1. INTRODUCTION

1.1 Overview

The proliferation of multimedia content has raised the need for automatic content-based indexing and information retrieval systems. Textual information in video and images proves to be a source of high-level semantics closely related to the concept of the video. There exist mainly two kinds of text occurrences in videos, namely artificial and scene text. Artificial text is artificially added in order to describe the content of the video or given additional information related to it. This makes it highly useful for building keyword indexes. Scene text is textual content that was captured by camera as part of scene such as text on T-shirts or road signs. Scene text can appear in any kind of surfaces, in any orientation and perspective and often under occlusion, making its extraction particularly difficult. Moreover scene text usually brings less related to video information.

In Figure 1, green boxes denote artificial text while red boxes bound the scene text. Text can also be classified into normal or inverse. Normal is called any text whose characters have lower intensity values than the background while inverse text is the opposite. In Figure 2 “EURO” is inverse while “SPORT” is normal text.

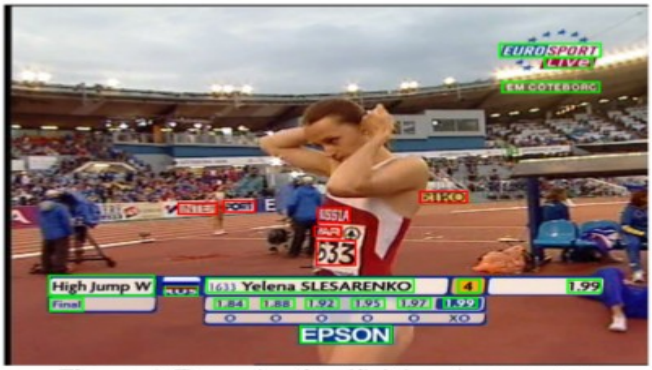


Figure 1 Example of artificial and scene text

Figure 2 Example of inverse and normal text

Fig: 1. Examples of a) Artificial and scene text
b) Inverse and Normal text

Although current visual search technologies have reached a certain level of maturity, they have largely ignored a class of informative features often observed in images: text. In fact, text is particularly interesting because it provides contextual clues for the object appearing inside an image. Given the vast number of text-based search engines, retrieving an image using the embedded text offers an efficient supplement to the visual search systems .

The procedure of textual information extraction from images is usually split into three steps: detection, segmentation and recognition. This project investigates the usage of Haar Classifier in the detection and segmentation stage.

2. SYSTEM ANALYSIS

The system analysis involves the identification of the objectives and requirements, evaluation of alternative solution and recommendation for a more feasible solution. New system must be developed for the organization to remain effective and competitive.

The objectives of system analysis are:

- Identifying the need
- Analysis the existing and proposed system
- Evaluating the feasibility study
- Perform economical and technical analysis
- Identifying the hardware and software requirements
- Allocating functions to the hardware and software.
- Creating system software

  The above objectives are amalgamated and further sectioned as below.

2.1 Proof Of Concept

Proof of concept (abbreviated PoC) is a short or incomplete realisation of a certain method or idea to demonstrate its feasibility or a demonstatration in principle whose purpose is to verify that the same concept or theory is probably capable of being useful. In this project, PoC targeted was to extract the text in frame obtained from video using Edge Detection algorithm and Haar Classifier. The efficiency of Haar Classifier (based on Viola Jones Algorithm) system is well known in the scientific community. The extrapolation of the same to text recognition is a novel idea.

2.2 Scope and Study

In recent years, visual search systems have been developed for applications such as product recognition and landmark recognition. In these systems, local image features are extracted from images taken with a camera-phone and are matched to a large database using visual word indexing techniques . The most reliable among the existing systems are using the Edge Detection and making use of neural nets. This system, though gains 70% accurate result, the recall ability of the system is not up to the benchmark.

