Context Free Application Development Plan

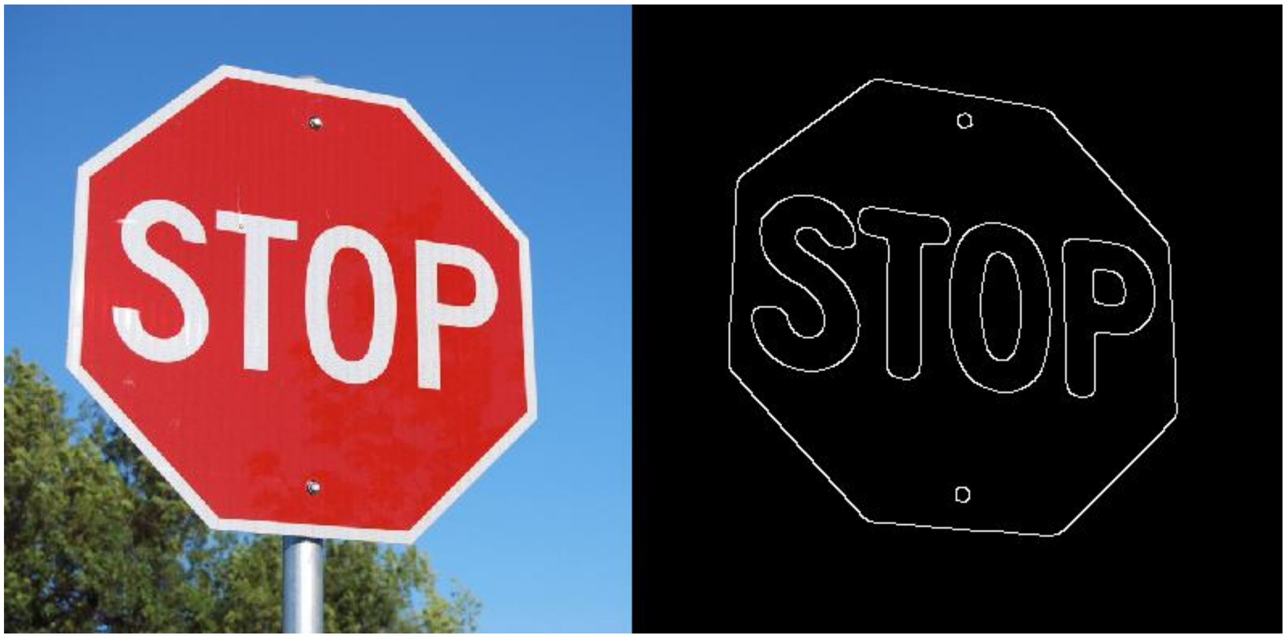


Figure 1: Canny Edge Detection run on a blurred version of (Red – (Green + Blue)) channel subtraction. Removes everything not related to the stop sign, including the pole and trees in the background.

1 Team

- Team name: Context Free
- Team members: Kevin Cherry, Robert Firth, Dennis Castleberry

2 Requirements

- **Main goal**: to minimize the likelihood of pedestrian and driver casualties due to inaccuracies and inefficiencies in sign, crosswalk, and pedestrian detection algorithms.

  To accomplish this, we propose implementing three algorithms for stop sign detection – one implementation of SURF and two new faster algorithms specifically devised for the task of fast stop sign detection. These algorithms are listed below as objectives.

- **Supplementary goals, objectives, and tasks**:
  1. **Goal**: stop sign detection. Detect stop signs in real-time with sufficient efficiency to allow for stopping time. Test SURF against two alternate algorithms, described below.
     (a) **Objective**: construct an R-channel integral-image-based detector.
        i. **Task**: compute integral image on the R-channel of the image.
        ii. **Task**: superimpose an*mxm*grid on the image and compute the sums of the R-channel intensity per 2x2 region.
        iii. **Task**: for the 2x2 region with the maximum sum, recursively apply (ii) until the region with the maximum R-channel intensity density is obtained.

   iv. **Task**: in the neighborhoods of the vertices inside the maximum-density region, find the four box such that the diagonal along those boxes separates a maximum-density R-channel triangle from a minimum-density R-channel triangle.

   v. **Task**: obtain the 8 vertices of the sign from the above.

  (b) **Objective**: construct octagon detector.

   – **Task**: apply a shape detector to yield sets of vertices.

   – **Task**: discard all sets where *n*6= 8.

   – **Task**: check to see if the opposing edges of the 8-vertex shape are parallel (within a certain threshold).

