

BOOK DETECTION USING OPENCV

Review Report – II

**Submitted To:**

Professor

Dr. Monali Gulhane

**Submitted By:**  
Bharat Singh Parihar ,22070521005

3rd Sem, Section A

Tirthak Girish Likhar, 22070521041

3rd Sem, Section A

Ameen Khan ,22070521007

3rd Sem, Section A

6 April, 2024

# Identification of the Tools for Designing and Developing a Book Detection System for Blind People using OpenCV

**1. Development Environment:**

Programming Language: Python is used due to its simplicity and extensive libraries for computer vision (OpenCV) and text processing.

IDE (Integrated Development Environment): Visual Studio Code offer code editing, debugging, and project management functionalities for this project.

**2. Computer Vision Library:**

OpenCV (Open Source Computer Vision Library): This is the core library for image processing and computer vision tasks. It provides functions for image capture, manipulation, feature detection, and object recognition.

**3. Additional Libraries Used:**

Tesseract OCR (Optical Character Recognition): The system to read text snippets from the book cover (e.g., title, author), Tesseract is used for character recognition.

Text-to-Speech (TTS) Engine: Libraries like gTTS (Google Text-to-Speech) is used to convert text (extracted book title) into audio for user feedback.

**4. Hardware:**

Camera: A webcam or a smartphone camera can be used to capture images of the book.

Processing Unit: A computer with a decent CPU or a single-board computer like Raspberry Pi can handle the processing tasks.

**5. Development Process Tools:**

Version Control System (VCS): Git is used as it is a popular VCS for tracking code changes and collaboration.

Text Editor: A basic text editor like Notepad is used for configuration files or simple scripts.

**Detailed Explanation:**

Development Environment: Python offers a good balance of ease of use and power for computer vision tasks.

OpenCV: This is the workhorse for image processing. We'll use OpenCV functions to capture an image of the book, convert it to grayscale, apply filters for noise reduction, detect edges, and potentially identify rectangular shapes like a book cover.

Tesseract OCR: Once a book cover is identified, Tesseract can be used to extract text from the cover image. This allows the system to announce the book title or author using a TTS engine.

Text-to-Speech: TTS engine converta extracted text into audio for user feedback. This allows the user to receive information about the book through voice.

Hardware: A camera is essential to capture the book image. The processing unit should be able to handle image processing tasks efficiently

Additional Considerations:

User Interface (Optional): A basic graphical user interface (GUI) is developed to provide a user-friendly experience for visually impaired users who rely on screen readers or audio instructions.

By combining these tools and following a structured development process, we can create a book detection system that empowers visually impaired people to interact with physical books through computer vision and text-to-speech functionalities.

# Identify Suitable Criteria/Parameters for Evaluating a Book Detection System for Blind People (using OpenCV)

**1. Accuracy:**

Book Detection Rate: Measuring the percentage of times the system correctly identifies a book in the image compared to the total number of test images.

False Positives: Tracking the number of times the system incorrectly identifies a non-book object as a book.

False Negatives: Counting the number of times the system misses a book present in the image.

**2. Speed:**

Processing Time: Measuring the average time taken by the system to process an image and provide feedback (audio or text).

**3. Robustness:**

Lighting Variations: Evaluating how well the system performs under different lighting conditions (bright light, low light, etc.).

Background Clutter: Testing the system's ability to detect books in images with background clutter (e.g., bookshelf, table).

Book Orientation: Assess if the system can identify books regardless of their orientation (upright, flat, slightly tilted).

**4. Text Recognition:**

Character Recognition Accuracy: Measures the percentage of characters correctly recognized by Tesseract from the book cover image.

Word Recognition Accuracy: Evaluates how well the system can recognize entire words extracted from the cover.

**5. User Experience:**

Ease of Use: A user interface is implemented, assessed how easy it is for visually impaired users to navigate and interact with the system.

Audio Feedback Clarity: Evaluates the clarity and understandability of the Text-to-Speech generated audio for book information.

By evaluating our book detection system using these criteria, we can objectively assess its strengths and weaknesses and make data-driven decisions for improvement. This ensures the system effectively meets the needs of visually impaired users in interacting with physical books.

# Designing and Developing a Book Detection System (OpenCV, Python) using modern tools

**1. Development Environment Setup:**

* Install Python (version 3.x recommended).
* Choose an IDE like PyCharm or Visual Studio Code.
* Install OpenCV library using `pip install opencv-python`.
* Install additional libraries: `pip install pytesseract gTTS` (for Tesseract OCR and Text-to-Speech).

**2. System Design:**

* The system will capture an image using the camera.
* OpenCV functions will process the image for book detection.
* Tesseract OCR can be used to extract text from the book cover.
* A Text-to-Speech engine can convert extracted text to audio feedback.

3. Code Implementation (Basic Structure):

import cv2

# Import other libraries (Tesseract, gTTS)

def detect\_book(image):

# Pre-processing (grayscale conversion, noise reduction)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Apply filters (e.g., blurring)

# Edge detection (identify rectangular shapes)

edges = cv2.Canny(gray, 75, 200)

# Find contours (closed shapes)

contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# Loop through contours and identify potential book shapes

for cnt in contours:

approx = cv2.approxPolyDP(cnt, 0.01 \* cv2.arcLength(cnt, True), True)

# Check if the contour has 4 corners (rectangle)

if len(approx) == 4:

# Further checks (aspect ratio, area within limits) for book detection

# Book detected! (Optional: draw rectangle around detected book)

x, y, w, h = cv2.boundingRect(cnt)

cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Use Tesseract OCR to extract text from the book cover region

break # Assuming only one book needs to be detected

# Display the processed image (with or without detected book rectangle)

cv2.imshow("Book Detection", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Capture video from camera

cap = cv2.VideoCapture(0)

while True:

# Read a frame from the camera

ret, frame = cap.read()

# Call the book detection function on the frame

detect\_book(frame.copy())

# Exit on 'q' press

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

**4. Explanation:**

1. This code captures video frames from the camera.
2. Each frame is processed for book detection using grayscale conversion, edge detection, and contour analysis.
3. The code identifies contours with four corners (potential rectangles) and performs additional checks for book-like shapes (e.g., aspect ratio).
4. If a book is detected,a green rectangle is drawn around it.
5. Extract text from the book cover using Tesseract OCR.
6. Convert extracted text to audio using Text-to-Speech for user feedback.

**5. Testing and Refinement:**

Test the system with various images and videos containing books under different conditions.

Analyse performance based on the evaluation criteria mentioned earlier.

Refine the code (e.g., adjusting thresholds, filtering techniques) based on the test results.

# Generate information through appropriate tests to validate the solution

1. Dividing the dataset into training and testing sets (e.g., 70% training, 30% testing). Using the training set to potentially refine your algorithms for object detection.
2. Run the system on the testing set images and capture results for each evaluation metric.
3. Analyse the results:

Identifying areas for improvement (e.g., specific lighting conditions causing issues, low text recognition accuracy).

1. Refine the system based on the analysis:

Adjusting image processing parameters, exploring different filtering techniques, or considering retraining the model.

1. Conduct user studies after refining the system to ensure it meets the needs and expectations of visually impaired users.

# References (If Any):