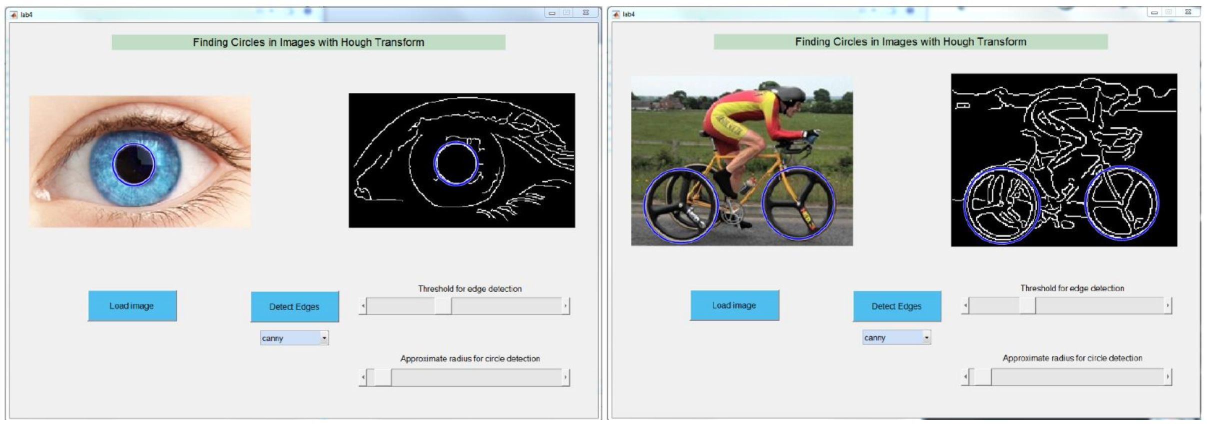
MCTE 7106 MACHINE VISION: QUIZ 2 (20 marks)
Name : Mohammad Najib Bin Abdul Rahim
Matric No: G1923437

Pupillometry is the study of the pupil diameter. It is used in psychology and medicine. A pupil can be well approximated with a circle. Moreover, its darker colour with respect to the iris gives a very strong edge response. As a machine vision programmer, you are required to develop an algorithm which will load an image and detect any number of (relevant) circles in it.


lab4
Finding Circles in Images with Hough Transform
Load image
Detect Edges
canny
Threshold for edge detection
Approximate radius for circle detection
lab4
Finding Circles in Images with Hough Transform
Load image
Detect Edges
canny
Threshold for edge detection
Approximate radius for circle detection
Figure 1: Using circular hough transform to detect (a) left: eye pupil and (b) right: bike's wheels.

Concretely, your task is to perform circular hough transform so that you can detect and localize the significant circle in an image (e.g. pupil in Figure 1(a) and wheels in Figure 1(b)).

1) Using any image on the internet containing eye pupil, write a Python script that reads the image, binarizes it, converts it into an edge map and then performs the circular hough transform to detect the eye pupil. **(10 marks)**
2) Write a comment at each important code line to describe the operation. Make a flowchart to discuss the flow of the algorithm. **(5 marks)**
3) Make a Graphical User Interface (GUI) that will load an image and detect any number of circles in it (bonus marks for designing a slider where you can change the relevant parameters in real-time). Your GUI should look similar to the one shown in Figure 1. **(5 marks)**

