MULTILINGUAL ACCESSIBILITY OF WEB IMAGES FOR PERSONS WITH VISUAL IMPAIRMENTS USNG STEGANOGRAPHY (product)

Submitted by

KATHIRVEL K
(Register No. 20394011)

Under the guidance of

Dr K S KUPPUSAMY

Project phase 2 report submitted for the degree of

MASTER OF TECHNOLOGY

IN

NETWORK AND INFORMATION SECURITY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF ENGINEERING AND TECHNOLOGY

PONDICHERRY UNIVERSITY

PUDUCHERRY 605014

MAY 2022

MULTILINGUAL ACCESSIBILITY OF WEB IMAGES FOR PERSONS WITH VISUAL IMPAIRMENTS USNG STEGANOGRAPHY (product)

Submitted by

KATHIRVEL K
(Register No. 20394011)

Under the guidance of

Dr K S KUPPUSAMY

Project phase 2 report submitted for the degree of

MASTER OF TECHNOLOGY

IN

NETWORK AND INFORMATION SECURITY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF ENGINEERING AND TECHNOLOGY

PONDICHERRY UNIVERSITY

PUDUCHERRY 605014

MAY 2022

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF ENGINEERING AND TECHNOLOGY

PONDICHERRY UNIVERSITY

CERTIFICATE

This is to certify that the project work entitled **"MULTILINGUAL ACCESSIBILITY OF WEB IMAGES FOR PERSONS WITH VISUAL IMPAIRMENTS USNG STEGANOGRAPHY"** is a genuine and original research work carried out by **Mr KATHIRVEL K (Reg. No. 20394011**) - **MASTER OF TECHNOLOGY** in **NETWORK AND INFORMATION SECURITY** of **Pondicherry University, Puducherry** under my guidance and supervision. The thesis work is not submitted earlier for the award of any degree/diploma or associateship of any other University / Institution.

DATE: 25 May 2022

PROJECT GUIDE

(Dr. K S KUPPUSAMY)

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF ENGINEERING AND TECHNOLOGY

PONDICHERRY UNIVERSITY

BONAFIDE CERTIFICATE

This is to certify that the Project Work titled **"MULTILINGUAL ACCESSIBILITY OF WEB IMAGES FOR PERSONS WITH VISUAL IMPAIRMENTS USNG STEGANOGRAPHY"** is a bonafide work of **Mr KATHIRVEL K (Reg. No. 20394011**) Carried out in PROJECT PHASE 2 for the award of the degree of **MASTER OF TECHNOLOGY** in **NETWORK AND INFORMATION SECURITY** of **Pondicherry University, Puducherry** under my guidance. This project work is original and not submitted earlier for the award of any degree/diploma or associateship of any other University / Institution.

HEAD OF THE DEPARTMENT

(Dr. S SIVA SATHYA)

PROJECT GUIDE

(Dr. K S KUPPUSAMY)

Submitted for the University Examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

TABLE OF CONTENTS

Universal Access in the Information Society
https://doi.org/10.1007/s10209-017-0607-z

LONG PAPER

Accessible images (AIMS): a model to build self-describing images for assisting screen reader users

Ab Shaqoor Nengroo[1] · K. S. Kuppusamy[1]

Abstract
Non-visual web access depends on the textual description of various non-text elements of web pages. The existing methods of describing images for non-visual access do not provide a strong coupling between described images and their description. If an image is reused multiple times either in a single Web site or across multiple times, it is required to keep the description at all instances. This paper presents a tightly coupled model termed accessible images (AIMS) which utilizes a steganography-based approach to embed the description in the images at the server side and updating alt text of the web pages with the description extracted with the help of a browser extension. The proposed AIMS model has been built, targeting toward a web image description ecosystem in which images evolve into a self-description phase. The primary advantage of the proposed AIMS model is the elimination of the redundant description of an image resource at multiple instances. The experiments conducted on a dataset confirm that the AIMS model is capable of embedding and extracting descriptions with an accuracy level of 99.6%.

Keywords Image accessibility · Alternative text · Steganography · Least significant bit (LSB) · Browser extension

1 Introduction

Worldwide, there are 285 million persons with visual impairments [39] many of whom use screen readers like JAWS [8] and NVDA [19] to access the web. A screen reader is a software application that reads the on-screen information or web page content and uses its text-to-speech (TTS) engine to translate it into speech or refreshable Braille. Screen readers require accessible web content or elements to work. Screen readers can read only textual information from a web page and not the graphics. The human brain holds and transmits significantly more information when it is conveyed graphically, so the web pages are packed with images that are compelling in catching attention. Images are one of the most widely used content types on web pages [31]. Screen readers read the alternative text of images if it is present. If the alternative text of an image is not present, then screen readers may skip the image or read the file name. Alternative text is a method of providing the textual description of non-textual content like images. To comply with W3C guidelines, web page developers must provide alternative text for images [13]. However, on web pages, alternative text is often not found or not meaningful [10, 36]. According to a study by Bigham et al. [3], on Alexa's top 500 Web sites homepages, only 39.6% of images were assigned alternative text. Our study of a data set of 18,259 images showed that only 5930 images (32.47%) contain some form of alternative text. Alternative text for images is provided using the *alt* attribute of the *img* tag of HTML.

Although W3C recommends providing alternative text for images to make them accessible to assistive technologies, developers find it is a time-consuming and repetitive task as they need to define alternative text for every single image on the web page. It will be less time-consuming and efficient for developers if this task of describing images is diverted to graphic designers or if images are self-describing.

This paper proposes a method to automate the process of supplying alternative text for images. Our method uses image steganography on the back-end to store alternative text inside the images before uploading them. This task can be performed by graphic designers or anyone without the

✉ K. S. Kuppusamy
kskuppu@gmail.com

Ab Shaqoor Nengroo
shakoor.ab.phd.cse@gmail.com

[1] Department of Computer Science, Pondicherry University, Pondicherry, India

Published online: 12 December 2017

need of any coding knowledge. This task is performed on an interface (AIMS.Builder) which can also be implemented by web hosts where users are allowed to provide the text description for images while uploading them. On the client side, we have developed a browser extension (AIMS browser extension) which reads the alternative text from the images and supplies it to the *alt* attribute of the *img* tag to make them accessible for users with visual impairments. Our proposed method can be an effective solution for providing alternative text to images automatically on the client side, which will increase the accessibility of web pages and reduce the workload of developers without having any adverse impact on user's browsing experience. The objectives of this paper are:

(a) To present a model to build self-describing images that establish the tight coupling between an image and its description;
(b) To harness image steganography in embedding image description and evaluating its performance regarding accuracy and time.

The remaining of this paper is organized as follows: Sect. 2 provides an overview of related work. Section 3 provides an overview of different types of images used on web pages and their accessibility. Section 4 explains the proposed AIMS model and its implementations. Section 5 explains the various experimental results of our proposed model. In Sect. 6, actual user evaluation of the proposed method is provided. Finally, we conclude and present future research directions. Query ID="Q4" Text="Please check the clarity of the phrase ‘ or across multiple times’ in the sentence ‘If an image is reused multiple…the description at all instances’."

2 Related work

The web contains diverse content, which includes simple text, images, animation, video. Images have a significant share on web pages and are found in almost all kinds of Web sites, such as social networks, e-commerce, newspapers, and educational services. However, images are not accessible to a significant fraction of web users, who are lacking visual cognition and are using screen readers to access the web. Providing alternative text for images makes them accessible to these users. W3C recommends the alternative text for images; however, many web page developers find it to be a time-consuming and repetitive process. Petrie et al. [26] noticed that images on web pages lack proper or descriptive alternative text and focused on encouraging developers to do the task of providing alternative text. To lessen the burden of developers, researchers are working to develop automated alternative text systems, which can be broadly classified into two categories.

2.1 Semiautomated systems

Semiautomated systems use crowd-powered systems to gather captions for images, which can be later used as alternative text for images on web pages to increase accessibility.

von Ahn et al. developed two computer games called *Phetch* [36] and *ESP Game* [35] where humans are used to label the images. Images used in these games are stored with their captions in a centralized database which can later be used to provide alternative text to images on web pages by a browser extension. They claim that if their game is played as much as some popular online games, then it is possible to provide alternative text to all images on the web in few months.

Bigham et al. introduced a talking mobile phone application called *VizWiz* [2] that can be used by persons with visual impairments to find answers for visual questions on a real-time basis. *VizWiz* allows visually impaired users to recruit online web users to answer their queries related to images. Persons with visual impairments can use *VizWiz* on mobile phones. Users take a picture with their cell phone, ask a question about the picture, and then receive multiple answers of the query aurally.

Brady et al. developed a variant of *VizWiz* called *VizWiz social* [4], an iPhone application that uses social networks of a visually impaired user to get answers for images. Instead of *crowdsourcing*, which uses strangers to answer the queries, *VizWiz social* uses *friendsourcing* to answer the visual questions, which improves privacy.

Guo et al. [11] developed another crowd-powered system called *VizLens*, a system that combines on-demand crowdsourcing and real-time computer vision to make real-world interfaces accessible to persons with visual impairments.

McCoy et al. [23, 24] developed methods for adding personalized information to digital images using metadata. Image metadata are summarized information about an image that is stored in the image file itself or contained in an external file that is linked to it. Several types of metadata can be stored in an image file like *technical metadata*, which includes the camera device model, shutter speed, focal depth, DPI, image dimensions, location, date, and time information. *Content metadata* or descriptive metadata contain information like caption, title, and comments. This information can be added manually using specialized imaging software. Image metadata are used for image categorization and retrieval. Content metadata can be used to store the textual description of an image, which can be later provided as an alternative text of an ⟨img⟩ tag. However, using metadata to store alternative text has certain limitations as described below:

- Metadata can be easily stripped from an image file [5];
- Storing metadata with an image file raises security and privacy concerns as it can be easily used to get sensitive information about the image creator;
- To preserve privacy some web applications like Facebook strip metadata from images before uploading them to their servers [5].

Semiautomated systems involve humans to provide labels or alternative text for images. Such systems like *Phetch* and *ESP Game* depend on the accuracy of humans to provide labels for random images. Since these approaches involve a centralized database to store images and captions, system updating and performance remain a problem. *Crowdsourcing* and *friendsourcing* approaches like *VizWiz*, *VizWiz social*, and *VizLens* which recruit online web users to provide labels for images involve various concerns like privacy, cost, accuracy, and response time. These methods for providing alternative text loosely couple images and their description, making it essential to provide alternative text repeatedly. The proposed system overcomes this problem by providing a tightly coupled mechanism in which textual description is embedded into the images and can be used any number of times across multiple web pages.

2.2 Fully automated systems

Fully automated systems combine the approaches of computer vision and machine learning to read text from images and supply it as alternative text.

Various automatic image captioning methods are available, which can read text and objects from image and generate a descriptive caption [7, 15, 16, 33, 34, 41]. These automatic captioning methods are employed to provide alternative text for images.

Park et al. [21, 22] described a method, which automatically extracts text from images using optical character recognition (OCR) and uses the extracted text as alternative text for images on web pages. They mention that this method can be an effective way to add alternative text for images, as 51.4% of images on top 20 Web sites contain text.

Bigham et al. [3] proposed a system called *WebInSight*, which automatically creates and inserts alternative text for images on web pages. To generate the alternative text for images, they used enhanced OCR, human labeling, and web context analysis.

The above methods for providing automatic alternative text for images depend on the accuracy of computer vision like OCR. Computer vision algorithms have evolved in the last few decades; however, they are not yet able to correctly describe a complex image which contains many objects and text on it, with much accuracy. Images with some new scenarios and objects are not described very well, which can mislead or confuse screen reader users.

3 Image concepts and accessibility

To ensure that images are accessible to persons with various disabilities, the W3C recommends providing the alternative textual description for images on web pages. This alternative text should describe the functions or information represented by the images. Images on web pages can be categorized into two groups, *significant images* and *insignificant images* [3].

3.1 Significant images

Significant images convey information to users and add to the content of a web page. Significant images are important to understand the content and to activate various functions of a web page. It can be an image with a news article, a map showing the internal layout of a building, a button submitting a form, or a navigational image linking to some other web page. To make the content and functionality of these images accessible to persons with visual impairments, they must have appropriate alternative text description.

Significant images can be described by *alt, longdesc, ARIA*, or *title* attributes. The *alt* attribute of an ⟨img⟩ tag can describe images that convey simple information. It is designed to provide a concise description or functional equivalent for an image which is typically invisible to users who can see images though is exposed to users using assistive technologies like screen readers or braille displays. It is a good practice to keep alt text between 75 and 100 characters. If the alt text is too long, it may cause reading issues in some assistive technologies [32]. For complex images like graphs, charts, and maps, more details are required to convey all the information contained within the image; *longdesc* attribute is designed for that purpose. The *longdesc* attribute should be used carefully as it is supported by a limited number of browsers and assistive technologies [32]. Accessible rich Internet applications (ARIA) aim to make web content and web applications more accessible to assistive technology users by providing additional attributes to HTML elements. The attributes *aria-labelledby* and *aria-label* can be used to provide text alternatives for images; *aria-labelledby* and *aria-describedby* can be used to provide an accessible description for HTML elements. The *alt* attribute text is not displayed by browsers to users experiencing disabilities. Using the *aria-labelledby* attribute to provide alternative text for images promotes the use of on-screen text, which can be beneficial to users with low vision and cognitive impairments. The text provided inside the *title* attribute is displayed when the user places the mouse pointer

Table 1 Image types and alternative text approaches

Image type	Examples	Alt text needed	Accessible via semiautomated systems	Accessible via fully automated systems	Accessible via AIMS model
Informative images	–Simple pictures –Icons –Succinct images –Photographs –Images of text	✓	✓	✓	✓
Navigational images	–Logos –Image links	✓	✗	✗	✓
Functional images	–Button images –Icons	✓	✗	✓	✓
Decorative images	–Bullet points –Borders –Fillers	✗	✗	✗	✗

over the image. Although it is not keyboard accessible, it can benefit users with low vision and cognitive impairments that use the mouse.

