# AUTOVISION X – HOME PARK ASSIST

**Project Statement –**

The challenge is to design and implement a dynamic object detection system to identify and track moving objects in a given environment. The primary application is to enhance autonomous driving capabilities by providing real-time detection of dynamic obstacles in the surroundings. The solution should be implemented in C++ and optimized for the x64 architecture.

**Important –** While running the exe make sure that dll file and exe are in the same folder.

**Team Members –**

|  |  |
| --- | --- |
| 1. N.Nachiappan Adhithiyan | [Adhi2712@gmail.com](mailto:Adhi2712@gmail.com) |
| 1. Lakshan Damodharasami | Lakshandamu2904@gmail.com |

**DATE -**

08/02/2024

**Contact Information –**

N.Nachiappan Adhithiyan – 7022321653

Lakshan Damodharasami - 8220000287

|  |  |  |
| --- | --- | --- |
| S.NO | TABLE OF CONTENTS | PAGE NUMBER |
| 1)  2)  3)  4)  5)  6)  7)  8) | Introduction  Project Description  Plan of Action  Source Code  Memory Management  Sequence Diagram  Tools Used  Summary | 3  4  5-6  7-11  12  13  14  15 |

## **Introduction**

The challenge at hand is to design and implement a dynamic object detection system tailored for autonomous parking applications. This system will enable real-time identification and tracking of moving objects within a given environment, thereby enhancing the safety and efficiency of autonomous vehicles. Leveraging C++ programming and optimized for the x64 architecture, the project aims to develop robust algorithms and integrate them seamlessly into autonomous driving frameworks. By providing autonomous vehicles with the ability to perceive and respond to dynamic obstacles in their surroundings, the project strives to advance the capabilities of autonomous driving technology, paving the way for safer and more efficient transportation solutions.

Home Park Assist is a compact yet essential feature designed to simplify parking maneuvers within the confines of residential environments. This system is tailored to assist drivers in safely navigating into tight or restricted parking spaces typically found in home driveways .

Can implement a Home Park Assist system with a microcontroller and external sensors like cameras and LiDAR involves initializing the sensors, capturing data, processing it to extract relevant information for parking assistance, and developing algorithms to analyze the data and provide guidance to the driver. The system then integrates user interface components for feedback and control logic to assist the driver during parking maneuvers.

Project Description

The objective of the project is to design and implement a dynamic object detection system capable of identifying and tracking moving objects in a given environment. Specifically, the aim is to develop algorithms and techniques to detect dynamic obstacles in real-time, thereby enhancing the capabilities of autonomous driving systems.

The need to detect objects arises from the critical requirement to ensure the safety and efficiency of autonomous driving systems. By accurately identifying and tracking moving objects in the vehicle's surroundings, such as other vehicles, pedestrians, or obstacles, the autonomous driving system can make informed decisions and adapt its behavior accordingly. Real-time detection of dynamic obstacles is essential to prevent collisions, navigate complex environments, and ensure smooth and reliable autonomous driving experiences. Therefore, the development of a dynamic object detection system is vital to address these safety and performance needs in autonomous vehicles.

A person standing next to a car

Description automatically generated

**Plan Of Action**

As a team of two, we faced quite a puzzle at the start as we tried to understand the problem we were tackling. We had some experience with a similar challenge involving object detection for autonomous driving, where we used a tool called YOLOv7 and datasets from Kaggle. This gave us a good foundation to work from.

However, there was a twist this time: we needed to use C++, which was a bit haunting for us. We had to do some digging to find the right tools, and that's when we stumbled upon OpenCV, a powerful library for computer vision tasks. It took us a while to get the hang of it and set everything up properly in our development environment.

One of the trickiest parts was managing memory usage efficiently. We spent a lot of time researching and experimenting to figure out the best way to do this. But with patience and perseverance, we finally cracked it.

Overall, it took us about a week of hard work and determination to understand the problem, find the right tools, and come up with a solution. But in the end, we were able to build a solid foundation for our dynamic object detection system, and we're proud of what we've accomplished together.

Even we were in between our busy college schedule , so had to pull out sometime to work on our project.

