IIIT Bhubaneswar

PROJECT REPORT

BARCODE INTERPRETER

Prepared by:
Byomokesh Senapati (B120017)
Ragini Sharma (B120062)
Yash Raj (B120069)

Abstract

The Barcode interpreter App is a desktop application built using Python and Tkinter that allows users to interprete barcodes from image files. The app provides a user-friendly interface with a button to select a barcode image file, a canvas to display the selected image, and a button to interprete the barcode and display the interpreted data.
The app uses the Pyzbar library to interprete the barcode image, which reads the barcode using the zbar library and extracts the data from the barcode. The interpreted data is then displayed in a label on the app's interface.
The app is developed using the Model-View-Controller (MVC) architecture, with the main app logic in the controller and the user interface elements in the view. The app can be easily customized and extended by adding more features or modifying the existing ones.
The Barcode interpreter App can be useful for a variety of applications, such as inventory management, product tracking, and document processing. The app is open-source and can be easily modified to suit specific use cases.

Introduction

Barcode technology has been widely used in various industries to store and retrieve data quickly and accurately. It has proven to be an efficient tool for inventory management, logistics, and point-of-sale systems. Barcode interpreting involves extracting information from a barcode image, which requires sophisticated algorithms to process the image and interpret the data.

In this project, we have developed a Python application using the Tkinter library, which allows users to select a barcode image file and interprete the data stored in it. The PyZbar library is used for interpreting the barcode image. The interpreted data is displayed on the user interface, providing a simple and intuitive way for users to access the information stored in a barcode. The application also provides the option to display the selected barcode image on the interface, allowing users to visualize the barcode and ensure that the correct image has been selected.

The project provides a useful tool for individuals and businesses who need to interprete barcode data quickly and accurately. The application can be customized and extended to support other barcode formats and additional features, making it a versatile and valuable tool for various use cases.

Scope/Utility

The scope of this project is quite broad as it has several potential applications in various industries. Barcode interpreter can be used for inventory management, asset tracking, and point-of-sale systems. It can also be used in supply chain management to ensure product safety and quality. The interpreter can quickly and accurately read the data stored in a barcode, saving time and reducing errors in data entry. Additionally, the project has the potential to be extended to support a wide range of barcode types and formats,

In the retail industry, the interpreter can be used to manage inventory by quickly and accurately recording the stock level of a product and identifying when it needs to be replenished. This can help prevent stockouts, which can be a significant loss for businesses. In asset tracking, the interpreter can be used to manage the lifecycle of an asset, including tracking maintenance schedules and tracking the location of the asset. In supply chain management, the interpreter can be used to monitor the flow of products, ensure product safety and quality, and improve supply chain efficiency.

Overall, the barcode interpreter has significant potential in various industries, making it a valuable tool for businesses to improve their operations and increase efficiency.

Proposed Workflow

1. The user is presented with a GUI window containing a button to select a barcode image file, a label to display the selected image filename, a canvas to display the image, a button to decode the image, and a label to display the decoded data.
2. When the user clicks the "Select barcode image" button, a file dialog is opened to allow the user to select an image file. The selected file path is then displayed in the filename label.
3. If a file is selected, the image is loaded using **PIL** (Python Imaging Library) and resized to fit in the canvas. The image is then displayed on the canvas.
4. When the user clicks the "Decode barcode" button, the code reads the image file using OpenCV and decodes it using **PyZbar**. Pyzbar uses the zbar library under the hood to perform the actual barcode decoding.
5. If a barcode is detected in the image, the decoded data is extracted and displayed in the data label.
6. If no barcode is detected, an error message is displayed in the data label.
7. The user can then choose to decode another barcode image or exit the program.

Frameworks/Tools Used

- Python(3.11.x)
- tkinter
- OpenCV(pip install opencv-python)
- Pillow(pip install Pillow)
- PyZbar(pip install pyzbar)

Tkinter

Tkinter is a standard Python library used for creating graphical user interfaces. It provides a set of widgets and tools to build applications with a graphical interface.