3.2 Insignificant images

Insignificant images are used as a part of a web page's design and do not add any information to the content of the web page. Insignificant images are mostly used to make a web page design more attractive, such as borders, fillers, round corners, bullet points. These images need not to be accessible and do not need any alternative text. These images should contain a *null* (empty) *alt* text or *alt* = " ", so that they can be ignored by screen readers. An *img* tag with no *alt* should be avoided as they can distract or confuse screen reader users because when a screen reader encounters an *img* tag with no *alt* attribute, it reads it as "*image*," which leaves users wondering if they have missed anything important.

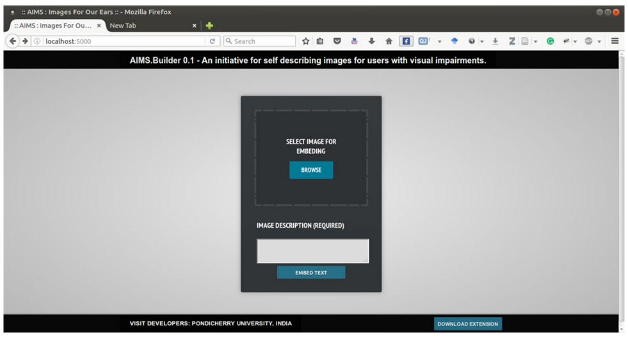
Table 1 compares two widely used approaches to providing alternative text with the proposed system. Semiautomated systems cannot provide valid alternative text for navigational and functional images as online users used in labeling cannot describe the working of these images on a real-time basis. These images can be correctly described by web page developers and designers only. For example, a button image with a "*submit*" text on it can be described by the semiautomated system as "*submit button*," though its actual functionality can be provided by a web page developer or designer as "*confirm and submit your registration form*." Our proposed system allows every image to be annotated at the web developers end, which can eliminate this problem.

4 Accessible images: the AIMS model

In this paper, we are proposing a method that aims to enhance the accessibility of web pages with regard to image content. The proposed method provides a tightly coupled mechanism in which textual description is embedded into the images using image steganography. Browsing assistant is the module of AIMS to which users are exposed. It provides alternative text for images which have been previously annotated by our system. Screen reader users can activate the extension by using a keyboard shortcut Ctrl + Shift + U or Command + Shift + U on Mac. Activation of the extension is conveyed to persons with visual impairments by a sound alert. Upon activation, our extension extracts the text from the image and supplies that text as the alternative text for the image and hence makes it accessible to screen reader users. The proposed system contains two core modules, i.e., *AIMS.Builder* and *AIMS browser extension*.

4.1 AIMS.Builder

The first step of the system is to embed the text description of an image into it. This task is performed by *AIMS.Builder*. *AIMS.Builder* is an accessible interface designed using Python and Flask used to embed the text description of an image into an image itself as shown in Fig. 1. *AIMS.Builder* takes input as the cover image (image used for embedding) and the text description of an image supplied by the designer or author of the web page (Fig. 2). The output is an annotated image which contains the embedded text. This embedded text description is read by the browser extension and is provided as the alternative text for the image to make it accessible for screen reader users.



AIMS.Builder 0.1 - An initiative for self describing images for users with visual impairments.

:: AIMS : Images For Our Ears :: - Mozilla Firefox
:: AIMS : Images For Ou...
New Tab
localhost:5000
Search
SELECT IMAGE FOR EMBEDING
BROWSE
IMAGE DESCRIPTION (REQUIRED)
EMBED TEXT
VISIT DEVELOPERS: PONDICHERRY UNIVERSITY, INDIA
DOWNLOAD EXTENSION

Fig. 1 AIMS.Builder interface

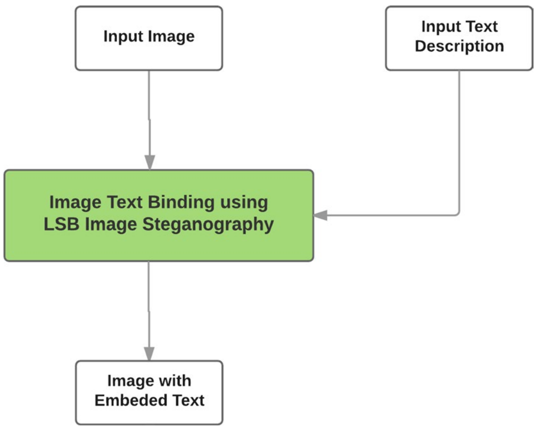


Fig. 2 AIMS.Builder method

AIMS.Builder uses the technique of image steganography to store text description inside the image without losing much of the quality of the image. Image steganography [20, 29] is a technique where information is embedded within an image. Due to the low cost of storing images and digital transmission, and the natural weakness of the human visual system (HVS), steganography is being widely used in the last few decades. Image steganography hides a secret message or text inside a carrier (an image). This message is secured using some *stego-key* from intruders [18]. To embed text description with the image for accessibility, we do not need the security component of steganography and ignoring the security component shall optimize the overall process regarding "*time*." The literature on image steganography shows a variety of approaches available for hiding text into images [28]; however, a technique which is simple for better performance and has a high payload capacity to store maximum text to describe the image for better understandability of persons with visual impairments was necessary. To embed text within an image, we chose the least significant bit (LSB) technique as it fulfilled our criteria of simple, fast, and high payload capacity [12, 20, 30]. LSB can store a good amount of text in informative and functional images as these

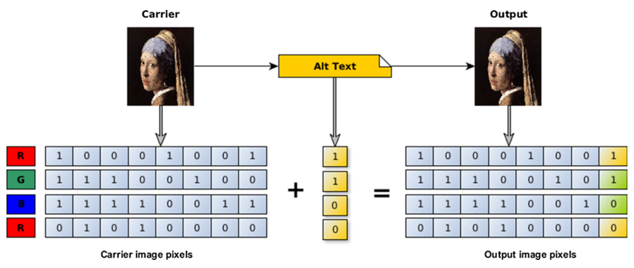


Fig. 3 Working of LSB technique

images usually have proper dimensions and quality for the better visual appearance of a web page.

LSB works by replacing the right-most bit, i.e., least significant bit (LSB) position of randomly selected pixels within the carrier image by the message bits (alt text in our case). The embedding operation of LSB steganography may be described by the following equation:

$$y_i = \left\lfloor \frac{x_i}{2} \right\rfloor + m_i$$

where m_i, x_i, and y_i are the ith message bit, the ith selected pixel value before embedding, and that after embedding, respectively [17]. LSB is described in Fig. 3. Pixels are chosen randomly from an RGB carrier image, and the LSB value of each color of a pixel (i.e., red, green, and blue) is used to carry the message bits resulting in an output image annotated with alternative text information. Algorithm 1 describes the steps for embedding text into images using the LSB technique, and Algorithm 2 shows the process of extracting text description from images.

Algorithm 1 Embedding Algorithm for LSB

```
1: procedure LSBWRITE($carrier\_img$,$alt\_text$)
2:     $C \leftarrow carrier\_img$, $M \leftarrow alt\_text$
3:     for $i = 1$ to $len(M)$ do
4:         Compute $ij$ of $C_{ij}$ pixel to store $i^{th}$ bit of M
5:         $p \leftarrow$ LSB bit of $C_{ij}$ pixel
6:         if $p! = M_i$ then
7:             $C_{ij} \leftarrow M_i$                ▷ Store message bit in LSB
8:     return $annotated\_image$
```

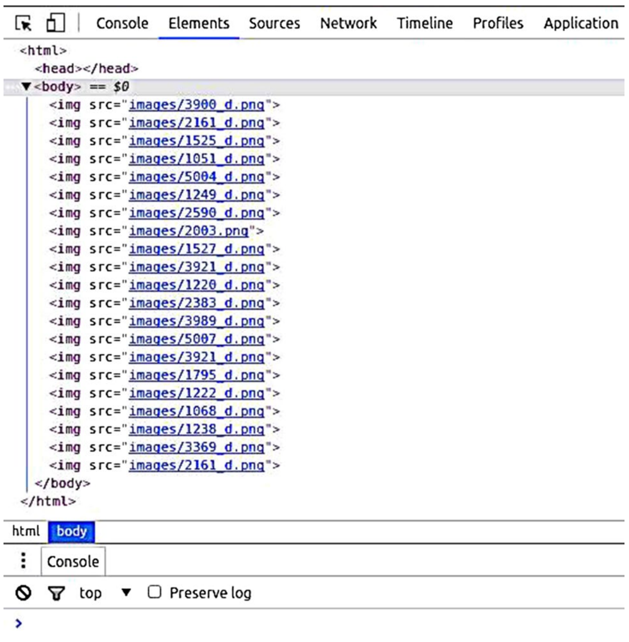
Algorithm 2 Extraction algorithm for LSB

```
1: procedure LSBREAD($carrier\_img$)
2:     for $i = 1$ to $len(M)$ do
3:         Compute $ij$ where $M_i$ is stored
4:         $M_i \leftarrow C_{ij}$                ▷ Get message bit from LSB
5:     return $alt\_text$
```

Image manipulations can destroy the embedded text as most image steganography schemes are sensitive to simple geometric attacks, namely rotation, cropping, and scaling [1, 27]. Our method will not suffer from loss of information when the annotated images are displayed on small screen devices like mobile phones or tablets, as according to the W3C recommendation [37], Media Queries is a standard way to render images for these types of devices. Among the media features that can be used in media queries are width and height. These two features are used in HTML/CSS to render images for small screen devices. For example, if we want to change the size of an image to adapt a particular environment all we need to do is change the width and height attributes. This will not have any effect on information stored in the annotated image when displayed on small screen devices like mobile phones or tablets.

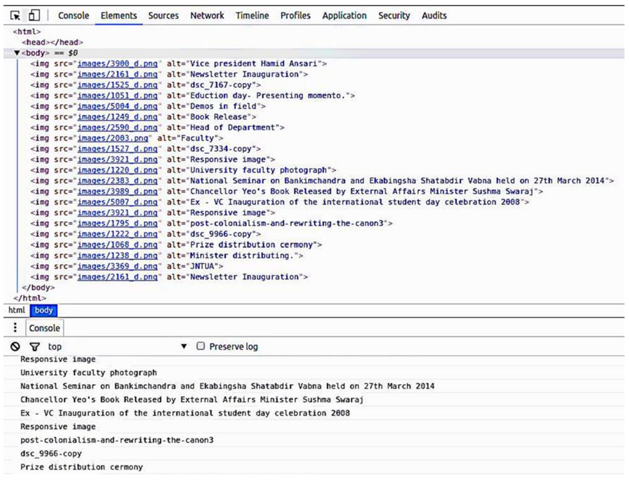
4.2 AIMS browser extension

We have developed an accessible browser extension. This extension, when activated by a keyboard or mouse, provides alternative text for images, which have been previously annotated by *AIMS.Builder*. The extension reads HTML source code of a web page visited by a user and identifies all images on the web page using ⟨img⟩ tags. Our JavaScript code will extract those images having *alt* = “ ” and images with no *alt* text (*alt* = *null*) as our target images for providing alternative text description. These images are scanned one by one to check whether an image contains any textual description inside it. During the process, if any text is found inside the image, it is read by the extension and provided as the alternative text of the image. Our results have shown that this process only takes 0.15 s to describe a web page containing 21 images. Figures 4 and 5 show the source code view of a user’s web page before and after activating our extension, respectively, for providing alternative text. The sample web page shown in Fig. 4 contained 21 images with text description embedded using our AIMS.Builder. Our extension successfully provided *alt* text for all 21 images as shown in Fig. 5.



```
<html>
  <head></head>
▼<body> == $0
    <img src="images/3900_d.png">
    <img src="images/2161_d.png">
    <img src="images/1525_d.png">
    <img src="images/1051_d.png">
    <img src="images/5004_d.png">
    <img src="images/1249_d.png">
    <img src="images/2590_d.png">
    <img src="images/2003.png">
    <img src="images/1527_d.png">
    <img src="images/3921_d.png">
    <img src="images/1220_d.png">
    <img src="images/2383_d.png">
    <img src="images/3989_d.png">
    <img src="images/5007_d.png">
    <img src="images/3921_d.png">
    <img src="images/1795_d.png">
    <img src="images/1222_d.png">
    <img src="images/1068_d.png">
    <img src="images/1238_d.png">
    <img src="images/3369_d.png">
    <img src="images/2161_d.png">
  </body>
</html>
```

Fig. 4 Web page before providing alt text



```
<html>
  <head></head>
▼<body> == $0
    <img src="images/3900_d.png" alt="Vice president Hamid Ansari">
    <img src="images/2161_d.png" alt="Newsletter Inauguration">
    <img src="images/1525_d.png" alt="dsc_7167-copy">
    <img src="images/1051_d.png" alt="Eduction day- Presenting momento.">
    <img src="images/5004_d.png" alt="Demos in field">
    <img src="images/1249_d.png" alt="Book Release">
    <img src="images/2590_d.png" alt="Head of Department">
    <img src="images/2003.png" alt="Faculty">
    <img src="images/1527_d.png" alt="dsc_7334-copy">
    <img src="images/3921_d.png" alt="Responsive image">
    <img src="images/1220_d.png" alt="University faculty photograph">
    <img src="images/2383_d.png" alt="National Seminar on Bankimchandra and Ekabingsha Shatabdir Vabna held on 27th March 2014">
    <img src="images/3989_d.png" alt="Chancellor Yeo's Book Released by External Affairs Minister Sushma Swaraj">
    <img src="images/5007_d.png" alt="Ex - VC Inauguration of the international student day celebration 2008">
    <img src="images/3921_d.png" alt="Responsive image">
    <img src="images/1795_d.png" alt="post-colonialism-and-rewriting-the-canon3">
    <img src="images/1222_d.png" alt="dsc_9966-copy">
    <img src="images/1068_d.png" alt="Prize distribution cermony">
    <img src="images/1238_d.png" alt="Minister distributing.">
    <img src="images/3369_d.png" alt="JNTUA">
    <img src="images/2161_d.png" alt="Newsletter Inauguration">
  </body>
</html>
```

```
Responsive image
University faculty photograph
National Seminar on Bankimchandra and Ekabingsha Shatabdir Vabna held on 27th March 2014
Chancellor Yeo's Book Released by External Affairs Minister Sushma Swaraj
Ex - VC Inauguration of the international student day celebration 2008
Responsive image
post-colonialism-and-rewriting-the-canon3
dsc_9966-copy
Prize distribution cermony
```

Fig. 5 Web page after providing alt text

4.3 Implementation

Figure 6 shows the overall architecture of AIMS. Our system contains two main components i.e., AIMS.Builder and AIMS browsing assistant or extension.

