International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

An Approach to Image Extraction and Accurate Skin Detection from Web Pages

Moheb R. Girgis, Tarek M. Mahmoud, and Tarek Abd-El-Hafeez

***Abstract*—**This paper proposes a system to extract images from web pages and then detect the skin color regions of these images. As part of the proposed system, using BandObject control, we built a Tool bar named "Filter Tool Bar (FTB)" by modifying the Pavel Zolnikov implementation. The Yahoo! Team provides us with the Yahoo! SDK API, which also supports image search and is really useful. In the proposed system, we introduced three new methods for extracting images from the web pages (after loading the web page by using the proposed FTB, before loading the web page physically from the localhost, and before loading the web page from any server). These methods overcome the drawback of the regular expressions method for extracting images suggested by Ilan Assayag. The second part of the proposed system is concerned with the detection of the skin color regions of the extracted images. So, we studied two famous skin color detection techniques. The first technique is based on the RGB color space and the second technique is based on YUV and YIQ color spaces. We modified the second technique to overcome the failure of detecting complex image's background by using the saturation parameter to obtain an accurate skin detection results. The performance evaluation of the efficiency of the proposed system in extracting images before and after loading the web page from localhost or any server in terms of the number of extracted images is presented. Finally, the results of comparing the two skin detection techniques in terms of the number of pixels detected are presented.

***Keywords*—**Browser Helper Object, Color spaces, Image and URL extraction, Skin detection, Web Browser events.

I. Introduction

THE first project to address the accessible browser issue was initiated in 1997 under the supervision of Jon Gunderson. It introduced the concept of using Microsoft's Component Object Model (COM) objects and the C++ programming language to interface with Internet Explorer (IE). Most groups used the Active Template Library to simplify the programming of COM objects, the Microsoft Foundational Classes to develop their window managers, and Microsoft Visual Studio to develop their code [13].

Browser Helper Object (BHO) is an IE Accessible, meaning that it runs in the background whenever IE runs. It has the ability to monitor user commands and manipulate what IE downloads and displays. BHOs are used to essentially create a new web browser without having to recreate IE's existing features, or encountering the incompatibility that normally comes with using a standard browser. The toolbar functionality of IE Accessible was created more to aid the developers and users that can use a mouse to navigate through web pages. The optional toolbar allows the users to fully customize any components of IE Accessible they want to be represented with buttons in the toolbar. This allows for an easier access to different components in IE Accessible.

The primary goal of IE Accessible is to make a web browser usable by people with disabilities that make using IE difficult. This can be achieved with two strategies: more options in IE's controls, and more options in IE's display. The idea is to give the user as much control as possible using only the keyboard. Display options are intended to let users specialize web page design for their needs. Whereas standard IE is primarily focused on giving options to designers about how to display a page, control of the same options to the user will be given. The secondary goal of IE Accessible is to help designers make their pages more accessible to general users (including those who are not using IE Accessible). HTML contains many elements, which the typical user will never see, but they are essential for people who can't see the standard images or table layouts [4,5]. BHO will give designers a quick way to check things like table headers and image descriptions.

With the development of WWW, the dramatically falling cost of data storage and the advancing in coding technology are generating dazzling array of photography, animation, graphics, sound and video. Filter the adult images is very important for search engines to avoid offensive content on the web. Now, there are some ways to stop naked images arriving on computers, such as blocking unwanted sites, identifying images depicting naked or scantily dressed people. "X-Stop" is a tool to block pornographic sites. It provides the parents with necessary method to safeguard their children who are using Internet. Another way to identify pornographic images is by means of text analysis or computer vision. Forsyth and Fleck designed software to detect naked people [18]. This software begins by analyzing the color and texture of a photograph. When it finds matches for skin colors, it runs an algorithm that looks for cylindrical areas that might be corresponds to an arm or leg. It then seeks other flesh-colored cylinders, positioned at certain angles, which might confirm the presence of limbs.

Skin color has proven to be a useful and robust cue for face detection, localization and tracking. Image content filtering, contentaware video compression and image color balancing applications can also benefit from automatic detection of skin in images. Numerous techniques for skin color modelling and recognition have been proposed during several past years. Most existing skin segmentation techniques involve the classification of individual image pixels into skin and non-

Moheb R. Girgis is with the Computer Science Dept., Faculty of Science, Minia University, El-Minia, Egypt (e-mail: mrgirgis@mailer.eun.eg).

Tarek M. Mahmoud is with the Computer Science Dept., Faculty of Science, Minia University, El-Minia, Egypt (e-mail: tarek_2ms@yahoo.com).

Tarek Abd-El-Hafeez is with the Computer Science Dept., Faculty of Science, Minia University, El-Minia, Egypt (e-mail: tarek_1_2@yahoo.com).

skin categories on the basis of pixel color [24]. It is not an easy task to extract regions of specific color from a given color image, since the color of an object varies with changes in illumination color, illumination geometry, and miscellaneous sensor parameters [25]. A few papers comparing different approaches have been published [21], [22], [23].

The proposed system is a Browser Helper Object, which runs in the background whenever the IE runs. It reads the web page as an HTML document, extracts all images and links (URLs) from it and saves them before and after displaying the web page. We compared the number of extracting images before and after loading the web page from the localhost and any server. The system toolbar contains a button called "skin recognition" that can be used to detect the skin color region of human images. We studied two skin color images detection techniques. The first one is based on the RGB color space and the second is based on the YUV and YIQ color spaces. We modified the second technique to overcome the failure of detecting complex image's background by using the saturation parameter to obtain an accurate skin detection results. This system is a part of a large system that is being developed to filter undesirable images from web pages.

This paper is organized as follows: section II describes the Internet Explorer methods, events, and prosperities. Section III describes HTML Screen Scraping. In section IV, the Regular Expression for Extracting URLs is presented. Section V describes extracting images after loading the web page by using our tool bar. Section VI describes extracting images before loading the web page from the Localhost. In section VII, we describe extracting images before loading the web page from any server. Filter Tool Bar (FTB) interface is described in section VIII. Skin recognition techniques are described in section IX. Finally, the experimental results are presented in section X.

