A Project Report On

**Parking Slots Counting**

Submitted in partial fulfillment of the requirement for the award of the degree

Bachelor of Computer Application (BCA)/ Bachelor of Science (IT)

Academic Year 2023 – 24

**Student-1 – Rohit Luni**

**Enrollment. No - 92100588045**

|  |
| --- |
| **Internal Guide** |
| Vinod Pal |



Rajkot-Morbi Road, At & PO : Gauridad, Rajkot 360 003. Gujarat. India.



**Faculty of Computer Applications (FCA)**

****

**This is to certify that the project work entitled**

**Text Summarizer**

**submitted in partial fulfillment of the requirement for**

**the award of the degree of**

**Bachelor of Computer Application/ Bachelor of Science (IT)**

**of the**

**Marwadi University**

**is a result of the bonafide work carried out by**

**Rohit Luni - 92100588045**

**during the academic year 2023-24.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Faculty Guide** |  | **HOD** |  | **Dean** |

**DECLARATION**

I/Wehereby declare that this project work entitled **Parking Slots Counting** is a record done by me.

I also declare that the matter embodied in this project is genuine work done by me and has not been submitted whether to this University or to any other University / Institute for the fulfillment of the requirement of any course of study.

Place :

Date :

**Rohit Luni – 92100588045 Signature :**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapters** | **Particulars** | **Page No.** |
| **1** | **SYNOPSIS** |  |
| **2**  2.1  2.2 | **PREAMBLE**  General Introduction  Module description |  |
| **3** | **REVIEW OF LITERATURE** |  |
| **4**  4.1  4.2 | **TECHNICAL DESCRIPTION**  Hardware Requirement  Software Requirement |  |
| **5**  5.1  5.2  5.2.1  5.3  5.3.1  5.3.2  5.3.3  5.4  5.4.1  5.4.2 | **SYSTEM DESIGN AND DEVELOPMENT**  **(Only applicable diagrams)**  **Architectural Design**   * Class Diagram   **Dynamic Modeling**   * Use Case Diagram * Sequence Diagram * Activity Diagram * Any other applicable diagram (applicable)   Database Design (If applicable)  Relationship Diagram (ER)  Menu Design  Screen Design |  |
| **6** | **TESTING** |  |
| **7** | **CONCLUSION** |  |
| **8** | **LEARNING DURING PROJECT WORK** |  |
| **9** | **BIBLIOGRAPHY** |  |

**1.SYNOPSIS**

The project aims to develop a computer vision-based system for counting the available parking slots in a designated parking area. It leverages the power of OpenCV (Open Source Computer Vision Library) and a camera module connected to a Raspberry Pi single-board computer.

The system works by capturing live video feed from the camera, which is strategically positioned to overlook the parking area. The video frames are then processed using OpenCV algorithms to detect and count the vacant parking slots.

**The core functionality involves several steps:**

**1.Image Acquisition:** The Raspberry Pi captures video frames from the connected camera module at a specified frame rate.

**2.Image Processing:** The captured frames undergo various image processing techniques, such as color space conversion, filtering, and edge detection, to enhance the visibility of parking slot markings and vehicles.

**3.Object Detection:** Computer vision algorithms, like blob detection or contour analysis, are applied to identify and localize the parking slots and vehicles within the processed frames.

**4.Vacancy Detection:** By analyzing the detected objects, the system determines which parking slots are occupied by vehicles and which ones are vacant.

**5.Counting and Display:** The number of available parking slots is calculated and displayed on a user interface or transmitted to a remote monitoring system.

**2.PREAMBLE**

**2.1.General Introduction**

**Project Overview:**

This project involves developing a computer vision-based system for counting available parking slots using OpenCV and a video feed from a parking camera. It processes the video frames, detects vacant and occupied parking slots, and updates the parking data in a Firebase Realtime Database. A Next.js front-end web application fetches this real-time parking data from Firebase and displays the current status of each parking slot, enabling users to quickly identify available spaces. The system aims to provide real-time parking availability information and improve the parking experience for drivers.

**Backend:**

**1.Video Acquisition and Processing:**

**-** The backend code reads a video file using OpenCV's `VideoCapture` function and processes the frames using various techniques like grayscale conversion, Gaussian blurring, adaptive thresholding, and median blurring to enhance the visibility of parking slots and vehicles.

**2.Parking Slot Detection:**

- It loads the positions of parking slots from a file containing the coordinates of each slot.

- For each detected slot, it extracts a cropped region from the processed image and counts the non-zero pixels to determine if the slot is occupied or vacant.

**3.Vacancy Detection and Counting:**

- Based on the non-zero pixel count, the code classifies each parking slot as occupied or vacant.

- It draws rectangles around the slots on the original image, with different colors indicating their status.

- The total number of available (vacant) parking slots is calculated.

**4.Firebase Integration:**

- The backend code creates a batch of updates containing the status (occupied or vacant) of each parking slot.

- This batch of updates is sent to the Firebase Realtime Database using the `update\_firebase\_batch` function, ensuring real-time updates.

**5.Technologies and Tools:**

- **OpenCV:** For image processing and computer vision tasks.

- **Python:** The programming language used for the backend code.

- **Pyrebase:** A Python library for interacting with the Firebase Realtime Database.

- **Firebase Realtime Database:** Used for storing and retrieving real-time parking data.

The backend component is responsible for processing the video feed, detecting vacant and occupied parking slots, and updating the real-time parking data in the Firebase Realtime Database. It acts as the data source for the front-end web application.

**Frontend:**

- Built using **Next.js**, a React framework for building server-rendered React applications.

- Fetches real-time parking data from the **Firebase Realtime Database**.

- Displays the current status (occupied or vacant) of each parking slot.

- Provides a user-friendly interface for visualizing the parking availability information.

- Utilizes **JavaScript** for interacting with the database and rendering the parking data.

- Offers a real-time view of parking availability, allowing users to quickly identify vacant spaces.

- Seamlessly integrates with the backend component, which continuously updates the parking data in the database.

The frontend component is a web application that presents the real-time parking data to users in an accessible and intuitive manner. It retrieves the parking slot status from the Firebase Realtime Database, which is continuously updated by the backend component, and renders this information in a user-friendly way, enabling users to easily find available parking spaces.

**2.2.Module description**

**1.cv2:** This is the OpenCV module for Python, which provides a wide range of computer vision and image processing functions. In this project, it is used for tasks such as video capture, image processing, and drawing rectangles on the processed frames.

**2.numpy:** NumPy is a fundamental scientific computing library for Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays. In this project, NumPy is likely used in conjunction with OpenCV for image processing and manipulation tasks.

**3.pyrebase:** Pyrebase is a Python library that simplifies the interaction with Google's Firebase services, including the Realtime Database, Authentication, and Cloud Storage. In this project, it is used to establish a connection with the Firebase Realtime Database and perform read/write operations.

**4.datetime:** The datetime module is part of the Python standard library and provides classes for working with dates and times. In this project, it is used to get the current date and time for logging updates to the Firebase Realtime Database.

**5.os:** The os module in Python provides a way to interact with the operating system. In this project, it is used to access environment variables for the Firebase configuration.

**6.dotenv:** dotenv is a Python module that loads environment variables from a `.env` file into the program's environment. In this project, it is used to load sensitive information, such as Firebase API keys, from an environment file, avoiding the need to hardcode these values in the source code.

**Utilities:**

**1.load\_parking\_positions(file\_path):** This function reads the coordinates of parking slots from a file and returns a list of tuples containing the (x, y) coordinates for each slot.

**2.update\_firebase\_batch(updates):** This function takes a dictionary of slot updates (slot number and status) and batches them together before sending the updates to the Firebase Realtime Database. This approach helps optimize the number of database writes and improves performance.

**3.checkParkingSpace(imgPro):** This function is the core of the parking slot detection and counting logic. It iterates through the list of parking slot positions, crops the corresponding region from the processed frame, and counts the non-zero pixels to determine if the slot is occupied or vacant. It then draws rectangles on the original frame to indicate the status of each slot and updates the Firebase Realtime Database with the current parking availability information.

These modules and utilities work together to enable the computer vision-based parking availability system to process video frames, detect vacant and occupied parking slots, and update the real-time parking data in the Firebase Realtime Database.

**3.REVIEW OF LITERATURE**

The development of text summarization tools has been a significant area of interest in the field of natural language processing and artificial intelligence. Researchers and developers have long sought efficient ways to condense lengthy texts, making information retrieval and content curation more manageable. In this context, OpenAI's Chat-GPT Turbo 3.5 model has emerged as a game-changer. It represents a culmination of advancements in language models, combining the capabilities of GPT-3.5 with a chat-based interface. Chat-GPT Turbo 3.5's ability to comprehend and generate coherent human-like text makes it a natural fit for text summarization tasks. Its integration in this project reflects the state-of-the-art in AI-driven summarization technology.

Django, a popular Python web framework, forms the backbone of the backend in this project. Django's robust and flexible design simplifies the creation of APIs, enabling the efficient handling of HTTP requests. It serves as a testament to the power of Python in web development and demonstrates how versatile web frameworks can be leveraged for AI-driven applications. The backend's integration with the Chat-GPT Turbo 3.5 model showcases the potential of combining mature web development tools with cutting-edge AI technology to deliver practical solutions.

On the frontend, Next.js and TypeScript make for a compelling combination. Next.js simplifies React application development with server-side rendering, enhancing the user experience. TypeScript, known for its strong typing and code maintainability, ensures the frontend remains robust and reliable. Together, they form the foundation for a user-friendly interface that complements the backend's capabilities. This literature review underscores the convergence of mature and innovative technologies in the "Text Summarizer Using Chat-GPT Turbo 3.5" project, marking it as a noteworthy advancement in the field of text summarization and AI-driven applications.

The project's blend of Chat-GPT Turbo 3.5, Django, Python, Next.js, and TypeScript illustrates the evolution of AI-driven text summarization, where the synergy of powerful AI models and reliable web development tools empowers users with an efficient and accessible solution for content summarization. It encapsulates the ongoing efforts to bridge the gap between sophisticated AI technology and real-world applications, offering the potential to revolutionize information processing and knowledge retrieval across diverse domains.

**4.TECHNICAL DESCRIPTION**

**4.1 Hardware Requirement**

* Any Operating System
* At least 2 GB of RAM
* At least 500 MB of free disk space

**4.2 Software Requirement**

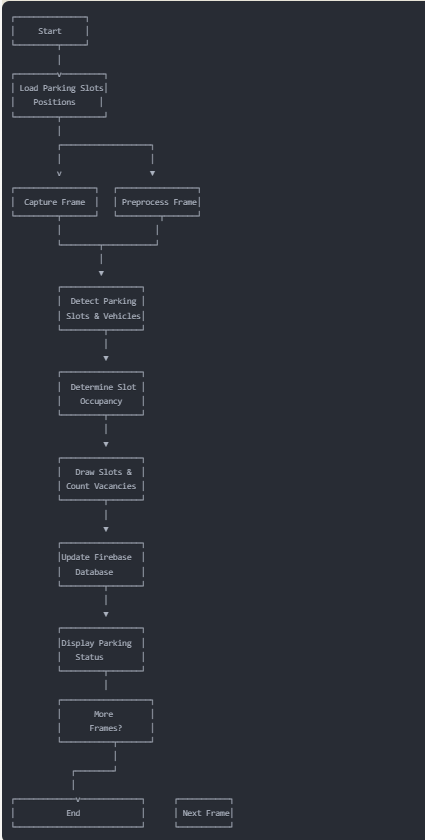
* Visual Studio Code (1.81)
* Python
* OpenCV
* Numpy
* Pyrebase
* Dotenv
* NextJS
* OS

