

AutoVisionX Hackathon

Dynamic Object Detection System Documentation

TEAM PIXEL TENSHI

-Madduri Satya Tejeswar & Tekkali Rahul Lakshmi Subrahmanyam

1. Introduction

Autonomous driving technology requires the ability to detect and respond to dynamic objects in real-time. This project develops a dynamic object detection system to enhance autonomous driving capabilities through advanced image processing and computer vision techniques. The system identifies and tracks moving objects within a certain camera view surroundings using C++, optimized for 64-bit architecture.

2. Problem Statement

The challenge is to design and implement a system capable of detecting and tracking dynamic objects, such as vehicles and pedestrians, in real-time to support autonomous driving. The system aims to provide immediate and accurate object detection to improve the safety and efficiency of autonomous vehicles.

3. Software Requirements

- CMake: Latest version for build process automation.
- OpenCV: Latest version for computer vision tasks.
- MinGW: Latest version for Windows users to support GCC compilation.
- Visual Studio 2022: Recommended IDE for development.

Hardware Requirements

- 64-bit (x64) architecture: To handle data and computations for real-time object detection.
- Development Environment Setup

- Install CMake, OpenCV, and MinGW.
- Proceed to add each application's "bin" files to the system's path environment variable.
- Use Visual Studio 2022 for project code, configuration and compilation.

4. Build Process

The project uses CMake for building. The project requires CMake, OpenCV, and MinGW for Windows users.

- First, the project folder is created in a desired location.
- Proceeding that, we open the project folder via Visual studio and make an initial "CMakeLists.txt" text file with the required lines of text as can be seen in the source code to connect OpenCV and initialize the project.
- The C++11 standard is set, and GNU extensions are disabled to maintain standard compatibility.
- The executable is built from `src/main.cpp` folder which is made within the project folder (ie, the same folder which the CMakeLists.txt doc exists) and linked against OpenCV libraries. After this and writing the source code in the main.cpp file, the code is now ready to run.

5. Using the Application

The application takes three command-line arguments: the path to the input video file, the base address for memory allocations, and the path where the output video will be saved. Since we have seen that the base address for memory allocation can be very context sensitive and if not done appropriately, can cause undesirable issues and unintended corruptions, we have simply added a dummy variable in its place. We believe that the base address feature can be updated by us if needed after gaining more knowledge about the environment it needs to be implemented in.

The code is run as follows:

- Code is built and run in Visual studio
- We open Cmd and go to the directory that has the executable file
- We then run the application by typing the following in command line "<executable_file_name.exe> <path/to/input/video> <dummy base address> </path/to/where/the/output/video/is/stored>"

(NOTE: FOR SMOOTH OPERATION, THE OUTPUT VIDEO SHOULD BE OF ".avi" EXTENSION AND NOT ANY OTHER EXTENSION LIKE ".mp4")

The code implements a background subtraction algorithm to detect moving objects and draws bounding boxes around them.

6. Dynamic Object Detection Algorithm

The dynamic object detection algorithm in the C++ code leverages background subtraction for identifying moving objects in a video stream. Here's how the algorithm works within the code:

1. Background Subtraction:

- The algorithm begins with the creation of a background subtractor object using ``cv::createBackgroundSubtractorMOG2``. This object models the background of a scene using a Gaussian mixture-based background/foreground segmentation algorithm.
- For each new frame, the background subtractor updates its model and produces a foreground mask (``fgMask``) by applying the ``apply`` method. The foreground mask highlights the moving objects as white blobs on a black background.

2. Noise Reduction:

- After obtaining the foreground mask, a morphological opening operation (``cv::morphologyEx``) with a small elliptical kernel is applied. This step helps in removing noise and small irrelevant objects from the foreground mask, improving the clarity of the object contours.

3. Contour Detection:

- The cleaned-up foreground mask is then used for contour detection with the ``cv::findContours`` function. Contours are simply the boundaries of the white blobs (moving objects) detected in the foreground mask.
- These contours are filtered based on area size, ignoring those below a certain threshold (100 pixels in this case) to exclude small, irrelevant detections that are not of interest.