The proposed system involves three stages:

a) Image extraction from Video

b) Text Detection, Segmentation and Recognition

c) Text Extraction

2.3 Software Specifications

2.3.1 Python

Python is a general purpose, high-level language whose design philosophy emphasizes code readibility. Python claims to be “remarkable power with very clear syntax”. And its standard library is large and comprehensive.

Python is a multi- paradigm programming language that is often applied in scripting roles. It is commonly defined as an object-oriented scripting language—a definition that blends support for Object Oriented Programming with an overall orientation toward scripting roles . Rather than forcing programmers to adopt a particular style of programming, it permits several styles. Along with OOP, structured programming is also fully supported, and there are a number of language features which support functional and aspect-oriented programming(including by meta programming and by magic metods). Many other paradigms are supported by the extensions pyDBC and Contracts for Python which allow design by contrat. The reference implementation of Python (Cpython) is free and open source software and has a community-based development model, as do mostly all of its alernative implementations. Cpython is managed by the non-profit Python Software Foundation.

2.3.2. OpenCV (Open Computer Vision Library)

OpenCV [OpenCV] is an open source (BSD License) cross-platform computer vision library The library is written in C and C++ and runs under Linux, Windows and Mac OS X. There is active development on interfaces for Python, Ruby, Matlab, and other languages. OpenCV was designed for computational efficiency and with a strong focus on real- time applications.

OpenCV automatically uses the appropriate IPP library at runtime if that library is installed. One of OpenCV’s goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The

OpenCV library contains over 500 functions that span many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Because computer vision and machine learning often go hand-in- hand, OpenCV also contains a full, general-purpose Machine Learning Library (MLL). This sub library is focused on statistical pattern recognition and clustering. The MLL is highly useful for the vision tasks that are at the core of OpenCV's mission, but it is general enough to be used for any machine learning problem.

2.4 Requirements

2.4.1. Hardware requirements

Processor: Core2Duo

RAM : 1 GB

Hard Disk: 500 GB

2.4.2. Software requirements

Language used : Python (v 2.7.3)

Operating System : Linux OS

Compiler: GNU Compiler

Library: OpenCV

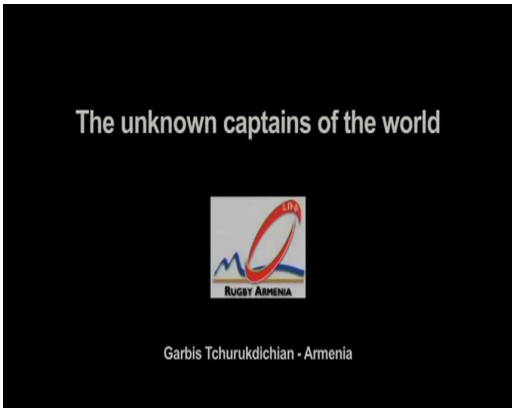
3. DEVELOPMENT AND IMPLEMENTATION

As mentioned above, the entire process was divided into 3 segments. The processes within each segment is explained below.

3.1 Image Extraction

Video is composed of frames with normal frame rate ranging from 24 fps (frames per second) to 29 fps. OpenCV provides us the option to work with the existing video files and videos captured from Webcam or any other compatible devices. There is in-built function to read and play '.avi' format videos. It is also possible to process videos of any other format.

The initial step of extracting the videos into frames approximately at the rate of 1 frame per second was written and compiled. The sample images , along with the essential background details of the obtained at the end of extraction are furnished below:



(a)

(b)

(d)

3.1 Frames from the Video : “unknowncaptains.flv” (fps: 29, Duration: 1:01)

3.2. Text Detection, Segmentation and Recognition

Locating text within an image/ video is a challenging task due to the wide variety of text appearances, such as variations in font and style, geometric and photometric distortions, partial occlusions, and different lighting conditions. Text can be detected by exploiting the discriminate properties of text characters such as the vertical edge density, the texture or the edge orientation variance.