AIMS.Builder uses the technique of LSB image steganography described in Sect. 4.1 to embed the text description inside the image. The AIMS.Builder interface is created using HTML/CSS with python as a scripting language on flask framework. A Python Imaging Library (PIL 1.1.6)[1] with LSB is used to annotate images. The annotated image is then included in a web page say “*abc.com*” and stored on the web server. The complete process of AIMS.Builder is explained in Algorithm 3.

[1] https://pypi.python.org/pypi/PIL.

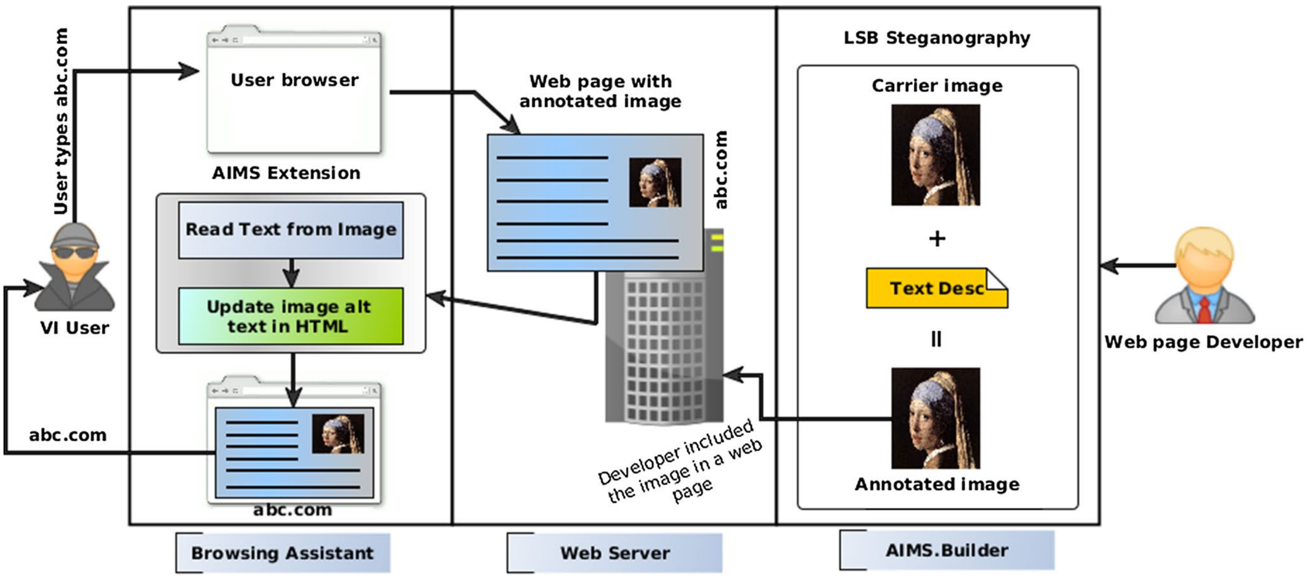


Fig. 6 AIMS architecture

Algorithm 3 AIMS.Builder algorithm

```
1: procedure AIMS.Builder($carrier\_img$,$text\_desc$)
2:     $image\_capacity \leftarrow getImgCapacity(carrier\_img)$
3:     $displayCapacity(image\_capacity)$
4:     if $image\_capacity >= \text{len}(text\_desc)$ then
5:         $annot\_img \leftarrow LSBWrite(carrier\_img, text\_desc)$
6:     else
7:         Display "Please provide a short text description"
8:     return $annot\_img$
```

Algorithm 4 AIMS.Extension algorithm

```
1: procedure AIMS.Extension($webpage$)
2:     $html \leftarrow getHTML(webpage)$
3:     $img\_list \leftarrow getImages(html)$
4:     while $img\_list \neq 0$ do
5:         $img \leftarrow img\_list[i]$
6:         $alt \leftarrow getAltText(html)$
7:         if $alt == "" || alt == NULL$ then
8:             $text\_desc \leftarrow LSBRead(img)$
9:             $setImageAltText(text\_desc)$
10:    return $webpage$
```

Annotated images once included on a web page are ready to be read by the browser extension. When a visually impaired user requests to access the web page "*abc.com*," it is fetched from the web server and scanned by the browser extension for the annotated images. The image is passed to LSBRead() procedure for retrieval of textual description. The retrieved text description is then provided as alt text of that particular image to make it accessible to users using assistive technologies such as screen readers or braille displays. The browser extension functionality has been implemented using JavaScript, which makes it faster. Algorithm 4 provides the description of the AIMS browser extension. The LSBWrite() and LSBRead() procedures are defined in Sect. 4.1. Figure 7 shows the overall flow of the AIMS model, and the time sequence interaction of various components of the model is shown in Fig. 8.

5 Experiments and results

For our study, we have built an *image data set* which contains 18,259 *image URL*, *alternative text of image*, *image dimensions*, and *HTML* ⟨img⟩ *tag* of images collected from their source web pages.[2] To gauge the accessibility of these web images, we ran a test on the data set, and the results showed that out of 18,259 images, only 5930 (32.47%) contain some form of alternative text. To evaluate the performance of the proposed system, we ran another test on the *image data set*. For this test, we selected only those images with non-empty alternative text and having both width and height greater than or equal to 50, treating the rest of the images as insignificant. Out of 18,259 images, 5126 images (28.07%) fit the selection criteria and were chosen for the

[2] https://github.com/shakoorst/AIMS-Dataset.

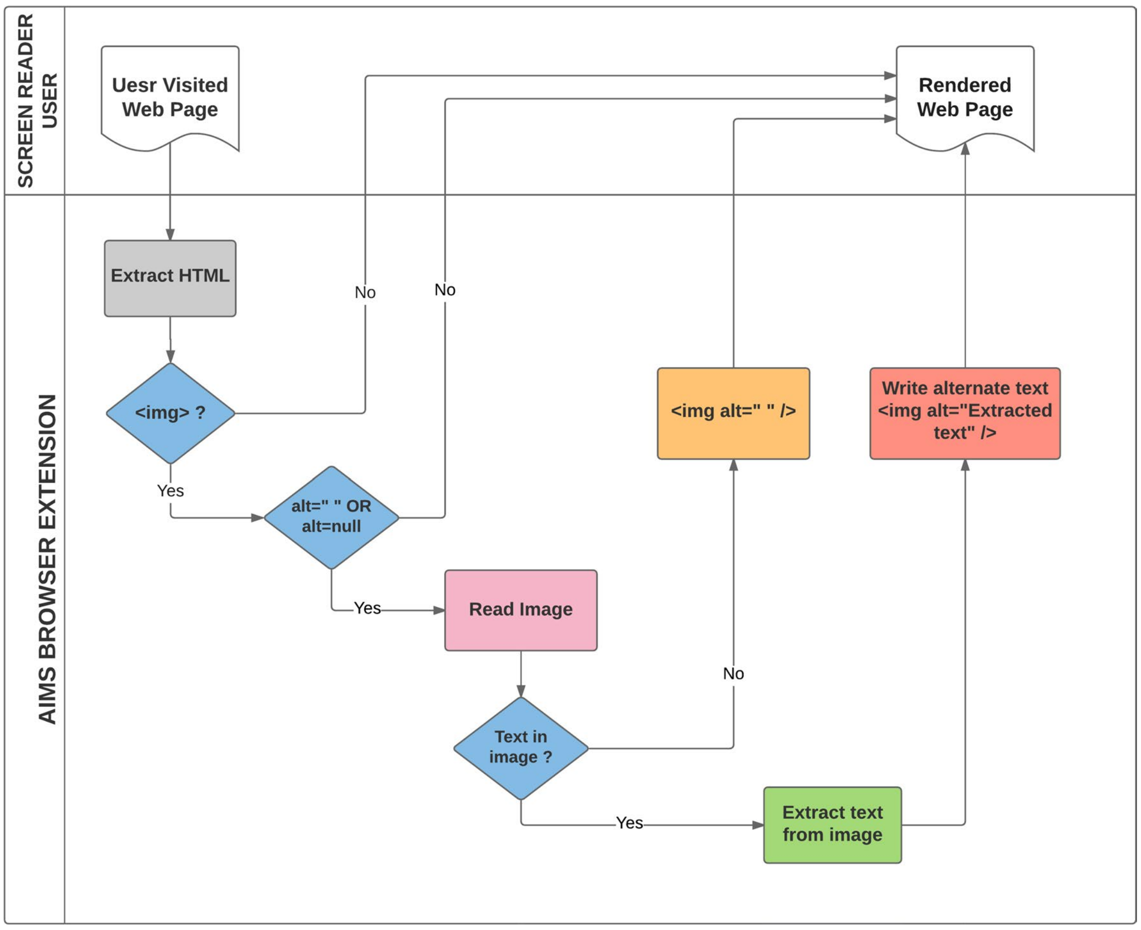


Fig. 7 AIMS flow diagram

test. We designed a python script to get these images from their source Web sites and passed them with their *alt* text to the system for analysis.[3] Out of 5126 images, the script successfully fetched 4595 images (89.6%). The remaining 531 images (10.4%) were not successfully fetched due to some expired image links (HTTP 404 Not Found Error) and connection timeout error. (Server was taking too long to respond.)

Successfully retrieved images (89.6%) were passed to our *AIMS.Builder* (Fig. 2) which embedded the *alt* text into them. The results of the test were recorded in a *log file* (.csv) for analysis. Our *log file* successfully recorded the results of 4595 images processed by the *AIMS.Builder*. The objectives of the test were to calculate the *embedding time*, calculate *text similarity*, and calculate *image distortion*.

[3] The machine used for the test was an Intel Pentium (Core 2 Duo) with 2 GB of RAM and an Internet connection of 512 Kbps.

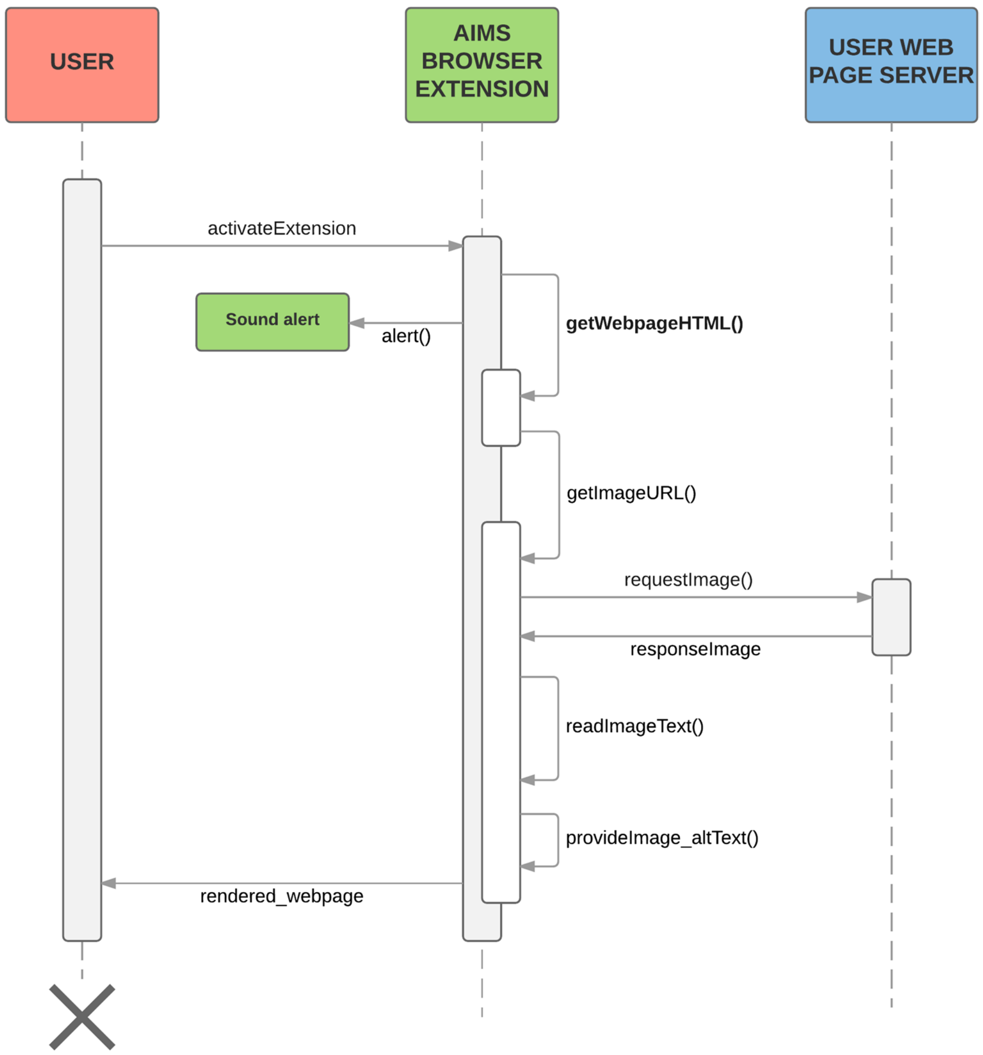
5.1 Embedding time

Embedding time is the time taken by the system to embed or hide the text description or alternative text inside the image for accessibility. Embedding is usually a one-time process performed at the developer's site before including the image to a web page. This process is carried out by our *AIMS.Builder*, described in Sect. 4.1. The analysis of the *log file* revealed that for a sample of $N = 4571$, mean embedding time was $M = 2.58$ with SD = 11.1 as shown in Fig. 9. This includes the time taken by our script to fetch the image from the parent Web site and save it on the local system, which may not be required when the image builder is used for embedding offline images locally.

5.2 Text similarity

Text similarity is the measure of similarity between the input text embedded into the image and text retrieved from the image. During the embedding process, it may sometimes happen that the input text and the retrieved text are not

Fig. 8 AIMS sequence diagram



same due to small image size or large text input. In order to analyze the text similarity rate of our system, we used Jaro–Winkler [14, 40] edit distance:

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{Otherwise} \end{cases}$$

Jaro–Winkler distance is a method to check the similarity of two strings. It returns a value between 0 and 1, where 0 means no similarity and 1 means an exact match. We applied Jaro–Winkler distance on our sample of $N = 4571$, where mean similarity between input text and retrieved text was $M = 0.996$ with SD = 0.05 indicating near equivalence between the input text and retrieved text.