II. The Web Browser Properties, Methods and Events

To display an HTML code, we need a web browser. IE is one of the most commonly used web browsers and it has a decent API. So we will use IE to display our reports. The *WebBrowser* control fires a number of different events to notify user applications —and browser— about the generated activity. When the browser is about to navigate to a new location, it triggers a *BeforeNavigate2* event that specifies the URL or path of the new location and any other data that will be transmitted to the Internet server through the http transaction. The data can include the http header, http post data, and the URL of the referrer. The *BeforeNavigate2* event also includes a flag that can be set to cancel the pending navigation request. This event can be useful for checking the requested URL against a database of unauthorized World Wide Web sites or local and network folders, and for canceling the navigation request. The *WebBrowser* control fires the *NavigateComplete2* event after it has navigated to a new location. This event includes the same information as *BeforeNavigate2*, except *NavigateComplete2* does not include the cancel flag.

Whenever the browser is about to begin a download operation, it triggers the *DownloadBegin* event. The control also generates a number of *ProgressChange* events as the operation progresses, and then it triggers the *DownloadComplete* event after completing the operation. Applications typically use these three events to indicate the progress of the download operation, often by displaying a progress bar. An application would show the progress bar in response to *DownloadBegin*, update the progress bar in response to *ProgressChange*, and hide the progress bar in response to *DownloadComplete*. More details about the properties, methods and events of the web browser can be found in [9].

In the proposed system, a new instance of the IE is created. Then we make this instance visible. The events of this instance are determined. After this step, the specified home page is navigated and the images are extracted. Fig. 1 gives the steps of the proposed system which runs in the background of the IE.

```
public void Run()
{
  1- Start the browser.
  2- Set the browser events.
  3- Go to the specified home page and extract  images.
  4- Start navigating to different URL.
}
```

Fig. 1 The proposed system steps

III. HTML Screen Scraping

Screen Scraping means reading the contents of a web page. When we visit any site, such as yahoo.com, we see the interface which includes buttons, links, images, …, etc. What we don't see is the target URL of the links, the name of the images, the method used by the button which can be POST or GET. To extract an image from a web page, we create an *HttpWebRequest* object from the Image *URL*, which is basically a class that manages the request to the given URL. If we're behind a proxy, then we will have to create a *WebProxy* object and assign it to wReq's *Proxy* property. Calling *GetResponse* will get the resulting response from the request we made. What we're really interested in is the data stream of the URL, which we obtain from the *GetResponseStream* method of the *HttpWebResponse* class. Once we have the stream, it's a simple matter to create an Image object from the data stream. Also, by default the *HttpWebRequest* method is *GET*. At this point, nothing has hit the network because an object needs to be created in order to set properties before the request is initiated.

The next step is to issue the request and receive the response from the destination. This is done by calling the *GetResponse* method, which returns a *WebResponse* object that can be cast to an *HttpWebResponse* object if we need to access the HTTP-specific properties [10].

Screen Scraping pulls the HTML code of the web page. This HTML code includes every HTML tag that is used to make up the page. Extracting images from the web page using the proposed system is based on the following steps:

```
public Image GetImage(string URL)
{
 1- Create a web request to the URL containing the image.
 2- Gets the response from the web request.
 3- Return the image stream from the URL specified earlier.
}
```

Fig. 2 Extracting image steps

IV. Extracting Information from the Retrieved HTML Code using Regular Expressions

We can easily view the HTML code that was generated when a request for a web page was made, by just viewing the source code of the web page. In the IE, we select View -> Source or click the "view source" on the proposed FTB described in section VIII. The notepad will open with the complete HTML code generated of the page. This HTML code is quite complex. It will be really useful if we can extract out all the links from the generated source to upgrade the list of undesirable URLs automatically. First we need to introduce a regular expression that can extract all URLs from the generated HTML code. There are many regular expressions already made which can be found in [11]. The regular expression would look like this:

```
Regex r = new Regex("((((http[s]?|ftp)[:]//)([a-zA-Z0-9.-]+([:][a-zA-Z0-9.&amp;%$-
    ]+)*@)?[a-zA-Z][a-zA-Z0-9.-]+|[a-zA-Z][a-zA-Z0-9]+[.][a-zA-Z][a-zA-Z0-9.-
    ]+)[.](com|edu|gov|mil|net|org|biz|pro|info|name|museum|ac|ad|ae|af|ag|ai|al|a
    m|an|ao|aq|ar|as|at|au|aw|az|ax|ba|bb|bd|be|bf|bg|bh|bi|bj|bm|bn|bo|br|bs|bt|bv|
    bw|by|bz|ca|cc|cd|cf|cg|ch|ci|ck|cl|cm|cn|co|cr|cs|cu|cv|cx|cy|cz|de|dj|dk|dm|do|d
    z|ec|ee|eg|eh|er|es|et|eu|fi|fj|fk|fm|fo|fr|ga|gb|gd|ge|gf|gg|gh|gi|gl|gm|gn|gp|gq|gr
    |gs|gt|gu|gw|hk|hm|hn|hr|ht|hu|id|ie|il|im|in|io|iq|ir|is|it|je|jm|jo|jp|ke|kg|kh|ki|k
    m|kn|kp|kr|kw|ky|kz|la|lb|lc|li|lk|lr|ls|lt|lu|lv|ly|ma|mc|md|mg|mh|mk|ml|mm|mn|
    mo|mp|mq|mr|ms|mt|mu|mv|mw|mx|my|mz|na|nc|ne|nf|ng|ni|nl|no|np|nr|nu|nz|o
    m|pa|pe|pf|pg|ph|pk|pl|pm|pn|pr|ps|pt|pw|py|qa|re|ro|ru|rw|sa|sb|sc|sd|se|sg|sh|s
    i|sj|sk|sl|sm|sn|so|sr|st|sv|sy|sz|tc|td|tf|tg|th|tj|tk|tl|tn|to|tp|tr|tt|tv|tw|tz|ua|ug|uk|u
    m|us|uy|uz|va|vc|ve|vg|vi|vn|vu|wf|ws|ye|yt|yu|za|zm|zw)([:][0-9]+)*(/([a-zA-Z0-
    9.,;?'\\+&amp;%$#=~_-]+)*)",RegexOptions.IgnoreCase |
    RegexOptions.Compiled);
```

Using the regular expression, the URLs are extracted from the web page by clicking the "Extract Links" at the "Filter Tool Bar", and displayed as shown in Fig. 3.