**5.SYSTEM DESIGN**

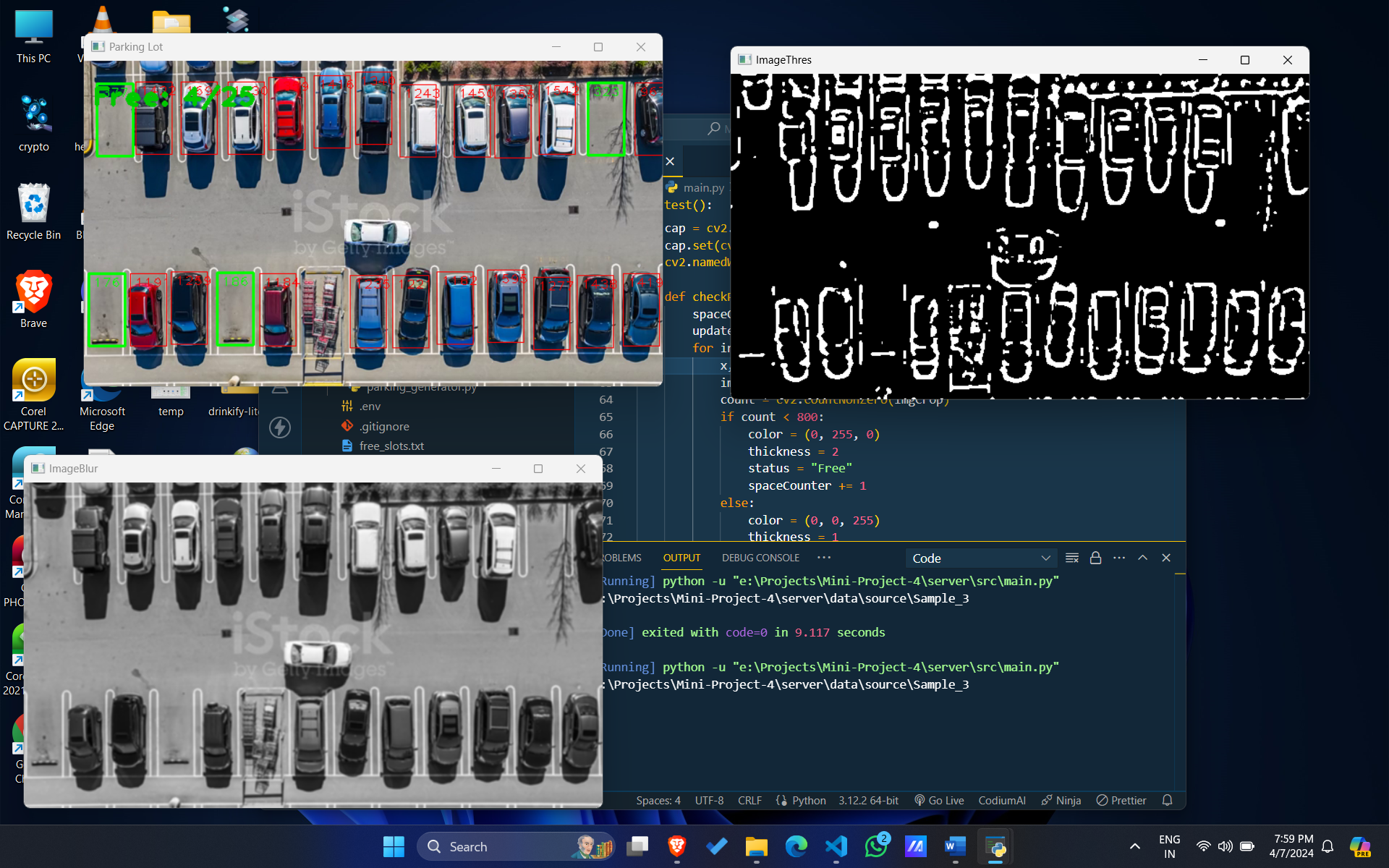
**AND DEVELOPMENT**

**5.1 Architectural Design**

* **Flow Chart**



**6. TESTING**





**7. CONCLUSION**

This project presents a comprehensive solution for managing parking availability by combining computer vision techniques, real-time data synchronization, and a user-friendly web application. The backend component leverages OpenCV to process video feed and detect vacant and occupied parking slots, continuously updating the parking data in the Firebase Realtime Database. The frontend component, built with Next.js, fetches this real-time data and displays the current status of each parking slot, enabling users to easily locate available spaces. The integration of computer vision, real-time data synchronization, and a modern web application provides a practical and efficient solution for improving the parking experience for drivers.

**8.LEARNING DURING PROJECT WORK**

* **Computer Vision and Image Processing with OpenCV**
* **Real-time Data Processing and Synchronization with Firebase**
* **Front-end Web Development with Next.js and React**
* **System Integration and Architecture Design**
* **Problem-Solving and Optimization Skills**

**9.BIBLIOGRAPHY**

* "NextJS Documentation"

<https://vercel.com/solutions/nextjs>

* "Python"

<https://python.org>

* "OpenCV"

<https://opencv.org>