2. **Goal**: pedestrian crosswalk sign detection.

  (a) **Objective**: construct and RG-channel integral-image-based detector.

   i. **Task**: compute integral image on the R-channel of the image.

   ii. **Task**: superimpose an*mxm*grid on the image and compute the sums of the R-channel intensity per 2x2 region.

   iii. **Task**: for the 2x2 region with the maximum sum, recursively apply (ii) until the region with the maximum R-channel intensity density is obtained.

   iv. **Task**: in the neighborhoods of the vertices inside the maximum-density region, find the two boxes such that the diagonal along those boxes separates a maximum-density RG-channel triangle from a minimum-density RG-channel triangle.

   v. **Task**: obtain the 3 vertices of the sign from the above.

3 Design

- Classes in namespace OMS.CVApp.SignDetector SurfStopSignDetector, IntegralImageStopSignDetector, OrientedSurfStopSignDetector are concrete implementations of the StopSignDetector class. Likewise, HogPedestrianDetector is an implementation of the abstract PedestrianDetector class. Both StopSignDetector and PedestrianDetector are subclasses of Detector.

- **Modules**
  1. **Input module:**
     a) **No input module. All of our detector code assumes data is already loaded into an Emgu.CV.Image<Bgr, Byte>.**
  2. **MS.CVApp.SignDetector.SURFStopSignDetector**
     a) **Functionalities:** Implementation of a stop sign detector using a SURF implementation provided by EmguCV.
     b) **Outputs:** Array of System.Drawing.Rectagles, each rectangle containing bounds of detected stop sign
        - **Also using classes** OMS.CVApp.Detector, OMS.CVApp.SignDetector.StopSignDetector, OMS.CVApp.SignDetector.StopSignDetectorA
  3. **OMS.CVApp.SignDetector.IntegralImageStopSignDetector**:
     a) **Functionalities**: computes and uses integral image on the R-channel to find maximum density areas used to approximate the region occupied by the stop sign.
     b) **Outputs**: Array of System.Drawing.Rectagles, each rectangle containing bounds of a detected stop sign

c) **Data structures**:

- **Namespaces**: EmguCV.CV.
- **Classes** (Image) and **class members**: Resize(), Convert(),
- **Also using classes** OMS.CVApp.Detector, OMS.CVApp.SignDetector.StopSignDetector, OMS.CV.AppStopSignDetectorA

4. **OMS.CVApp.SignDetector.OrientedSurfWarningSignDetector**:

a) **Functionalities**: detects a warning sign, determines its orientation, rotates the warning sign models, and finds a match using SURF. Used to extend the robustness to out-of-plane rotation.

b) **Outputs**: Array of System.Drawing.Rectangles, each rectangle containing bounds of a detected warning sign

c) **Data structures**: Image, LineSegment2D, boxList, triangleList, Contour

- **Namespaces**: EmguCV.CV.
- **Classes** (Image, Contour) and **class members**: Canny(), HoughLinesBinary(), Resize(), Convert(), PyrUp(), PyrDown(), FindContours().
- **Also using classes** OMS.CVApp.Detector, OMS.CVApp.SignDetector.WarningSignDetector, OMS.CVApp.SignDetectorOrientedSurfWarningSignDetector

5. **OMS.CVApp.HogPedestrianDetector**:

a) **Functionalities**: detects pedestrians using built-in EmguCV classes.

b) **Outputs**: Array of System.Drawing.Rectangles, each containing the bounds of a detected pedestrian

c) **Data structures**: System.Drawing.Rectangle[].

- **Namespaces**: EmguCV.CV.
- **Classes** (Image, HOGDescriptor) and **class members**: Draw(), SetSVMDetector(), DetectMultiScale(), GetDefaultPeopleDetector().
- **Also using classes** OMS.CVApp.Detector, OMS.CVApp.PedestrianDetector, OMS.CVApp.HogPedestrianDetector

6. **OMS.CVApp.**

a) **Functionality:** Implements the main app. Implements the Abstract Factory design pattern to switch between implementations of algorithms to allow us to compare accuracy.
b) **Inputs:** Static images, camera stream, or saved video.
c) **Outputs:** Annotated images or streams of images.