4. Bounding Box Calculation:

- For each valid contour, the ``cv::boundingRect`` function calculates the bounding box that encloses the contour. These bounding boxes are rectangles that represent the position and size of the detected moving objects.

5. Drawing and Display:

- The bounding boxes are drawn onto the original frame using ``cv::rectangle``, which visually indicates the detected objects.
- The modified frame is then shown in a window using ``cv::imshow`` and written to an output file using ``cv::VideoWriter``. This results in a video stream where moving objects are highlighted in real-time.

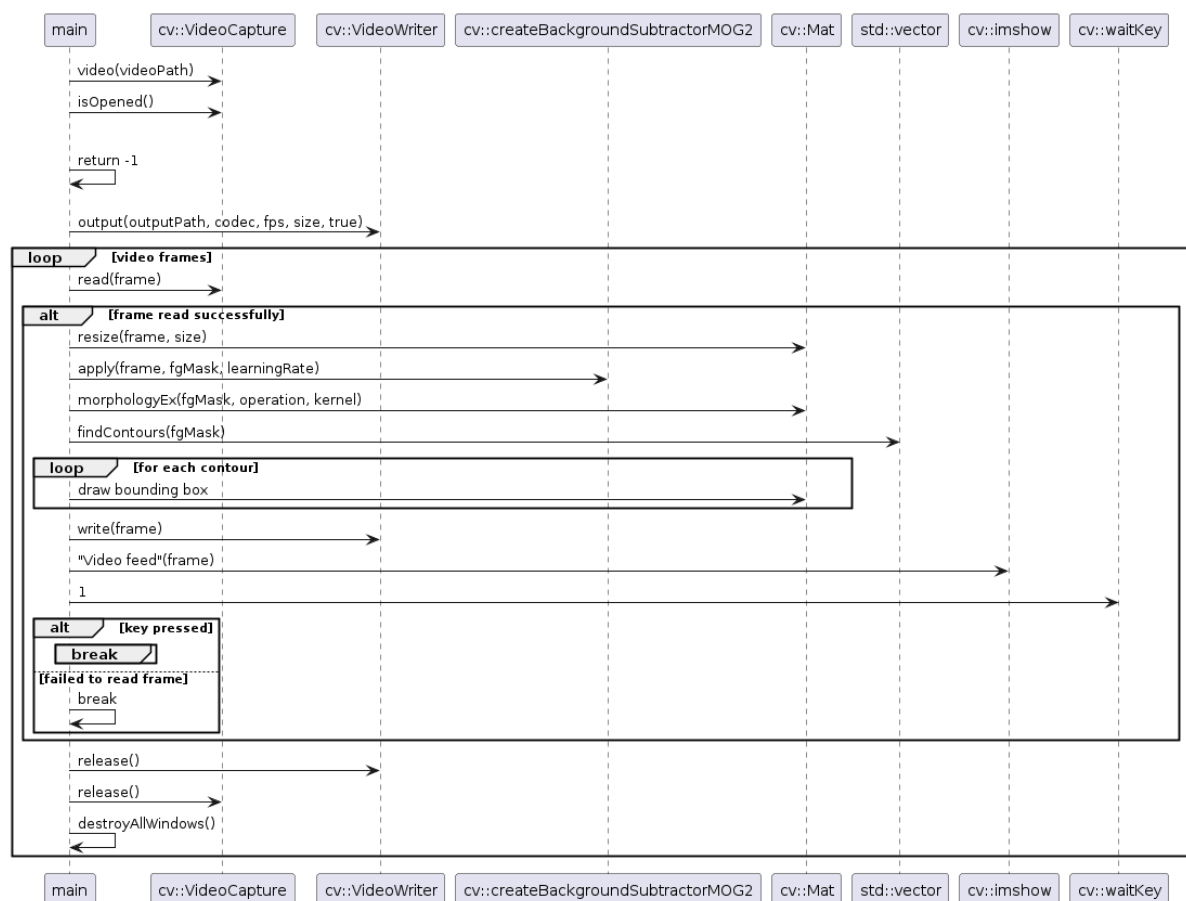
6. Looping Through Video:

- This process repeats for each frame in the video until there are no more frames to read or a break condition is triggered (e.g., a key press).