ANSWER

Question 1: Refer to GitHub

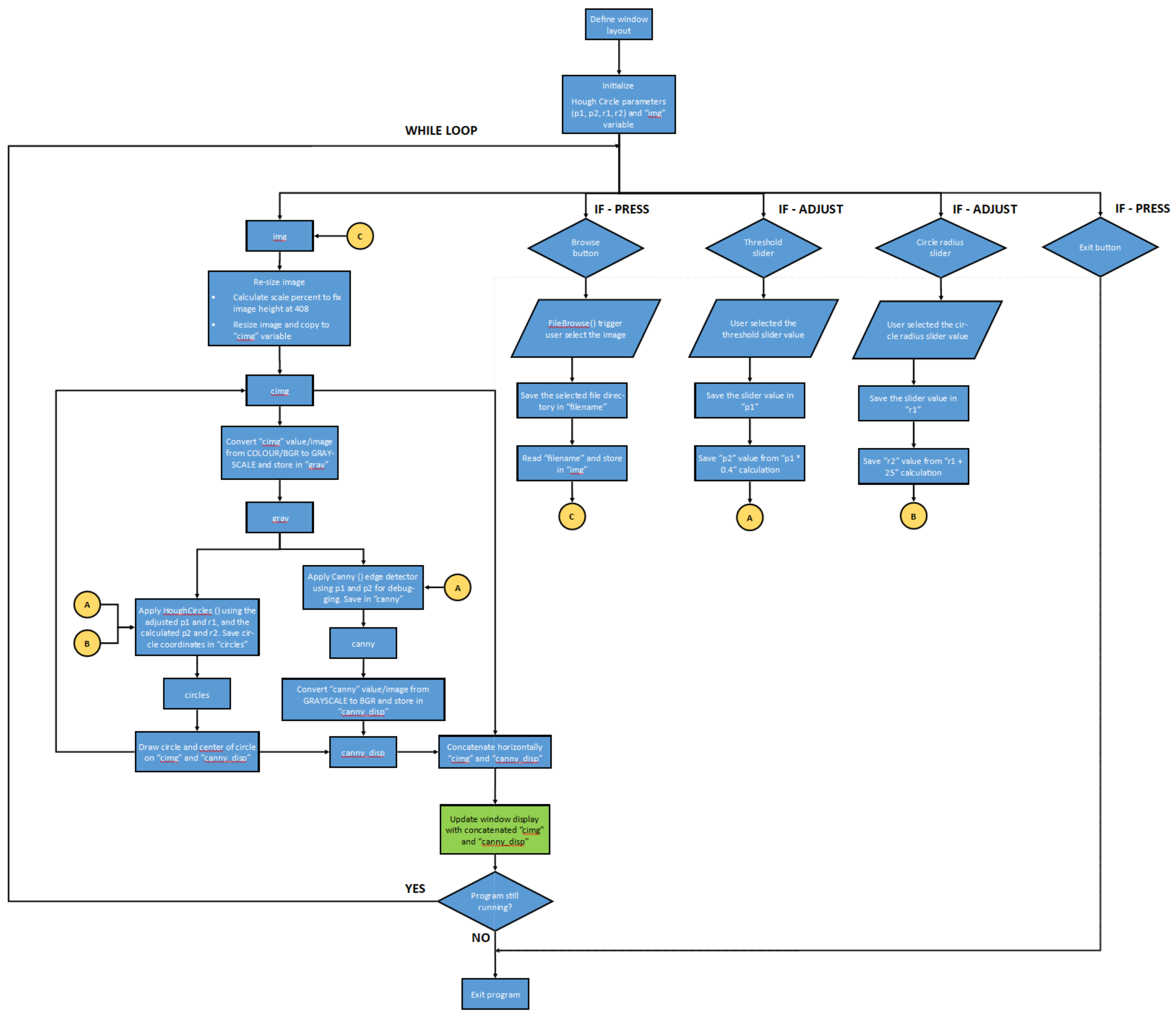
```
import numpy as np
import cv2
import tkinter as tk
from tkinter import *
from tkinter import filedialog
import os
import PIL
from PIL import Image, ImageTk
import PySimpleGUI as sg


#######################################
# Best Configuration by file:                  #
# eye01.jpg: T87, CR43                         #
# eye02.jpg: T90, CR43                         #
# cyclist03.jpg (wheel): T215, CR56            #
# cyclist03.jpg (sprocket): T93, CR11          #
#######################################

def main():
    sg.theme("Purple")

    # Define the window layout
    layout = [
        [sg.Text("Hough Circle Transform", size=(60, 1), justification="center")],
        [sg.Image(filename="", key="-IMAGE-")],
        [sg.Text('Load Image')],
        [sg.Input(key='-FILE-', visible=False, enable_events=True), sg.FileBrowse()],
        [
            sg.Text('Threshold:', size=(15, 0)),
            sg.Slider((0, 255), 87, 1, orientation="h", size=(40, 15), key="-THRESH SLIDER-",),
        ],
        [
            sg.Text('Circle Radius:', size=(15, 0)),
            sg.Slider((0, 255), 43, 1, orientation="h", size=(40, 15), key="-CIRCLE RADIUS-",),
        ],
        [sg.Button("Exit", size=(10, 1))],
    ]

    # Create the window, append the layout
    window = sg.Window("Circle Detection", layout, location=(1000, 400))

    # Store the initial image in "img" variable
    img = cv2.imread("eye01.jpg")

    # Initialize the Hough Circle parameters.
    # p1 = the higher threshold of the two passed to the Canny edge detector (the lower one is twice smaller)
    # r1 = minimum circle radius
    # r2 = maximum circle radius
    p1 = 90
    p2 = p1 * 0.4
    r1 = 43
    r2 = r1 + 25

    while True:
        event, values = window.read(timeout=20)

        if event == "Exit" or event == sg.WIN_CLOSED:
            break

        # Trigger when Browse button is pressed to load image
        # Image file directory is stored in "filename" and stored in "img" for further processing
        if event == '-FILE-':
            filename = values['-FILE-']
            window['-FILE-'].update(filename)
            img = cv2.imread(filename)

        # Store threshold slider value in p1 and p2 (p1*0.4) whenever slider is adjusted