Dividing our task into equal halves, we decided to split our work into two clear stages. The first half involved handling input from the user, specifically the video file path. This meant creating a user-friendly interface where users could input the path easily. Then, using OpenCV's built-in functions, we aimed to read the video file effectively, ensuring smooth processing of frames.

In the second half, we focused on the core functionality: tracking and overlaying moving objects in each frame. To accomplish this, we planned to implement tracking modules that would help us establish a reference frame. From there, our strategy was to continuously monitor each frame for changes and overlay the detected moving objects with bounding boxes. This step required meticulous attention to detail, as we needed to accurately detect and track moving objects while minimizing false positives.

By breaking down our tasks in this way, we aimed to streamline our workflow and ensure each team member had a clear role to focus on. This initial thought process laid the groundwork for our subsequent development efforts, guiding us towards an organized and efficient approach to tackling the dynamic object detection challenge.

A group of colorful circles

Description automatically generated

**Source Code**

* Just refer to the source code attached with the exe in this folder, implemented on C++ Language.
* The provided source code implements a class-based approach for processing a video and detecting moving objects. Here's an overview of the main components.

1. Header Directories:

**#include <opencv2/opencv.hpp>** This line includes the OpenCV library header file. OpenCV (Open Source Computer Vision Library) is a powerful open-source computer vision and machine learning software library. By including this header, the code gains access to a wide range of functions and classes provided by OpenCV for image and video processing tasks.

**#include <iostream>** This line includes the standard input/output stream header file. This header provides input and output functionality in C++ through objects like cin, cout, etc. It's commonly used for printing messages to the console and reading user input.

1. Using Namespace:

**using namespace cv** This line introduces the cv namespace into the current scope. OpenCV functions and classes are defined within the cv namespace to avoid naming conflicts with other libraries or user-defined code. By using the using namespace cv; directive, all functions and classes from the cv namespace become accessible directly without prefixing them with cv::.

**using namespace std** This line introduces the std namespace into the current scope. The C++ Standard Library provides its functionality within the std namespace. Similar to OpenCV, this directive allows direct access to functions and classes from the std namespace without prefixing them with std::.

1. Within the private section of the VideoProcessor class, several member variables are declared.

**VideoCapture \*Capture :**

This capture is a pointer pointing to VideoCapture object provided by OpenCV. Helps in reading the video file as input.

**Mat frame:**

This variable represents a single frame of the video being processed.The Mat class is a fundamental structure in OpenCV used to store and manipulate images and matrices.In this context, frame stores the current frame read from the video source.

**Mat background:**

This variable represents the background model used for background subtraction.It stores a single image frame that serves as the initial background reference.

The background model is subtracted from each frame of the video to detect moving objects.

**Mat foregroundMask:**

This variable represents the foreground mask obtained through background subtraction.

After subtracting the background model from the current frame, a binary mask is generated.The foreground mask highlights the regions of the frame where significant changes have occurred compared to the background.

It is used to isolate and identify moving objects within the video frame.

**Ptr<BackgroundSubtractor> bgSubtractor:**

This variable represents a smart pointer to a background subtractor object.The BackgroundSubtractor class is an abstract base class in OpenCV used for background subtraction algorithms.In this code, a concrete implementation of a background subtractor (such as the Gaussian Mixture-based algorithm) is assigned to bgSubtractor.

1. the **constructor of the VideoProcessor class** is declared within the public section. The constructor initializes the member variables of the class and sets up the initial state for video processing.

The constructor of the VideoProcessor class initializes the member variables (capture, bgSubtractor, background, and foregroundMask) and sets up the initial state for video processing.It ensures error handling by checking if the video file was opened successfully.It initializes the background subtractor using the MOG2 algorithm, which is a commonly used method for background subtraction in video processing applications.The constructor also reads the first frame of the video file to initialize the background model and applies background subtraction to generate the initial foreground mask.Overall, the constructor prepares the VideoProcessor object for video processing tasks, including background subtraction and moving object detection.

1. About the ProcessVideo function –

**Frame Processing Loop:**The function enters an infinite loop to continuously process each frame of the video until the end of the video file or until interrupted by the user.