7. Overlay Output

The output contains the input video but now with rectangular boxes drawn on dynamic objects. The overlay boxes are drawn from frame to frame. If a person or object stops moving, the box is removed as that object has become static. The boxes are redrawn once movement is observed again. As the program processes each frame and displays it, each frame is also written onto the output video file.

8. Sequence Diagram



Explanation:

--Main Functionality

- The ``main`` process begins by creating an instance of ``cv::VideoCapture`` to capture video from a file specified by ``videoPath``.
- It checks if the video capture has been successfully opened with ``isOpen()``. If not, it returns -1.
- Then, a ``cv::VideoWriter`` is instantiated to output the processed video to a specified ``outputPath``. The ``cv::VideoWriter`` is configured with a codec, frames per second (fps), and size, and is set to write in color (``true``).

--Loop for Video Frames

- The diagram enters a loop to read each frame of the video.
- The ``read(frame)`` function is called to read a frame from the video.
- If the frame is successfully read, several operations are performed:
 - The frame is resized with ``resize(frame, size)``.
 - A foreground mask is applied using a background subtraction algorithm, which is an instance of ``cv::BackgroundSubtractorMOG2``.
 - Morphological operations (like erosion or dilation) are performed on the foreground mask with ``morphologyEx(fgMask, operation, kernel)``.
 - Contours in the foreground mask are found with ``findContours(fgMask)``.

-- Loop for Each Contour

- For each contour found in the foreground mask, a bounding box is drawn around it.

-- Writing and Displaying Frames

- The processed frame is written out using ``write(frame)``.
- The ``cv::imshow`` displays the frame in a window titled "Video feed".

-- Alternate Paths

- The ``alt`` block represents alternative paths of execution depending on certain conditions:
 - If a key is pressed, the loop breaks, and the program will move towards cleanup and exit.
 - If the frame fails to be read, the loop also breaks.

-- Cleanup and Exit

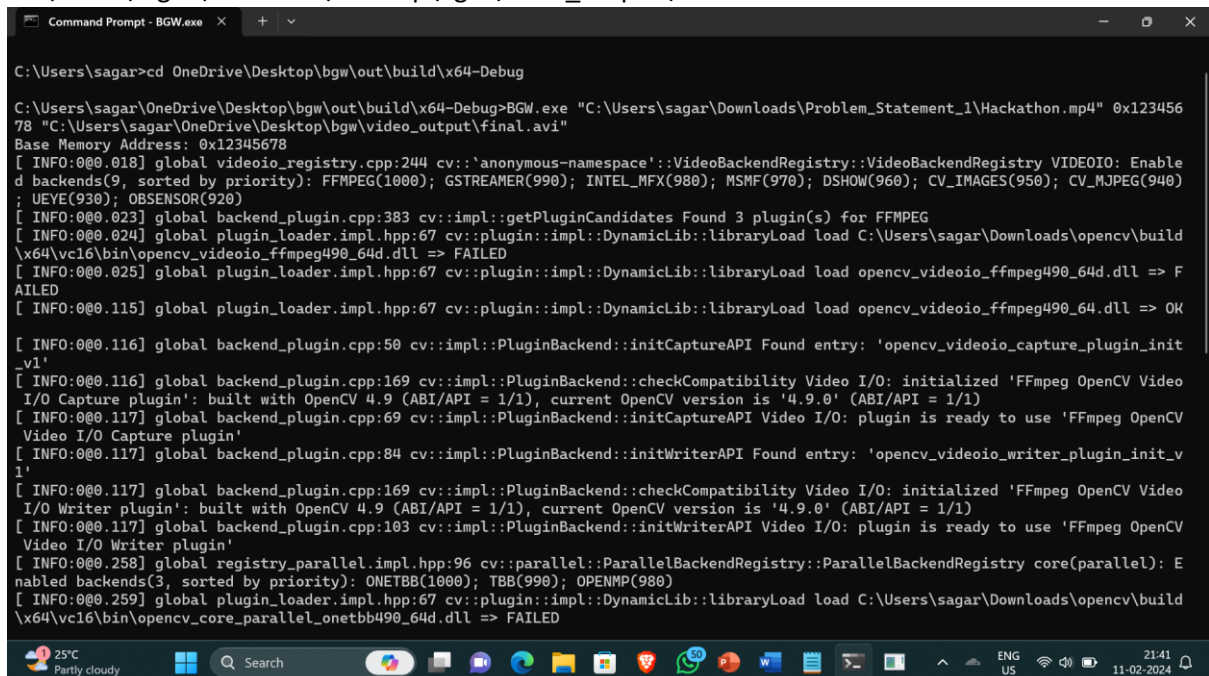
- The video capture and writer are released with ``release()``.
- All windows created by ``cv::imshow`` are destroyed with ``destroyAllWindows()``.