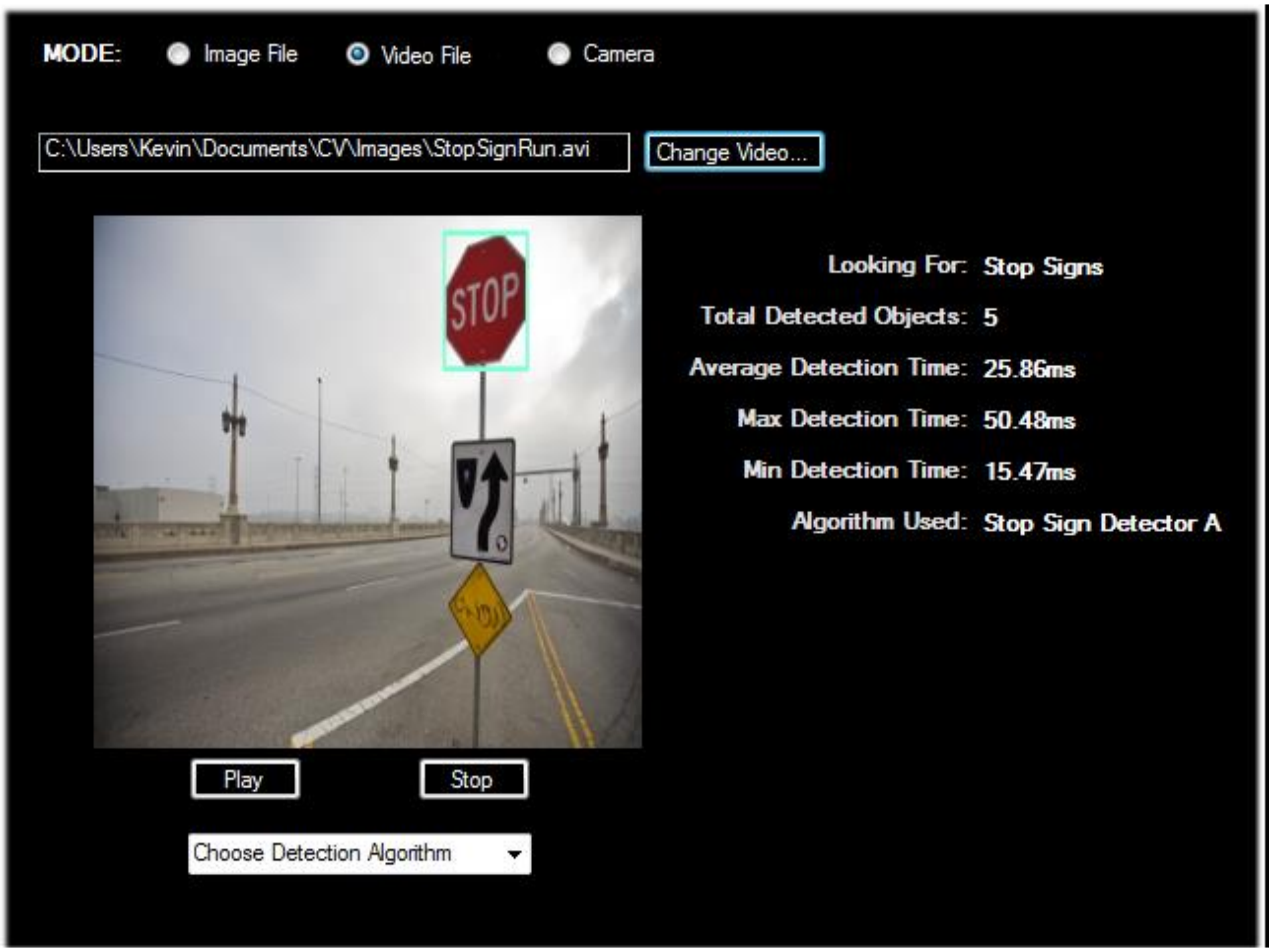
4 Data

We will capture video from areas dense in pedestrian crosswalk and child crossing signs in a variety of weather conditions (normal weather, nighttime, light and heavy rain); then segment the video to isolate the target objects. We require two data sets; one to explore existing implementations and make incremental improvements, and a second to test the improved implementation.