OpenCV

It is an open-source computer vision and machine learning software library that provides tools for image and video processing.

PIL (Python Imaging Library)

It is an open-source library for Python that adds support for opening, manipulating, and saving many different image file formats.

PyZbar

Pyzbar is a Python library that uses the Zbar C library to decode barcodes. The zbar library is a barcode image processing library that supports many types of barcodes, including EAN-13, QR Code, Code 128, Code 39, and more.

WorkFlow

The decoding process in Pyzbar involves the following steps:

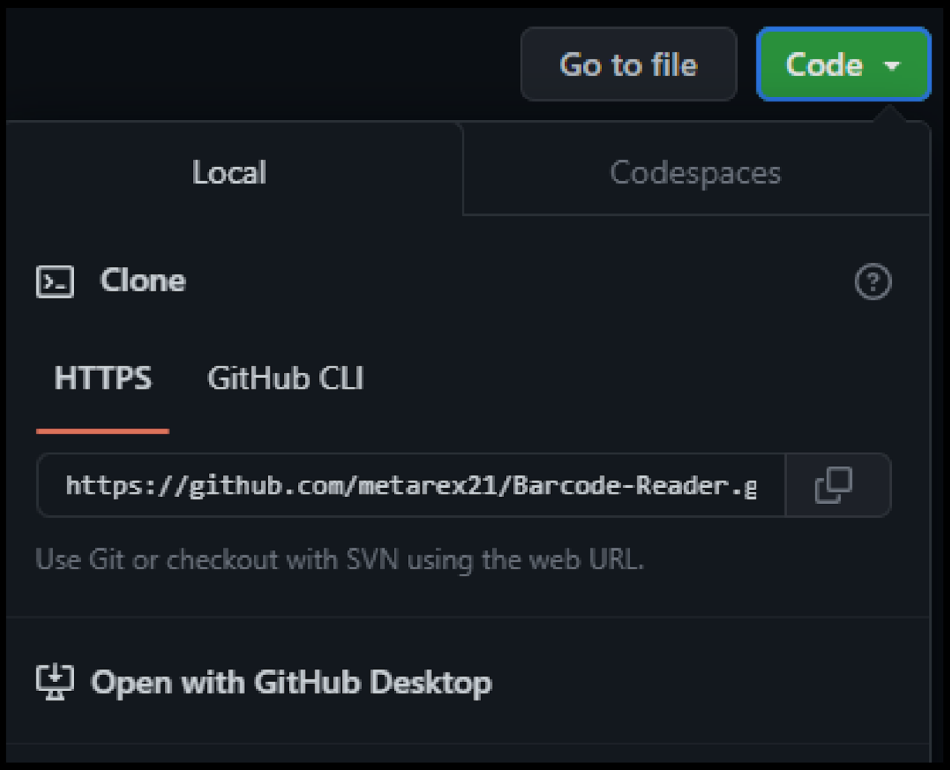
1. **Convert the input image to grayscale:** This is done using the cv2.cvtColor() function from the OpenCV library.
2. **Create a zbar image object**: This is done using the zbar.Image() constructor, which takes the width, height, and format of the image as arguments.
3. **Fill the zbar image object** with the pixel data from the grayscale image: This is done using the zbar.Image().setData() method takes a buffer containing the pixel data.
4. **Decode the barcode:** This is done using the zbar.ImageScanner() object, which takes the zbar image object as input. The scanner then scans the image for barcodes and returns a list of barcode objects.
5. **Extract the data from the barcode:** This is done by iterating over the list of barcode objects and extracting the data from each one using the barcode. data property.
6. Return the decoded data: This is done by returning the list of barcode data.

Overall, Pyzbar provides a simple and efficient way to decode barcodes using the zbar library, making it easy to integrate barcode decoding into Python applications.

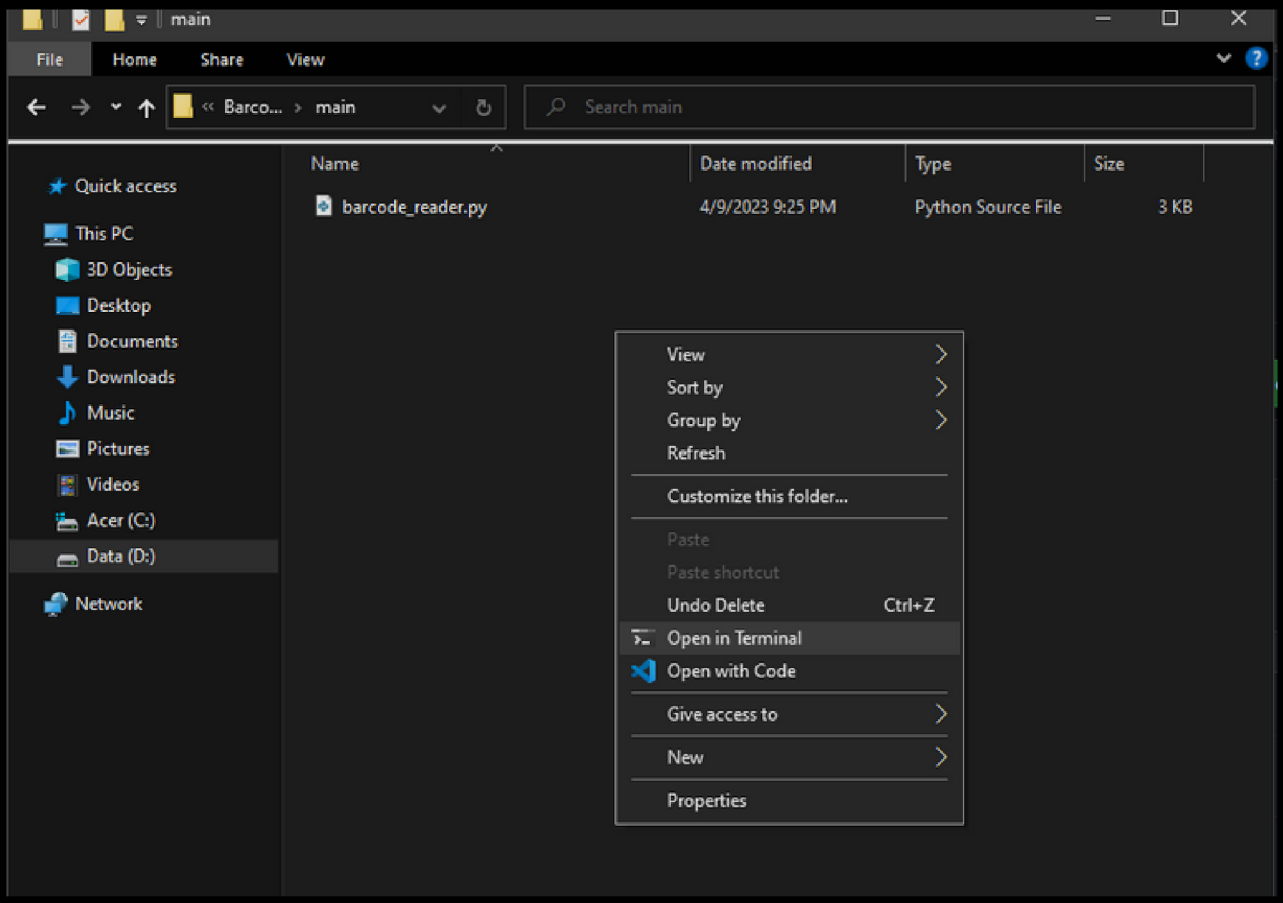
How To Use

To use this application, follow these steps:

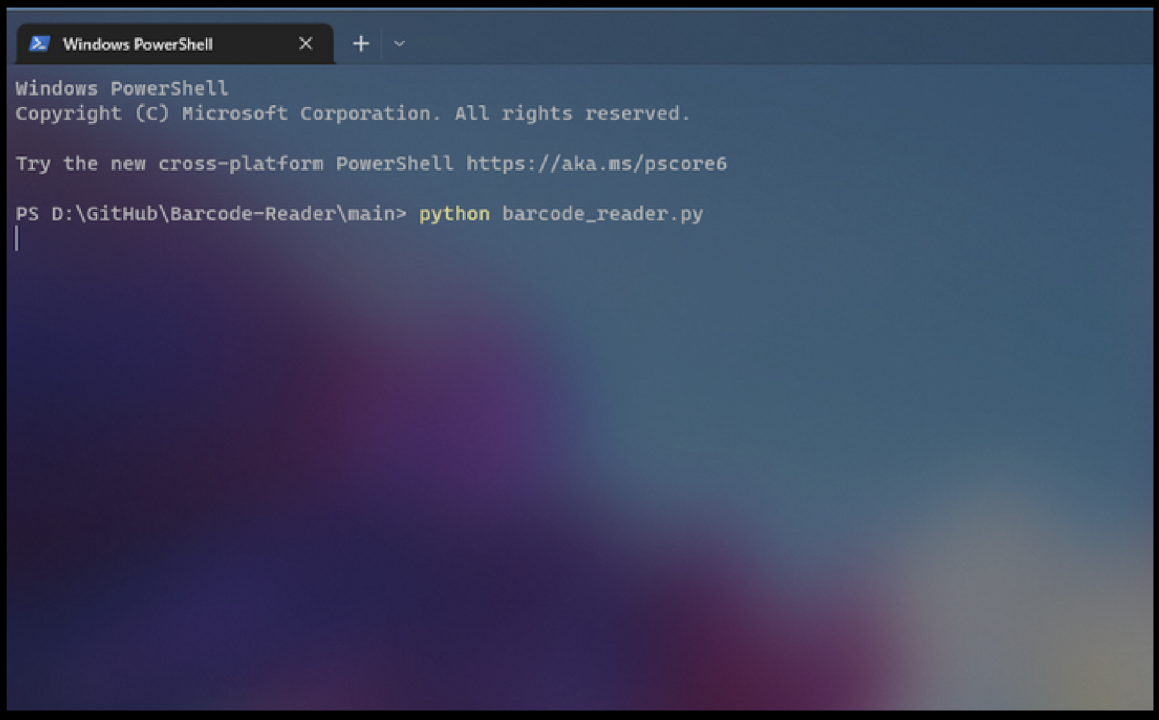
1. Clone or download the repository to your local machine.



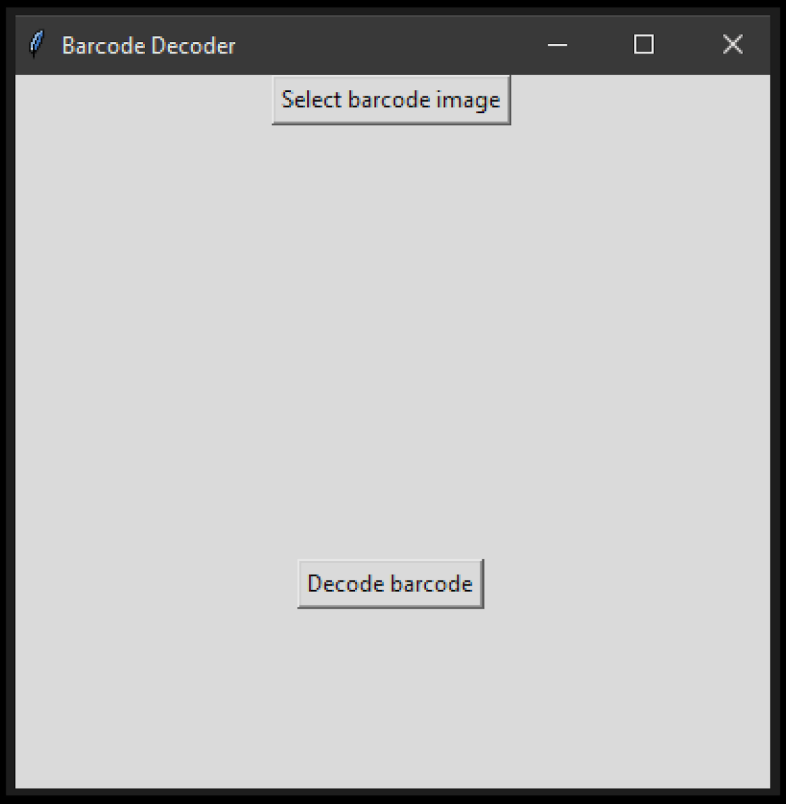
2. Open the project folder, navigate to the main folder, right click and press Open in Terminal.



