Prelim Exam (Hands-on Exam)	
Name: Bernardo, Christian Emmanuel	**Date Performed: 19/02/2024**
Course/Section: CPE018 CPE31S1	**Date Submitted: 19/02/2024**
Instructor: Dr. Jonathan Taylar	**Semester and SY: 2nd 2023-2024**

```
import cv2
import numpy
import time

class CaptureManager(object):
    def __init__(self, capture, previewWindowManager=None, shouldMirrorPreview=False):
        self.previewWindowManager = previewWindowManager
        self.shouldMirrorPreview = shouldMirrorPreview

        self._capture = capture
        self._channel = 0
        self._enteredFrame = False
        self._frame = None
        self._imageFilename = None
        self._videoFilename = None
        self._videoEncoding = None
        self._videoWriter = None
        self._startTime = None
        self._framesElapsed = 0
        self._fpsEstimate = None
```

```
    @property
    def channel(self):
        return self._channel

    @channel.setter
    def channel(self, value):
        if self._channel != value:
            self._channel = value
            self._frame = None

    @property
    def frame(self):
        if self._enteredFrame and self._frame is None:
            _, self._frame = self._capture.retrieve()
        return self._frame

    @property
    def isWritingImage(self):
        return self._imageFilename is not None

    @property
    def isWritingVideo(self):
        return self._videoFilename is not None
```

Analysis:

- Is it necessary to use object-oriented approach in creating the application interface for our project
    - Yes it is because the CaptureManager class is needed to make sure that the image and videos are properly saved in the folder
- Why are certain attributes of the class defined with an underscore in the name? What does it do?
    - They are named internal implementation details. They are a naming convention to denote that these attributes are part of the class's implementation and not part of its public interface
- Explain the use of @property and other confusing parts of this code. Refer to the appropriate Python documentation.
    - They are a decorator in Python used to create read-only properties. The @property decorator is applied to the channel method, turning it into a read-only property. The @property decorator is also used with the frame, isWritingImage, and isWritingVideo methods in creating a read-only properties for these attributes.

```
def enterFrame(self):
    """Capture the next frame, if any."""
    assert not self._enteredFrame, 'previous enterFrame() had no matching exitFrame()'
    if self._capture is not None:
        self._enteredFrame = self._capture.grab()

def exitFrame(self):
    """Draw to the window. