'Xc',
    '10c':'X0c/10c', '20c':'X0c/20c', '50c':'X0c/50c', 'X0c':'X0c',
    '1e':'1e', '2e':'2e'}


for d in denominations:

    print('Processing "' + d + '" denomination...')

    coins = {}
    new_count = 0
    cache_count = 0

    # If cache file exists, read from cache
    try:
        with open('cache/' + d + '.csv') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                coins[row['filename']] = Coin(row)
    except:
        pass

    for root, dirs, files in os.walk('data/' + denominations[d]):
        for f in files:
            if root.startswith('data/.git'):
                # Skip .git folder
                continue
            if f in coins:
                # Skip cached data
                cache_count += 1
                continue

            file_name = os.path.join(root, f)
            #print('Processing ' + file_name + '...')
            img = cv2.imread(file_name)
            coins[f] = Coin(img)

            new_count += 1

    # Cache processed data
    with open('cache/' + d + '.csv', 'w') as csvfile:
        fieldnames = ['filename', 'hue', 'saturation', 'lightness',
            'luma', 'blue_diff', 'red_diff']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

        # Write the coins to cache
```

```
        for f in coins:
            writer.writerow({
                'filename': f,
                'hue': coins[f].hue,
                'saturation': coins[f].saturation,
                'lightness': coins[f].lightness,
                'luma': coins[f].luma,
                'blue_diff': coins[f].blue_diff,
                'red_diff': coins[f].red_diff
            })

    print("Gathered " + str(cache_count + new_count) + " items, " +
        str(cache_count) + " of which are from cache.")

    hue = ContinuousFeature([c.hue for c in coins.itervalues()])
    hue = hue.clean(coins, "hue")
    saturation = ContinuousFeature([c.saturation for c in coins.itervalues()])
    saturation = saturation.clean(coins, "saturation")
    lightness = ContinuousFeature([c.lightness for c in coins.itervalues()])
    lightness = lightness.clean(coins, "lightness")
    luma = ContinuousFeature([c.luma for c in coins.itervalues()])
    luma = luma.clean(coins, "luma")
    blue_diff = ContinuousFeature([c.blue_diff for c in coins.itervalues()])
    blue_diff = blue_diff.clean(coins, "blue_diff")
    red_diff = ContinuousFeature([c.red_diff for c in coins.itervalues()])
    red_diff = red_diff.clean(coins, "red_diff")

    result = [hue, saturation, lightness]

    classifier['classification'][d] = {
        'hue': hue.to_dict(),
        'saturation': saturation.to_dict(),
        'lightness': lightness.to_dict(),
        'luma': luma.to_dict(),
        'blue_diff': blue_diff.to_dict(),
        'red_diff': red_diff.to_dict()
        }

    # Generate Reports
    df = pandas.DataFrame([
        hue.to_list(),
        saturation.to_list(),
        lightness.to_list(),
        luma.to_list(),
        blue_diff.to_list(),
        red_diff.to_list()

        ],
    index=['Hue', 'Saturation', 'Lightness', 'Luma', 'Blue Difference',
        'Red Difference'])
    df.columns = ContinuousFeature.columns
    df.to_csv('./reports/' + d + '.csv')

with open('euro_coin_detector_classifier.json', 'w') as fp:
    json.dump(classifier, fp, indent=4, separators=(',', ': '),
        sort_keys=False)
```

9.2.8 Euro Coin Classification

```
#! /usr/bin/python
"""
------------------------
    Euro Coin Classification
------------------------
```

```
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier
-----------------------
"""
import numpy as np
import cv2
import math

class Coin:
    """Coin class for a euro coin."""

    def __init__(self, x, y, r):
        """Initializes the euro coin object."""
        self.x = x;
        self.y = y;
        self.r = r;
        self.result = {
            'p': {}, # the p value, probability
            'z': {}, # the z score
        }
        self.feature = {}
        self.__process_color()
        self.__classification()

    ...
    ... (Other methods are omitted to conserve space.)
    ...

    def __classification(self):
        """Classify the coins using their features."""
        for classification in Coin.classifier:
            z = {}
            for feature in self.feature:
                f = Coin.classifier[classification][feature]
                x = self.feature[feature]

                # Calculate the z-score of this feature
                z[feature] = ((x - float(f['median']))
                    / float(f['std_deviation']))

            # Euclidean distance
            z = sum(x**2 for x in z.itervalues()) ** 0.5

            # Convert the z-score to p value (probability)
            p = 1.0 + math.erf(-z / math.sqrt(2.0))

            self.result['z'][classification] = z
            self.result['p'][classification] = p
```

9.2.9 Demo App – Server