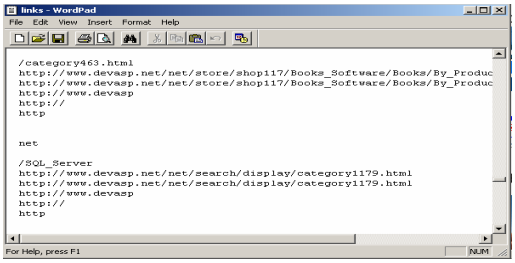


Fig. 3 The URLs which are extracted from the web page when we click the "Extract Links" in the FTB

The HTML code is parsed to extract the number of available results for the user query as well as information about the retrieved images. Ilan Assayag [12] used Google patterns that accurately allow us to locate each of these interesting information bits in the HTML code after analyzing it. But the problem with these patterns is that if Google changes the format of the returned HTML code the parsing will fail!

To overcome this problem, we introduced three new methods to extract images before and after loading the web page as described in sections 5, 6, and 7.

Following are the different patterns used in the API:

1. *Regex imagesRegex = new Regex(@"(\x3Ca\s+href=/imgres\" + @"x3Fimgurl=)(?<imgurl>http" + @"[^&>]*)([>&]{1})" + @"([^>]*)(>{1})(<img\ssrc\"@"x3D)""{0,1})(?<images> /images" +@"[^""\s>]*)([\s])+(width=)"+@"(?<width>[0-9,]*)\s+ (height=)" + @"(?<height>[0-9,]*)");*

This pattern is used to retrieve information about each image. The URL of the original image is captured into the "imgurl" group, the URL of the thumbnail is captured into the "images" group and the width and height of the thumbnail image are captured in the "width" and "height" groups, respectively.

2. *Regex dataRegex = new Regex(@"([^>]*)(>)\s{0,1}(
) {0,1}\s {0,1}" + @"(?<width>[0-9,]*)\s+x\s+(?<height>[0-9,]*)" + @"\s+pixels\s +-\s+(?<size>[0-9,]*)(k)");*

This pattern is used to retrieve additional information about each image - the original images' widths, heights and sizes (in groups "width", "height" and "size", respectively). Ilan Assayag could not find a way to use the same pattern for all the images' information.

V. Extracting Images After Loading the Web Page Using the Filter Tool Bar

To stream images from the web, we need to use classes from the *System.Net* and *System.IO* classes, and we need an Internet access as well. Fig. 4 shows the steps of extracting image after loading the web page by clicking on the "Image Data" of the FTB.

```
protected void ImageData(ButtonEntry frm_in)
 {
  1- Attach the Internet Explorer document.
  2- Get the current URL.
  3- Call GetImage method and save its data.
  4- Display the Image Data Window as shown in Fig. 10.
 }
```

Fig. 4 The steps of extracting image after loading the web page by clicking on the "Image Data" of FTB

VI. Extracting Images Before Loading the Web Page from the Localhost Server

The *System.DirectoryServices* namespace provides access to the Active Directory. The classes in this namespace can be used with any of the Active Directory service providers including Internet Information Services (IIS), the Lightweight Directory Access Protocol (LDAP), the Novell Directory Services in NetWare (NDS), and WinNT.

The core of this Active Directory Service is creating an instance of the *DirectoryEntry* object. This object can be used to manipulate any Active Directory Entry stored in an IIS, LDAP, NDS, or NT Metabase. In order to understand the syntax for binding to IIS objects, we need to understand the

components of the paths to the IIS Metabase. The metabase is organized in a hierarchical structure that mirrors the structure of our IIS installation. Each node in the metabase structure is called a key, and each key can contain one or more IIS configuration values, called metabase properties.

Before manipulating entries on IIS with *DirectoryEntry* class we need to know basic ideas about how IIS manage these entries. Each IIS entry (web site, web virtual directory, ftp virtual directory,..) has a schema name and a collection of properties. All entries follow a hierarchy, with parent entries and children entries. This hierarchy is defined by its metabase path, which means that every entry has a corresponding physical path. To instantiate a *DirectoryEntry* we basically need its metabase path, as follows:

```
DirectoryEntry directoryEntry = new
            DirectoryEntry("IIS://localhost/w3svc/1/root");
```

Indexers should access the most of the *DirectoryEntry* properties.

```
Console.Write( directoryEntry.Properties["Path"].Value.ToString() );
```

Let's look at the following IIS Metabase path and break it down into its key components:

IIS://ComputerName/WebService/Server/Root/VirtualDirectoryName

IIS://	Lets the DirectoryEntry Object know that we are dealing with an IIS directory rather than an LDAP or NDS entry.
ComputerName	Simply reflects the Web server we will be accessing. This can be a name or IP address.
WebService	Can be W3SVC which indicates that we are dealing with the Web service or MSFTPSVC, which would indicate we wanted to use the FTP service [1,2,3,7].
Server	Is typically an integer value that allows us to attach to different Web servers on the same server, each with its own unique IP: Port Address combination. Typically the default Web server has a value of 1.
VirtualDirectory Name	Is the name of our virtual directory?

