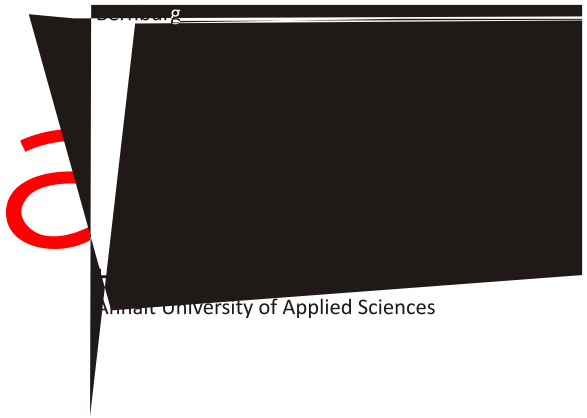

Ibrahim Hassan, MET 2023, 5152780

Eshwar Srinivasan, MET 2023, 5130032

Improving Traffic Sign Recognition (Traffic Light)

Declaration of independence

	Ibrahim. H
	S. Eshwar

Contents

1 Motivation and goal setting 4

2 Theoretical basics 7

3 Optimizing Blob Detection 16

4 Alternative Algorithm: Pixel Counting 22

5 Conclusion and Future Thoughts 28

6 Figure directory 29

7 Refrences 30

1 Motivation and goal setting

1.1 Introduction

1.2 Objective

Optimizing the Blob Detection Algorithm:

Implementing a Pixel Counting Mechanism:

1.3 Methods

1.4 Scope

Current Algorithm Assessment:

Algorithm Modification:

New Algorithm Proposal:

Method Evaluation:

1.5 Software

2 Theoretical basics

2.1 OpenCV

2.1.1 Gaussian Blur

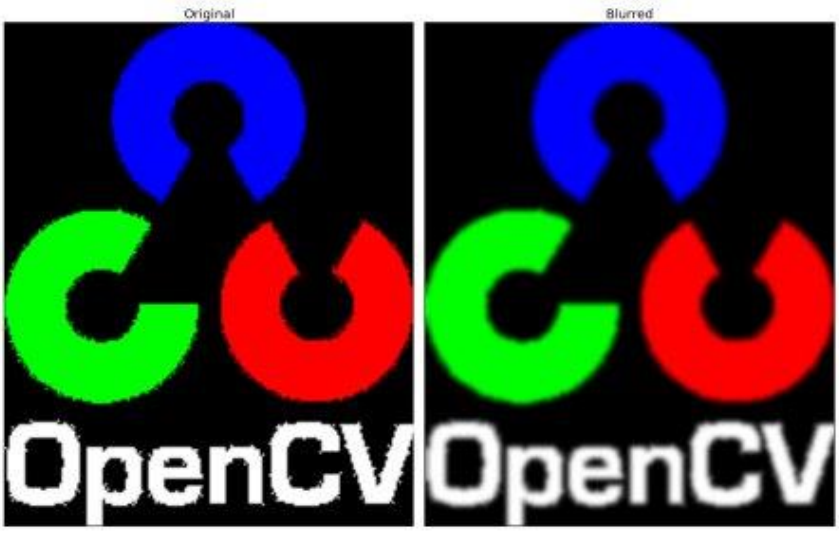

Figure 2.1*: Applying Gaussian blur using OpenCV*

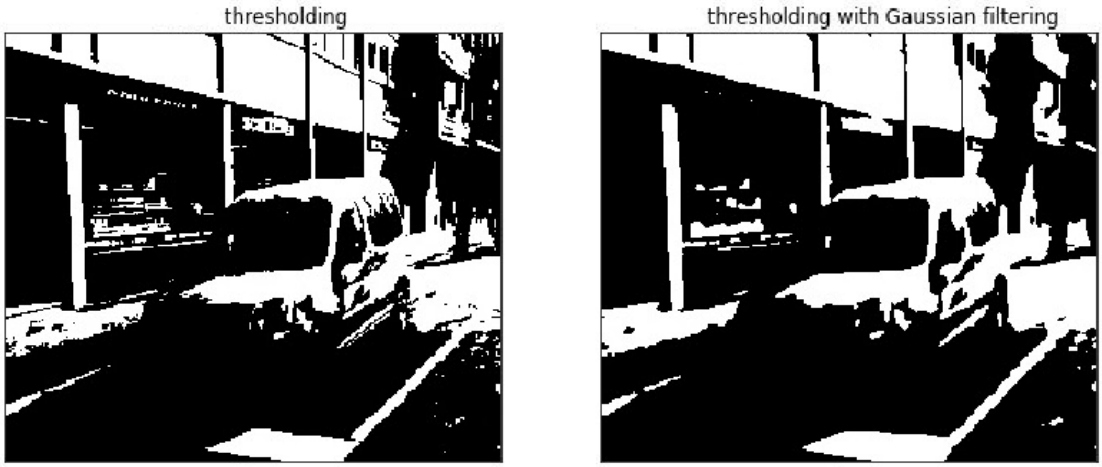

Figure 2.2 Effect of Gaussian Blur on Thresholding[6]