5.3 Image distortion

One disadvantage of essentially all information or text embedding strategies is that the image gets distorted by a small amount of noise caused due to text embedding [9]. W3C recommends that the alternative text should be a concise description which conveys the image's purpose [6]. Because of the restriction within a popular screen reader program, many experts recommend *alt* text of 125 characters or fewer [25].

Due to the usage of high-quality images nowadays and recommendations for concise *alt* text, our system may suffer very less (if any), image distortion. To analyze the image distortion caused by our system, due to text embedding, we saved the original copy of the image before embedding and compared it with result image using the Structural SIMilarity (SSIM) index, developed by Wang et al. [38]:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x + c1)(2\sigma_x y + c2)}{(\mu_x^2 + \mu_y^2 + c1)(\sigma_x^2 + \sigma_y^2 + c2)}$$

SSIM index is a method for measuring the similarity between two images and returns a value between − 1 and 1, where − 1 indicates no similarity and 1 shows perfect

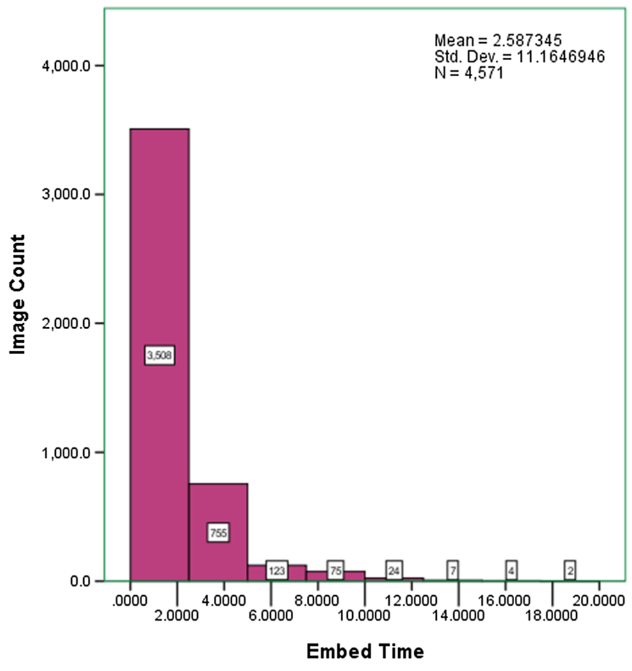


Fig. 9 Time taken to embed text (s)

similarity. We applied SSIM on our sample of $N = 4569$, where mean similarity between original and resultant images was $M = 0.96$ with SD = 0.18 (Fig. 10), indicating a negligible amount of distortion, not visible to the human visual system (HVS).

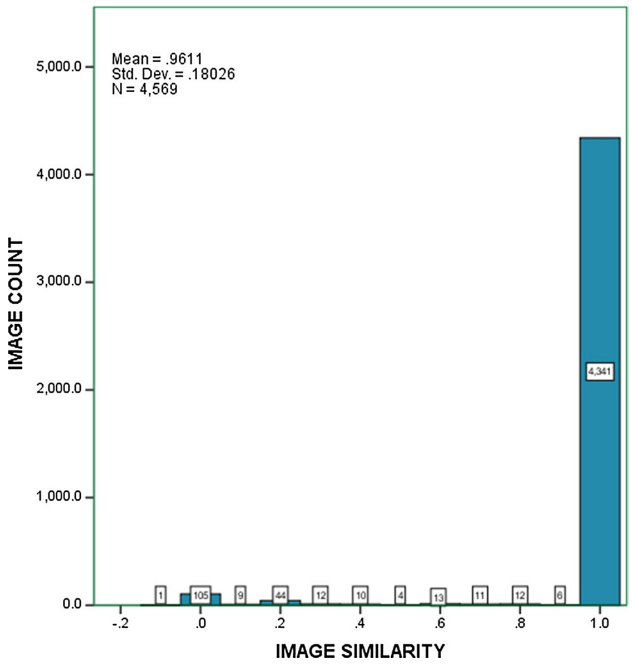


Fig. 10 Similarity of images after embedding text

5.4 Performance of AIMS browser extension

As discussed in Sect. 4.2, browser extension retrieves or reads the text description stored inside the image and provides it as alternative text (*alt text*) or *aria-label* or *longdesc* of the image for accessibility. To analyze the performance of AIMS browser extension, we designed a set of web pages containing the images created by our AIMS.Builder (with *alt* text embedded inside the images). Images were randomly selected from a sample of 500 images by a python script which then automatically designed the web pages. The number of images on each web page was chosen according to *Leonardo Pisano Series* which is generated using the recurrence,

$$F_n = F_{n-1} + F_{n-2}$$

We selected the image bucket size according to the value of F_n from $n = 2–8$ assuming that a web page may not contain more than 21 informative and functional images at one time. No alternative text was provided for these images in HTML ⟨img⟩. Sample web pages were uploaded on a live web host to check actual performance of the system. These web pages were visited one by one, and the *AIMS.Extension* was activated on each web page for the automatic provision of alternative text that was earlier stored inside these images. The success rate of providing *alt* text was 100%. The performance of the system was recorded with the browser's developer tools as shown in Fig. 11, and it shows that the total time taken by our system to provide alternative text for an image is ≈ 6.6 ms. The complete results of the test are recorded in Table 2.

Performance evaluation tests on image dataset were carried out using Python. Text similarity before and after the embedding process was calculated using Jaro–Winkler distance for which we used Python jellyfish 0.5.6[4] library. To check image distortion due to embedding, we used pyssim,[5] which implements the Structural Similarity Image Metric (SSIM).

6 User evaluation

The goal of our user study was to evaluate how AIMS Extension performs in assisting persons with visual impairments to understand the web content. We conducted seven sessions for our user study, in which users aged 20–50 years old, participated. All the participants were congenitally blind and participated voluntarily in our study while attending a national conference on "Rights of Persons with Disabilities

[4] http://github.com/jamesturk/jellyfish.

[5] https://github.com/jterrace/pyssim.

Table 2 Time taken by our system to provide alternative text for web pages containing different image quantities

Image bucket size	1	2	3	5	8	13	21
Time taken to provide alt text (in s)	0.01	0.012	0.03	0.04	0.05	0.09	0.15

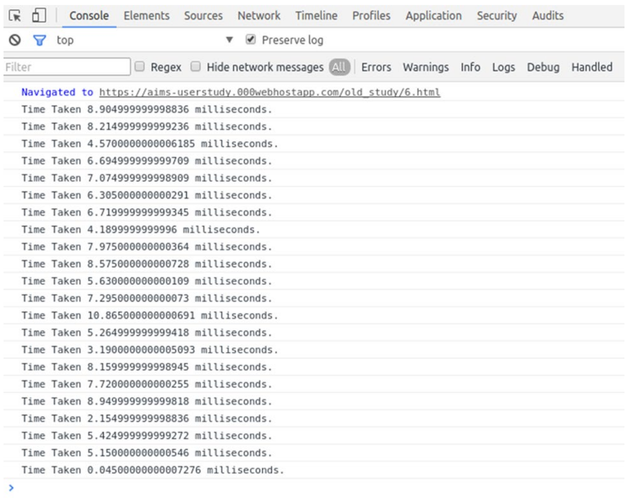
Table 3 User demographics summary

Age	Gender	Education	Impairment	Screen reader	Computer experience
> 50 = 0%		Ph.D = 14%			> 7 years = 14%
36–50 = 14%	M = 100	Masters = 57%	Blind = 100%	JAWS = 70%	5–7 years = 43%
20–35 = 86%	F = 0%	Bachelors = 29%	Low vision = 0%	NVDA = 30%	2–5 years = 43%
< 20 = 0%		Other = 0%		Other = 0%	Other = 0%

Act (RPwD-Act)" in India. Participants were experienced JAWS and NVDA screen reader users, well-educated with good computer knowledge. Basic demographic characteristics were recorded for each user, summarized in Table 3.

We designed a demo Web site of four web pages. The home page of the Web site contained instructions about the user study, links to questionnaires, and a link to download the extension. After installing the extension, users were asked to go through the demo Web site before and after activating our extension. Each visit was followed by a short questionnaire. Below, we summarize the findings and user feedback about the AIMS Extension:

- 86% of the participants agreed that the extension had improved the browsing experience by describing the images;
- 72% of the participants found our extension very fast as they were not forced to wait and the remaining 28% found it average;
- Participants agreed that our extension described all the images very well as images contain sufficient text description. Responses were measured on a Likert-type scale (from 0—low, to 10—high) with $M = 7.57$ and $SD = 3.51$;
- All participants agreed that the extension described the images before navigating to them and were not forced to wait for description;
- 71.4% of the participants agreed that the extension improved the overall usability and understandability of the given web pages by describing the images which were critical to understanding the content of the web page.



```
Navigated to https://aims-userstudy.000webhostapp.com/old_study/6.html
Time Taken 8.904999999998836 milliseconds.
Time Taken 8.214999999999236 milliseconds.
Time Taken 4.5700000000006185 milliseconds.
Time Taken 6.694999999999709 milliseconds.
Time Taken 7.074999999998909 milliseconds.
Time Taken 6.305000000000291 milliseconds.
Time Taken 6.719999999999345 milliseconds.
Time Taken 4.1899999999996 milliseconds.
Time Taken 7.975000000000364 milliseconds.
Time Taken 8.575000000000728 milliseconds.
Time Taken 5.630000000000109 milliseconds.
Time Taken 7.295000000000073 milliseconds.
Time Taken 10.865000000000691 milliseconds.
Time Taken 5.264999999999418 milliseconds.
Time Taken 3.1900000000005093 milliseconds.
Time Taken 8.159999999998945 milliseconds.
Time Taken 7.720000000000255 milliseconds.
Time Taken 8.949999999999818 milliseconds.
Time Taken 2.154999999998836 milliseconds.
Time Taken 5.424999999999272 milliseconds.
Time Taken 5.150000000000546 milliseconds.
Time Taken 0.0450000000007276 milliseconds.
>
```

Fig. 11 Time analysis of the system for a web page containing 21 images

7 Conclusions and future directions

The Internet has revolutionized almost every field, and the web has become an important resource in various facets of life. To provide barrier-free access to this valuable resource, web pages should be developed according to web accessibility standards and guidelines. Images are one of the most widely used content types on web pages. The lack of alternative textual description for many images has become an accessibility challenge for persons with visual impairments, as the alternative text is often missing or not meaningful. Our study of 18,259 images showed that only 5930 images (32.47%) contain some form of alternative text. Providing the alternative textual description for images is an important task, however, requiring significant effort and time from web page developers, who often find it to be a repetitive task. We introduced a new system that is capable of providing the alternative text for web images in a tightly coupled manner, without having any adverse effect on the user's browsing experience. We have described two modules of our system, which embeds the text description into the image using the steganography technique. This text is extracted and provided as the alternative text by our extension when visited by a screen reader user. Our method can provide alternative text to all images annotated by the system, and our results have shown a good performance. Our system will be effective as developers do not need to provide the alternative textual description of the image repeatedly at multiple instances. Alternative text provision is not limited to developers. Anyone can annotate the image.

As the AIMS ecosystem evolves further, the mechanism of extracting the descriptions from the images shall be made a part of the browser rendering engine code itself. The extraction of description shall be attempted at the server side itself to reduce any delay on the client side. Another major future direction is the development of a tailor-made speed-focused steganography algorithm which would further optimize the temporal performance of the algorithm. As future work, we are also looking forward to providing multilingual alternative text for web images and extract the alternative text according to the language of the web page selected by the user.

References

1. Amin, P., Subbalakshmi, K.P.: Rotation and cropping resilient data hiding with Zernike moments. In: 2004 International Conference on Image Processing, 2004. ICIP'04, vol. 4, pp. 2175–2178. IEEE (2004)
2. Bigham, J.P., Jayant, C., Ji, H., Little, G., Miller, A., Miller, R.C., Miller, R., Tatarowicz, A., White, B., White, S., et al.: Vizwiz: nearly real-time answers to visual questions. In: Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology, pp. 333–342. ACM (2010)
3. Bigham, J.P., Kaminsky, R.S., Ladner, R.E., Danielsson, O.M., Hempton, G.L.: Webinsight:: making web images accessible. In: Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility, pp. 181–188. ACM (2006)
4. Brady, E.L., Zhong, Y., Morris, M.R., Bigham, J.P.: Investigating the appropriateness of social network question asking as a resource for blind users. In: Proceedings of the 2013 Conference on Computer Supported Cooperative Work, pp. 1225–1236. ACM (2013)
5. Condron, M.: Managing the Digital You: Where and How to Keep and Organize Your Digital Life, 1st edn. Rowman & Littlefield Publishers, Lanham (2017)
6. Eggert, E., Abou-Zahra, S.: Images concepts. https://www.w3.org/WAI/tutorials/images/tips/ (2015)
7. Fang, H., Gupta, S., Iandola, F., Srivastava, R.K., Deng, L., Dollár, P., Gao, J., He, X., Mitchell, M., Platt, J.C., et al.: From captions to visual concepts and back. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1473–1482 (2015)
8. Freedom Scientific: The world's most popular windows screen reader. http://www.freedomscientific.com/Products/Blindness/JAWS (2016)
9. Goljan, M., Fridrich, J.J., Du, R.: Distortion-free data embedding for images. In: International Workshop on Information Hiding, pp. 27–41. Springer (2001)
10. Goodwin, M., Susar, D., Nietzio, A., Snaprud, M., Jensen, C.S.: Global web accessibility analysis of national government portals and ministry web sites. J. Inf. Technol. Politics **8**(1), 41–67 (2011)
11. Guo, A., Chen, X., Qi, H., White, S., Ghosh, S., Asakawa, C., Bigham, J.P.: Vizlens: A robust and interactive screen reader for interfaces in the real world. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology, pp. 651–664. ACM (2016)
12. Hamid, N., Yahya, A., Ahmad, R.B., Al-Qershi, O.M.: Image steganography techniques: an overview. Int. J. Comput. Sci. Secur. **6**(3), 168–187 (2012)
13. Henry, S.L., McGee., L.: Accessibility. https://www.w3.org/standards/webdesign/accessibility (2016)
14. Jaro, M.A.: Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. J. Am. Stat. Assoc. **84**(406), 414–420 (1989)
15. Karpathy, A., Fei-Fei, L.: Deep visual-semantic alignments for generating image descriptions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3128–3137 (2015)
16. Krause, J., Johnson, J., Krishna, R., Fei-Fei, L.: A hierarchical approach for generating descriptive image paragraphs (2016). arXiv preprint arXiv:1611.06607
17. Li, B., He, J., Huang, J., Shi, Y.Q.: A survey on image steganography and steganalysis. J. Inf. Hiding Multimed. Signal Process. **2**(2), 142–172 (2011)
18. Lin, E.T., Delp, E.J.: A review of data hiding in digital images. In: IS and TS PICS Conference, pp. 274–278. Society for Imaging Science & Technology (1999)
19. Michael Curran, J.T.: Empowering lives through non-visual access to technology. https://www.nvaccess.org/ (2017)
20. Morkel, T., Eloff, J.H., Olivier, M.S.: An overview of image steganography. In: ISSA, pp. 1–11 (2005)
21. Park, E., Lim, H.: A study on improvement of evaluation method on web accessibility automatic evaluation tool's <img> alternative texts based on OCR (2015)
22. Park, E., Lim, H.: A study on providing alternative text of image for web accessibility improvement. Int. J. Appl. Eng. Res. **11**(2), 762–765 (2016)
23. Parulski, K.A., McCoy, J.R.: Method for adding personalized metadata to a collection of digital images. US Patent 6629104, 2003
24. Parulski, K.A., McCoy, J.R.: Digital camera for capturing images and selecting metadata to be associated with the captured images. US Patent 7171113, 2007
25. Pennsylvania State University: Image alt tag tips for html. http://accessibility.psu.edu/images/imageshtml/ (2016)
26. Petrie, H., Harrison, C., Dev, S.: Describing images on the web: a survey of current practice and prospects for the future. In: Proceedings of Human Computer Interaction International (HCII), vol. 71 (2005)
27. Potdar, V.M., Han, S., Chang, E.: Fingerprinted secret sharing steganography for robustness against image cropping attacks. In: 2005 3rd IEEE International Conference on Industrial Informatics, 2005. INDIN'05, pp. 717–724. IEEE (2005)
28. Pradhan, A., Sahu, A.K., Swain, G., Sekhar, K.R.: Performance evaluation parameters of image steganography techniques. In: International Conference on Research Advances in Integrated Navigation Systems (RAINS), pp. 1–8. IEEE (2016)
29. Queirolo, F.: Steganography in images. Final Communications Report, vol. 3. http://eric.purpletree.org/file/Steganography%20In%20Images.pdf (2011)