Frame Reading:

**\*capture >> frame:** This line reads the next frame from the video file using the VideoCapture object (capture).

Frame Empty Check:

**if (frame.empty())**  This condition checks if the frame is empty, indicating the end of the video file or a capture failure. If the frame is empty, an appropriate message is displayed, and the loop is exited.

**Frame Preprocessing:**

The current frame (frame) is converted to grayscale using cvtColor() to simplify subsequent processing steps.

Background Subtraction:

**The background subtractor (bgSubtractor)** is applied to the grayscale frame (grayFrame) to generate a foreground mask (foregroundMask). This mask highlights the moving objects in the frame by identifying the pixels that differ significantly from the background model.

Foreground Mask Processing:

**Previous contours** are removed from the foreground mask by setting their corresponding pixels to zero. This helps in updating the mask for the current frame's contours.

Morphological opening operation is applied to the foreground mask to remove noise and smooth the mask. This is achieved using the morphologyEx() function with MORPH\_OPEN operation and an appropriate structuring element.

**Contour Detection:**

Contours are detected in the processed foreground mask using the findContours() function. Detected contours represent the outlines of the moving objects in the frame.

Contour Visualization:Detected contours are drawn on the original frame using the drawContours() function. This overlays the contours as green lines on the frame.

**Frame Display:**

The frame with overlaid contours is displayed to the user using the imshow() function.

**User Interaction:**

The function waits for 30 milliseconds for user input. If the user presses the 'Esc' key (ASCII code 27), the video playback is stopped, and the loop is exited.

**Main Function:**

the main() function orchestrates the video processing workflow by instantiating a VideoProcessor object, starting the video processing pipeline, and handling user interaction. It encapsulates the high-level logic of the program and serves as the control center for video processing tasks using the OpenCV library.

**Memory Management**

In the context of our program, setting up the buffer and utilizing it as the base memory address entails a strategic approach to memory management. We initiate this process by dynamically allocating a buffer of suitable size, meticulously considering the expanded memory space available in a 64-bit (x64) architecture.

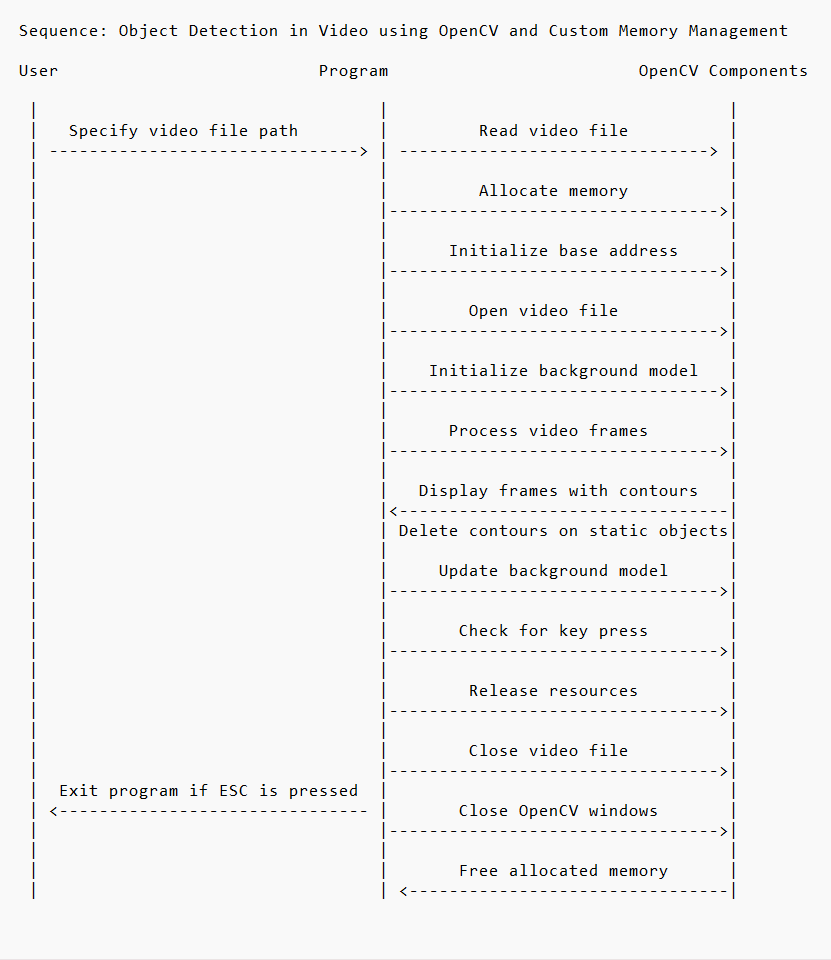
**(400\*sizeof(long long)) 3200 bytes**- By leveraging memory allocation functions like **malloc()** or **new**, we ensure the buffer's allocation aligns with the system's memory capabilities, facilitating optimal resource utilization.