2.1.2 Color Filter

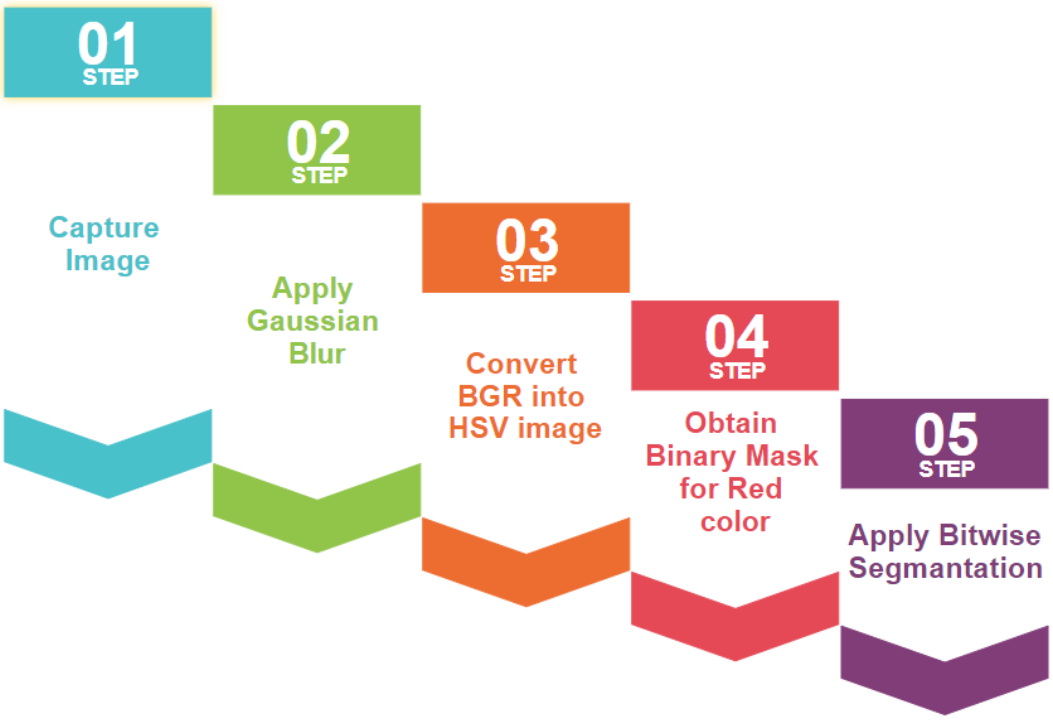

Figure 2.3 Steps to filter Red color

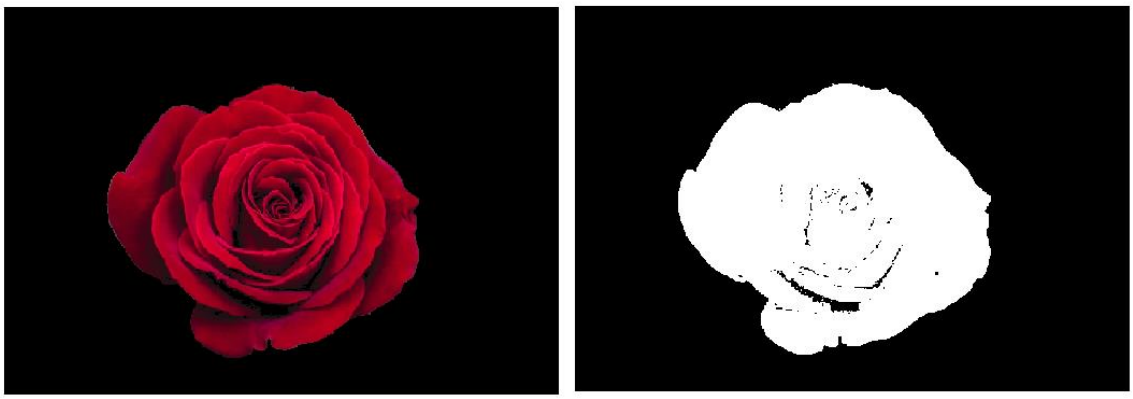

Figure 2.4 Isolating Red of an image.[7]

Advantages of Using HSV Color Space

- **Hue**
- **Saturation**
- **Brightness**

2.1.3 Blob Detection

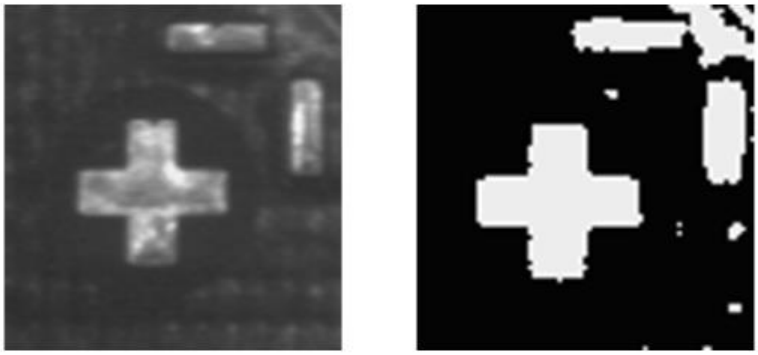

Figure 2.5 Different Blobs in an image

SimpleBlobDetector()

SimpleBlobDetector() Parameters

- **Filter by Area**

filterByArea = 1

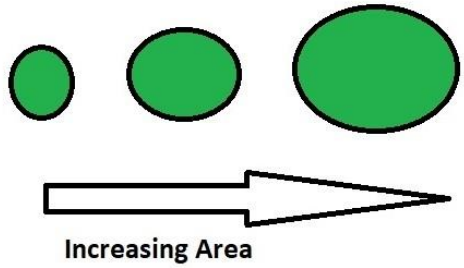

Figure 2.6 Blob Area Parameter

- **Filter by Circularity**

filterByCircularity = 1

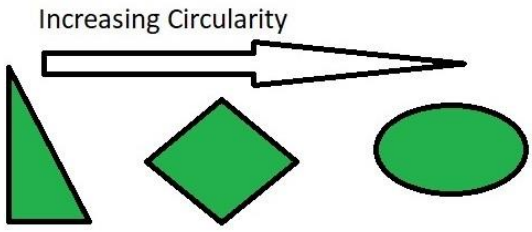

Figure 2.7 Blob Circularity Parameter

- **Filter by Convexity**

filterByConvexity = 1

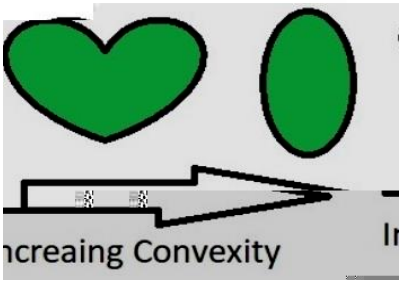

Figure 2.8 Blob Convexity Parameter

- **Filter by Inertia**

  filterByInertia = 1

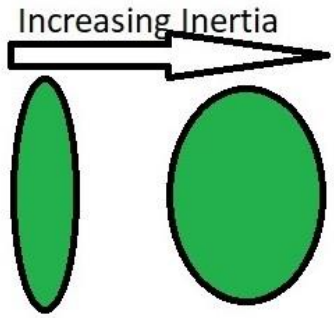

Figure 2.9 Blob Inertia Parameter

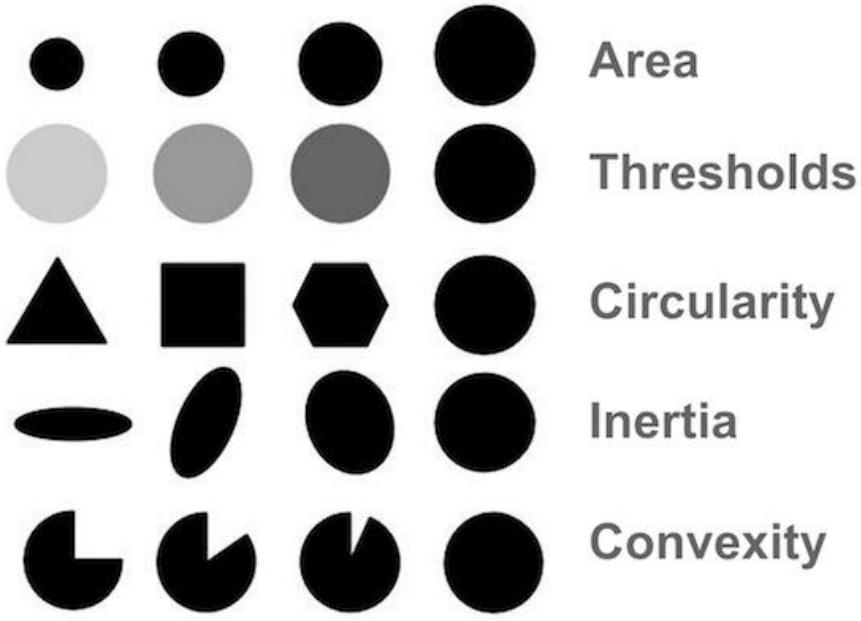

Figure 2.10 SimpleBlobDetector() Parameters.[11]