- The program then exits the `main` function.

9. Testing and Results

We have tested the code, building and running procedure and have obtained the desired output ie a video with overlays on dynamic objects and stored in a new directory.

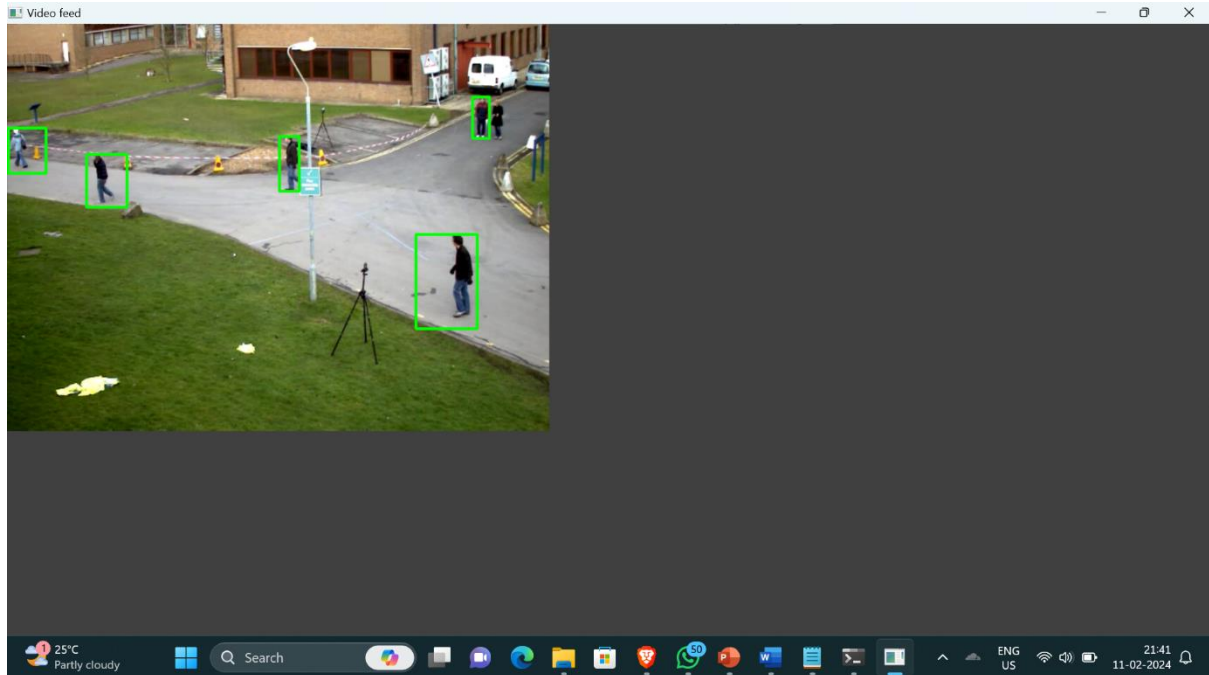
Opening cmd, changing directory to the directory having the executable file("BGW.exe") and typing --BGW.exe "C:\Users\sagar\Downloads\Problem_Statement_1\Hackathon.mp4" 0x12345678 "C:\Users\sagar\OneDrive\Desktop\bgw\video_output\final.avi"