5 Implementation Plan and Timeline

• Setup project	10/19	Done
• Implement User Interface	10/22	Done
• Implement class hierarchy	10/23	Done
• Implement SurfStopSignDetector	10/26	Done
• Implement IntegralImagesStopSignDetector	11/30	In progress
• Implement OrientedSurfStopSignDetector	11/30	In progress
• Implement HogPedestrianDetector	10/26	Done
• Compare methods	12/2	In progress
• Testing	ongoing	
• Documentation	ongoing	

User Interface

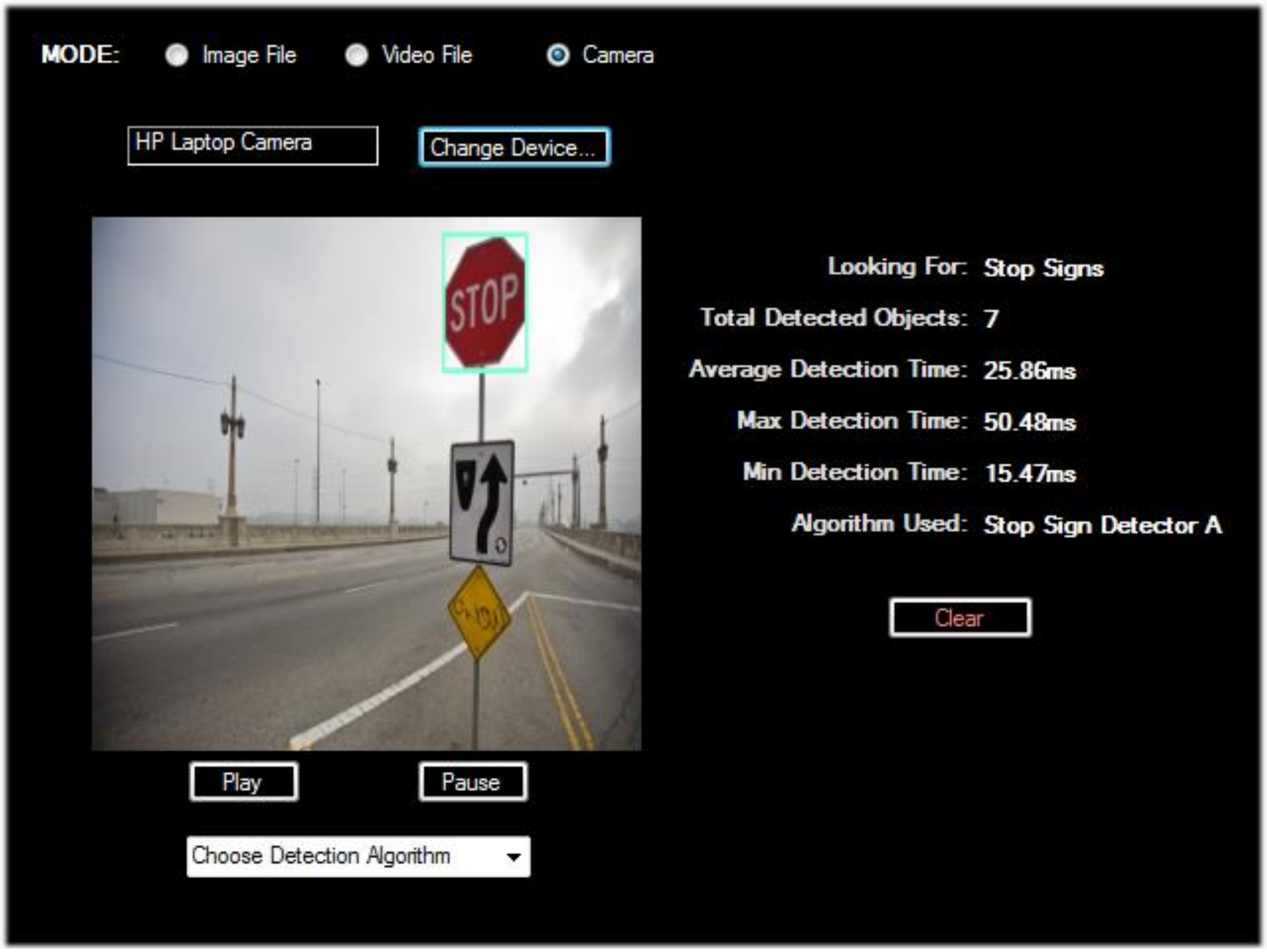


MODE: Image File [✓ Video File] Camera

C:\Users\Kevin\Documents\CV\Images\StopSignRun.avi

Looking For: Stop Signs
Total Detected Objects: 5
Average Detection Time: 25.86ms
Max Detection Time: 50.48ms
Min Detection Time: 15.47ms
Algorithm Used: Stop Sign Detector A

