AutoVisionX Hackathon

**Dynamic Object Detection for Autonomous Driving**

By

Sanchita Ghosh(sanchitag150@gmail.com,9875696944),

Sayan Sarkar(sayansarkar1121@gmail.com,7753876965)



**Problem 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.

**Architecture:-**

1. Input Handling:
   * The program takes a video file path as a command-line argument.
   * It checks if the video file is provided and if it can be opened successfully.
2. Object Detection Pipeline:
   * The main loop iterates over each frame of the video.
   * For each frame, it performs the following steps:
     + Applies background subtraction to detect moving objects.
     + Performs morphological operations (opening and closing) to refine the detected object shapes.
     + Thresholds the foreground mask to obtain a binary mask.
     + Finds contours in the binary mask to outline the detected objects.
     + Approximates the contours with polygons and draws them on the original frame.
3. Memory Management:

* The code uses smart pointers (**unique\_ptr** and **shared\_ptr**) to manage the memory for the **VideoCapture** object and the background subtractor. Matrices like **frame**, **fg\_mask**, and **binary\_mask** are created and released within the loop to manage memory efficiently.
  + Uses dynamic memory allocation for storing contours and polygon approximations.
  + Releases memory for contours and polygons after they are used.

1. Output Display:
   * Displays the processed frames with detected objects outlined.
   * Waits for the user to press 'q' to exit the program.
2. Resource Release:
   * Releases the VideoCapture object to close the video stream.
   * Destroys all OpenCV windows.
3. Error Handling:
   * Checks for errors during input handling and video stream opening.
   * Prints error messages if any issues occur.

**Algorithm:-**

Step 1: Take the input video file and base memory address via the command line.

Step 2: Check the video file open successfully.

Step 3: Create a background subtractor object using the MOG2 method.

Step 4: Initialise variables for storing frames.

Step 5: Read the next frame from the video stream.

Step 6: Exit the loop if the frame is empty.

Step 6: Apply the background subtractor to the current frame to obtain the foreground mask.

Step 7: Perform morphological operations on the foreground mask to remove noise.

Step 8: Calculate the mean value of the foreground mask to determine the threshold for binarization.

Step 9: Threshold the foreground mask to obtain a binary mask, separating the moving objects from the background.

Step 10: Find the contours in the binary mask representing the outline of the objects.

Step 11: Compute the contour area to determine its size.

Step 12: Draw the overlay on the detected objects if the contour area falls within a specified range.

Step 13: Display the processed frame with the detected objects highlighted.

Step 14: Release the memory for Mat variables.

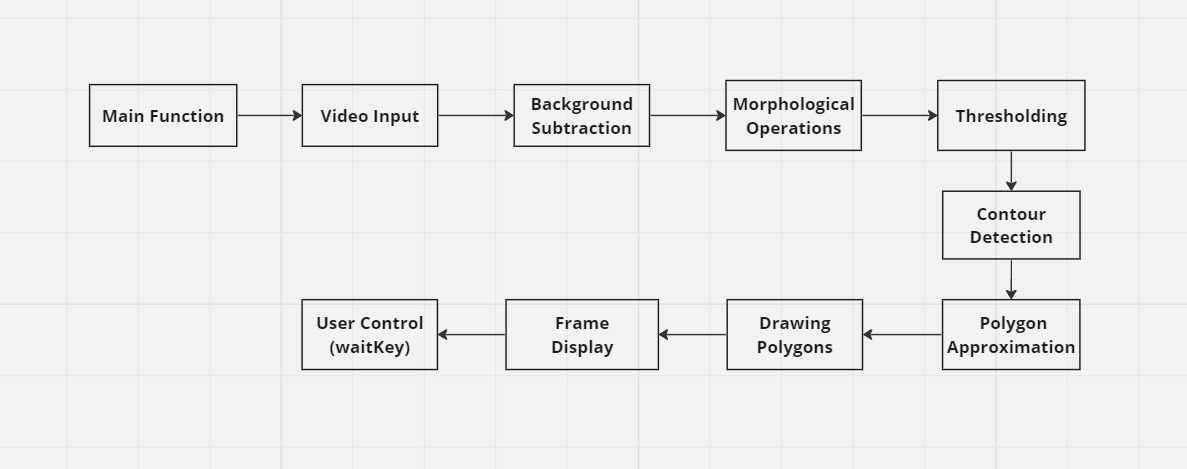
Step 15: Repeat step 5.