2.2 Traffic light Detection Algorithm

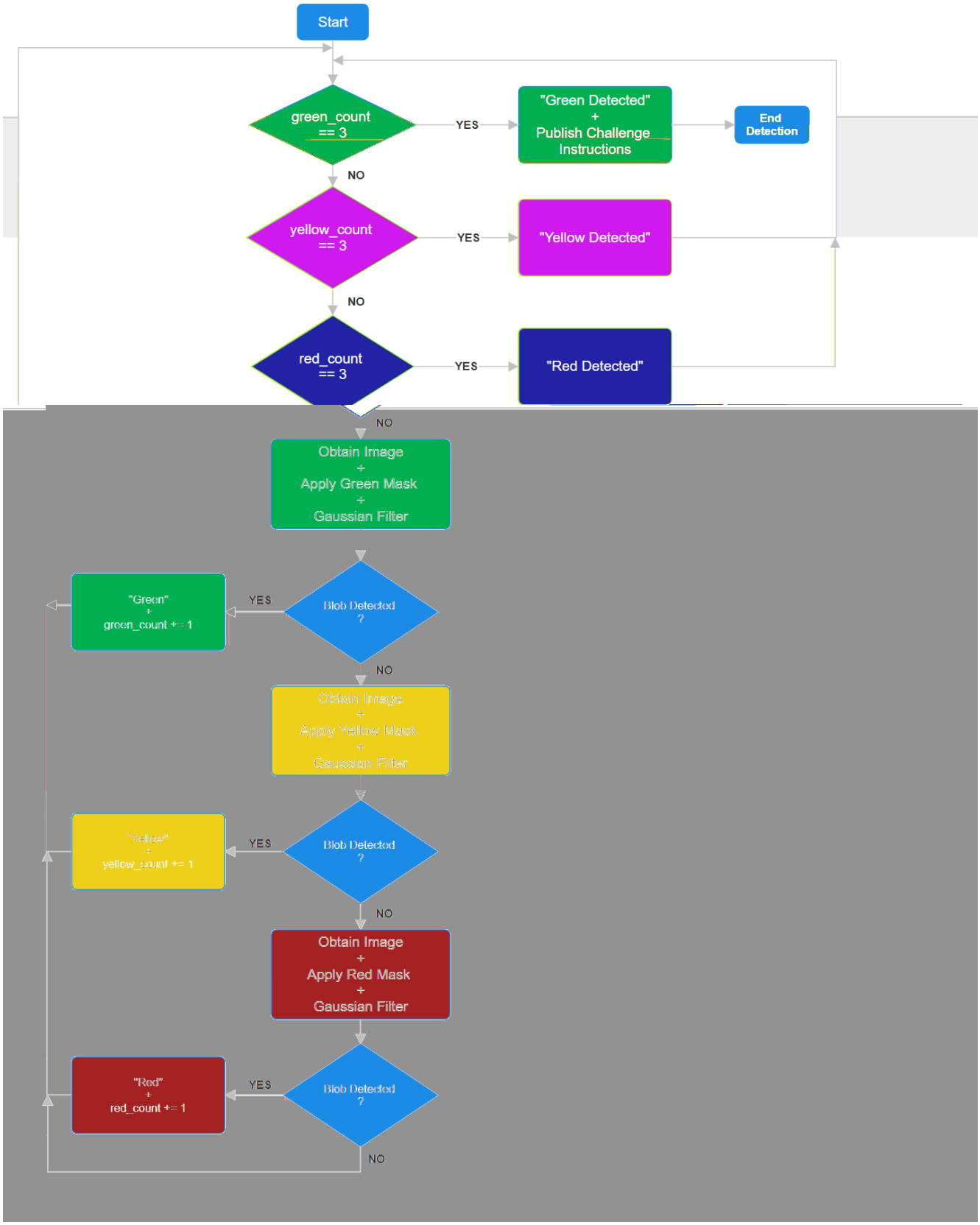

Figure 2.11 Traffic Light Detection Flow Chart

Initialization:

```
self.green_count = 0        #Initializing Counters
self.yellow_count = 0
self.red_count = 0
```

Green Detection:

- I

```
def fnMaskGreenTrafficLight(self):
    image = np.copy(self.cv_image)

    # Convert BGR to HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    Hue_l = self.hue_green_l
    Hue_h = self.hue_green_h
    Saturation_l = self.saturation_green_l
    Saturation_h = self.saturation_green_h
    Lightness_l = self.lightness_green_l
    Lightness_h = self.lightness_green_h

    # define range of green color in HSV
    lower_green = np.array([Hue_l, Saturation_l, Lightness_l])
    upper_green = np.array([Hue_h, Saturation_h, Lightness_h])

    # Threshold the HSV image to get only green colors
    mask = cv2.inRange(hsv, lower_green, upper_green)

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(image, image, mask = mask)
```

-

```
def fnFindTrafficLight(self):
    rospy.loginfo("[Deting traffic light color]")
    cv_image_mask = self.fnMaskGreenTrafficLight()
    cv_image_mask = cv2.GaussianBlur(cv_image_mask,(5,5),0)
```