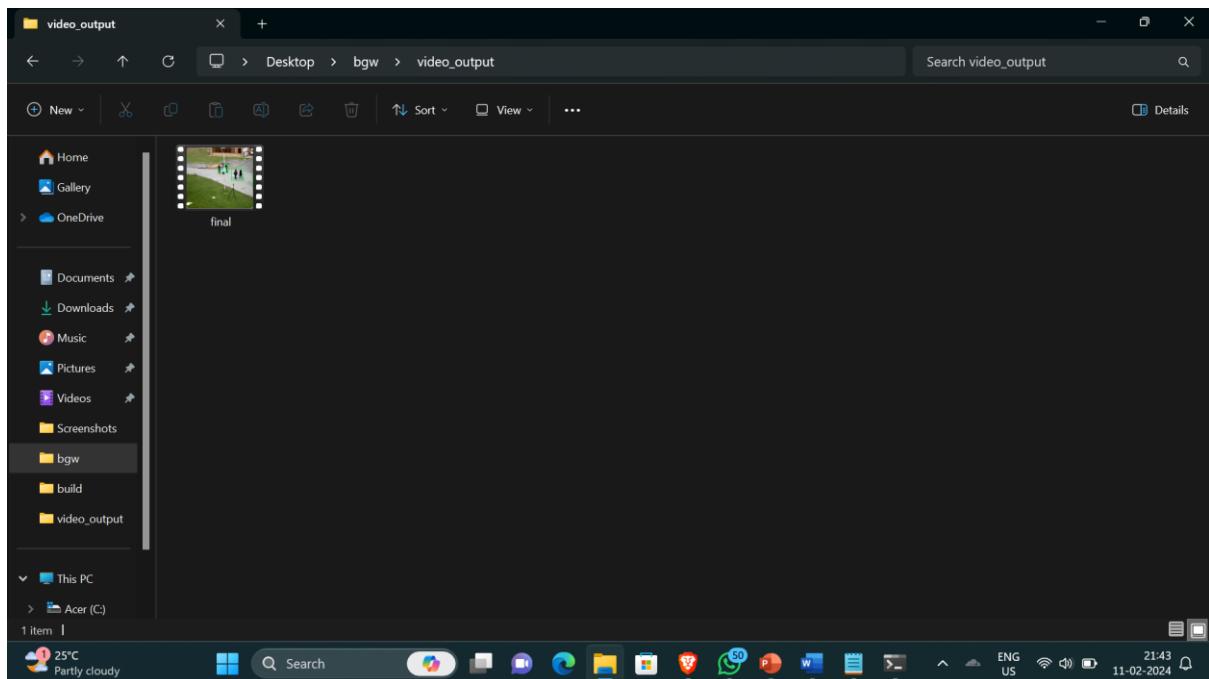
```
C:\Users\sagar>cd OneDrive\Desktop\bgw\out\build\x64-Debug

C:\Users\sagar\OneDrive\Desktop\bgw\out\build\x64-Debug>BGW.exe "C:\Users\sagar\Downloads\Problem_Statement_1\Hackathon.mp4" 0x12345678 "C:\Users\sagar\OneDrive\Desktop\bgw\video_output\final.avi"
Base Memory Address: 0x12345678
[ INFO:000.018] global videoio_registry.cpp:244 cv::'anonymous-namespace':::VideoBackendRegistry::VideoBackendRegistry VIDEOIO: Enabled backends(9, sorted by priority): FFMPEG(1000); GSTREAMER(990); INTEL_MFX(980); MSMF(970); DSHOW(960); CV_IMAGES(950); CV_MJPEG(940); UEYE(930); OBSSENSOR(920)
[ INFO:000.023] global backend_plugin.cpp:383 cv::impl::getPluginCandidates Found 3 plugin(s) for FFMPEG
[ INFO:000.024] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\Users\sagar\Downloads\opencv\build\x64\vc16\bin\opencv_videoio_ffmpeg490_64d.dll => FAILED
[ INFO:000.025] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_videoio_ffmpeg490_64d.dll => FAILED
[ INFO:000.115] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_videoio_ffmpeg490_64d.dll => OK
[ INFO:000.116] global backend_plugin.cpp:50 cv::impl::PluginBackend::initCaptureAPI Found entry: 'opencv_videoio_capture_plugin_init_v1'
[ INFO:000.116] global backend_plugin.cpp:169 cv::impl::PluginBackend::checkCompatibility Video I/O: initialized 'FFmpeg OpenCV Video I/O Capture plugin': built with OpenCV 4.9 (ABI/API = 1/1), current OpenCV version is '4.9.0' (ABI/API = 1/1)
[ INFO:000.117] global backend_plugin.cpp:69 cv::impl::PluginBackend::initCaptureAPI Video I/O: plugin is ready to use 'FFmpeg OpenCV Video I/O Capture plugin'
[ INFO:000.117] global backend_plugin.cpp:84 cv::impl::PluginBackend::initWriterAPI Found entry: 'opencv_videoio_writer_plugin_init_v1'
[ INFO:000.117] global backend_plugin.cpp:169 cv::impl::PluginBackend::checkCompatibility Video I/O: initialized 'FFmpeg OpenCV Video I/O Writer plugin': built with OpenCV 4.9 (ABI/API = 1/1), current OpenCV version is '4.9.0' (ABI/API = 1/1)
[ INFO:000.117] global backend_plugin.cpp:103 cv::impl::PluginBackend::initWriterAPI Video I/O: plugin is ready to use 'FFmpeg OpenCV Video I/O Writer plugin'
[ INFO:000.258] global registry_parallel.impl.hpp:96 cv::parallel::ParallelBackendRegistry::ParallelBackendRegistry core(parallel): Enabled backends(3, sorted by priority): ONETBB(1000); TBB(990); OPENMP(980)
[ INFO:000.259] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\Users\sagar\Downloads\opencv\build\x64\vc16\bin\opencv_core_parallel_onetbb490_64d.dll => FAILED
```