Upon allocation from user, the allocated buffer becomes our primary memory region, with its memory address serving as the anchor, or base address, for subsequent memory operations within the program. This base address acts as a reference point from which we conduct efficient memory access, manipulation, and release throughout the program's lifecycle.

Through this methodical setup, we aim to streamline memory access and management, particularly critical in our video processing application's real-time data processing demands. With the base memory address established, we orchestrate seamless memory interactions, harnessing the architecture's expansive memory resources to enhance program performance and efficiency.

Furthermore, by embracing this approach, we empower our program to navigate the complexities of memory management with finesse, ensuring optimal utilization of system resources while mitigating potential memory-related bottlenecks or inefficiencies. Ultimately, this meticulous handling of memory resources fortifies our program's capability to deliver reliable and high-performance video processing functionalities.

**SEQUENCE DIAGRAM**

****

**Tools Used**

**Visual Studio** is a versatile integrated development environment (IDE) designed by Microsoft, catering to the diverse needs of software developers. It offers a robust code editor, advanced debugging capabilities, and extensive project management tools. With support for various programming languages and platforms, Visual Studio enables developers to build applications efficiently for Windows, Android, iOS, web, and more. Its seamless integration with version control systems, testing frameworks, and cloud services streamlines the development process. Furthermore, Visual Studio's extensibility through plugins and extensions allows developers to customize and enhance their development experience according to their specific requirements. Overall, Visual Studio stands as a comprehensive and indispensable tool for modern software development.

**OpenCV** (Open Source Computer Vision Library) is a powerful open-source library for computer vision and machine learning tasks in C++ (and other languages). It offers a wide range of functions and algorithms for image and video processing, including object detection, tracking, facial recognition, and more. OpenCV provides a rich set of tools for handling images and videos, such as loading, saving, and manipulating pixels. Its modular design makes it easy to use and extend, allowing developers to build complex vision applications efficiently. With its cross-platform support, OpenCV is widely adopted in academia and industry, serving as a foundation for research, prototyping, and production-level applications in fields like robotics, augmented reality, medical imaging, and surveillance. Overall, OpenCV empowers developers with the tools and capabilities needed to tackle a diverse range of computer vision challenges in C++.

**CMake** is a versatile build system that simplifies software project compilation and management. It generates platform-specific build files from a single configuration file, allowing developers to define project settings and dependencies efficiently. With CMake building OpenCV into the Visual studio project was done without a hassle.

**Summary**

Our journey with OpenCV was an exciting exploration of computer vision possibilities. We immersed ourselves in tutorials, documentation, and practical examples, gaining proficiency in image processing and computer vision tasks. With newfound knowledge, we tackled the challenge of creating a program to track moving objects in a video file.

Our program dynamically initializes a video capture object, applies background subtraction, and detects moving objects using contour detection and morphological operations. Through teamwork and problem-solving, we overcame challenges, fine-tuning our code to achieve precise object tracking.

The project not only deepened our understanding of OpenCV but also sharpened our problem-solving skills and fostered collaboration within the team. Each obstacle we encountered became an opportunity for growth, as we experimented with different approaches and learned from our mistakes. In the end, we emerged with a robust solution that showcases our dedication and passion for computer vision.

In conclusion, our journey with OpenCV was a rewarding experience, culminating in the creation of a practical solution that demonstrates our dedication and passion for computer vision. We look forward to applying our newfound skills to future projects and continuing our exploration of the exciting field of computer vision**.**