        if values["-THRESH SLIDER-"]:
            p1 = values["-THRESH SLIDER-"]
            p2 = p1 * 0.4

        # Store circle radius slider value in r1 and r2 (r1+25) whenever slider is adjusted
        if values["-CIRCLE RADIUS-"]:
            r1 = int(values["-CIRCLE RADIUS-"])
            r2 = r1 + 25

        # Calculate scale percent to fix image height at 408, resize image, and copy
        scale_percent = 408/img.shape[0]
        width = int(img.shape[1] * scale_percent)
        height = int(img.shape[0] * scale_percent)
        dsize = (width, height)
        img = cv2.resize(img, dsize, interpolation = cv2.INTER_AREA)
        cimg = np.copy(img)

        # Convert from BGR to gray-scale
        gray = cv2.cvtColor(cimg, cv2.COLOR_BGR2GRAY)
        # Detect circles using HoughCircles transform and store position in circles array
        circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, cimg.shape[0]/64,
                                   param1=p1, param2=p2,
                                   minRadius=r1, maxRadius=r2)
        # Edge image using Canny edge detector for debugging
        canny = cv2.Canny(gray, p1, p2)
        canny_disp = cv2.cvtColor(canny, cv2.COLOR_GRAY2BGR)


        # If at least 1 circle is detected
        if circles is not None:
            cir_len = circles.shape[1] # store length of circles found
            circles = np.uint16(np.around(circles))
            for i in circles[0, :]:
                # Coloured image - Draw the outer circle
                cv2.circle(cimg, (i[0], i[1]), i[2], (0, 255, 0), 2)
                # Coloured image - Draw the center of the circle
                cv2.circle(cimg, (i[0], i[1]), 2, (0, 0, 255), 3)

                # Canny Edge - Draw the outer circle
                cv2.circle(canny_disp,(i[0],i[1]),i[2],(0,255,0),2)  #img,center,radius,color[, thickness[, lineType[, shift]]]
                # Canny Edge - Draw the center of the circle
                cv2.circle(canny_disp,(i[0],i[1]),2,(0,255,0),3)

        else:
            cir_len = 0 # no circles detected

        # Display image
        img_concat = np.hstack((cimg, canny_disp))
        imgbytes = cv2.imencode(".png", img_concat)[1].tobytes()
        window["-IMAGE-"].update(data=imgbytes)


    window.close()

main()
```

Question 2:

(a) Code comments - Refer to Question 1 and GitHub

(b) Algorithm Flowchart

The Hough Circle detection with load image, threshold adjustment, and circle radius adjustment features are summarized in the following flow chart

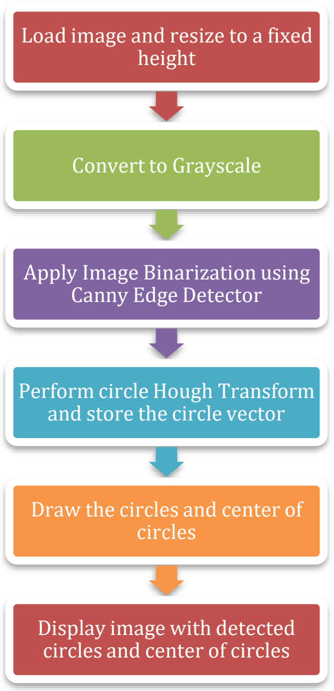


Define window layout
Initialize
Hough Circle parameters (p1, p2, r1, r2) and "img" variable
WHILE LOOP
IF - PRESS
Browse button
FileBrowse() trigger user select the image
Save the selected file directory in "filename"
Read "filename" and store in "img"
C
IF - ADJUST
Threshold slider
User selected the threshold slider value
Save the slider value in "p1"
Save "p2" value from "p1 * 0.4" calculation
A
IF - ADJUST
Circle radius slider
User selected the circle radius slider value
Save the slider value in "r1"
Save "r2" value from "r1 + 25" calculation
B
IF - PRESS
Exit button
img
C
Re-size image
Calculate scale percent to fix image height at 408
Resize image and copy to "cimg" variable
cimg
Convert "cimg" value/image from COLOUR/BGR to GRAYSCALE and store in "gray"
gray
Apply Canny () edge detector using p1 and p2 for debugging. Save in "canny"
A
canny
Convert "canny" value/image from GRAYSCALE to BGR and store in "canny_disp"
A
B
Apply HoughCircles () using the adjusted p1 and r1, and the calculated p2 and r2. Save circle coordinates in "circles"
circles
Draw circle and center of circle on "cimg" and "canny_disp"
canny_disp
Concatenate horizontally "cimg" and "canny_disp"
Update window display with concatenated "cimg" and "canny_disp"
YES
Program still running?
NO
Exit program

After defining the window layout and initializing the Hough Circle parameters and image variable, the loaded image is processed as follow:

1) Resize to a fixed height. This is to ensure any images loaded into the program can be properly processed and displayed
2) Convert the colour image to grayscale
3) Apply image binarization using Canny Edge Detector. For HoughCircle() function, canny detector is built-in. Suitable param1 or p1 (as declared in this program) need to be identified using the slider to reduce or eliminate false circles.
4) Perform hough circle transform to detect the true center point which is common to all parameter circles, and can be found with a Hough accumulation array. The output vector of found circles are stored in circles array. Each vector is encoded as a 3-element floating-point vector (x, y, radius).
5) Draw the circles and center of circles

6) Display image with detected circles and center of circles. The displayed image comprises of the original image and binarized image after applying Canny operation.


Load image and resize to a fixed height
Convert to Grayscale
Apply Image Binarization using Canny Edge Detector
Perform circle Hough Transform and store the circle vector
Draw the circles and center of circles
Display image with detected circles and center of circles

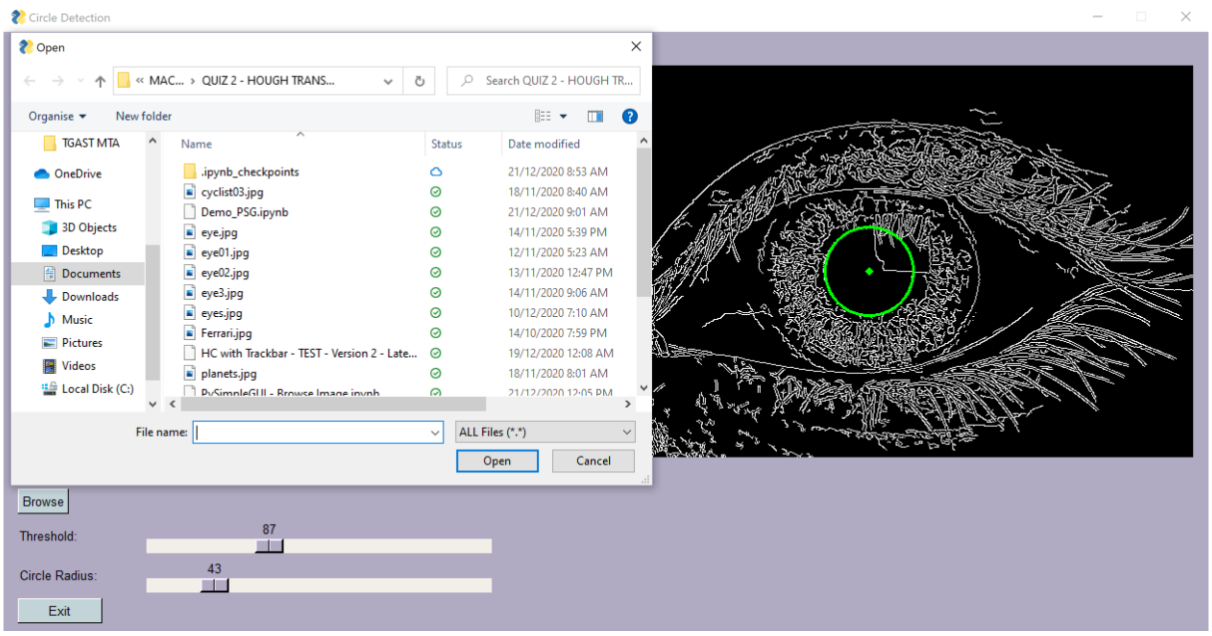
The core algorithm of the Hough Circle is in the HoughCircle() function which accepts the following parameters:

- **image:** 8-bit, single-channel, grayscale input image.
- **circles:** Output vector of found circles. Each vector is encoded as a 3-element floating-point vector (x, y, radius) .
- **circle_storage:** In C function this is a memory storage that will contain the output sequence of found circles.
- **method:** Detection method to use. Currently, the only implemented method is CV_HOUGH_GRADIENT , which is basically 21HT
- **dp:** Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height.
- **minDist:** Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
- **param1:** First method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the higher threshold of the two passed to the Canny() edge detector (the lower one is twice smaller).
- **param2:** Second method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
- **minRadius:** Minimum circle radius.
- **maxRadius:** Maximum circle radius.

Question 3:

Graphical User Interface (GUI) is created to load any images and perform adjustment on canny thresholding parameter and circle radius. Following are the final results of the GUI and circle detections with optimum Threshold and Circle Radius:

(a) Load Image feature

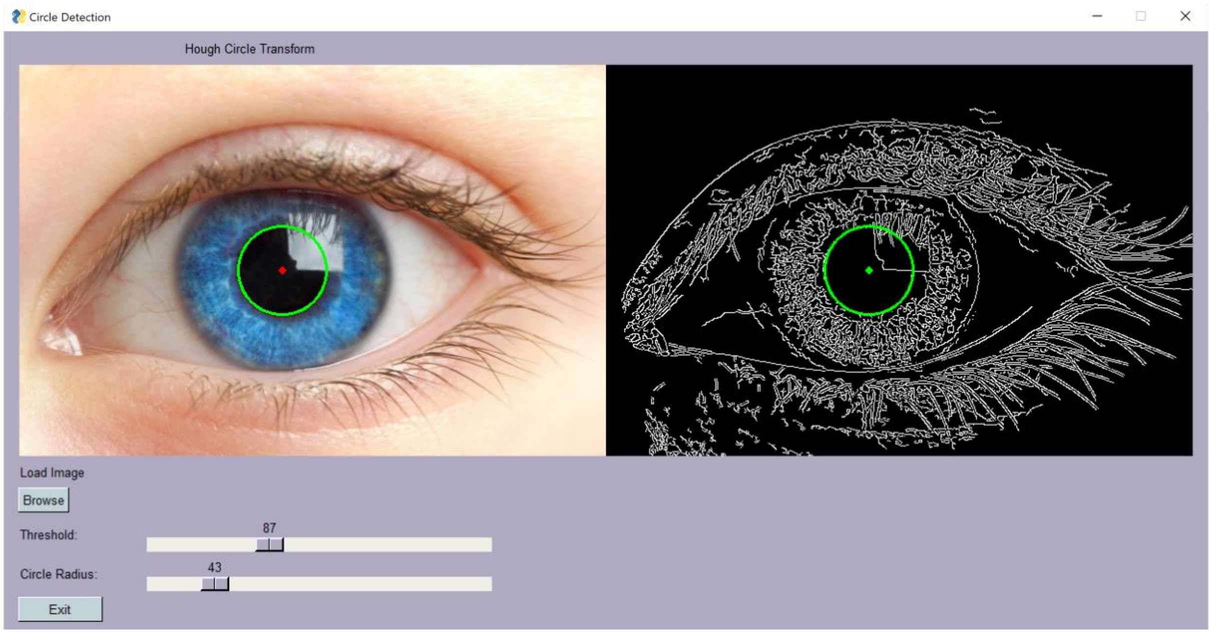


Name	Status	Date modified
.ipynb_checkpoints		21/12/2020 8:53 AM
cyclist03.jpg		18/11/2020 8:40 AM
Demo_PSG.ipynb		21/12/2020 9:01 AM
eye.jpg		14/11/2020 5:39 PM
eye01.jpg		12/11/2020 5:23 AM
eye02.jpg		13/11/2020 12:47 PM
eye3.jpg		14/11/2020 9:06 AM
eyes.jpg		10/12/2020 7:10 AM
Ferrari.jpg		14/10/2020 7:59 PM
HC with Trackbar - TEST - Version 2 - Late...		19/12/2020 12:08 AM
planets.jpg		18/11/2020 8:01 AM
PySimpleGUI - Browse Image.ipynb		21/12/2020 12:05 PM