Text detection has been considered in many recent studies and numerous methods are reported in the literature. These techniques can be classified into two categories: texture-based and connected component (CC)-based. Texture-based approaches view text as a special texture that is distinguishable from the background. Typically, features are extracted over a certain region and a classifier (trained using machine learning techniques or by heuristics) is employed to identify the existence of text. As opposed to texture-based methods, the CC-based approach extracts regions from the image and uses geometric constraints to rule out non-text candidates. The top scoring contestant in applies an adaptive binarization method to find CCs. Text lines are then formed by linking the CCs based on geometric properties.

In this Project, exploits the fact that text lines produce strong vertical edges horizontally aligned and follow specific shape restrictions. Using edges as the prominent feature of our system gives us the opportunity to detect characters with different fonts and colours since every character present strong edges, despite its font or color, in order to be readable .

Two are the main steps here, text area detection and text line detection, applied in a multi resolution manner. In the first step, an edge map is created using the Canny edge detector.

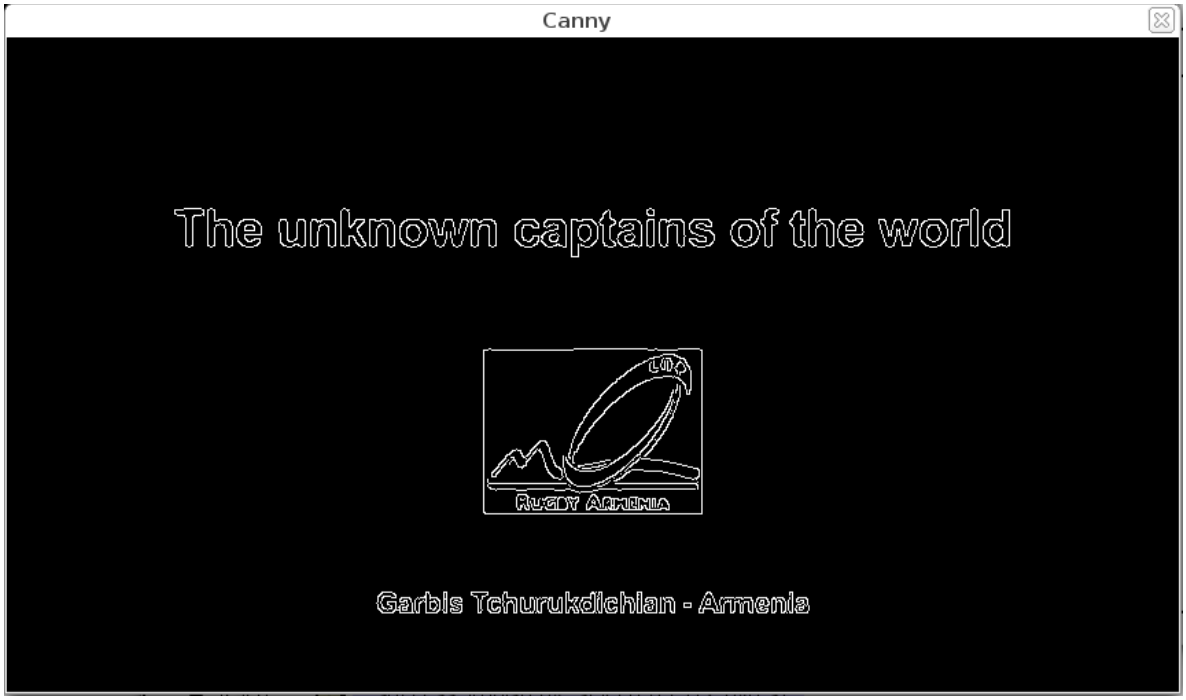


Fig: 3.2.1 Original, Greyscale and Canny Edge Detected version of image 3.1(a)