Video processing in progress



Final video output stored in "video_output" folder.



Hence testing and results are successful.

11. Challenges and Solutions

The following are the possible challenges that can be faced by the code, followed immediately by their respective solutions which can be implemented with more time and more understanding of the context-

1. Dynamic Backgrounds

- Challenge: Backgrounds with moving elements (e.g., swaying trees, rain) can be mistakenly identified as foreground objects.
- Possible solution: Implementing more sophisticated background models that can differentiate between background motion and foreground objects is essential. The `cv::createBackgroundSubtractorMOG2`` function provides options to control the sensitivity to detect shadows and dynamically update the background, which can be fine-tuned.

2. Processing Speed and Real-time Performance

- Challenge: Processing each video frame with complex operations might not meet real-time performance requirements, especially in high-resolution videos.
- Solution: Optimization of the algorithm

3. Detection Accuracy and False Positives

- Challenge: Maintaining high detection accuracy while minimizing false positives and negatives is challenging, especially in crowded scenes.
- Possible Solution: Applying additional filters based on object size, shape, or motion characteristics can help refine detection. Integrating machine learning models trained to recognize specific objects of interest could also enhance accuracy.

4 . Resource Utilization

- Challenge: Intensive computations can lead to high CPU and memory usage.

12. Conclusion

We have built a viable solution to the given problem state using softwares such as CMake and OpenCV. The code provided has its certain challenges regarding optimization and scalability but is reliable and works positively while also being easy to update.

We have taken advantage of OpenCV to fully process a video frame by frame, output an overlay with rectangular boxes over dynamic objects and store the final video in a directory of our choice. Thus achieving what was needed and going beyond too.