Threshold: 87

Circle Radius: 43

(b) Eye pupil #1: Threshold 87, Circle Radius 43

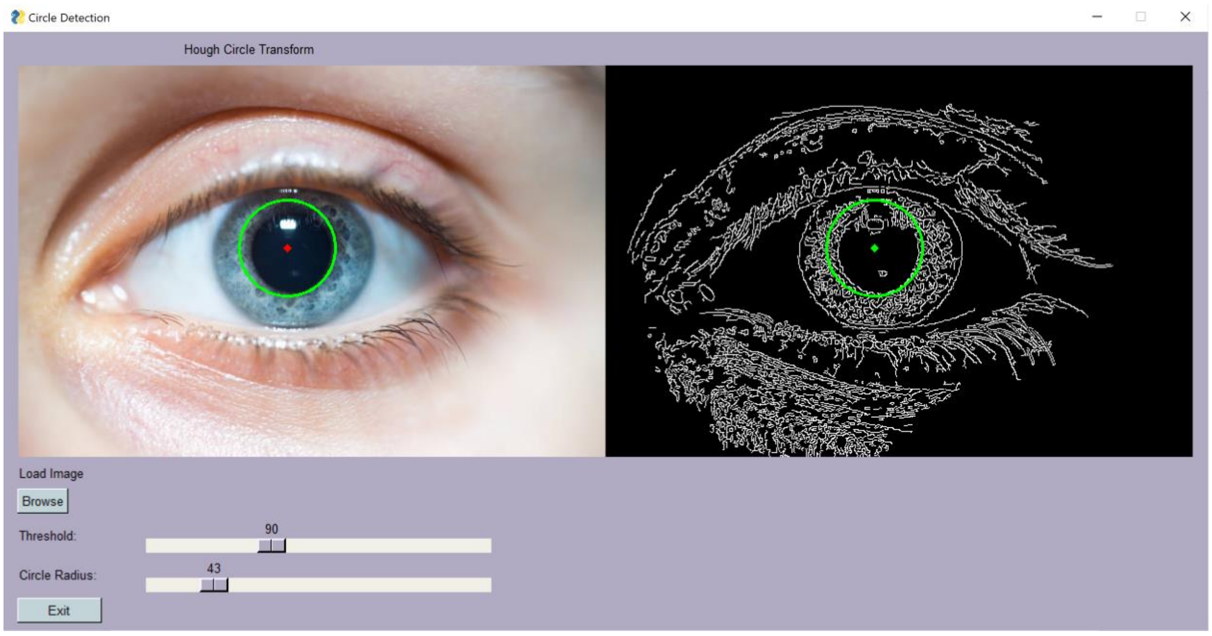


Threshold: 87

Circle Radius: 43

Open

(c) Eye pupil #2: Threshold 90, Circle Radius 43


Circle Detection

Hough Circle Transform

Load Image

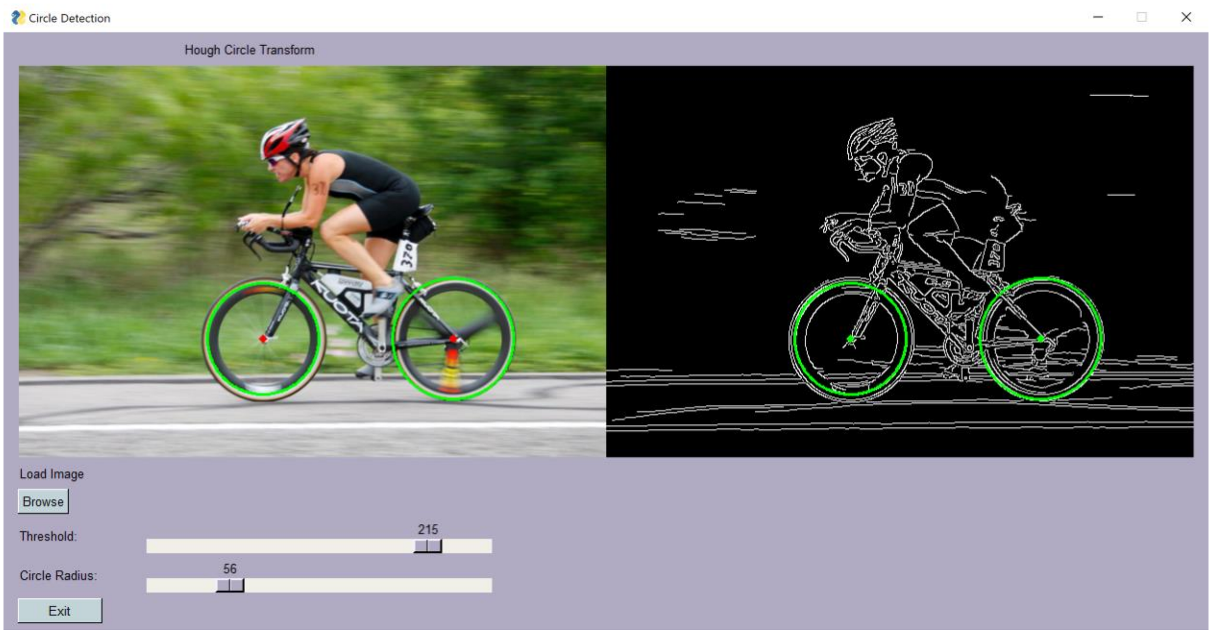
Browse

Threshold: 90

