Software Requirement Specification

AutoMart
dot slash

#YOCO
You Only Code Once

Schnetler Robin	Sinovcic Marko	Tanna Yashu
12220702	10693034	12076164
robin.schnetler@gmail.com	markosinovcic@gmail.com	tanna.yashu@gmail.com

October 19, 2014

1 Introduction

This document is the Software Documentation for the AutoMart system. The document will explain the problem that AutoMart is currently facing. AutoMart's vision to fix their problem will be explained and from which the scope of the system will be discussed.

2 Vision

Dot Slash^{TM} currently hosts a website for AutoMart for the sale of new and second hand vehicles, motorcycles and trucks. Users place their advertisements by entering textual data along with images of the vehicle onto the website. The system currently has an intelligent textual parser that parses the data to extrapolate relevant vehicle properties and create a valid advertisement. Once this parsing is complete, the advert is placed into a temporary database which is then manually approved to ensure that the advert complies with their terms and conditions.

AutoMart is currently facing spamming, poor quality images and misrepresentation problems. At the moment bots and users are able upload any images onto the website. Many of the adverts do not contain car images or contain car images of inferior quality.

Dot Slash has a vision of implementing a car recognition system that will prevent users from uploading any irrelevant or inferior quality images onto the AutoMart website. The subsystem needs to determine if there is a car in the image wanting to be uploaded. If there is car in the image, the make, the model and the colour of the car need to be identified. The subsystem also needs to generate a rating for the image.

Dot Slash has assigned #YOCO team to implement the subsystem.

3 Architecture Requirements

3.1 Access Channel Requirements

The system will be deployed as a .dll library that will be linked to a web service hosted by an IIS.NET server. The web service communicates via requests and responses sent between the server and the client. The #YOCO utility will be accessible from any browser on desktop.

3.2 Quality Requirements

3.2.1 Performance

The system must be used in real time when a user creates an advert for a vehicle that they wish to sell. The system must therefore process and classify the image in a maximum of 2 seconds per feature (car, image quality, colour, contradictions between images), and a further 2 seconds to generate a rating, based on the above results, for the image.

The #YOCO utility makes use of a Pipes and Filters architectural pattern.

The car detection is done through OpenCV's API. The first run of this API is at maximum 10 seconds, after which all libraries are already in cache. Thereafter the classification takes maximally 2 seconds per classifier. There are 3 different classifiers for front, side and back views of a car.

The blur detection algorithm (which is essential in determining the overall quality of the image) is self implemented and works per pixel. With a normalised image size of 480 x 320 pixels, this results in 153 600 pixels to query. We will need to ensure that all pixel information is gathered in a reliable and efficient manner. The algorithm that we use to get all pixel information (such as colour) guarantees a linear representation of the image. The algorithm execution time is maximally 2 seconds.

3.2.2 Reliability

The system will classify the feature at least 8 out of 10 times.

The OpenCV framework suggests usage of atleast 5000 images when classifying faces. This has worked and historical data proves that this had been effective. Our system will be using 5000 images for each characteristic.

3.2.3 Flexibility

To allow for classification of other object in the future(make detection for future development), all that is required is an XML file representing the AI knowledge base.

3.2.4 Maintainability

We are using SOLID design principles and have implemented the pipes and filters architectural pattern to classify images. If a new feature is required, our system has ensured that it is easy to plug in the new functionality by means of creating a new module that implements the Filter interface into the existing framework with minimal hassle.

It is also essential that the information obtained about the car from the image is accurate.

3.3 Trade-offs

In order to achieve processing speeds of 2 seconds per feature in the image as well as a further 2 seconds to generate a rating for the image, a lot of memory will be needed for buffering so that a lot of data is immediately at hand. This requires a large amount of memory as the images used in the system will require sizeable chunks of RAM. A lot of processing power will be used since there is a lot of processing that happens on the image. Thus, to achieve these speeds, some memory and processing power will have to be sacrificed.

To ensure that the system correctly classifies features 80% of the time a lot of training will be needed in order for the system to achieve this level of accuracy. Thus this will take time, hence to achieve accuracy a lot of time will have to be sacrificed to train the program. As mentioned, approximately 2500 images are required for every feature that needs to be trained. The training process is also very memory intensive as the training process has been adapted to use 4GB of memory. Processing power required for the training also needs to be considered.

Lastly, flexibility will be achieved by ensuring the system can be trained for each classification. This again requires a lot of training which, as above, is very time consuming and memory intensive as well as needing string processing power.

3.4 Integration Requirements

The final product needs to integrate with the current AutoMart web server. To achieve this,an API will be provided documenting all of the system functions as well as a .dll library that will integrate with the web server and be called whenever a user uploads an image.

3.5 Architecture Constraints

The #YOCO utility must be written in C# using the .NET framework.

4 Functional Requirements

4.1 Introduction

4.1.1 Scope

The #YOCO utility should be implemented within the current AutoMart system and should not replace it. The subsystem needs to provide features that will help AutoMart reduce the number of poor and irrelevant advertisements on their website.

In this regard, the utility acts as a firewall that blocks bad quality images from entering the database while also validating that images that do pass as good quality, are well represented in the advertisement.

The system determines the image validity and quality by performing car detection, car coverage, blur and colour detection. Car detection is performed first as further tests and analysis require car location in the image in order to perform their own analysis. Car detection needs to check if the image contains a car. If the image contains a car, detection rating is set to one, and if no car is found detection rating is set to zero. Further tests are independent and can be conducted in any order.