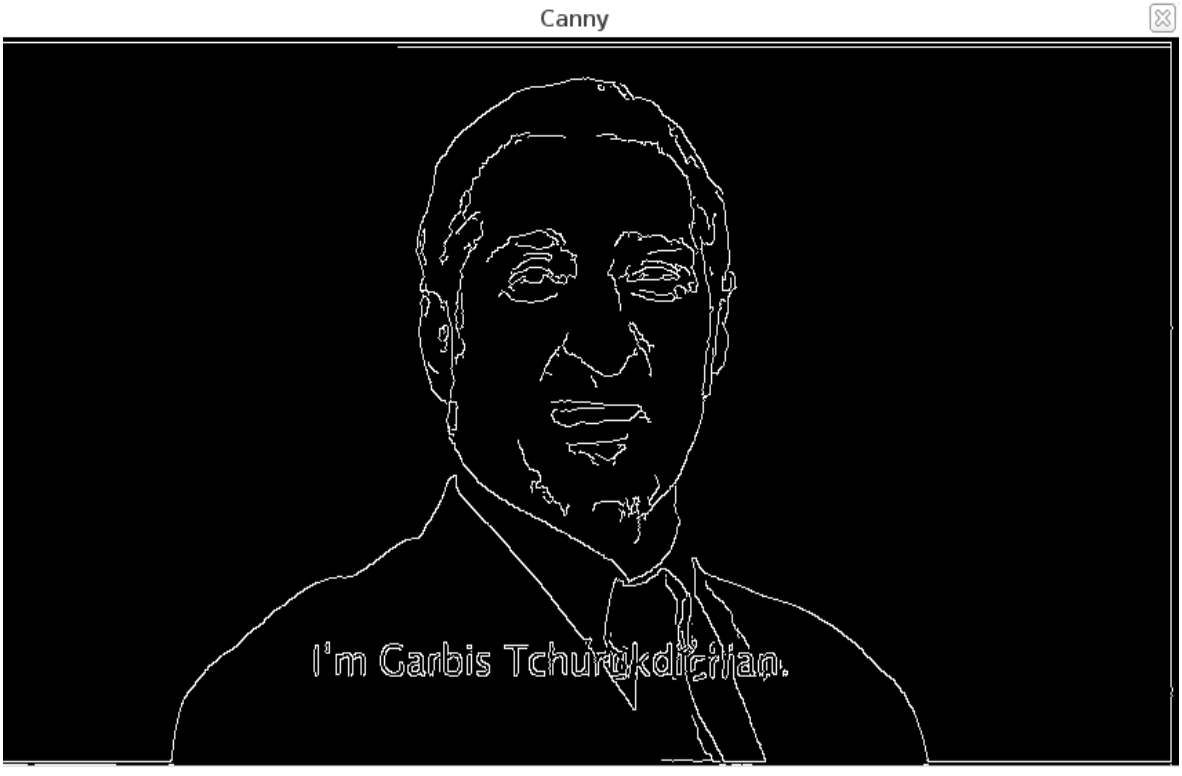


Fig: 3.2.2 Original, Greyscale and Canny Edge Detected version of image 3.1(b)

Fig: 3.2.3 Original, Greyscale and Canny Edge Detected version of image 3.1(b)