The steps of extracting and saving all images of the current web page before loading it from the localhost are as follows:

```
static void LocalHost( )
 {
    1- Read the contents of the web page.
    2- Read the HTML code into an html document to enable parsing
    3- Get Image Data from the HTML document..
    4- foreach (HTMLImgClass image in doc.images)
      {
       4.1  Get the Virtual path of the image.
       4.2  Convert the virtual path to the physical path.
       4.3   Save image data.
      }
 }
```

Fig. 5 The steps of extracting all images before loading the web page from the localhost

VII. Extracting Images Before Loading the Web Page from Any Server

The *WebClient* class in the *System.NET* namespace has two pretty neat methods that let us download data from the internet:

- public byte[ ] DownloadData(string address) – downloads to an in-memory buffer.
- public void DownloadFile(string address, string fileName) – downloads to the hard drive.

We used the *DownloadFile* method to download data from the URI specified by the address parameter to a local file. Fig. 6 shows the steps of extracting all images of the current web page before loading it from any server.

```
static void Extract_Image( )
 {
    1- Read the contents of the web page.
    2- Read the HTML code into an HTML document to enable parsing
    3- Get Image Data from the HTML document..
    4- foreach (HTMLImgClass image in doc.images)
      {
       4.1  Download  the image from the server.
      }
 }
```

Fig. 6 The steps of extracting all images before loading the web page from any server

VIII. Filter Tool Bar

As shown in Fig. 7, the proposed FTB contains six buttons with captions “Extract Links”, “Thumbnail Image”, “Skin Recognition”, “View Source”, “Image Data”, and “Internet Options”. Fig. 10 shows the IE browser with the proposed FTB. The purpose of the “Extract links” button is to extract all URLs in the current web page and save these links after loading the web page. The elapsed time of the extraction process is computed after clicking this button. The purpose of “Thumbnail Image” button is to display all the extracted images in the web page as a photo album, as shown in Fig. 11. The purpose of “Skin Recognition” button is to recognize the skin in every image of the current web page. We will use this button to determine the skin percentages of each image of the current web page. Some filter techniques can be used with these images to prevent the undesirable images from displaying. The purpose of the “view source” button is to display the HTML source code of the current web page. The purpose of the “Image Data” button is to display the Image Data window as shown in Fig. 12. Finally, the purpose of the “Internet Options” button is to display the Internet Options window.

Fig. 7 The “Filter Tool Bar” buttons

IX. Skin Color Detection

Skin color detection is a very important step in many vision systems, like gesture recognition, hand tracking, video indexing, region of interest, face detection, etc. Pixel based skin detection can narrow the search space prior to high-level

layers, however this is not an easy task. Skin pixels can vary with ambient light, such as color lamps acting as filters, brightness and specularities, shadows, daylight, etc. Since different cameras return different values for the same scene, pixel-based skin detection becomes a cumbersome task [17].

The main goal of skin color detection is to build a decision rule that will discriminate between skin and skin pixels. Identifying skin colored pixels involves finding the range of values for which most skin pixels would fall in a given color space. The purpose of a color space is to facilitate the specification of colors in some standard, generally accepted manner. A color space is a specification of a coordinate system and subspace within a system where each color is represented by a single point. Most color spaces today are oriented toward hardware such as color monitors or toward applications where color manipulation is a goal such as the creation of color graphics or animation [16]. Various color spaces are used for processing digital images. For some purposes, one color space may be more appropriate than others. For skin detection, researchers do not quite agree on whether the choice of color space is critical to the overall performance of the detection system. One obvious reason for this is the lack of a standard set of images that can be used as a benchmark for algorithms using different color spaces. This may be as simple as explicitly classifying a pixel as a skin pixel if R>G or R>B or both [19] or may be as complex as models using neural networks and Bayesian methods [20]. In general, a good skin color model must have a high detection rate and a low false positive rate. That is, it must detect most skin pixels while minimizing the amount of skin pixels classified as skin. This section presents two techniques for detecting skin color as a first step to filter undesirable images.

A. The RGB Skin Detection Technique

The RGB color space is one of the most widely used color spaces for storing and processing digital image. However, the RGB color space alone is not reliable for identifying skin-colored pixels since it represents not only color but also luminance. Skin luminance may vary within and across persons due to ambient lighting so it is not dependable for segmenting skin and skin regions. Chromatic colors are more reliable and these are obtained by eliminating luminance through some form of transformation.

In the RGB space each color appears in its primary spectral component of red, green, and blue. Images represented in the RGB space consist of three component images, one for each primary color. When fed into an RGB monitor, these images combine on the phosphor screen to produce a composite color image [16]. The RGB True Color is an additive color system based on tri-chromatic theory. It is one of the most commonly used color spaces, with a lot of research activities being based on it. Therefore, skin color is classified by heuristic rules that take into account two different conditions: uniform daylight and flash or lateral illumination. The chosen skin cluster for RGB is [14]:

(R,G,B) is classified as skin if:

$R > 95$ and $G > 40$ and $B > 20$
$\max\{R,G,B\} - \min\{R,G,B\} > 15$
$|R-G| > 15$ and $R > G$ and $R > B$,

In case of flashlight or daylight lateral illumination:

(R,G,B) is classified as skin if:
R>220 , G>210 , B>170
| R-G | ≤ 15 , B<R , B<G.
where $R,G,B = [0 .. 255]$.

B. YUV - YIQ Skin Detection Technique

Duan et al. [15] convert the pixels' value from RGB to YUVand YIQ respectively. The RGB values are transformed into YUV values using the formulation.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1471 & -0.2888 & 0.4359 \\ 0.6148 & -0.5148 & -0.1000 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The chromaticity information is encoded in the U and V components. Hue and saturation are obtained by the following transformation.

$$ch = \sqrt{|U|^2 + |V|^2} \quad , \quad \theta = \tan^{-1}(|V|/|U|) \ * (180 / 3.14)$$

Where θ represents hue, which is defined as the angle of vector in YUV color space. *Ch* represents saturation, which is defined as the modulus of U and V. Like YUV color space, YIQ is the color primary system adopted by NTSC for color TV broadcasting. Conversion from RGB to YIQ may be accomplished using the color matrix:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.2990 & 0.5870 & 0.1140 \\ 0.5957 & -0.2745 & -0.3213 \\ 0.2115 & -0.5226 & 0.3111 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

where I is the red-orange axis, and Q is roughly orthogonal to I. The less I value means the less blue-green and the more yellow.

Duan et al. [15] found that the combination of YUV and YIQ color space is more robust than each other. The most people's skin varies in the range from 20 to 90 in the term of I.

Fig. 8 illustrates the result of the implementation of Duan et al. used to detect skin color.

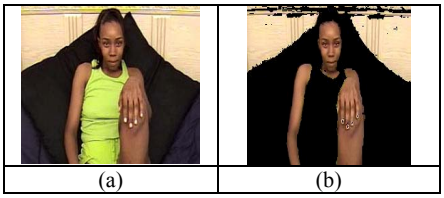

(a) (b)

Fig. 8 (a) Original image (b) Result with Duan et al. method

The drawback of Duan et al. method is that, if the image background contains pixels similar to human skin pixels (i.e. $20 \leq I \leq 90$) and these pixels doesn't belong to the skin region, the method fails to detect it (see Fig. 8).

To overcome this drawback we use the saturation parameter (ch). More accurate detected skin can be produced by varying the saturation parameter as shown in Fig. 9.

If we change the value of saturation, we get various values of images under test. Fig. 9 shows the original image and different skin images by changing the saturation values:

Fig. 9 (a) original image (b) Result with ch ≥ 20
(c) Result with ch ≥ 30 (d) Result with ch ≥ 40

We found that, If a pixel satisfies the condition $I \in [20,90]$ *and hue* $\in [30,75]$ *and saturation* $\in [20,220]$ it is possible to be relevant to skin color.

X. Experimental Results

The main purpose of the proposed system is extracting the images and URLs from the current web page and then detecting the skin color regions of these images. In our implementation, images can be extracted before and after loading the web page from the localhost or any server. Using the FTB, we can obtain different data about the extracted images. The implementation of the proposed system is based on C#. Net.

As shown in Fig. 11, the image thumbnail of the extracted images can be obtained after clicking the "Thumbnail Image" button in the FTB.

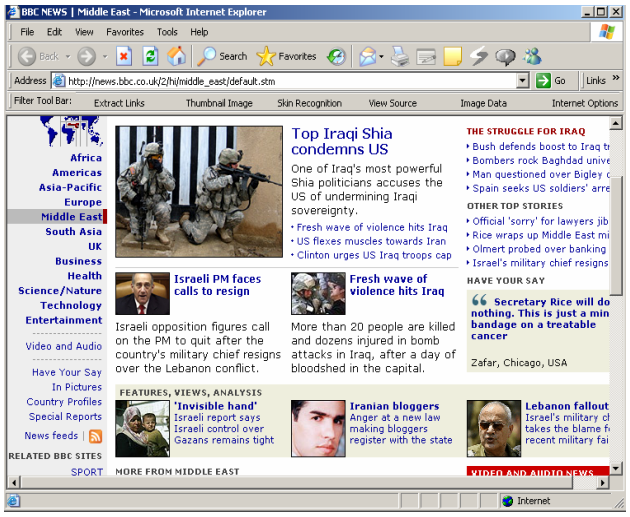


Fig. 10 The IE browser with the proposed FTB showing the web page to be analyzed

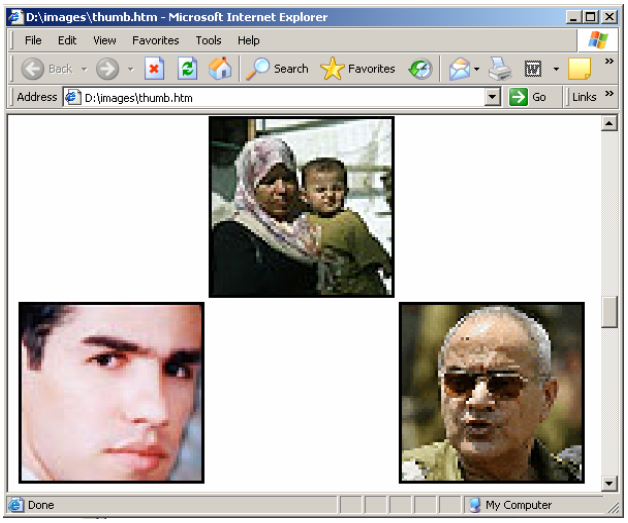


Fig. 11 The image thumbnail of the extracted images

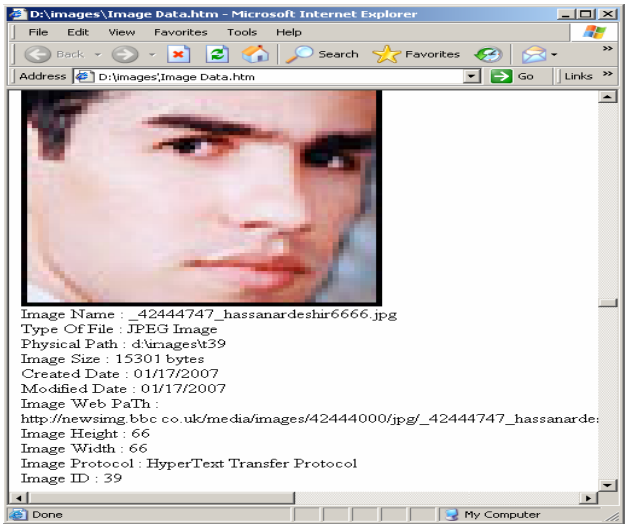


Fig. 12 The data of the extracted image after loading using FTB

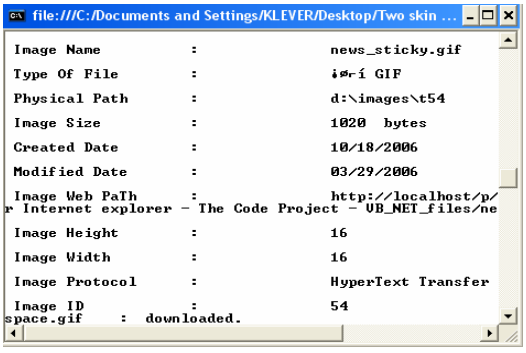


Fig. 13 The data of the extracted image before loading the web page

All image data of the extracted images, such as image name, type, physical path …etc., can be obtained after loading the web page. Fig. 12 shows the image data obtained after

clicking the "Image Data" button. Fig. 13 shows the data of every extracted image before loading the web page. It will appear automatically at the IE background.