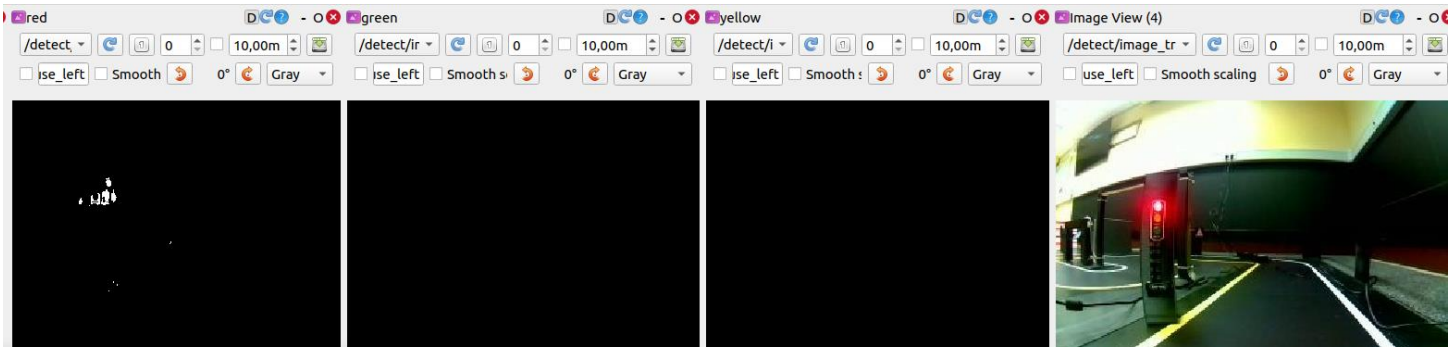

Figure 2.12 The three published images after color and noise filteration

-

```python
def fnFindCircleOfTrafficLight(self, mask, find_color):
    status = 0

    params=cv2.SimpleBlobDetector_Params()
    # Change thresholds
    params.minThreshold = 0
    params.maxThreshold = 255

    # Filter by Area.
    params.filterByArea = True
    params.minArea = 50
    params.maxArea = 600

    # Filter by Circularity
    params.filterByCircularity = True
    params.minCircularity = 0.4

    # Filter by Convexity
    params.filterByConvexity = True
    params.minConvexity = 0.6

    det=cv2.SimpleBlobDetector_create(params)
```

-

Yellow and Red Detection:

-

Looping Algorithm:

-
-
-
-
-

3 Optimizing Blob Detection

3.1 Optimizing Procedure

I

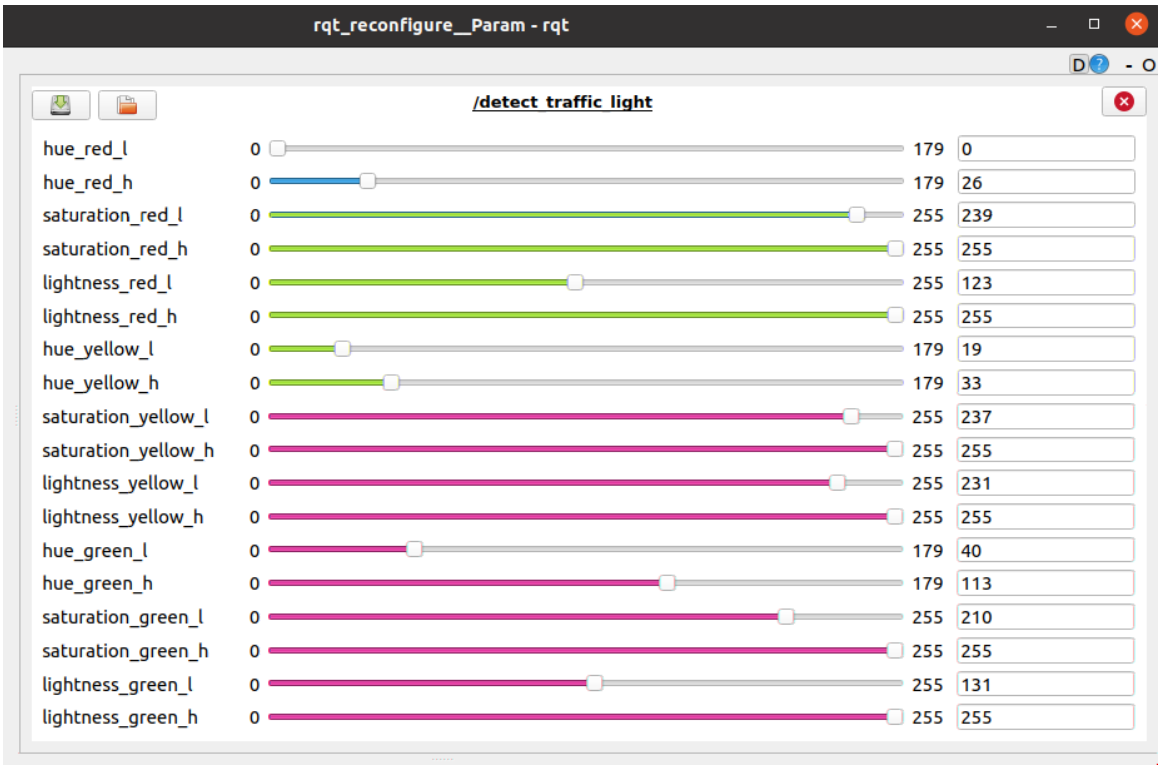

Figure 3.1 Dynamic Callibration for Traffic light detection

rqt dynamic

Figure 3.2 Steps to optimize Blob Detection

File and Code Location:

-
-

Tuning Process:

-

Testing and Evaluation:

-
-

3.2 Implementation

5 files

/home/user/catkin_ws/devel/lib/python3/dist-packages/turtlebot3_autorace_detect/cfg/DetectTrafficLightParamsConfig.py

/home/user/catkin_ws/src/turtlebot3_autorace_HSA23/turtlebot3_autorace_detect/nodes/detect_traffic_light

/home/user/catkin_ws/src/turtlebot3_autorace_HSA23/turtlebot3_autorace_detect/cfg/DetectTrafficLightParams.cfg

/home/user/catkin_ws/src/turtlebot3_autorace_HSA23/turtlebot3_autorace_detect/param/traffic_light/traffic_light.yaml

/home/user/catkin_ws/devel/include/turtlebot3_autorace_detect/DetectTrafficLightParamsConfig.h