Step 16: Release the video capture object, closing the video stream.

Step 17: Close all windows displaying the frame.

Step 18: End of program.

**Component Diagram:**



**Sequence Diagram:**

User Program video capture BackgroundSubtractor Morphological Operations

| | | | |

|-- Start video -->| | | |

| |-- Open video ----->| | |

| | |-- Read frame --------->| |

| | |<-- Frame captured ---| |

| | | |-- Apply background subtraction ->|

| | | |<-- Foreground mask ------------------|

| | | |-- Morphological operations ------->|

| | | |<-- Processed foreground mask -----|

| | | |-- Calculate mean value ------------->|

| | | |<-- Mean value --------------------------|

| | | |-- Thresholding ------------------------>|

| | | |<-- Binary mask -------------------------|

| | | |-- Find contours ----------------------->|

| | | |<-- Contours -----------------------------|

| | | |-- Approximate polygons ----------->|

| | | |<-- Approximated polygons ---------|

| | | |-- Draw polygons --------------------->|

| | | |<-- Frame with polygons -------------|

| | |-- Display frame ------>| |

| |<-- Display frame --| | |

| | | | |

| |-- Next frame ----->| | |

| | | | |

| |-- End video ------->| | |

| | | | |

| | |<-- Video released -----| |

| | | | |

| |-- Close program ---| | |

| | | | |

**Code Explanation:**

1. Header Includes:
   * The code includes necessary header files from the OpenCV library for image processing, video capture, and object detection.
2. Main Function:
   * The main function is the entry point of the program.
   * It takes command-line arguments, specifically the path to the video file to be processed.
   * If no video file is provided, the program prints an error message and exits.
3. Video Capture:
   * A VideoCapture object is created to read frames from the input video file.
   * If the video file fails to open, an error message is printed, and the program exits.
4. Background Subtraction:
   * A BackgroundSubtractorMOG2 object is created for background subtraction.
   * This object is used to generate a foreground mask that highlights moving objects in each frame.
5. Object Detection Loop:
   * The program enters a loop to process each frame of the video.
   * It reads a frame from the video capture object.
   * Background subtraction is applied to the frame to obtain a foreground mask.
   * Morphological operations (opening and closing) are performed on the foreground mask to clean up noise.
   * The mean value of the foreground mask is calculated to determine a threshold for creating a binary mask.
   * Contours of objects in the binary mask are detected using the findContours function.
   * Each contour is approximated by a polygon using the approxPolyDP function.
   * The detected polygons are drawn onto the original frame.
6. Memory Management:

* The code uses smart pointers (**unique\_ptr** and **shared\_ptr**) to manage the memory for the **VideoCapture** object and the background subtractor. Matrices like **frame**, **fg\_mask**, and **binary\_mask** are created and released within the loop to manage memory efficiently.
  + Dynamic memory allocation is used for storing contours and polygon approximations.
  + Memory allocated during each iteration of the loop is properly deallocated to prevent memory leaks.

1. Display:
   * The processed frame, with detected objects outlined, is displayed in a window.
2. User Interaction:
   * The program waits for the user to press the 'q' key to quit.
   * While the program is running, the user can view the processed video and the detected objects in real time.
3. Cleanup:
   * After processing all frames or when the user quits, the video capture object is released, and all windows are closed.

**User Manual:**

1. Compilation:
   * Ensure you have OpenCV installed on your system. You can install it using package managers like apt (for Linux) or brew (for macOS) or by building it from source.
   * Compile the code using a C++ compiler like g++.
   * You can use the following command:

**g++ -o bosch bosch.cpp ‘pkg-config –cflags –libs opencv4’**

* + This will generate an executable file named bosch.

1. Execution:
   * Run the compiled executable along with the path to the video file you want to process as a command-line argument. For example:

**bosch.exe video\_file\_path.mp4 0x7fffe8b05570**

* + Replace video\_file\_path.mp4 with the actual path to your video file and replace 0x7fffe8b05570 the address with your base memory address.

3 Operation:

* + Once the program is running, it will display the processed video frames in a window.
  + Moving objects in the video will be outlined with maroon-colored polygons.
  + You can press the 'q' key to quit the program at any time.

1. Exiting:
   * To exit the program, simply close the window displaying the video frames or press the 'q' key while the window is active.
2. Cleanup:
   * After the program finishes, it will automatically release all resources and close any open windows.

**Output**

****