A. Image Extracting Results

Firstly, we present the performance evaluation of the proposed system in terms of the number of the extracted images before and after loading the web page. The performance evaluation was carried out on 8 URLs using localhost and different web servers.

TABLE I
THE NUMBER OF EXTRACTED IMAGES FROM THE LOCALHOST AND ANY SERVER BEFORE AND AFTER LOADING THE WEB PAGE

URL	No. of Images	Percentage & No. of extracted images after loading	Percentage & No. of extracted images before loading	Average After	Average before
http://localhost/codeproject/codeproject.htm	8	7 (87.5%)	7 (87.5%)		
http://localhost/yahoo/yahoo.htm	18	18 (100%)	18 (100%)	294	296
http://localhost/cnn/cnn.htm	280	235 (98.07%)	258 (99.2%)	(98%)	(98.6%)
http://localhost/microsoft/microsoft.htm	14	14 (100%)	13 (92.8%)		
http://www.yahoo.com	18	16 (88.8%)	17 (90.9%)		
http://www.codeproject.com	8	8 (100%)	7 (87.5%)	288	295
http://www.microsoft.com	14	14 (100%)	14 (100%)	(96%)	(98.3%)
http://www.cnn.com	260	250 (96.1%)	257 (98.8%)		
Total Number of Images	600	582 (97%)	591 (98.5%)		

In Table I, the first four URLs are representing the localhost results and the last four URLs are representing the results of any server. As can be seen in Table I:

1. The number of extracted images *after* loading the web page from the *localhost* is more than the number of extracted images *after* loading the web page from *any server*.

2. The number of extracted images *before* loading the web page from the *localhost* is more than the number of extracted images *before* loading the web page from *any server*.

3. The total number of extracted images before loading the web page from the localhost and any server is more than the number of extracted images after loading the web page from the localhost and any server.

In our proposed filtering system, we will employ the method for extracting images before loading the web page from any server, which is more relevant to the main purpose of the system.

B. Skin Recognition Results

Finally, we present a comparison between the two skin recognition techniques described in section IX. For each method we present a table showing the number of skin pixels, the total number of pixels, the skin percentage, and the skin pixels for each image. Each technique is applied on a wide variety of images taken under different lighting conditions and with different backgrounds.

The images also have areas containing skin from other parts of the body like hands, and areas with color very similar to that of skin. These areas get classified as skin and they form skin regions accordingly. Table II describes the first technique (YUV-YIQ) results for some variety images. Each row contains the results of an image data.

TABLE II
THE FIRST TECHNIQUE (YUV-YIQ) WITH CH ≥ 30 RESULTS FOR VARIOUS IMAGES

Skin Pixels	Non– skin pixels	Total number of pixels	Skin percentage	Non - skin percentage
2373	7655	10028	23.66	76.33
2721	7307	10028	27.13	72.86
896	9774	10670	8.397	91.6
4694	7406	12100	38.79	61.2
5515	6585	12100	45.57	54.4
8226	3874	12100	67.983	32.016
5445	7055	12500	43.56	56.44
5941	6559	12500	47.528	52.472
2406	10398	12804	18.79	81.20
3864	9816	13680	28.24	71.75
1249	12751	14000	8.92	91.07
4131	10299	14430	28.62	71.37
3856	10904	14760	26.12	73.87
2866	12374	15240	18.80	81.19
4025	12004	16029	25.110	74.88
4119	11910	16029	25.69	74.30
5993	10036	16029	37.38	62.61
7349	10801	18150	40.49	59.50
13	19187	19200	0.067	99.93
1515	17685	19200	7.8	92.10
1707	17493	19200	8.89	91.109
4432	17348	21780	20.348	79.65
13499	10801	24300	55.55	44.44
299	28501	28800	1.038	98.96
6706	22094	28800	23.28	76.71
7688	21112	28800	26.69	73.305
12634	17559	30193	41.84	58.15
18113	28523	46636	38.83	61.16
2107	4533	6640	31.73	68.26
17095	59705	76800	22.25	77.74

In Table III, the second technique (RGB) results for the same set of images are presented.

TABLE III
THE SECOND TECHNIQUE (RGB) RESULTS FOR THE SAME IMAGES CONSIDERED IN TABLE II

Skin Pixels	Non – skin pixels	Total number of pixels	Skin percentage	Non - skin percentage
1640	8388	10028	16.3	83.6
2565	7463	10028	25.57	74.42
1074	9596	10670	10.06	89.93
2680	9420	12100	22.14	77.85
4094	8006	12100	33.83	66.16
5932	6168	12100	49.02	50.97
3707	8793	12500	29.656	70.344
5443	7057	12500	43.544	56.456
2330	10474	12804	18.19	81.80
3319	10361	13680	24.261	75.738
1496	12504	14000	10.68	89.31
3861	10569	14430	26.75	73.2
3196	11564	14760	21.653	78.3468
2711	12529	15240	17.78	82.21
2463	13566	16029	15.365	84.634
2480	13549	16029	15.47	84.528
4341	11688	16029	27.082	72.
5560	12590	18150	30.633	69.36
26	19174	19200	0.13	99.864
1200	18000	19200	6.25	93.75
1579	17621	19200	8.223	91.776
2614	19166	21780	12.0018	87.99
12300	12000	24300	50.61	49.38
289	28511	28800	1.00	98.99
4543	24257	28800	15.77	84.22
4939	23861	28800	17.14	82.850
10912	19281	30193	36.1408	63.859

Skin Pixels	Non – skin pixels	Total number of pixels	Skin percentage	Non - skin percentage
12989	33647	46636	27.85	72.148
1596	5044	6640	24.0361	75.967
16505	60295	76800	21.490	78.50

Fig. 14 shows sample of accurate results of the (YUV-YIQ) with ch ≥ 30 and (RGB) techniques. Fig. 15 shows sample of inaccurate results of both techniques. The first column shows the original images, the second column shows the skin images using YUV-YIQ technique, and the last column shows the skin images using RGB technique.