I

DynamicBlob

```python
class DetectTrafficLight():
    def __init__(self):
        '''>>>>>>>>>>>>>>>>>>ORIGINAL HSV PARAMETERS<<<<<<<<<<<<<<<<<<<<<<<<<<'''

        self.hue_red_l = rospy.get_param("~detect/traffic_light/red/hue_l", 0)
        self.hue_red_h = rospy.get_param("~detect/traffic_light/red/hue_h", 20)
        self.saturation_red_l = rospy.get_param("~detect/traffic_light/red/saturation_l", 1)
        self.saturation_red_h = rospy.get_param("~detect/traffic_light/red/saturation_h", 255)
        self.lightness_red_l = rospy.get_param("~detect/traffic_light/red/lightness_l", 1)
        self.lightness_red_h = rospy.get_param("~detect/traffic_light/red/lightness_h", 255)
        self.hue_yellow_l = rospy.get_param("~detect/traffic_light/yellow/hue_l", 25)
        self.hue_yellow_h = rospy.get_param("~detect/traffic_light/yellow/hue_h", 35)
        self.saturation_yellow_l = rospy.get_param("~detect/traffic_light/yellow/saturation_l", 1)
        self.saturation_yellow_h = rospy.get_param("~detect/traffic_light/yellow/saturation_h", 255)
        self.lightness_yellow_l = rospy.get_param("~detect/traffic_light/yellow/lightness_l", 1)
        self.lightness_yellow_h = rospy.get_param("~detect/traffic_light/yellow/lightness_h", 255)
        self.hue_green_l = rospy.get_param("~detect/traffic_light/green/hue_l", 45)
        self.hue_green_h = rospy.get_param("~detect/traffic_light/green/hue_h", 75)
        self.saturation_green_l = rospy.get_param("~detect/traffic_light/green/saturation_l", 1)
        self.saturation_green_h = rospy.get_param("~detect/traffic_light/green/saturation_h", 255)
        self.lightness_green_l = rospy.get_param("~detect/traffic_light/green/lightness_l", 1)
        self.lightness_green_h = rospy.get_param("~detect/traffic_light/green/lightness_h", 255)

        '''>>>>>>>>>>>>>>>>>>DYNAMIC BLOB PARAMETERS<<<<<<<<<<<<<<<<<<<<<<<<<<'''

        self.minThreshhold = rospy.get_param("~detect/traffic_light/blop/minThreshhold", 1)
        self.maxThreshhold = rospy.get_param("~detect/traffic_light/blop/maxThreshhold", 255)
        self.minArea = rospy.get_param("~detect/traffic_light/blop/minArea", 50)
        self.maxArea = rospy.get_param("~detect/traffic_light/blop/maxArea", 600)
        self.minCircularity = rospy.get_param("~detect/traffic_light/blop/minCircularity", 0.4)
        self.maxCircularity = rospy.get_param("~detect/traffic_light/blop/maxCircularity", 0.6)
```

Figure 3.3: Example to implementing Blob Parameters into dynamic rqt reconfigure

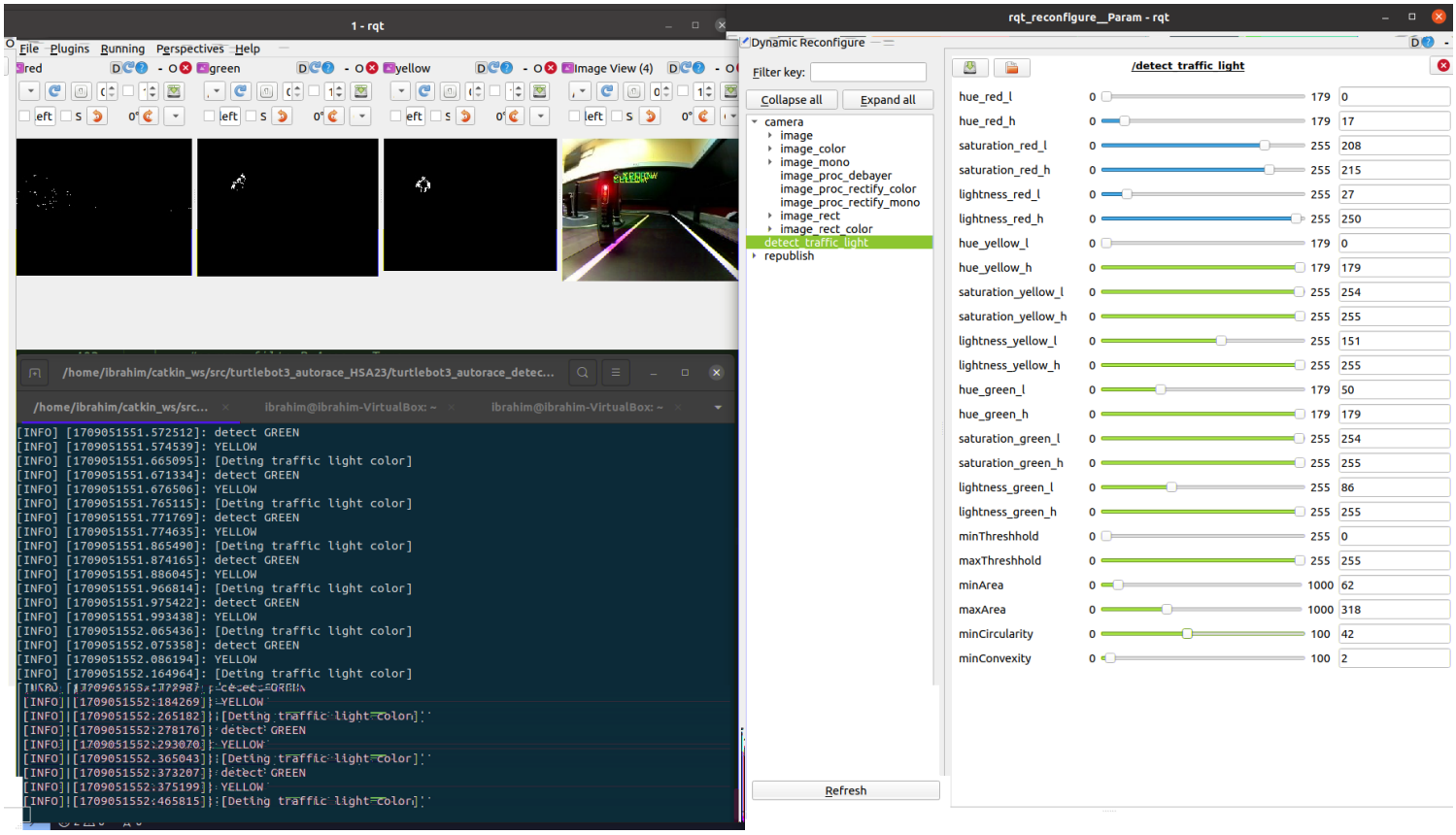

Figure 3.3 Blob detector parameters integrated into the calibration environment

3.3 Observation and Results

Improved Detection Performance:

Increased Crashes:

Less Tolerance for Errors:

Optimal Values:

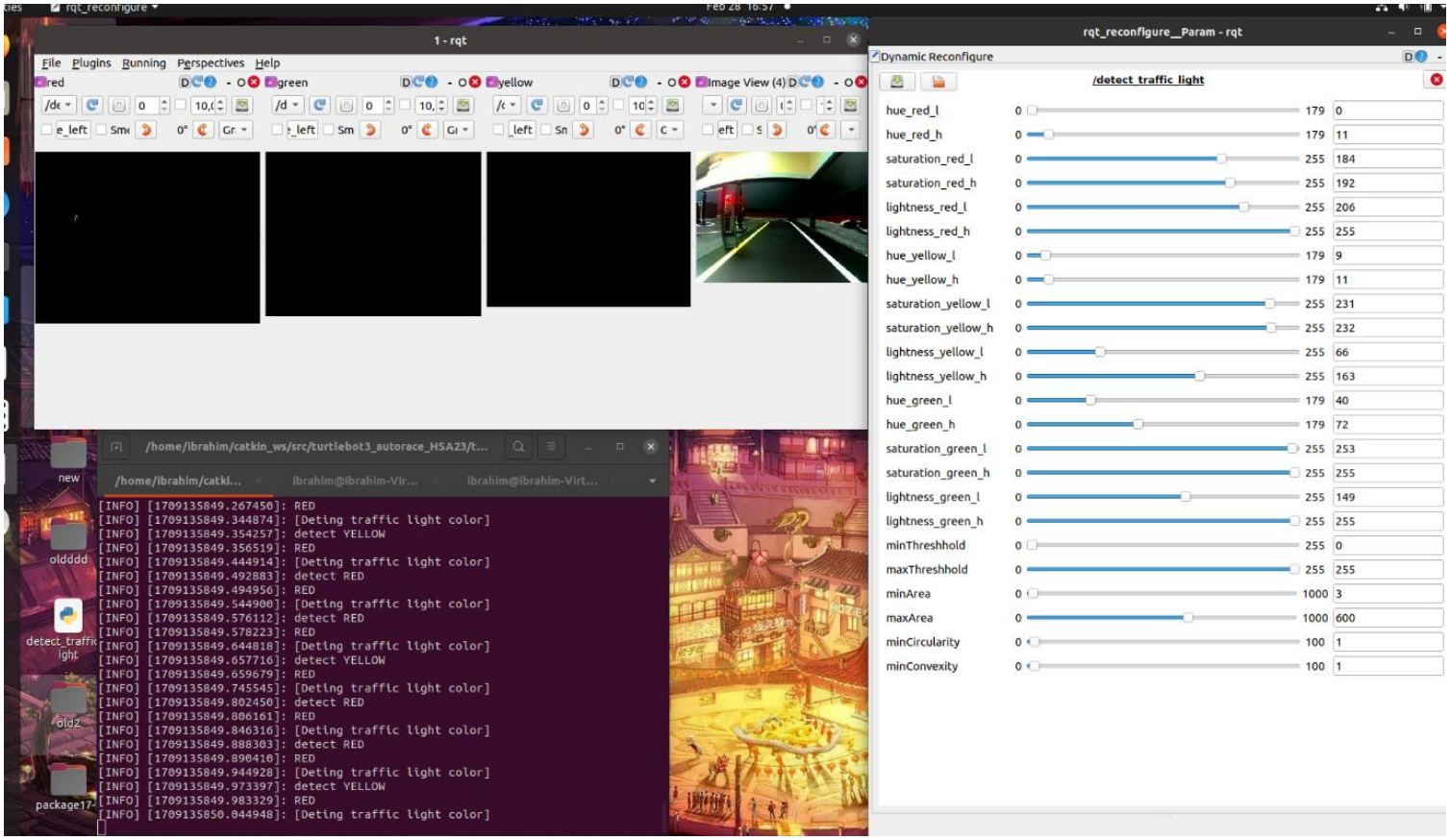

Figure 3.4 Most accurate Blob detection Parameters

New Approach:

4 Alternative Algorithm: Pixel Counting

4.1 Introduction

- **Optimal Blob Detection Parameters:**

- **Challenges with Perfect Color Filtering:**

Setting up Calibration Environment:

Calibrating Color Filters:

Implementing Detection Algorithm:

4.2 Algorithm and Implementation

np.sum() **NumPy**

```
if find_color == 'green':
    rospy.loginfo("Green Pixel Count =  %d", (19584000-np.sum(mask))/256)
elif find_color == 'yellow':
    rospy.loginfo("Yellow Pixel Count : %d", (19584000-np.sum(mask))/256)
elif find_color == 'red':
    rospy.loginfo("Red Pixel Count : %d", (19584000-np.sum(mask))/256)
```

