**Dynamic Object Detection in Video using C++ with OpenCV and CMake**

**Team Name:** AutoZone

**Members:**

* T Bala Saatvik
* T Karthik Venkatesh

**College:** Coimbatore Institute of Technology

**Executive Summary:**

This report encapsulates the development and functionality of a novel C++ application tailored for dynamic object detection within video streams. Leveraging the capabilities of OpenCV and the efficiency of CMake, the program not only identifies moving objects in real-time but also enhances performance through custom memory management techniques. Highlighting detected objects with red rectangular boxes aids in visualization and identification.

**1. Introduction:**

**1.1 Purpose:**

The primary purpose of this project is to engineer a sophisticated software solution capable of detecting and tracking moving objects in video samples. Beyond mere detection, the program aims to serve as a foundational tool for advanced computer vision applications, empowering tasks such as surveillance, traffic monitoring, and industrial automation.

**1.2 Scope:**

Focusing on real-time object detection, the project delves into the intricacies of identifying moving objects within video streams accurately. Developed using C++, OpenCV, and CMake, the application targets efficiency and scalability while maintaining a user-friendly interface for seamless integration into various systems.

**2. Features:**

**2.1 Object Detection:**

Harnessing the prowess of OpenCV, the program employs an array of algorithms and techniques to detect and localize moving objects within video frames. These techniques encompass background subtraction, motion estimation, and contour detection, ensuring robust object identification in diverse scenarios.

**2.2 Contour Labeling:**

Detected object contours are meticulously labeled using red rectangular boxes, enhancing the visual representation of moving objects within the video stream. This feature facilitates easy tracking and identification, thereby bolstering the application's utility in real-world scenarios.

**2.3 Custom Memory Management:**

Recognizing the critical role of memory management in real-time video processing, the program integrates custom memory management techniques to optimize resource allocation and deallocation. By mitigating memory fragmentation and overhead, the application ensures consistent performance across prolonged video processing tasks.

**3. Dependencies:**

The project relies on essential dependencies:

* **OpenCV:** A versatile computer vision library renowned for its comprehensive suite of functionalities, enabling robust object detection and processing.
* **CMake:** A cross-platform build system instrumental in streamlining the compilation and build process of the C++ program, fostering development efficiency and portability.

**4. Memory Management:**

Efficient memory management serves as the cornerstone of real-time video processing. To this end, the program implements tailored memory management strategies, including:

* Streamlined allocation and deallocation of memory resources to minimize processing overhead.
* Mitigation of memory fragmentation through proactive memory management techniques.
* Adoption of memory pooling and caching mechanisms to optimize resource utilization and enhance overall performance.

**5. Conclusion:**

In conclusion, the dynamic object detection program stands as a testament to the synergy between cutting-edge technologies and meticulous engineering. By amalgamating the capabilities of OpenCV with efficient memory management techniques, the project not only achieves its objectives but also lays the groundwork for future advancements in computer vision applications.

**6. Acknowledgments:**

The project extends its gratitude to the vibrant OpenCV community for their invaluable contributions and steadfast support. Additionally, the utilization of the CMake build system is acknowledged for its pivotal role in simplifying the development workflow and fostering collaboration.

**7. Source Code:**#include <opencv2/opencv.hpp>

#include <iostream>

// Define your custom memory block structure

struct MemBlock {

void\* ptr;

size\_t size;

};

// Define your custom memory allocator class

class CustomAllocator {

private:

void\* baseAddress;

public:

CustomAllocator(void\* baseAddr) : baseAddress(baseAddr) {}

MemBlock\* allocateMemory(size\_t size) {

MemBlock\* block = new MemBlock;

if (block) {

block->ptr = static\_cast<char\*>(baseAddress) + size; // Use the custom base address

block->size = size;

}

return block;

}

};

void detectAndDrawMovingObjects(cv::VideoCapture& cap, CustomAllocator& allocator) {

cv::Ptr<cv::BackgroundSubtractor> pMOG2 = cv::createBackgroundSubtractorMOG2();

cv::Mat frame, fgMask, fgMaskFiltered;

while (true) {

cap >> frame;

if (frame.empty())

break;

// Allocate memory using the custom allocator

size\_t imageSize = frame.total() \* frame.elemSize();

MemBlock\* imageBlock = allocator.allocateMemory(imageSize);

if (imageBlock) {

// Process video using the allocated memory

std::cout << "Memory allocated successfully at address: " << imageBlock->ptr << std::endl;

// Apply background subtraction

pMOG2->apply(frame, fgMask);

// Threshold the foreground mask to get binary image

cv::threshold(fgMask, fgMask, 128, 255, cv::THRESH\_BINARY);

// Apply morphological operations to reduce noise

cv::morphologyEx(fgMask, fgMaskFiltered, cv::MORPH\_OPEN, cv::getStructuringElement(cv::MORPH\_ELLIPSE, cv::Size(5, 5)));

cv::morphologyEx(fgMaskFiltered, fgMaskFiltered, cv::MORPH\_CLOSE, cv::getStructuringElement(cv::MORPH\_ELLIPSE, cv::Size(5, 5)));

// Find contours in the binary image

std::vector<std::vector<cv::Point>> contours;

cv::findContours(fgMaskFiltered, contours, cv::RETR\_EXTERNAL, cv::CHAIN\_APPROX\_SIMPLE);

// Draw a red rectangle around each moving object

for (const auto& contour : contours) {

cv::Rect boundingBox = cv::boundingRect(contour);

cv::rectangle(frame, boundingBox, cv::Scalar(0, 0, 255), 2); // Red rectangle

}

// Display the resulting video with red rectangles

cv::imshow("Moving Objects", frame);