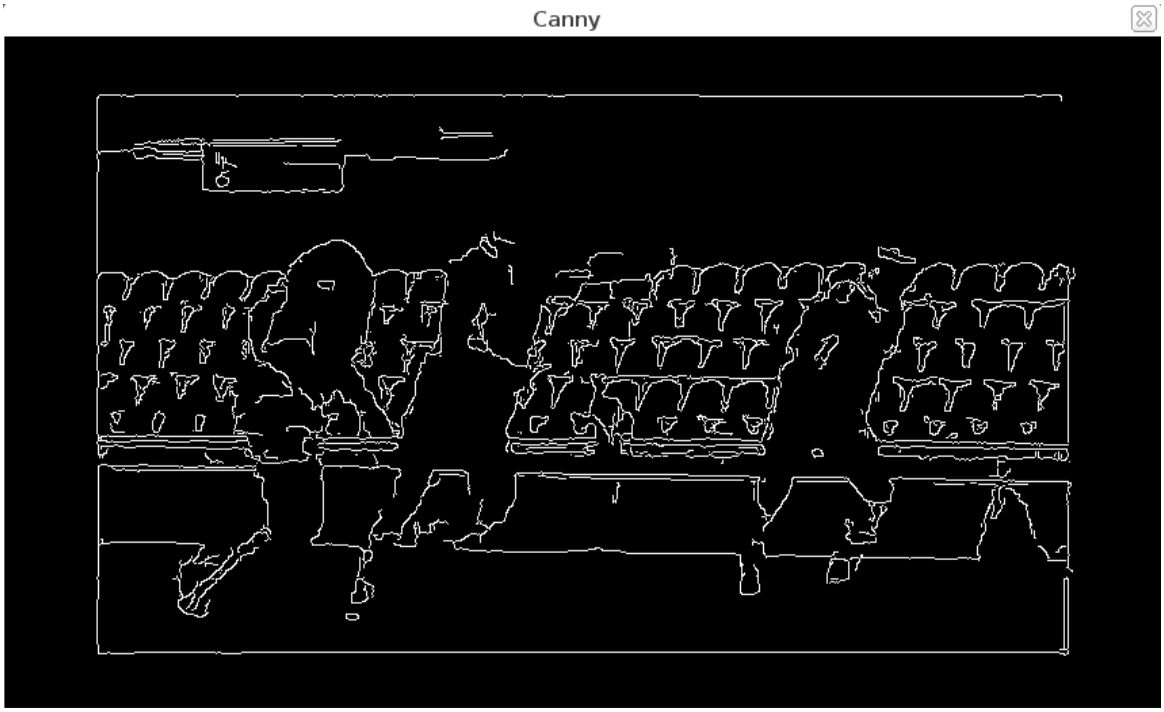


Fig: 3.2.4 Original, Greyscale and Canny Edge Detected version of image 3.1(d)

Then, morphological dilation and opening are used in order to connect the vertical edges and eliminate false alarms. Bounding boxes are determined for every non-zero valued connected component, consisting the initial candidate text areas. Finally, edge projection analysis is applied,

4.RESULT

5. CONCLUSION AND FUTURE SCOPE

6. APPLICATIONS

7. REFERENCES

APPENDIX :

A.

 # Python program to implemet the slicing of Video into frames : “capture.py”

```
from cv2.cv import *          % importing cv module of OpenCV library
capture = CaptureFromFile("unknowncaptains.flv")  % initialising video file
if capture:  %  task carried out if initialising is successful
 i=0
 f=1
 while(f):
        f=0
        f = QueryFrame(capture)
        if f:
         p= GetCaptureProperty(capture, CV_CAP_PROP_FPS);
         c= int(p)
         if( i%c ==0):
          SaveImage( "ku" + str(i/c)+ ".jpg",f)
         i=i+1
```

B.

 # Python program to implemet the slicing of Video into frames : “image_canny.py”

```
from cv2.cv import *
# read image
newImg= LoadImage("ku65.jpg")
NamedWindow("Gray")
grayImg= CloneImage(newImg)
grayImg = CreateImage(GetSize(newImg),IPL_DEPTH_8U,1 );
CvtColor(newImg, grayImg, CV_BGR2GRAY );
cannyImg = CreateImage(GetSize(newImg), IPL_DEPTH_8U, 1);
Canny(grayImg, cannyImg, 50,150, 3);
NamedWindow("Original");
```

```
NamedWindow("Canny");
ShowImage( "Original", newImg );
ShowImage("Gray",grayImg)
ShowImage("Canny",cannyImg)
WaitKey(0)
DestroyWindow("Original")
DestroyWindow("Gray")
DestroyWindow("Canny")
```