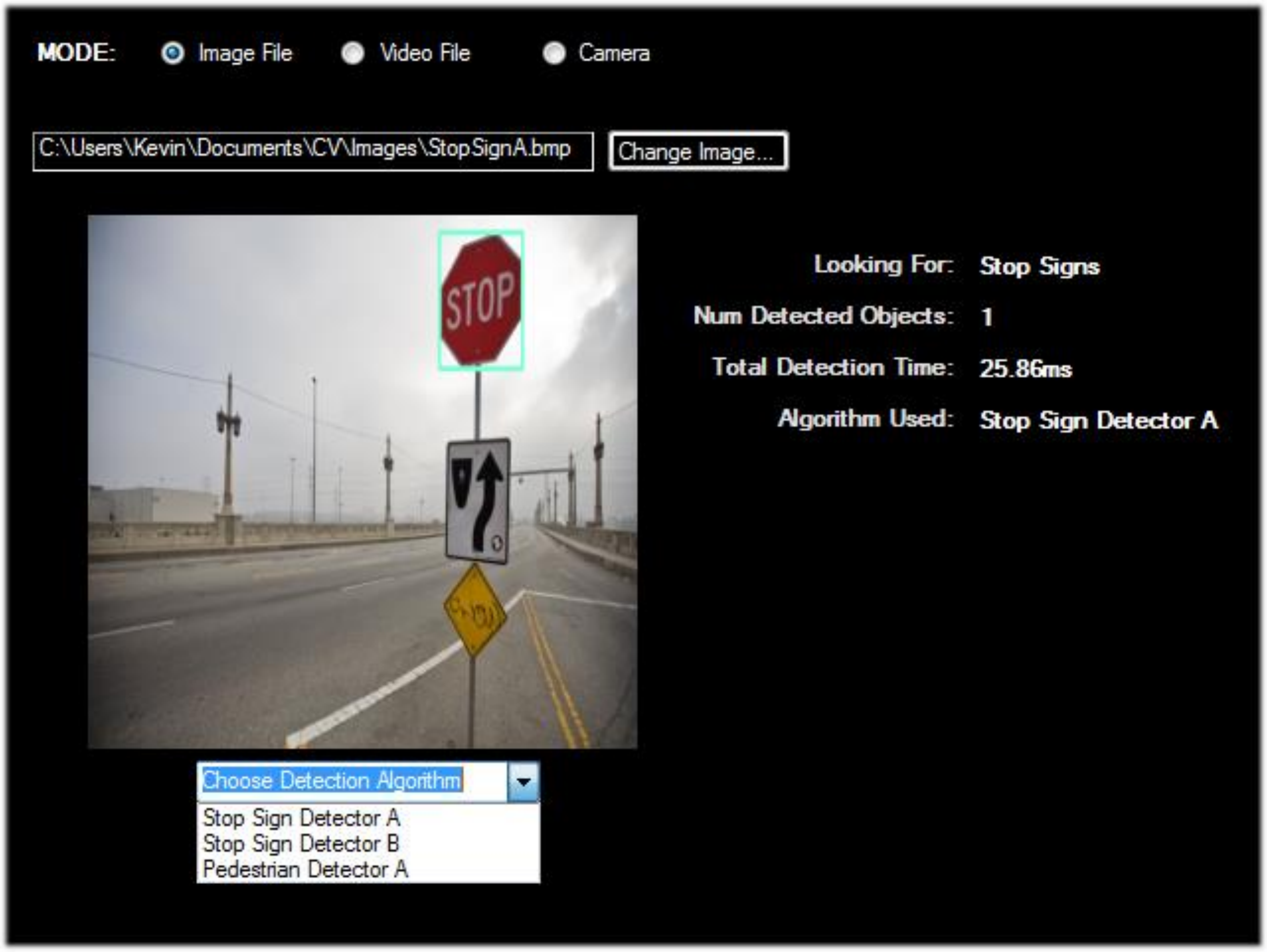
Figure 2: Stop sign detection run on pre-recorded video for OMS.CVApp.SignDetector.StopSignDetectorA



HP Laptop Camera

Looking For: Stop Signs
Total Detected Objects: 7
Average Detection Time: 25.86ms
Max Detection Time: 50.48ms
Min Detection Time: 15.47ms
Algorithm Used: Stop Sign Detector A

Figure 3: Stop sign detection run on live video, using OMS.CVApp.SignDetection.StopSignDetectorA


MODE: [✓ Image File] Video File Camera

C:\Users\Kevin\Documents\CV\Images\StopSignA.bmp

Looking For: Stop Signs

Num Detected Objects: 1

Total Detection Time: 25.86ms

Algorithm Used: Stop Sign Detector A

Figure 4: Stop sign detection run on a static image, using OMS.CVApp.SignDetection.StopSignDetectorA