As can be seen in Table II, III and Figs. 14, 15:

1- Using a single color space may limit the performance of the skin color filter. This study shows that better performance can be achieved by using two or more color spaces.

2- Contrary to some studies, experimental results in this study indicate that chrominance values still significantly differ for light and dark skin even if the luminance component is removed through a normalization or transformation of RGB values. Thus, a skin color model should not treat light and dark skin in the same manner. This may be more computationally expensive but it allows for more skin pixels to be labeled correctly.

3- The YUV-YIQ technique is more efficient than the RGB technique in terms of skin pixels detected.

Accordingly, we will employ the modified (YUV-YIQ) technique for skin color detection in our proposed system for detecting and filtering undesirable images.

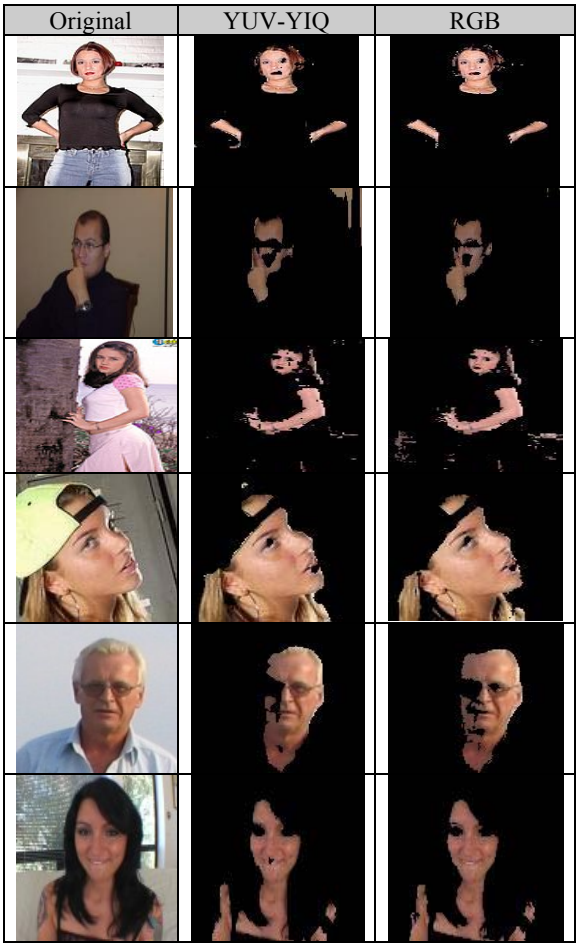
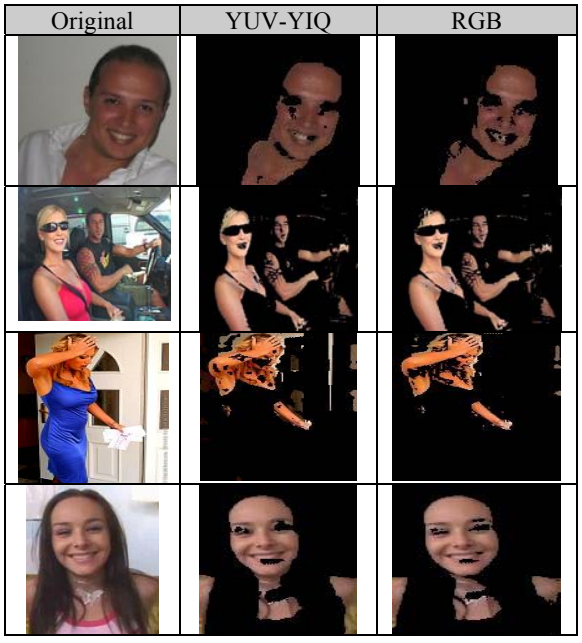
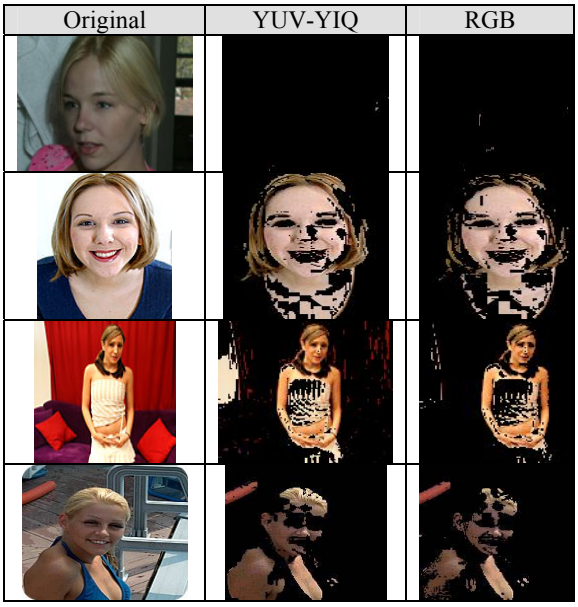