Car coverage calculates the area the car occupies in the image. Depending on the coverage percentage a value between zero and two is assigned to the coverage rating. Blur detection determines the image blurriness. Depending on the blurriness percentage a value between zero and two is assigned to the blur rating. Colour detection determines the colour of the car. AutoMart currently has sixteen colour buckets that are used to represent the colour of the car. The system calculate which colour bucket is the closest to the car's colour.

Finally the final image rating is calculated by adding detection rating, coverage rating and blur rating. The image rating and colour of the car are returned to the client side where further decisions can be performed based on clients preference.

4.1.2 Limitations

The system will not always be able to determine the correct colour of the car as aspects such as lighting, shadow and angle make it difficult to determine the colour correctly.

4.1.3 Exclusions

The system should only be implemented to work with cars and should not cater for the existence of any other vehicles such as motorcycles and trucks. The system is not able to determine the make and the model of the car. The recognition of different makes of the cars could be implemented in the future. However the recognition of models is not as no possible solution was found.

4.2 Required Functionality

4.3 Use Cases

Get Image Details

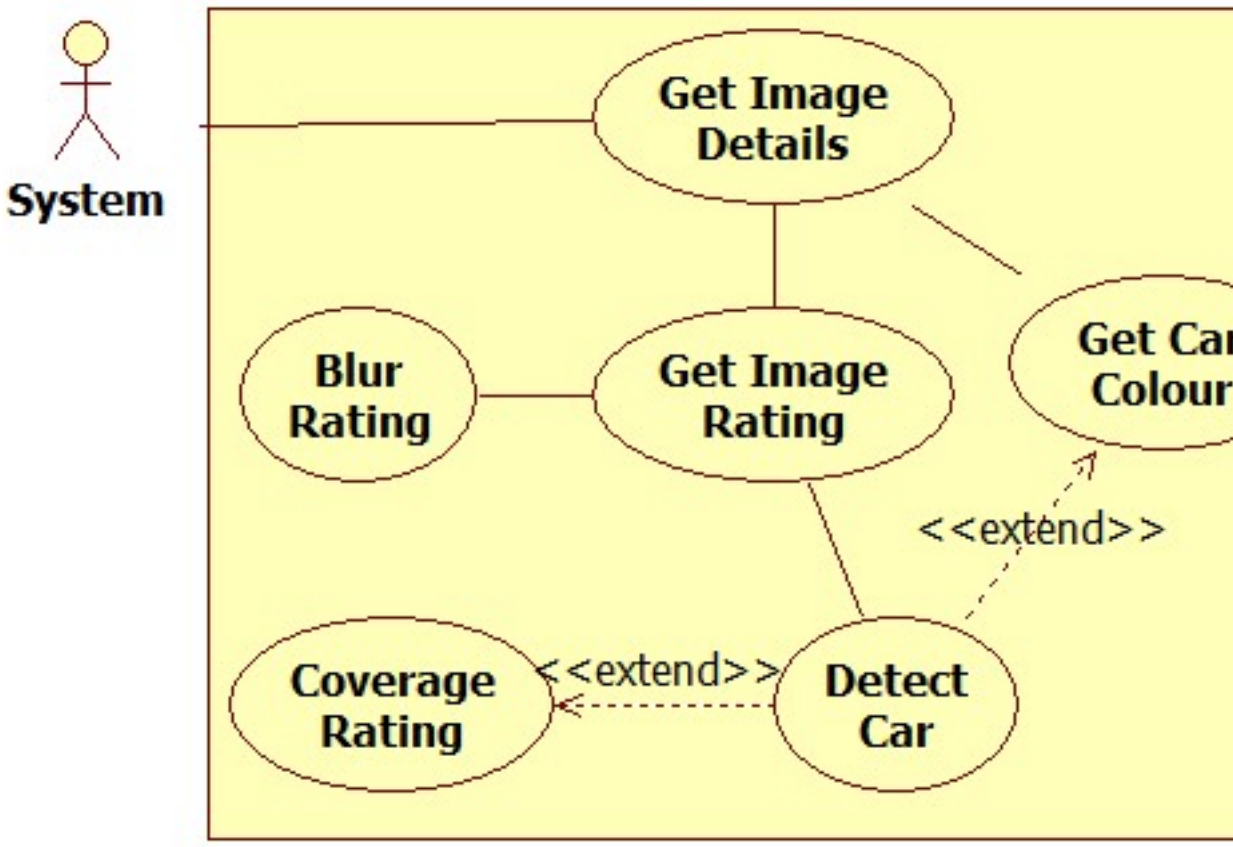
Find Bounding Box	
Analyse the image and return its details.	
Priority	Critical
Pre-Conditions	A valid image needs to be provided.
Post-Conditions	The image details acquired and returned.

Describe Car Subsystem

Determine Colour	
Determines the colour of the car and finds the closest 3 colour buckets.	
Priority	Important
Pre-Conditions	A car needs to exist in the bounding box.
Post-Conditions	Colour(s) of the car determined.

Calculate Blur	
Stretches the bounding box to 480x320 pixels and calculates and returns the blur value.	
Priority	Important
Pre-Conditions	Bounding box exists.
Post-Conditions	Blur value successfully calculated and returned.

High Level Use Case Diagram.



Calculate Coverage	
Calculates the coverage the car occupies in the photo.	
Priority	Important
Pre-Conditions	Bounding box exists.
Post-Conditions	Car coverage value successfully calculated and returned.

4.4 Process Specifications

Figure 1: Activity diagram of FindBoundingBox