```
#! /usr/bin/python
"""
-----------------------
    Demo App -- Server
-----------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier-demo-server
-----------------------
```

```
"""

import base64
from PIL import Image
from StringIO import StringIO
import numpy as np
from flask import Flask, request
from flask_cors import CORS
from euro_detect import euro_detect
import json
app = Flask(__name__)
CORS(app)

def readb64(base64_string):
    sbuf = StringIO()
    sbuf.write(base64.b64decode(base64_string))
    pimg = Image.open(sbuf)
    return np.array(pimg)

@app.route("/", methods=['POST'])
def on_request():
    roi = readb64(request.data)
    circles = euro_detect(roi);
    return json.dumps(circles.tolist(), ensure_ascii=False)

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

9.2.A Demo App – Client

```
/*
------------------------
    Demo App -- Client
------------------------
Author: Chen Yumin (hello@chenyumin.com)
Web:    http://chenyumin.com
The full source code can be found on:
Github: https://github.com/chen-yumin/euro-coin-classifier-demo-client
------------------------
*/
angular.module('euro', ['ionic', 'ngCordova'])
  .controller('EuroDetectCtrl', function($scope, $http, $ionicActionSheet,
$cordovaCamera, $cordovaImagePicker, $ionicPopup, $ionicModal) {
    $scope.start = function() {
      $ionicActionSheet.show({
        buttons: [{
          text: '<i class="icon ion-camera dark"></i> From Camera'
        }, {
          text: '<i class="icon ion-image dark"></i> From Gallery'
        }],
        titleText: 'Please choose a picture',
        cancelText: 'Cancel',
        buttonClicked: function(index) {
          var result = null;
          console.debug(index);
          if (index == 0) {
            result = getPictureFromCamera();
          } else {
            result = getPictureFromGallery();
          }
          console.debug(result);
          if (result) {
            result.then(function(imageData) {
```

```
            $http({
              method: 'POST',
              data: imageData,
              url: '//147.252.144.246:5000'
            }).then(function successCallback(response) {
              console.debug('Success');
              console.debug(response);
              // this callback will be called asynchronously
              // when the response is available
              $scope.imageData = imageData;
              $scope.circles = response.data[0];
              console.debug($scope.circles);
              $scope.openResultModal();
            }, function errorCallback(response) {
              console.debug(response);
              $ionicPopup.alert({
                title: "Error",
                template: "There's an error connecting to the euro detection
server.",
                okText: "OK"
              });

            });
          }, function(err) {

          });


        }
        return true;
      }
    });
  }


  getPictureFromCamera = function() {

    if (typeof Camera == "undefined") {
      console.debug("No Camera available!");
      // @todo: localization
      return;
    }

    var options = {
      destinationType: Camera.DestinationType.DATA_URL,
      sourceType: Camera.PictureSourceType.CAMERA,
      encodingType: Camera.EncodingType.JPEG,
      allowEdit: false,
      targetWidth: 640,
      targetHeight: 640,
      quality: 80,
      popoverOptions: CameraPopoverOptions,
      saveToPhotoAlbum: false,
      correctOrientation: true
    };

    return $cordovaCamera.getPicture(options);
  };

  getPictureFromGallery = function() {

    return $cordovaImagePicker.getPictures({
        maximumImagesCount: 1
      });
```

```
  };

  $ionicModal.fromTemplateUrl('result-view-modal.html', {
    scope: $scope,
    animation: 'slide-in-up'
  }).then(function(modal) {
    $scope.resultModal = modal;
  });

  $scope.openResultModal = function() {
    $scope.resultModal.show();

    const img = new Image();

    img.onload = function() {
      canvas = document.getElementById('resultcanvas');

      const hwRatio = this.height / this.width;
       // Set canvas dimensions
      canvas.width = window.innerWidth;
      canvas.height = window.innerWidth * hwRatio;
      const ctx = canvas.getContext('2d');
      ctx.drawImage(this, 0, 0, canvas.width, canvas.height);

      const fRatio = canvas.width / this.width;
      ctx.strokeStyle="#FF0000";
      $scope.circles.forEach(circle => {
        var c = circle.map(function(num) {
          return num * fRatio;
        });
        //var c = circle.map(coordinate => {coordinate * hwRatio});
        console.debug("Drawing circle...");
        console.debug(c);
        ctx.beginPath();
        ctx.arc(c[0], c[1], c[2], 0, 2*Math.PI);
        ctx.stroke();
        ctx.closePath();
      });
    }

    img.src = '' + $scope.imageData;
  };
  $scope.closeResultModal = function() {
    $scope.resultModal.hide();
  };

  $scope.$on('$destroy', function() {
    $scope.resultModal.remove();
  });


})
```