Fig. 14 Samples of accurate results of the (YUV-YIQ) and (RGB) techniques





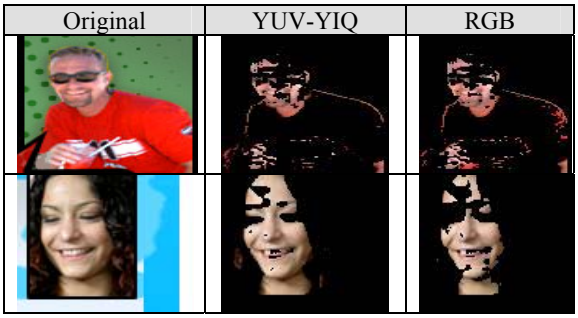


Fig. 15 Samples of inaccurate results of the (YUV-YIQ) and (RGB) techniques

XI. Conclusion

This paper suggests a system containing two major parts. The purpose of the first part is detecting and extracting images and URLs from web pages. A new Filter Tool Bar is created in the Internet Explorer interface. We presented three new methods for extracting the images from the web page. These methods overcome the drawback of the regular expression method suggested by Ilan Assayag. The performance evaluation of the proposed system in terms of the number of extracted images before and after loading the web page from local host and any server is presented. The purpose of the second part of this system is detecting skin color. So, we studied two famous skin detection techniques for color images. The first technique is based on the RGB color space and the second technique is based on YUV and YIQ color spaces. We modified the second technique to overcome the failure of detecting complex image's background by using the saturation parameter to obtain accurate results. Finally, the results of the two skin detection techniques, (YUV-YIQ) and (RGB), are presented. Future extension of this work includes more studies on the images background and lighting similar to skin color pixels to increase the accuracy of the skin color detection. Another extension of this work consists of using the proposed system as a filter to prevent the undesirable images and URLs of any web page from displaying.

References

[1] Postel, J., "Simple Mail Transfer Protocol", RFC 821, USC/Information Sciences Institute, August 1982.
[2] Postel, J., and Reynolds, J., "File Transfer Protocol (FTP), RFC 959, USC/Information Sciences Institute, October 1985.
[3] Postel, J., and Reynolds, J., "TELNET Protocol Specification", RFC 854, USC/Information Sciences Institute, May 1983.
[4] Postel, J.,"Media Type Registration Procedure", RFC 1590,USC/Information Sciences Institute, March 1994. URL:ftp://ds.internic.net/rfc/rfc1590.txt>
[5] Borenstein, N., and Freed, N., "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC1521, Bellcore, Innosoft, September 1993. URL:ftp://ds.internic.net/rfc/rfc1521.txt>
[6] Fielding, R.,"Relative Uniform Resource Locators", RFC 1808, June 1995 <URL:ftp://ds.internic.net/rfc/rfc1808.txt>
[7] Frank, H., and Mayer, S., "The Dexter Hypertext Reference Model", Communications of the ACM, pp. 30-39, vol. 37 no. 2, Feb 1994.
[8] http://msdn.microsoft.com/library/default.asp? url=/workshop/ browser/webbrowser/reflist_vb.asp
[9] http://www.microsoft.com/isapi/redir.dll?prd=ie&pver =6&ar=msnhome
[10] Anthony, J., Jim, O., and Lance, O., "Microsoft Network Programming for the Microsoft .NET Framework", Published By Microsoft Press, A Division of Microsoft Corporation, One Microsoft Way Redmond, Washington 98052-6399, Copyright © 2004.
[11] http://www.regexlib.com/
[12] Ilan, A., "An API for Google Image Search", http://www.codeproject.com.
[13] http://slappy.cs.uiuc.edu/fall03/team2/Final/.
[14] P. Peer, F. Solina, "An automatic human face detection method", in *Proc. 4th Computer Vision Winter Workshop (CVWW)*, Rastenfeld, Austria, Feb. 1999, pp. 122-130.
[15] Duan, L., Cui, G., Gao, W., and Zhang, H., "Adult image detection method base-on skin color model and support vector machine". In Asian Conference on computer Vision, pages 797-800, Melbourne, Australia, 2002.
[16] Gonzales R. and Woods R. E., "Digital Image Processing," Prentice Hall, Inc, New Jersey, 2002.
[17] Brand J., Mason J. S., Roach M., Pawlewski M.."Enhancing face detection in colour images using a skin probability map". Int. Conf. on Intelligent Multimedia, Video and Speech Processing, pp. 344-347, 2001.
[18] Forsyth D. A.. Fleek M., and Bregler C.. "Finding naked people". In Proc.Forth European Conference on Computer Vision. pp 593-602. 1996.
[19] Brown, D., Craw, I., & Lewthwaite, J." A SOM Based Approach to Skin Detection with Application in Real Time Systems". In *Proc. Of the British MachineVision Conference,* 2001.
[20] Chai, D. & Bouzerdoum, A. "A Bayesian Approach to Skin Color Classification in YCbCr Color Space". In *Proc. Of IEEE Region Ten Conference*, vol. 2, 421- 4124, 1999.
[21] Zarit, B. D., Super, B. J., and Quek, F. K. H. "Comparison of Five Color Models in Skin Pixel Classification". In ICCV'99 Int'l Workshop on recognition, analysis and tracking of faces and gestures in Real-Time systems, 58–63, 1999.
[22] Terrillon, J.-C., Shirazi, M. N., Fukamachi, H., and Akamatsu, S. "Comparative Performance Of Different Skin Chrominance Models and Chrominance Spaces for The Automatic Detection of Human Faces in Color Images". In Proc. of the International Conference on Face and Gesture Recognition, 54–61, 2000.
[23] Brand, J., and Mason, J. "A Comparative Assessment of Three Approaches to Pixel level Human Skin-Detection". In Proc. of the International Conference on Pattern Recognition, vol. 1, 1056–1059, 2000.
[24] Phung, S. L., Bouzerdoum, A. and Chai, D. "Skin Segmentation Using Color Pixel Classification: Analysis and Comparison" , IEEE Tran. On Pattern Analysis and Machine Intelligence, Vol. 27, No. 1, Jan. 2005.
[25] Cho, K. M., Jang, J. H. and Hong, K. S. "Adaptive Skin Color Filter", Pattern Recognition, Vol. 34, pp. 1067-1073, 2001.