// Release the allocated memory

delete imageBlock;

} else {

std::cout << "Memory allocation failed!" << std::endl;

}

// Break the loop if 'ESC' key is pressed

if (cv::waitKey(30) == 27)

break;

}

}

int main() {

void\* baseMemoryAddress;

// Prompt the user to enter the base address of memory

std::cout << "Enter the base address of memory: ";

std::string userInput;

std::getline(std::cin, userInput);

// Convert the user input to a void pointer

baseMemoryAddress = reinterpret\_cast<void\*>(std::strtoull(userInput.c\_str(), nullptr, 0));

// Prompt the user to enter the video location

std::cout << "Enter the video location: ";

std::string videoLocation;

std::getline(std::cin, videoLocation);

// Modified string

std::string modifiedString;

// Loop through the original string and add escape sequences for backslashes

for (char c : videoLocation) {

if (c == '\\') {

modifiedString += "\\\\";

} else {

modifiedString += c;

}

}

// Print the modified string

std::cout << "Modified String: " << modifiedString << std::endl;

cv::VideoCapture cap(modifiedString);

if (!cap.isOpened()) {

std::cerr << "Error opening video file." << std::endl;

return -1;

}

CustomAllocator allocator(baseMemoryAddress);

detectAndDrawMovingObjects(cap, allocator);

cap.release();

cv::destroyAllWindows();

return 0;

}

**8. Location of exe file:** Opencvtest \ build \ Debug \ opencvtest.exe

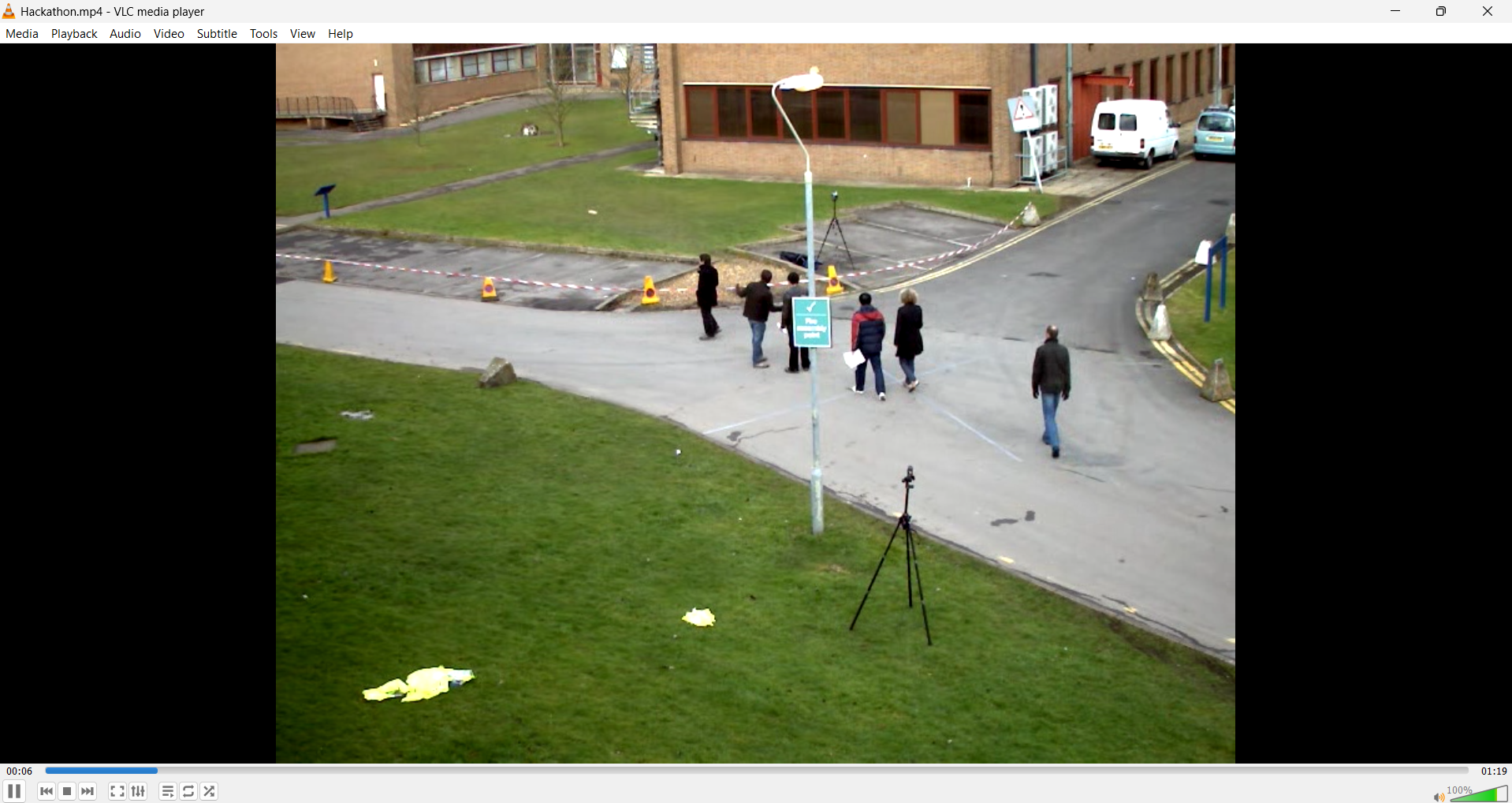
**9. Terminal Input from User-Side:**

Enter the base address of memory: 4000

Enter the video location: D:\CIT College\BGSW Auto Vision X\HPA\_Problem\_Statement\ Problem\_Statement\_1\Hackathon.mp4

**10. Simulation Result:**

**Input:**

****

**Output:**

****