Appendix

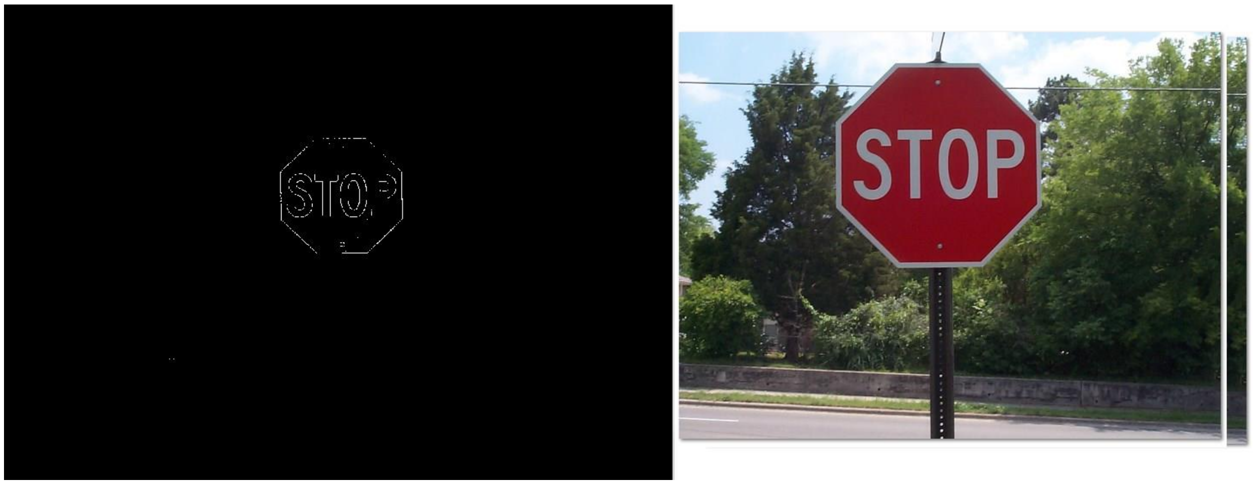


Figure 5: Edge detector by computing R-(G+B), clamped from 0 to 255

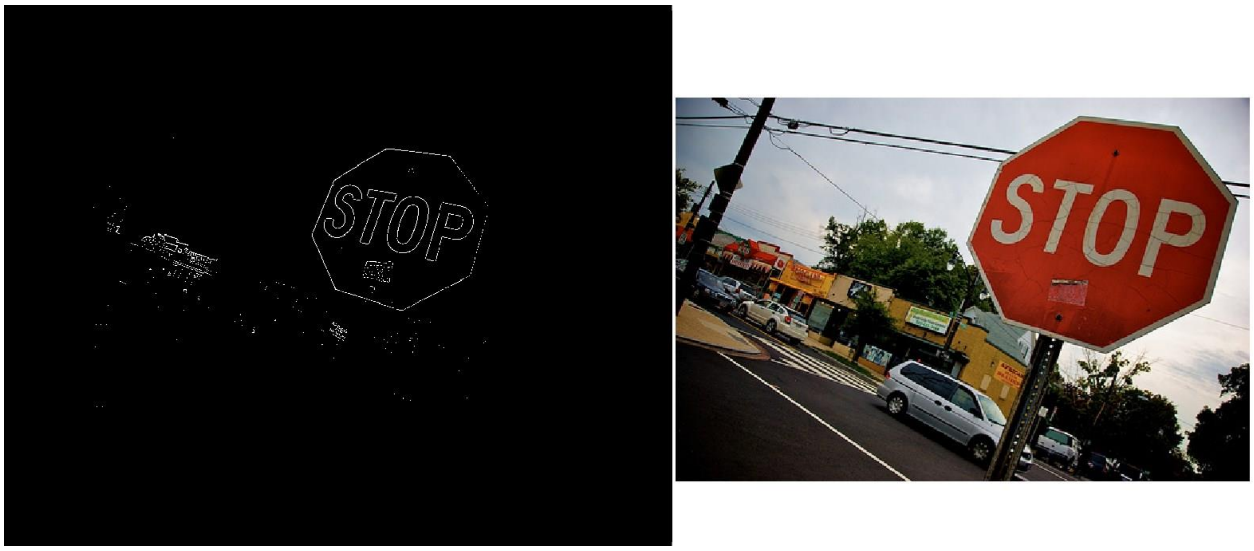


Figure 6: Edge detector by computing R-(G+B), clamped from 0 to 255