30. Rana, M.: Parameter evaluation and comparison of algorithms used in steganography. Int. J. Eng. Sci. Comput. https://doi.org/10.4010/2016.1901 (2016)
31. Shin, H., Lim, J., Park, J.: Information visualization and information presentation for visually impaired people. ETRI J. **28**(1), 81–91 (2013)
32. Steve, F.: Html5: Techniques for providing useful text alternatives. https://www.w3.org/TR/2011/WD-html-alt-techniques-20110113/ (2017)
33. Tran, K., He, X., Zhang, L., Sun, J., Carapcea, C., Thrasher, C., Buehler, C., Sienkiewicz, C.: Rich image captioning in the wild (2016). arXiv preprint arXiv:1603.09016
34. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: lessons learned from the 2015 MSCOCO image captioning challenge. IEEE Trans. Pattern Anal. Mach. Intell. **39**, 652–663 (2016)
35. Von Ahn, L., Dabbish, L.: Labeling images with a computer game. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 319–326. ACM (2004)
36. Von Ahn, L., Ginosar, S., Kedia, M., Liu, R., Blum, M.: Improving accessibility of the web with a computer game. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 79–82. ACM (2006)
37. W3C Working Group on Cascading Style Sheets: Media queries level 4-w3c working draft. https://www.w3.org/TR/mediaqueries-4/ (2016)
38. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)
39. WHO: “visual impairment and blindness”, fact sheet 2014. http://www.who.int/mediacentre/factsheets/fs282/en/ (2014)
40. Winkler, W.E.: String comparator metrics and enhanced decision rules in the Fellegi–Sunter model of record linkage (1990)
41. Wu, Q., Shen, C., Hengel, A.v.d., Wang, P., Dick, A.: Image captioning and visual question answering based on attributes and their related external knowledge (2016). arXiv preprint arXiv:1603.02814

CHAPTER 2

ABSTRACT

In today's modern world, user interfaces are built-in a way that attracts people to use with the help of various multimedia content like images, videos, graphs, charts etc., Especially images are the prominent content of the webpage since the human brain understands the information in pictorial form faster than in textual form. Screen readers are the special software designed to assist visually impaired persons to consume digital content with the help of audio. These screen readers are capable of reading the textual content on the screen, but multimedia content present on the webpage can't be read by screen readers. In that case, alt attributes/filename of images are read by most of the screen readers. But the majority of websites don't have proper alternative text. Even though W3C recommend the usage of alternative text to all multimedia files to make them accessible to assistive technology as well, developers find it is the time-consuming and repetitive task of giving alternative text to all instance of the same/different image used throughout the website. It can be solved by implementing a model that predicts the image's content using Machine learning. These predicted keywords are embedded into the images using steganography methods. Later These descriptions are extracted, translated and updated in the alt attribute by the browser extension which can be easily read by the screen readers on the client-side. Multilingual alt text updation is achieved based on the user's preference via the translator module in the extension. Thereby this AIMS design improves the usability for visually impaired users.

PROBLEM DEFINITION

A screen reader is a software application that reads the on-screen information and uses its text-to-speech (TTS) engine to translate it into speech or refreshable Braille. Screen reader increases the independence of visually impaired persons in accessing computing devices. These screen readers are often used by visually impaired people to access web content. Information in pictorial or graphical representations are easily conveyed to users than in textual form, so websites are developed with text, images, videos and other graphical content. But screen readers are only capable of reading the textual contents of the webpage. Important information in the form of images like pie charts, graphs etc., can be skipped by these screen readers. In this case, the alternative text given to the image can be read by screen readers. If the alt attribute is not given to the image, the file name can be read by screen readers that are often not meaningful/relevant. The existing design proposed by Ab Shaqoor Nengroo, and K. S. Kuppusamy [2018] on making self-describing images to assist screen readers fixes this problem by hiding the image description in the image which can be decoded by the browser extension on the client-side and updated in the alt attribute of the image tag. Even though this helps screen readers, but increases the work of Webmasters where they manually have to embed the appropriate alternative text in each image which is found to be a time-consuming and recurring task. And this also lacks the multilingual alt text updation based on the website's native language/user preferred language choice.

CHAPTER 3

ARCHITECTURE DIAGRAM

The existing AIMS architecture consists of two primary models which are given in detail below

- AIMS builder
- AIMS browser extension

AIMS Builder

AIMS builder helps in embedding images behind the image without reducing much of its quality using steganography methods. It can be done by LSB (Least significant bit) technique which is simple, fast and carries high payloads. LSB works by replacing the right-most bit, i.e., the least significant bit (LSB) position of randomly selected pixels within the carrier image by the message bits. LSB can store a good amount of text in informative and functional images as these images usually have proper dimensions and quality for a better visual appearance of a web page.

AIMS Browser Extension

AIMS extension is a browser extension which is installed as an add-on to the browser. It extracts the message embedded in the image by finding the image tag on the source code of the webpage. The extracted message is updated on the alt attribute of the respective <img> tags which can be easily read by the screen readers.

The overall existing architecture of the AIMS system is given in figure 1.

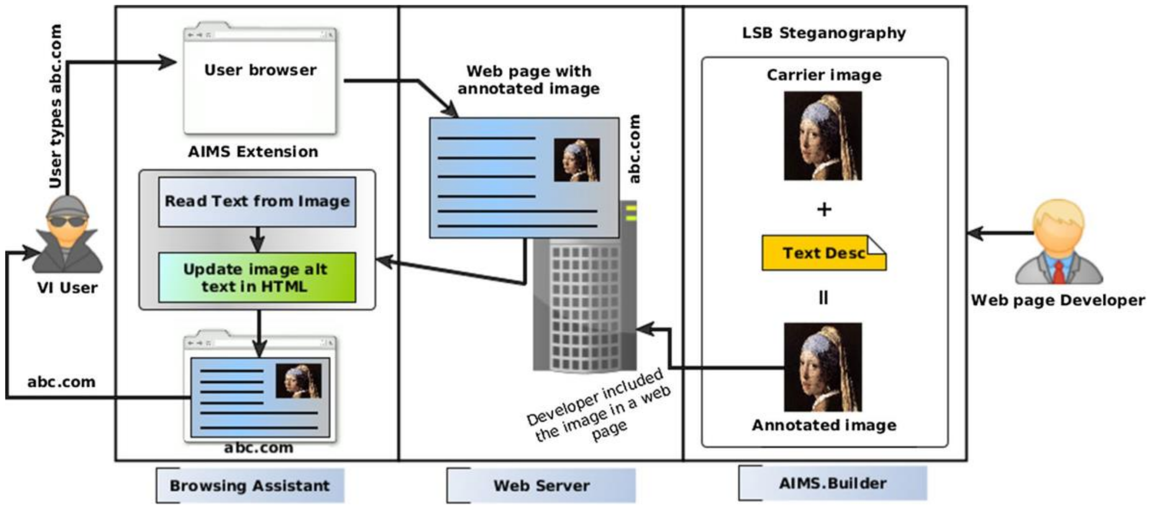


Figure 1 – Overall architecture

CHAPTER 4

DESIGN

AIMS Builder

- AIMS builder is the web application which gets the image as the input from the user and sends the data to the python flask server.
- Flask server performs object detection and returns the detected objects.
- Input image and detected objects are embedded into the image using the LSB steganography algorithm.

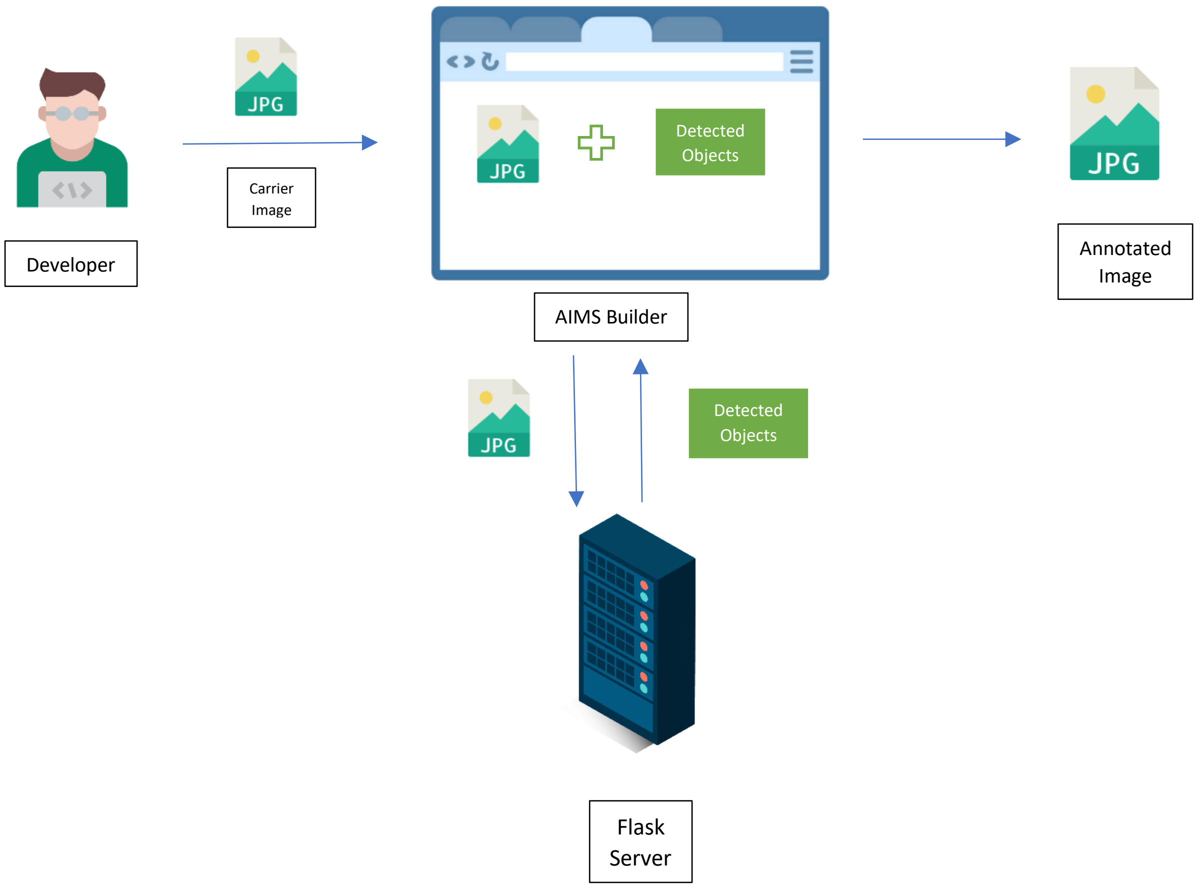


Figure 2 – AIMS builder architecture

- Then finally we got the annotated image as the output which can be downloaded and used on web pages.

AIMS Extension

- AIMS browser extension mail functionality is to extract the embedded message from the image.
- AIMS extension also gets the user preferred language as input and translates the extracted message as per the user preferences.
- The detailed flow of the AIMS browser extension is given as below.

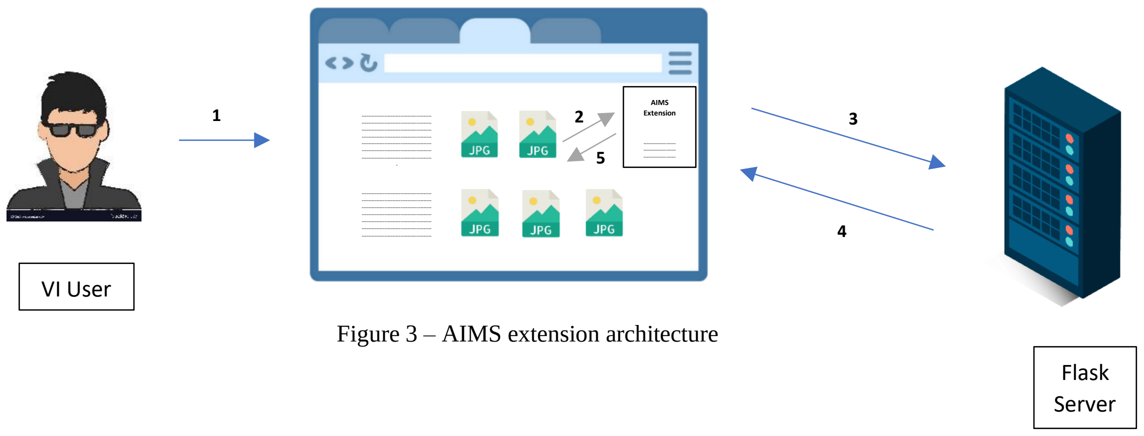


Figure 3 – AIMS extension architecture

1 - VI user surfs the websites

2 - Image src is passed to extension as input

3 - The embedded message is decoded from the image and the language user chosen is sent to the flask server

4 - The flask server translates the embedded message as per the user preference and returns the result to the extension.

5 - Extension set alt attribute to the image tags

CHAPTER 5

BACKGROUND STUDY

Least Significant Bit (LSB)

The entire AIMS design is based on image steganography. Steganography is the technique of hiding a secret message by embedding it into a file. In AIMS, the Least Significant Bit (LSB) technique is used to embed and extract the alternative text on the image. LSB works by replacing the right-most bit, i.e., the least significant bit (LSB) position of randomly selected pixels within the carrier image by the message bits. The embedding operation of LSB is given by the below equation,

$y_i = [\ x_i / 2\ ] + m_i$

where, m_i = ith Message bit

x_i = ith selected pixel value before embedding

y_i = ith selected pixel value after embedding

Embedding Algorithm for LSB

```
procedure LSBWrite(carrier img,alt text)
   C ← carrier img, M ← alt text
   for i = 1 to len(M) do
      Compute ij of Cij pixel to store ith bit of M
      p ← LSB bit of Cij pixel
      if p! = Mi then
         Cij ← Mi        //Store message bit in LSB
   return annotated image
```

Extraction algorithm for LSB

```
procedure LSBRead(carrier img)
   for i = 1 to len(M) do
      Compute ij where Mi is stored
      Mi ← Cij          //Get message bit from LSB
   return alt text
```

Browser extension

Another important component of the AIMS system is the chrome browser extension. Extensions are added to the browser as an addon. Extension components are created with web development technologies like HTML, CSS, and JavaScript. Extensions include manifest, content scripts, background scripts and UI elements.

Manifest file - Every extension must have JSON-formatted manifest file, named manifest.json, that provides important information like manifest version, name, icon, permission, included scripts, action etc.,

Background scripts - The background script is a script running in the background to handle the majority of chrome events that content scripts cannot. Unlike content scripts, the background script runs in the browser level context.

Content scripts - Content scripts are script files that run in the context of web pages.

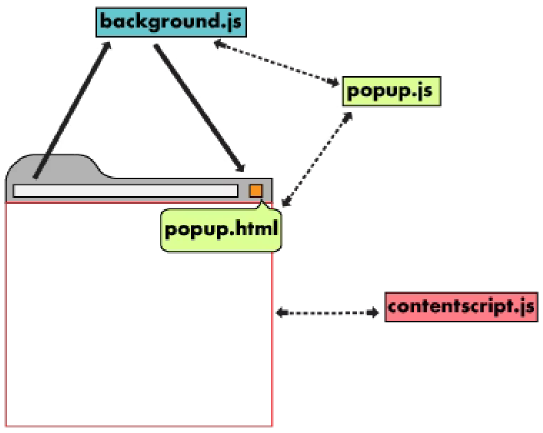


Figure 4 – Chrome extension components

UI elements – The user interface of the chrome extension is created in HTML and CSS which gets loaded when the user clicks on the extension.

ImageAI library

ImageAI is a python library built by the DeepQuest AI team to empower developers, researchers and students to build applications and systems with self-contained Deep Learning and Computer Vision capabilities.

With the Image AI library, anyone can easily perform custom image prediction, object detection, video detection and video object tracking with fewer lines of code.

CHAPTER 6

CODE

AIMS Builder

App.py

Initializing the variables

```
UPLOAD_FOLDER = 'static\\files'
ALLOWED_EXTENSIONS = {'JPG', 'JPEG', 'PNG', 'jpg', 'jpeg', 'png'}
```

Configuring the Flask application

```
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

Check file extension – validate whether the uploaded files are images. Support JPG, JPEG and PNG file formats

```
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

Routing the path to index.html

```
@app.route("/")
def index():
    return render_template("/index.html")
```

Routing the path to uploader.html

```
@app.route('/uploader', methods = ['GET', 'POST'])
def upload_file():
   result=""
   imagename=""
   image_name=""
   detectedobjects=''
   finaldetectedobjects=''
   if request.method == 'POST':
       file = request.files['pic']
       message=''
       if file.filename == '':
           message="Please select a file"
           flash(message,'error')
```

```
        elif not allowed_file(file.filename):
            message="Only images files are allowed"
            flash(message, 'error')
        else :
            file.save(os.path.join(app.config['UPLOAD_FOLDER'],'image.jpg'))
            message="File uploaded and analysed successfully"
            image_name = "static/files/image.jpg"
            flash(message, 'result')
            finaldetectedobjects=objdetection.detectobj()
            print(finaldetectedobjects)

    return render_template('upload.html', imagename=image_name,
result=finaldetectedobjects)
```

Driver code

```
if __name__ == "__main__":
    app.run(
        debug='true',
        threaded=False
    )
```

index.html

File upload form – gets the file uploaded and saves it in the server

```
<form action = "http://localhost:5000/uploader" method = "POST"
         enctype = "multipart/form-data">
          <input
          class="ui primary button"
          type="file"
          name="pic"
          accept="image/*"
          onchange="readURL(this);"/>
          <button class="ui secondary button" name="btn" value="upload"
style="height: 42px;"> Analyse </button>
</form>
```

Exception handling - Throw the error message to the user

```
{% for mesg in get_flashed_messages() %}
  <p class="errormsg">{{ mesg }}</p>
{% endfor %}
```

uploader.html

Update the image and result to the user

```
<img id="image1" src="{{imagename}}" alt="" />

<h5 id="result_text">{{result}}</h5>
```

Message extraction – extract the embedded message from the image

```
</form>
    <button id="hidetext_button" class="ui secondary button"
onclick="hideText()">
  Hide Message Into Image
</button>
```

index.js

Get the uploaded image URI

```
var imgdatauri;

function readURL(input) {
  if (input.files && input.files[0]) {
    var reader = new FileReader();
    reader.onload = function(e) {
      document.querySelector("#image1").src = e.target.result;
      imgdatauri = e.target.result;
    };
  }
  reader.readAsDataURL(input.files[0]);
}
```

Embed the message into the carrier image

```
function hideText() {
  document.querySelector("#image2").src =
steg.encode(document.querySelector('#result_text').innerHTML,
'http://localhost:5000/static/files/image.jpg');
}
```

Extract the embedded message from image

```
function decode(input) {
    if (input.files && input.files[0]) {
      var reader = new FileReader();

      reader.onload = function(e) {
          console.log(steg.decode(e.target.result));

          document.querySelector('#decoded').innerText =
steg.decode(e.target.result);
      };
    }
    reader.readAsDataURL(input.files[0]);
  }
```

objdetection.py

Perform object detection to the uploaded image and return the result

```
def detectobj():
    allObjects =[]
    execution_path = os.getcwd()

    detector = ObjectDetection()
    detector.setModelTypeAsRetinaNet()
    detector.setModelPath( os.path.join(execution_path ,
"resnet50_coco_best_v2.0.1.h5"))
    detector.loadModel()

    detections =
detector.detectObjectsFromImage(input_image=os.path.join(execution_path ,
"C:\\Users\\kathi\\PycharmProjects\\AIMS\\venv\\static\\files\\image.jpg"),
output_image_path=os.path.join(execution_path ,
"C:\\Users\\kathi\\PycharmProjects\\AIMS\\venv\\static\\files\\detectedobj\
\imagenew.jpg"))
    for eachObject in detections:
        allObjects.append(eachObject["name"])

    allObjectslist= list(dict.fromkeys(allObjects))
    values = {"value": "This image contains " +' '.join(allObjectslist)}
    return values['value']
```

steganography.js

Performs embedding and extraction of image description on the input image using the steganography algorithm [Developed by Peter Eigenschink]

```
! function(a, b, c) {
    "undefined" != typeof module && module.exports ? module.exports = c() :
"function" == typeof define && define.amd ? define(c) : b[a] = c()
}("steg", this, function() {
    var a = function() {},
        b = {
            isPrime: function(a) {
                if (isNaN(a) || !isFinite(a) || a % 1 || 2 > a) return !1;
                if (a % 2 === 0) return 2 === a;
                if (a % 3 === 0) return 3 === a;
                for (var b = Math.sqrt(a), c = 5; b >= c; c += 6) {
                    if (a % c === 0) return !1;
                    if (a % (c + 2) === 0) return !1
                }
                return !0
            },
            findNextPrime: function(a) {
                for (var c = a; !0; c += 1)
                    if (b.isPrime(c)) return c
            },
            sum: function(a, b, c) {
                var d = 0;
                c = c || {};
                for (var e = c.start || 0; b > e; e += c.inc || 1) d +=
a(e) || 0;
                return 0 === d && c.defValue ? c.defValue : d
            },
            product: function(a, b, c) {
                var d = 1;
                c = c || {};
                for (var e = c.start || 0; b > e; e += c.inc || 1) d *=
a(e) || 1;
                return 1 === d && c.defValue ? c.defValue : d
            },
            createArrayFromArgs: function(a, b, c) {
                for (var d = new Array(c - 1), e = 0; c > e; e += 1) d[e] =
a(e >= b ? e + 1 : e);
                return d
            },
            loadImg: function(a) {
                var b = new Image;
                return b.src = a, b
            }
        };
    return a.prototype.config = {
        t: 3,
        threshold: 1,
        codeUnitSize: 16,
        args: function(a) {
            return a + 1
        },
        messageDelimiter: function(a, b) {
            for (var c = new Array(3 * b), d = 0; d < c.length; d += 1)
c[d] = 255;
            return c
        },
        messageCompleted: function(a, b, c) {
            for (var d = !0, e = 0; 16 > e && d; e += 1) d = d && 255 ===
a[b + 4 * e];
            return d
        }
    }, a.prototype.getHidingCapacity = function(a, b) {
        b = b || {};
        var c = this.config,
            d = b.width || a.width,
            e = b.height || a.height,
            f = b.t || c.t,
            g = b.codeUnitSize || c.codeUnitSize;
        return f * d * e / g >> 0
    }, a.prototype.encode = function(a, c, d) {
        if (c.length) c = b.loadImg(c);
        else if (c.src) c = b.loadImg(c.src);
        else if (!(c instanceof HTMLImageElement)) throw new
Error("IllegalInput: The input image is neither an URL string nor an
image.");
        d = d || {};
        var e = this.config,
            f = d.t || e.t,
            g = d.threshold || e.threshold,
            h = d.codeUnitSize || e.codeUnitSize,
            i = b.findNextPrime(Math.pow(2, f)),
            j = d.args || e.args,
            k = d.messageDelimiter || e.messageDelimiter;
        if (!f || 1 > f || f > 7) throw new Error('IllegalOptions: 
Parameter t = " + t + " is not valid: 0 < t < 8');
        var l = document.createElement("canvas"),
            m = l.getContext("2d");
        l.style.display = "none", l.width = d.width || c.width, l.height = 
d.height || c.height, d.height && d.width ? m.drawImage(c, 0, 0, d.width, 
d.height) : m.drawImage(c, 0, 0);
        var n, o, p, q, r, s, t, u, v, w, x = m.getImageData(0, 0, l.width, 
l.height),
            y = x.data,
            z = h / f >> 0,
            A = h % f,
            B = [];
        for (v = 0; v <= a.length; v += 1) {
            if (s = a.charCodeAt(v) || 0, t = A * v % f, t > 0 && o) {
                if (u = Math.pow(2, f - t) - 1, p = Math.pow(2, h) * (1 - 
Math.pow(2, -t)), q = (s & u) << t, r = (o & p) >> h - t, B.push(q + r), v 
< a.length) {
                    for (u = Math.pow(2, 2 * f - t) * (1 - Math.pow(2, -
f)), w = 1; z > w; w += 1) n = s & u, B.push(n >> (w - 1) * f + (f - t)), u 
<<= f;
                    A * (v + 1) % f === 0 ? (u = Math.pow(2, h) * (1 - 
Math.pow(2, -f)), n = s & u, B.push(n >> h - f)) : f >= A * (v + 1) % f + 
(f - t) && (n = s & u, B.push(n >> (z - 1) * f + (f - t)))
                }
            } else if (v < a.length)
                for (u = Math.pow(2, f) - 1, w = 0; z > w; w += 1) n = s & 
u, B.push(n >> w * f), u <<= f;
            o = s
        }
        var C, D, E, F, G, H = k(B, g);
        for (C = 0; 4 * (C + g) <= y.length && C + g <= B.length; C += g) {
            for (G = [], v = 0; g > v && v + C < B.length; v += 1) {
                for (F = 0, w = C; g + C > w && w < B.length; w += 1) F += 
B[w] * Math.pow(j(v), w - C);
                G[v] = 255 - i + 1 + F % i
            }
            for (v = 4 * C; v < 4 * (C + G.length) && v < y.length; v += 4) 
y[v + 3] = G[v / 4 % g];
            E = G.length
        }
        for (D = C + E; D - (C + E) < H.length && 4 * (C + H.length) < 
y.length; D += 1) y[4 * D + 3] = H[D - (C + E)];
        for (v = 4 * (D + 1) + 3; v < y.length; v += 4) y[v] = 255;
        return x.data = y, m.putImageData(x, 0, 0), l.toDataURL()
    }, a.prototype.decode = function(a, c) {
        if (a.length) a = b.loadImg(a);
        else if (a.src) a = b.loadImg(a.src);
        else if (!(a instanceof HTMLImageElement)) throw new 
Error("IllegalInput: The input image is neither an URL string nor an 
image.");
        c = c || {};
        var d = this.config,
            e = c.t || d.t,
            f = c.threshold || d.threshold,
            g = c.codeUnitSize || d.codeUnitSize,
            h = b.findNextPrime(Math.pow(2, e)),
            i = (c.args || d.args, c.messageCompleted || 
d.messageCompleted);
        if (!e || 1 > e || e > 7) throw new Error('IllegalOptions: 
Parameter t = " + t + " is not valid: 0 < t < 8');
```

```
        var j = document.createElement("canvas"),
            k = j.getContext("2d");
        j.style.display = "none", j.width = c.width || a.width, j.height =
c.width || a.height, c.height && c.width ? k.drawImage(a, 0, 0, c.width,
c.height) : k.drawImage(a, 0, 0);
        var l, m, n = k.getImageData(0, 0, j.width, j.height),
            o = n.data,
            p = [];
        if (1 === f)
            for (l = 3, m = !1; !m && l < o.length && !m; l += 4) m = i(o,
l, f), m || p.push(o[l] - (255 - h + 1));
        var q = "",
            r = 0,
            s = 0,
            t = Math.pow(2, g) - 1;
        for (l = 0; l < p.length; l += 1) r += p[l] << s, s += e, s >= g &&
(q += String.fromCharCode(r & t), s %= g, r = p[l] >> e - s);
        return 0 !== r && (q += String.fromCharCode(r & t)), q
    }, new a
});
```