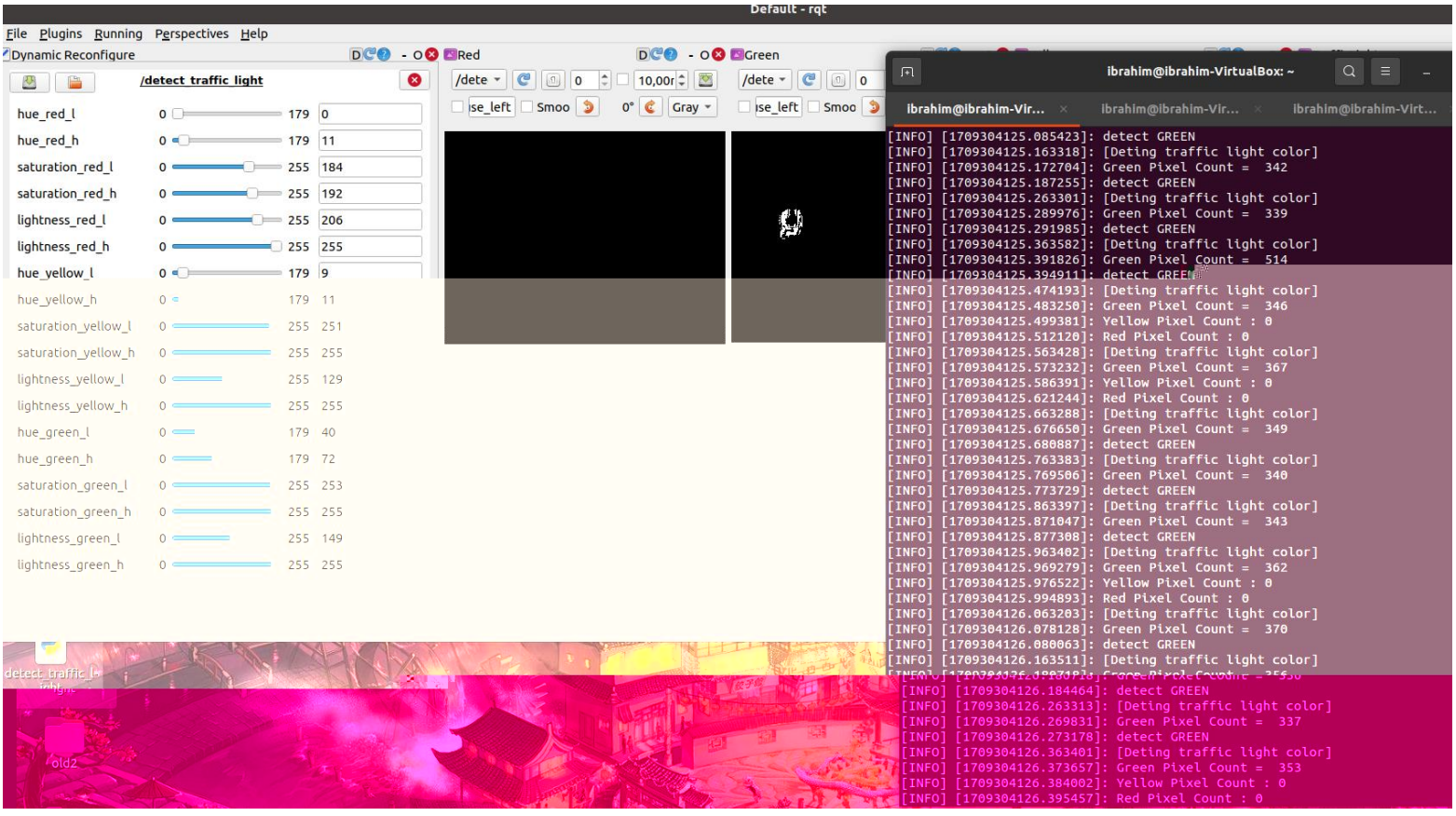

Figure 4.1 Callibrating Pixel Counter

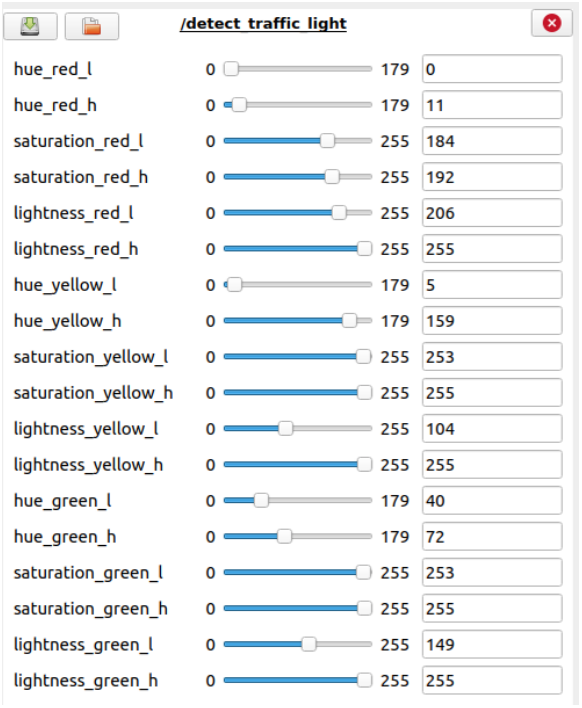

Figure 4.2 Optimal Parameters for Pixel Counter

	RED PIXELS DETECTED	YELLOW PIXELS DETECTED	GREEN PIXELS DETECTED
RED TRAFFIC LIGHT	1~4	0	0
YELLOW TRAFFIC LIGHT	0	15~30	10~20
GREEN TRAFFIC LIGHT	0	500~550	300~550

Figure 4.3 Pixel Detector Count Table

Red Pixel Count:

-

Yellow Pixel Count:

-
-
-

Green Pixel Count:

-
-
-

```
def fnFindTrafficLight(self):
    rospy.loginfo("[Deting traffic light color]")
    cv_image_mask = self.fnMaskGreenTrafficLight()                    #filter using green mask
    cv_image_mask = cv2.GaussianBlur(cv_image_mask,(5,5),0)           #Gaussian Blur
    GreenPixel = (19584000-np.sum(cv_image_mask))/256                 #Obtain GreenPixel count
    #status1 = self.fnFindCircleOfTrafficLight(cv_image_mask, 'green')  #Uncomment this line for debugging (printing detected pixel count)
    if GreenPixel >= 50:                                              #check condition
        rospy.loginfo("detect GREEN")                                 #publish message
        self.green_count += 1                                         #increment counter
        self.red_count = 0                                            #reset other counters
        self.yellow_count = 0
```

```
else:
    self.green_count = 0
    cv_image_mask = self.fnMaskRedTrafficLight()                  #filter using red mask
    cv_image_mask = cv2.GaussianBlur(cv_image_mask,(5,5),0)       #Gaussian Blur
    RedPixel = (19584000-np.sum(cv_image_mask))/256               #Obtain RedPixel count
    #status2 = self.fnFindCircleOfTrafficLight(cv_image_mask, 'yellow') #Uncomment this line for debugging (printing detected pixel count)
    if RedPixel >= 1:                                             #check condition
        rospy.loginfo("detect RED")                               #publish message
        self.red_count += 1                                       #increment counter
        self.green_count = 0
        self.yellow_count = 0
```

```
else:
    self.red_count = 0
    cv_image_mask = self.fnMaskYellowTrafficLight()             #filter using yellow mask
    cv_image_mask = cv2.GaussianBlur(cv_image_mask,(5,5),0)     #Gaussian Blur
    YellowPixel = (19584000-np.sum(cv_image_mask))/256          #Obtain YellowPixel count
    #status3 = self.fnFindCircleOfTrafficLight(cv_image_mask, 'red') #Uncomment this line for debugging (printing detected pixel count)
    if YellowPixel >= 10 and YellowPixel <= 50:                 #check condition
        rospy.loginfo("detect YELLOW")                          #publish message
        self.yellow_count += 1                                  #increment counter
        self.green_count = 0
        self.red_count = 0
```