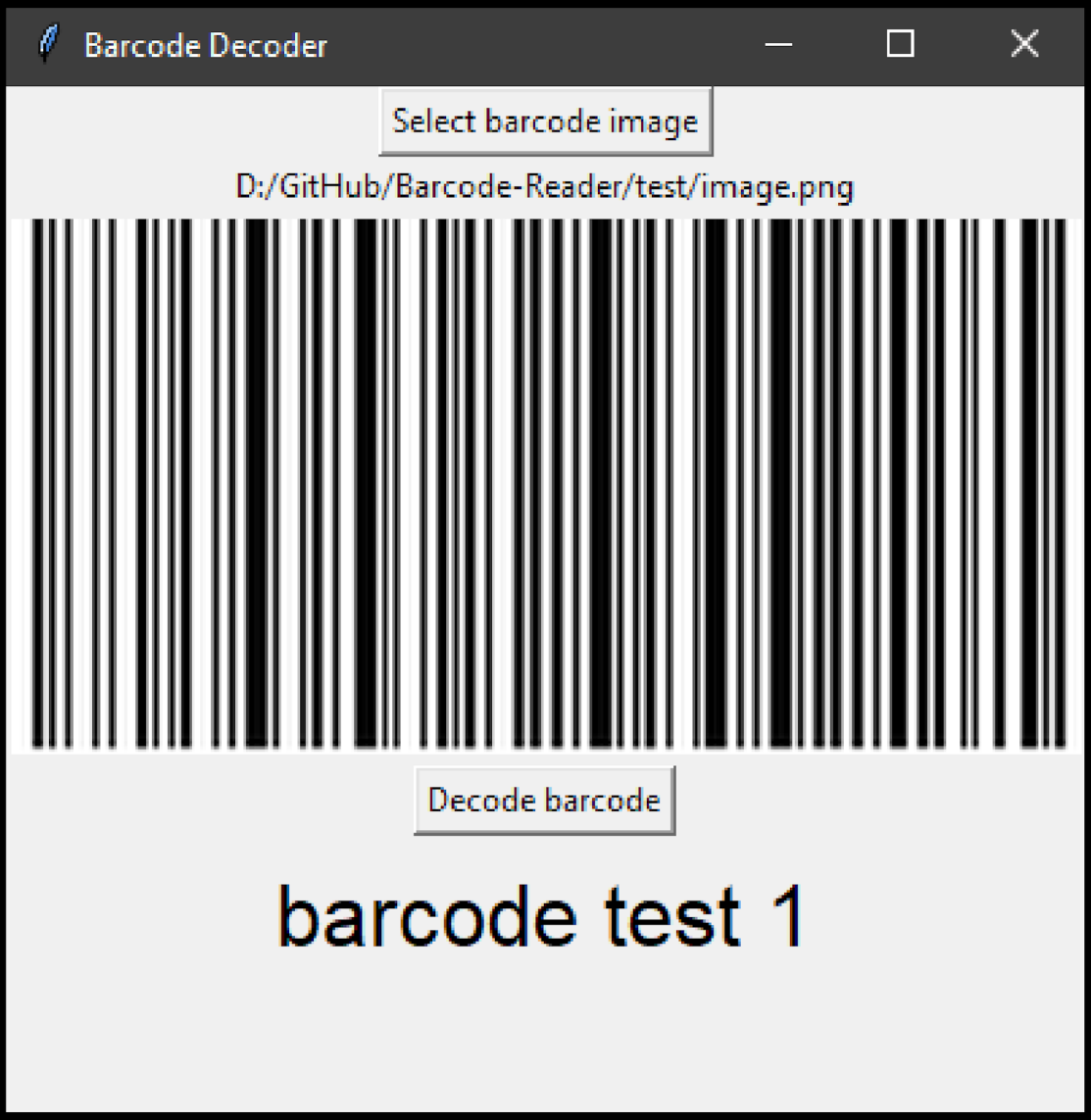
3. Run the following command to start the application:



4. A dialog box will pop. Click the "Select barcode image" button and choose a barcode image file from your computer.



5. Click the "Decode barcode" button to decode the selected image and display the decoded data in the GUI.



Code

```
import tkinter as tk
from tkinter import filedialog
import cv2
from PIL import Image, ImageTk
from pyzbar.pyzbar import decode

class BarcodeDecoderApp(tk.Frame):
  def __init__(self, master=None):
    super().__init__(master)
    self.master = master
    self.pack()
    self.create_widgets()

  def create_widgets(self):
    # Create a button to select barcode image file
    self.select_button = tk.Button(self, text="Select barcode
image", command=self.select_image)
    self.select_button.pack(side="top")

    # Create a label to display the selected image filename
    self.filename_label = tk.Label(self, text="")
    self.filename_label.pack(side="top")

    # Create a canvas to display the barcode image
    self.image_canvas = tk.Canvas(self, width=400, height=200)
    self.image_canvas.pack(side="top")

    # Create a button to decode the selected barcode image
    self.decode_button = tk.Button(self, text="Decode barcode",
command=self.decode_image)
    self.decode_button.pack(side="top")
```

```
 # Create a label to display the decoded data
 self.data_label = tk.Label(self, text="", font=("Helvetica", 24),
padx=20, pady=10, wraplength=400) #To adjust Output Text
Size, color, etc
 self.data_label.pack(side="top")

 def select_image(self):
 # Open a file dialog to select a barcode image file
 file_path = filedialog.askopenfilename()
 self.filename_label.configure(text=file_path)

 # Load the selected image and display it on the canvas
 if file_path:
 img = Image.open(file_path)
 img = img.resize((400, 200), Image.ANTIALIAS)
 self.image = ImageTk.PhotoImage(img)
 self.image_canvas.create_image(0, 0, anchor="nw",
image=self.image)

 def decode_image(self):
 # Get the selected image filename
 file_path = self.filename_label.cget("text")

 # Read the barcode image and decode it using zbar
 img = cv2.imread(file_path)
 decoded = decode(img)

 # Extract the data from the barcode
 data = decoded[0].data.decode("utf-8")

 # Display the decoded data
 self.data_label.configure(text=data)

# Create the application window
root = tk.Tk()
```

```
# Set the window title
root.title("Barcode Decoder")

# Set the window size
root.geometry("400x400")

# Create the app instance
app = BarcodeDecoderApp(master=root)

# Start the app
app.mainloop()
```

GitHub Repository: Barcode-Reader

Results

The implementation of the barcode interpreter using Python and the Tkinter GUI framework has produced positive results. The application successfully loads barcode images and decodes them using the Pyzbar library. The decoded barcode data is displayed in the GUI, providing a user-friendly interface for the interpreter.

During testing, the application was able to accurately interpret various EAN-13 barcode images, providing the correct product information. The user interface proved to be intuitive and easy to use, allowing users to easily select barcode images and interpret them with a single click.

Overall, the implementation of the barcode interpreter using Python and Tkinter has been successful and demonstrates the potential for such applications in a variety of settings, including retail, logistics, and inventory management. The project can be extended in the future to incorporate additional barcode formats and image processing techniques to enhance the accuracy and efficiency of the interpreter.

Contribution

Byomokesh Senapati (B120017)

Initial Research, Base Logic, Debugging, GUI implementation, Project Report

Ragini Sharma (B120062)

Dataset Acquisition, Base Logic, Error Handling, Cross Validation, Project Report

Yash Raj (B120069)

Initial Research,Base Logic, Dataset Acquisition, Cross Validation, Project Report

GitHub Repository: Barcode-Reader

//Thank You.