AIMS Extension

manifest.json

```
{

  "description": "This extension will provide alternate text for images.",
  "manifest_version":2,
  "name": "AIMS extension",
  "version": "3",
  "icons": {
    "48": "icons/icon.png"},
  "content_scripts":[
    {
    "matches":["<all_urls>"],
    "js":["content.js"]
    }
  ],
  "background":{
    "scripts":["background.js"]
  },

  "browser_action": {
    "default_popup": "popup.html",
    "default_title": "Accessible Images"
  },

  "permissions": ["tabs","downloads","activeTab","contextMenus"]

}
```

popup.html

Getting the languages as the input from the user

```
<script type="text/javascript" src="steganography.js"></script>
<form name="language_form">
Language
<select id="language_select" name="language">
<option value="en">English</option>
<option value="ta">Tamil</option>
<option value="fr">French</option>
<option value="hi">Hindi</option>
<option value="ka">Kanada</option>
<option value="te">Telugu</option>
</select><br><br>
<input type="button" id="setlang" value="Set">
</form>
<script type="text/javascript" src="popup.js"></script>
```

popup.js

Get the language selected by the user in select box and store it in local storage

```
document.getElementById('language_select').value=localStorage.getItem('sele
cted_value')


const btn = document.getElementById("setlang");


   btn.addEventListener('click', ()=>{
        console.log("button clicked");
      let queryoptions = {active:true, currentWindow: true };
      chrome.tabs.query(queryoptions, (tabs)=>{
         console.log(tabs[0].id);
         var lang = document.querySelector('#language_select').value;
         console.log(lang);
         localStorage.setItem("selected_value",lang);

         var svalue = localStorage.getItem('selected_value');
           console.log("lang "+svalue);
            let language = {
          a: svalue
           };

            chrome.tabs.query({active: true, currentWindow: true},
function(tabs) {
            chrome.tabs.sendMessage(tabs[0].id, language,
function(response) {
            console.log(response);
            });
           });
      });
   });
```

Get the language stored in local storage

```
var svalue = localStorage.getItem('selected_value');
   console.log("lang "+svalue);
    let language = {
      a: svalue
   };
```

Pass the value to content scripts

```
chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
  chrome.tabs.sendMessage(tabs[0].id, language, function(response) {
    console.log(response);
  });
});
```

content.js

Get the message from popup scripts

```
chrome.runtime.onMessage.addListener(receiver);

function receiver(request, sender, sendResponse){
   const lang = request.a;
   localStorage.setItem("langvalue",lang);
  }
```

Get the image tag present on the website

```
var images = document.getElementsByTagName('img');
for(var i = 0; i < images.length; i++)
{
    if(images[i].alt===""||images[i].alt==='null')
    {
        imgurl=images[i].src;
        console.log(imgurl);
         msg=steg.decode(imgurl);
         console.log(msg);
        x = localStorage.getItem("langvalue");
         let obj = {
            b: x,
            c: msg,
            d: imgurl
        };

        dotranslate(obj);
    }

}
```

Perform translation for the image description

```
function dotranslate(obj){

    fetch('http://localhost:5001/dotranslate',{
        method: "POST",
        body: JSON.stringify(obj),
        cache:"no-cache",
        headers: new Headers({
            "content-type": "application/json"
        })
    })
    .then(function(response){
        if(response.status !== 200){
            console.log(`Response status was not 200: ${response.status}`);
            return ;
        }

        response.json().then(function (data){

                for(var i = 0; i < images.length; i++)
                {
                    alter.push(images[i].src)
                }
                for(var i=0; i < images.length; i++)
                {
                    if (alter[i] == Object.keys(data)){
                    images[i].setAttribute('alt', data[Object.keys(data)]
);
                    }
                }

        })
    })
}
```

app.py

Configuring the flask application

```
app = Flask(__name__)
CORS(app)
PORT = 5001
```

Routing the path to dotranslate

dotranslate function translates the image description using python google trans module

```
@app.route('/dotranslate', methods=['GET', 'POST'])
def dotranslate():
    a = 'key'
    data = request.get_json()
    language = data['b']
    objects = data['c']
    imageurl = data['d']
    print("lang : "+ language +" obj : "+objects+ " imgurl : "+imageurl)

    translator = Translator()
    trans = translator.translate(objects, dest=language)

    print(trans)
    return {imageurl : trans.text}
```

Driver code

```
if __name__ == "__main__":
    app.run(
        host='0.0.0.0',
        port=PORT,
        debug=True,
        threaded=True
    )
```

CHAPTER 7

INPUTS

AIMS BUILDER

- AIMS builder takes **images** as input from the user. These images are passed to the machine learning model as inputs.

  For example.,

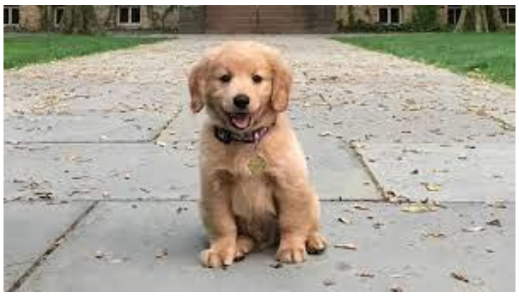


Figure 5 – Sample input

- AIMS builder uses the output (detected objects in the image) of the ML layer
  For example, For the above input image returns a **dog** which is used by the AIMS builder. AIMS builder uses a steganography algorithm to embed the detected objects into the input image.

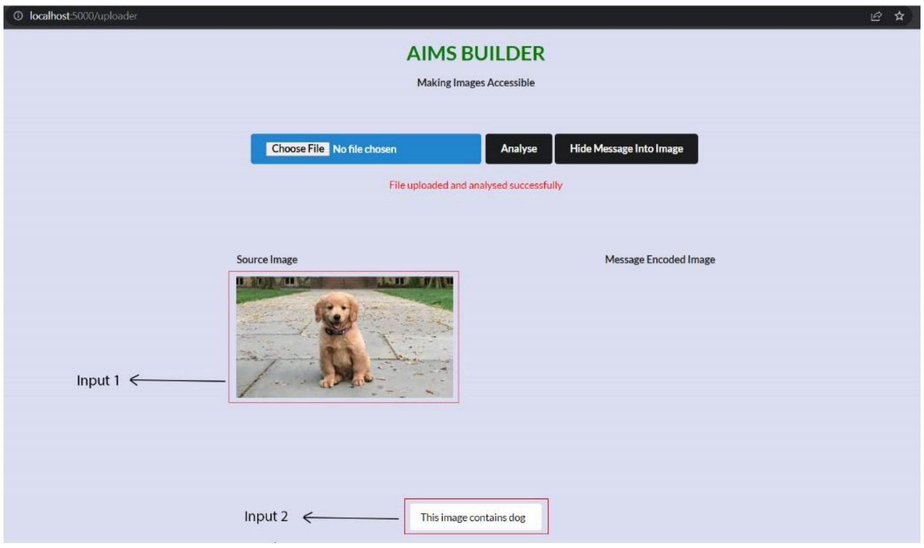


Figure 6 – AIMS builder user interface

AIMS EXTENSION

- AIMS extension takes the language as the input from the users.

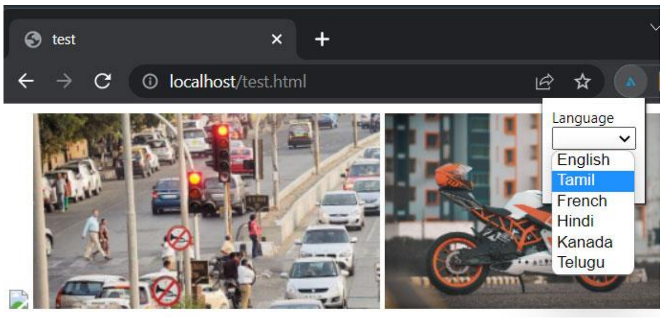
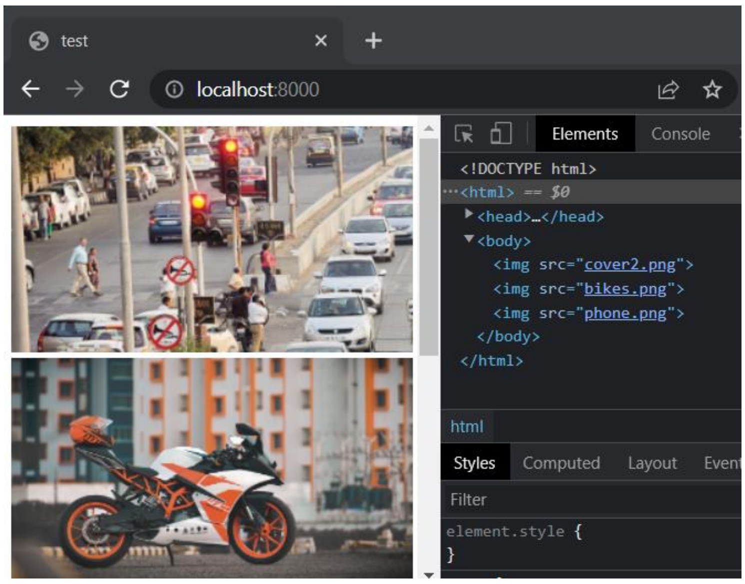


Figure 7 – AIMS extension user interface

- AIMS extension also uses the **imagesrc** (image URL) of the images present on the website as input.
  For example, http://localhost/images/bikes.png



```
<!DOCTYPE html>
<html> == $0
  <head>…</head>
  <body>
    <img src="cover2.png">
    <img src="bikes.png">
    <img src="phone.png">
  </body>
</html>
```

```
element.style {
}
```

Figure 8 – Test site

- Translator module – which is the part of the browser extension that takes the message in one language as input and processes it.
  For example, this image contains a motorcycle

CHAPTER 8

OUTPUTS

AIMS BUILDER

- On taking the plain image and image description as inputs, AIMS builder returns the message embedded image as outputs.

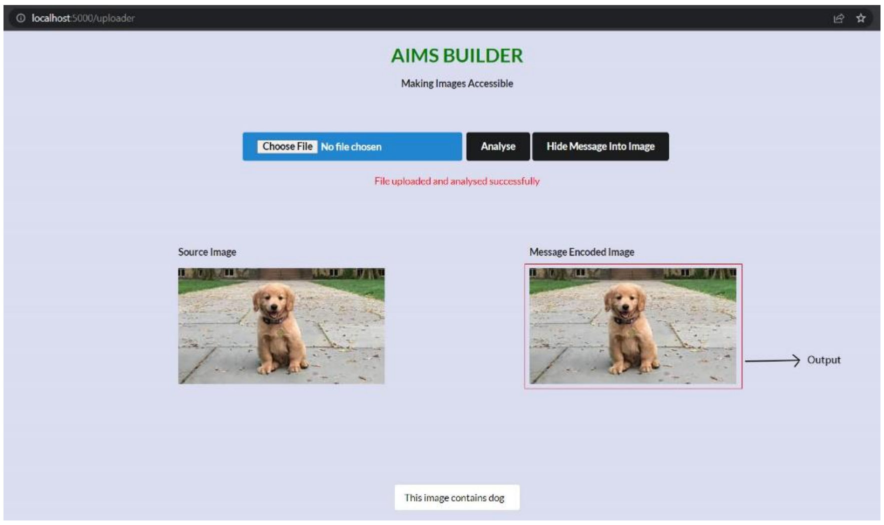
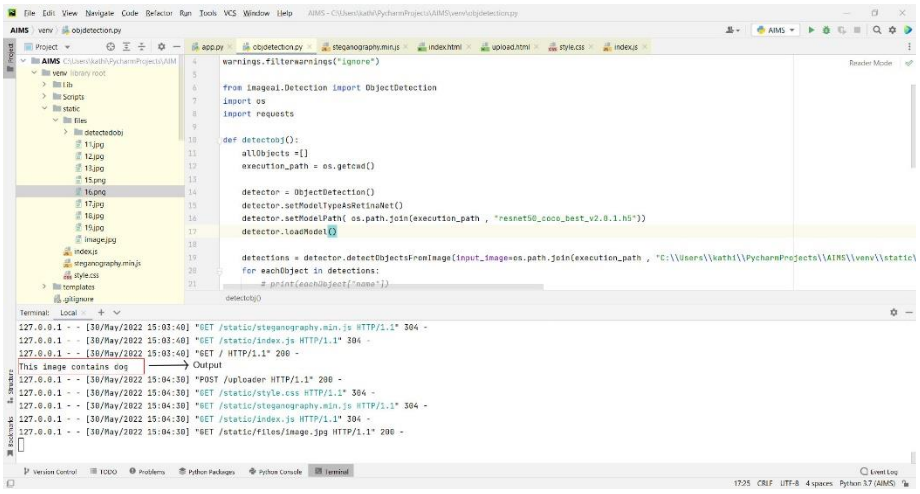


Figure 9 – Annotated image as output

- This output image can be used/included on the websites.
- The message description is generated automatically as output by the machine learning model.