Circle Radius: 43

Exit

(d) Cyclist – to detect wheel: Threshold 215, Circle Radius 56


Circle Detection

Hough Circle Transform

Load Image

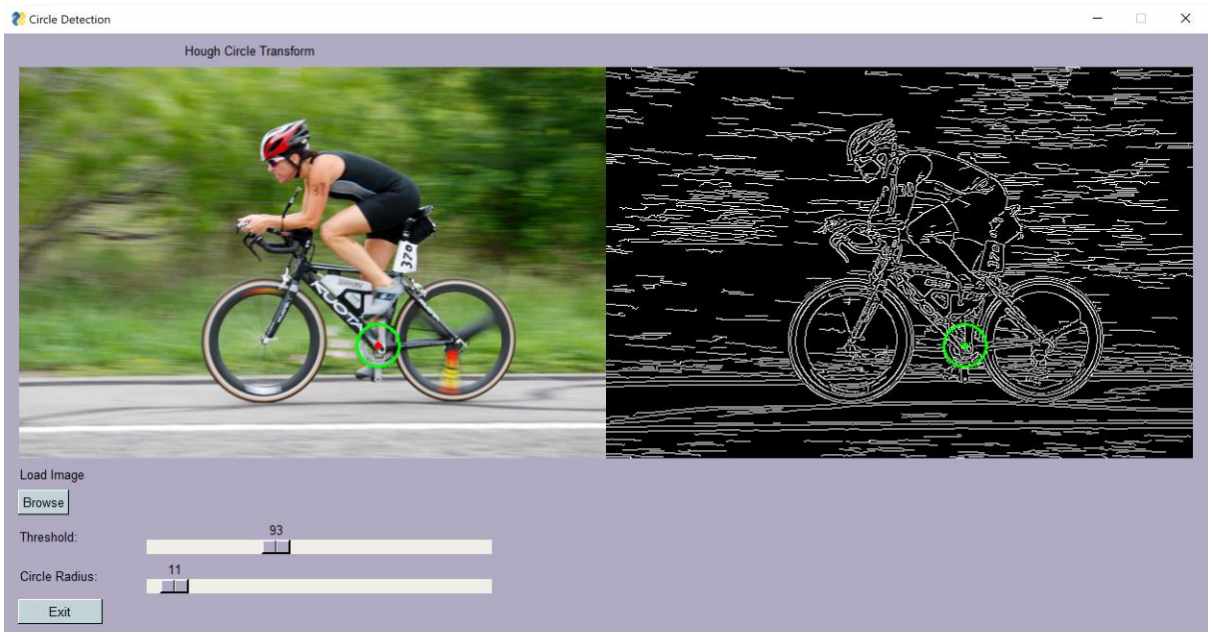
Browse

Threshold: 215

Circle Radius: 56

Exit

(e) Cyclist – to detect sprocket: Threshold 93, Circle Radius 11



Threshold: 93

Circle Radius: 11