excluding irrelevant steps

were previously explained

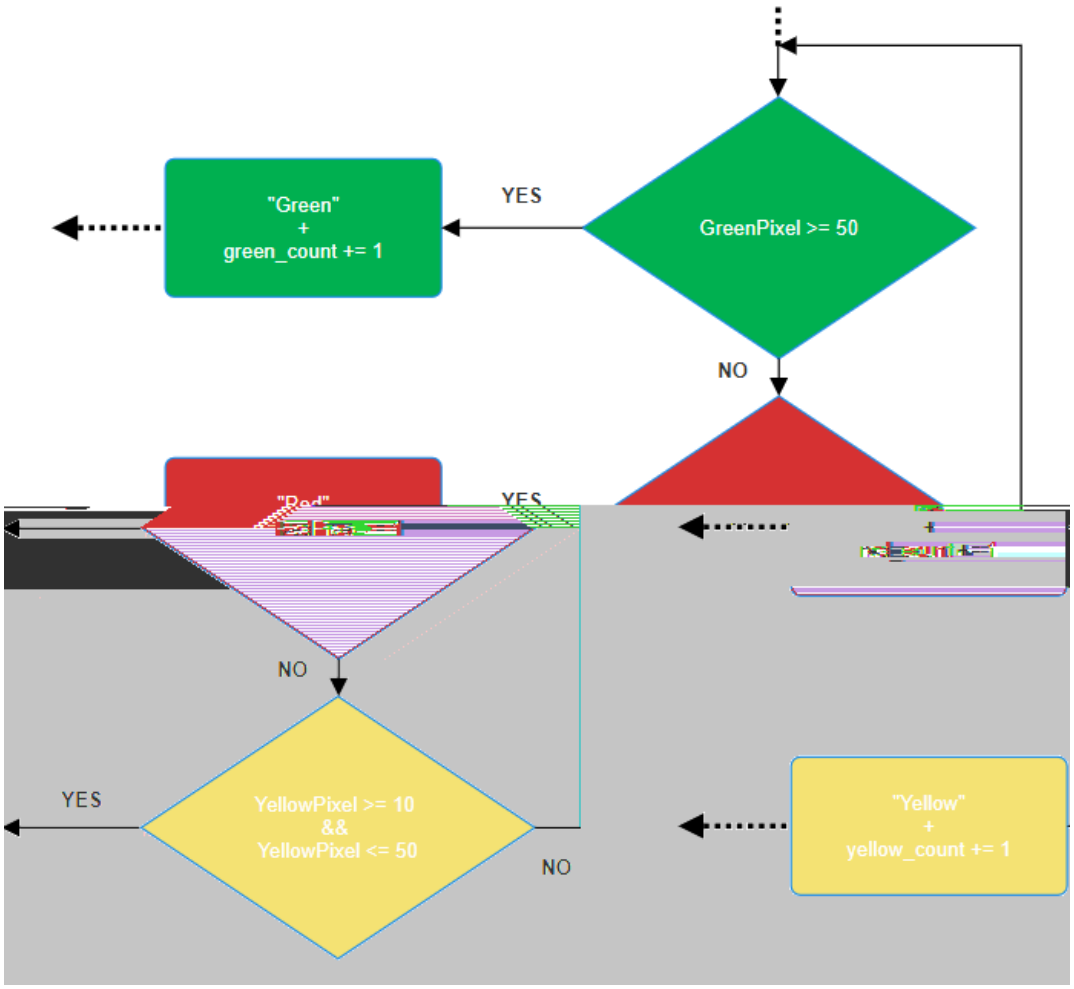

Figure 4.4 Pixel Counter Algorithm Flowchart

4.3 Observation and Results

Significant Improvement:

Successful Detection at Multiple Angles:

Highly Optimized:

Limited Background Effects:

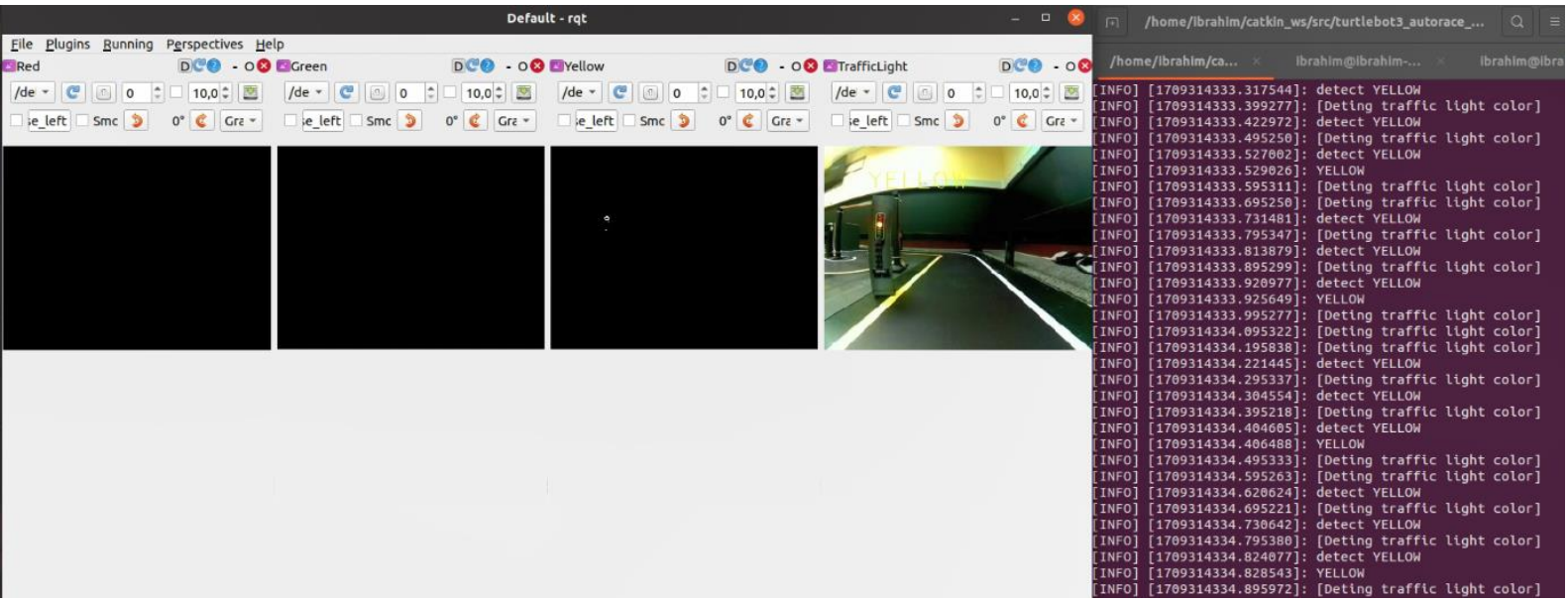

Figure 4.5 Background showing no effect on detection

Drawback of Pixel Counting:

5 Conclusion and Future Thoughts

6 Figure directory

Figure 2.1*:* Applying Gaussian blur using OpenCV 7

Figure 2.2 Effect of Gaussian Blur on Thresholding 8

Figure 2.3 Steps to filter Red color..... 9

Figure 2.4 Isolating Red of an image..... 9

Figure 2.5 Different Blobs in an image..... 10

Figure 2.6 Blob Area Parameter 11

Figure 2.7 Blob Circularity Parameter..... 11

Figure 2.8 Blob Convexity Parameter 11

Figure 2.9 Blob Inertia Parameter 12

Figure 2.10 *SimpleBlobDetector()* Parameters 12

Figure 2.11 Traffic Light Detection Flow Chart 13

Figure 2.12 The three published images after color and noise filteration..... 15

Figure 3.1 Dynamic Callibration for Traffic light detection..... 16

Figure 3.2 Steps to optimize Blob Detection 17

Figure 3.3 Blob detector parameters integrated into the calibration environment..... 20

Figure 3.4 Most accurate Blob detection Parameters..... 21

Figure 4.1 Callibrating Pixel Counter..... 23

Figure 4.2 Optimal Parameters for Pixel Counter 24

Figure 4.3 Pixel Detector Count Table..... 24

Figure 4.4 Pixel Counter Algorithm Flowchart..... 26

Figure 4.5 Background showing no effect on detection 27

7 Refrences

I

I

p.137.

I