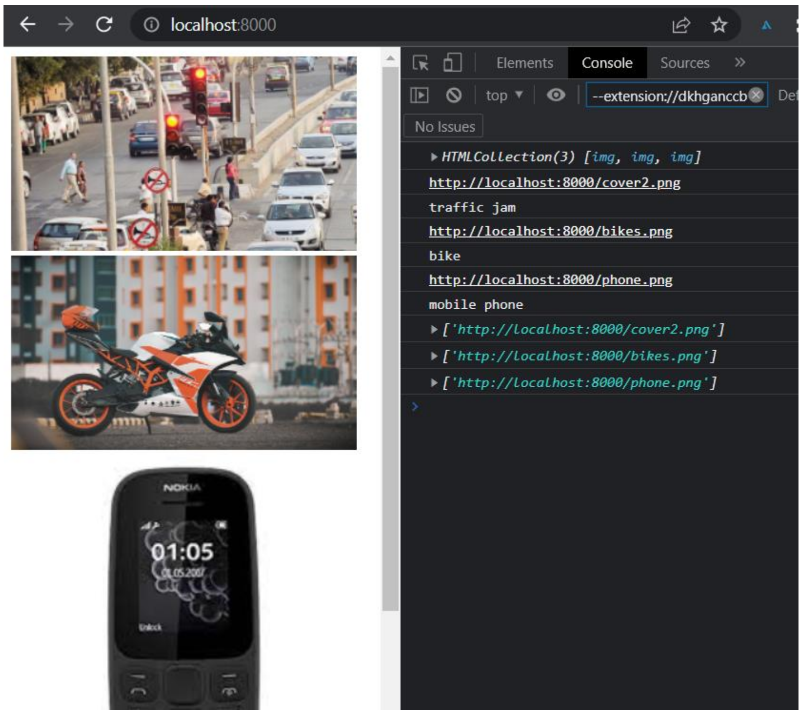

```python
warnings.filterwarnings("ignore")

from imageai.Detection import ObjectDetection
import os
import requests

def detectobj():
    allObjects =[]
    execution_path = os.getcwd()

    detector = ObjectDetection()
    detector.setModelTypeAsRetinaNet()
    detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.0.1.h5"))
    detector.loadModel()

    detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "C:\\Users\\kathi\\PycharmProjects\\AIMS\\venv\\static\
    for eachObject in detections:
        # print(eachObject["name"])
```

```
127.0.0.1 - - [30/May/2022 15:03:40] "GET /static/steganography.min.js HTTP/1.1" 304 -
127.0.0.1 - - [30/May/2022 15:03:40] "GET /static/index.js HTTP/1.1" 304 -
127.0.0.1 - - [30/May/2022 15:03:40] "GET / HTTP/1.1" 200 -
This image contains dog
127.0.0.1 - - [30/May/2022 15:04:30] "POST /uploader HTTP/1.1" 200 -
127.0.0.1 - - [30/May/2022 15:04:30] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [30/May/2022 15:04:30] "GET /static/steganography.min.js HTTP/1.1" 304 -
127.0.0.1 - - [30/May/2022 15:04:30] "GET /static/index.js HTTP/1.1" 304 -
127.0.0.1 - - [30/May/2022 15:04:30] "GET /static/files/image.jpg HTTP/1.1" 200 -
```

Figure 10 – Performs object detection

AIMS EXTENSION

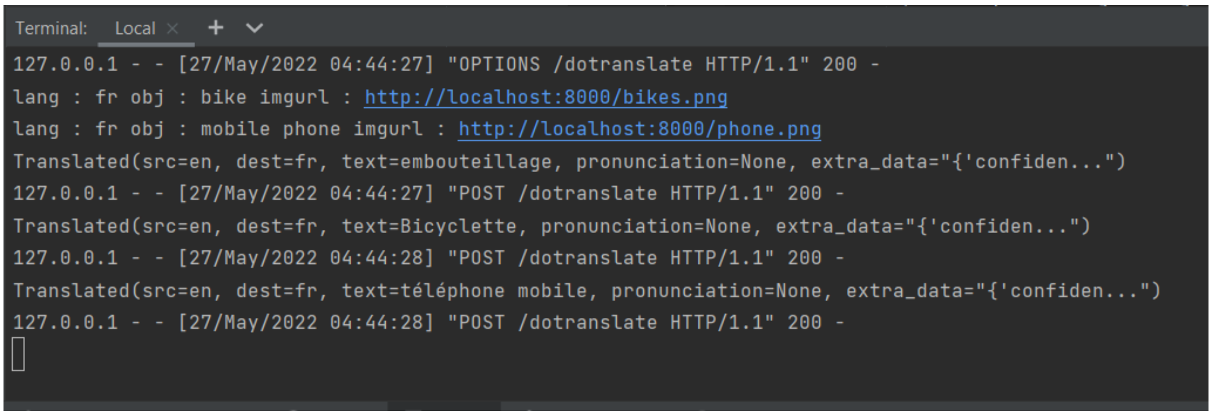
- AIMS extension extracts the embedded text in the images and returns the result as outputs.



```
▸HTMLCollection(3) [img, img, img]
http://localhost:8000/cover2.png
traffic jam
http://localhost:8000/bikes.png
bike
http://localhost:8000/phone.png
mobile phone
▸['http://localhost:8000/cover2.png']
▸['http://localhost:8000/bikes.png']
▸['http://localhost:8000/phone.png']
```

Figure 11 – AIMS extension in background

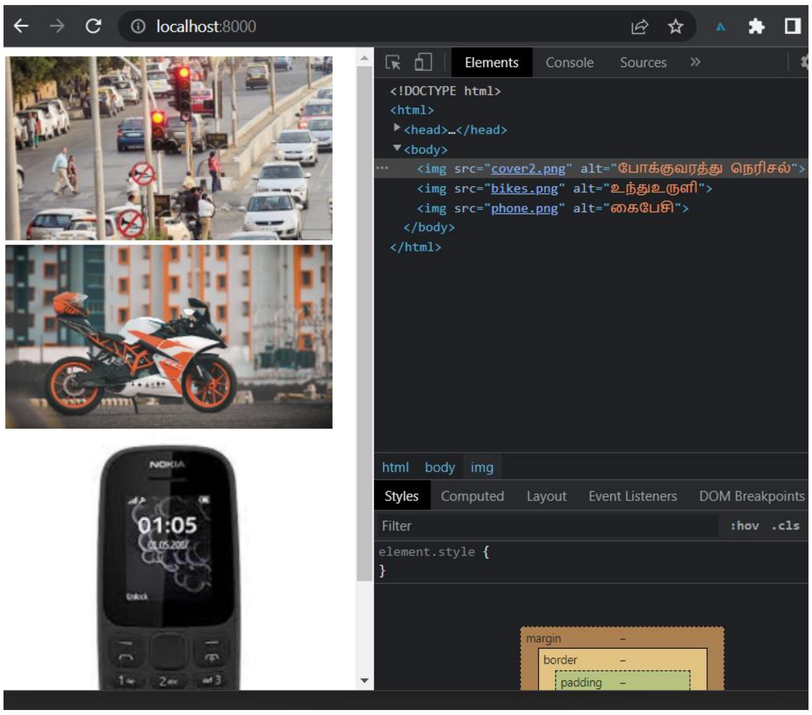
- Then the extracted messages are translated to the user's preferred language



```
127.0.0.1 - - [27/May/2022 04:44:27] "OPTIONS /dotranslate HTTP/1.1" 200 -
lang : fr obj : bike imgurl : http://localhost:8000/bikes.png
lang : fr obj : mobile phone imgurl : http://localhost:8000/phone.png
Translated(src=en, dest=fr, text=embouteillage, pronunciation=None, extra_data="{'confiden...")
127.0.0.1 - - [27/May/2022 04:44:27] "POST /dotranslate HTTP/1.1" 200 -
Translated(src=en, dest=fr, text=Bicyclette, pronunciation=None, extra_data="{'confiden...")
127.0.0.1 - - [27/May/2022 04:44:28] "POST /dotranslate HTTP/1.1" 200 -
Translated(src=en, dest=fr, text=téléphone mobile, pronunciation=None, extra_data="{'confiden...")
127.0.0.1 - - [27/May/2022 04:44:28] "POST /dotranslate HTTP/1.1" 200 -
```

Figure 12 – Performs language translation

- This translated message description is updated in the alt attribute of the img tag which can be read by screen readers.



```
<!DOCTYPE html>
<html>
 <head>…</head>
 <body>
    <img src="cover2.png" alt="போக்குவரத்து நெரிசல்">
    <img src="bikes.png" alt="உந்துஉருளி">
    <img src="phone.png" alt="கைபேசி">
  </body>
</html>
```

```
element.style {
}
```

Figure 13 – Alt text updated

CHAPTER 9
ANALYSIS

AIMS ecosystem is built to assist the screen reader users to access the web content more easily by providing the alt attributes to the image tags. This system improves usability and makes the life of visually impaired users easy. On analysis, we found that our AIMS system describes the images before navigating to them and VI users are not forced to wait for the description for a long time. This system improved the overall usability and understandability of the given web pages by describing the images which were critical to understanding the content of the web page. For evaluation, we designed a website which contains 4-6 images and it is tested with our AIMS system, the overall time we need to wait for our AIMS extension to describe images is less than 4 seconds. AIMS extension extracts the message from the image, then perform translation as per the user preferences and sets the alternative text in the alt attributes of the img tag. The entire process is done at a higher speed to as not to make the user force wait for the image description for a long time.

CHAPTER 10

CONCLUSION

In the modern world, the Internet is very popular and almost all the business operations are carried out through the internet only. In that particular, webpages are so common media they need to be designed in such a way that they can be accessed by all people of the world irrespective of their ability. Visually impaired persons access the computer as well as websites via screen readers only, but images present on the websites are acting barriers to them. To avoid that, developers need to follow the web accessibility standards and guidelines to make it accessible to all. Using alternative text to images is a good practice as well as it helps people with special needs. But on the other hand, it makes developers' life harder, which can be eliminated by the AIMS design. The ML model is built in such a way that uses computer vision to identify the object's detection and it can be embedded into an image using steganography methods. Then at the end-user (VI User) side, the alternate text is updated in the image as per the user's preferred language. If the user is visiting the site in another language, screen readers read the website in the language the webpage renders. But alt text is only read in English which can be solved by our AIMS design. This multilingual support helps the VI user to surf the webpage without any hustle.

CHAPTER 11

FUTURE ENHANCEMENTS

In the current design of the AIMS ecosystem, we use the steganography algorithm to embed the alt message inside the image which can be extracted using the AIMS browser extension. This AIMS system is enhanced by making a browser extension fully automatic, which scraps all the image tag and send it to the ML model to get the image captions and update them in the alt attributes of image tags. This AIMS design is further improvised by implementing Natural language processing (NLP) where the ML model itself identifies the content of the images based on the page context. This future design helps a lot of screen readers as well as saves a considerable amount of developers' time.

CHAPTER 12

REFERENCES

[1] Ab Shaqoor Nengroo & K. S. Kuppusamy : Accessible images (AIMS): a model to build self-describing images for assisting screen reader users [2017]

[2] Md Rahman, Md. Mahib Hosen Ornob, et.al : An Effective Text Steganographic Scheme Based on Multilingual Approach for Secure Data Communication [2021]

[3] Nandhini Subramanian, Omar Elharrouss, et.al : Image Steganography: A Review of the Recent Advances [2021]

[4] Salwa Shakir Baawi, Dhamyaa AL-Nasrawi, et.al : Improvement of "Text Steganography Based on Unicode of Characters in Multilingual" by Custom Font with Special Properties Improvement of "Text Steganography Based on Unicode of Characters in Multilingual" by Custom Font with Special Properties [2020]

[5] Soheyle Amiran, Khaled Rasheed, et.al : Automatic Image and Video Caption Generation With Deep Learning: A Concise Review and Algorithmic Overlap [2020]

[6] Omkar Sargar, Shakti Kinger : Image Captioning Methods and Metrics [2021]

[7] Priyanka Malhotra, Ekansh Garg : Object Detection Techniques: A Comparison [2020]

[8] Sarathak Haldar : Design and Implementation of an Image Classifier using CNN [2019]

[9] Uran oh, Hwayeon Joh, et.al : Image Accessibility for Screen Reader Users: A Systematic Review and a Road Map [2021]

[10] Meredith Ringel Morris, Jazette Johnson, et.al : Rich Representations of Visual Content for Screen Reader Users [2018]

[11] Dae-Jea Cho : A Study on Web Accessibility Improvement Using Alternative Text Watermarking [2017]

[12] Park, E., Lim, H.: A study on providing alternative text of image for web accessibility improvement. Int. J. Appl. Eng. Res. 11(2), 762–765 (2016)

[13] Pennsylvania State University: Image alt tag tips for html. http://accessibility.psu.edu/images/imageshtml/ (2016)

[14] Fang, H., Gupta, S., Iandola, F., Srivastava, R.K., Deng, L., Dollár,P., Gao, J., He, X., Mitchell, M., Platt, J.C., et al.: From captions to visual concepts and back. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp.1473–1482 (2015)

[15] Steve, F.: Html5: Techniques for providing useful text alternatives. Htt1ps://www.w3.org/TR/2011/WD-html-alt-techniques-20110113/ (2017)

[16] Park, E., Lim, H.: A study on improvement of evaluation method on web accessibility automatic evaluation tool's <img> alternative texts based on OCR (2015)

[17] Eggert, E., Abou-Zahra, S.: Images concepts. https://www.w3.org/WAI/tutorials/images/tips/ (2015)

[18] Hamid, N., Yahya, A., Ahmad, R.B., Al-Qershi, O.M.: Image steganography techniques: an overview. Int. J. Comput. Sci. Secur.6(3), 168–187 (2012)

[19] Python documentation: https://docs.python.org/3/tutorial/index.html

[20] JavaScript documentation: https://www.w3schools.com/js/DEFAULT.asp

[22] Python Flask Documentation: https://flask.palletsprojects.com/

[23] Image AI Documentation: https://imageai.readthedocs.io/en/latest/

[24] Tensorflow documentation: https://www.tensorflow.org/tutorials

[25] Chrome extension manifest V2 documentation: https://developer.chrome.com/docs/extensions/mv2/

[26] Googletrans documentation: https://py-googletrans.readthedocs.io/en/latest/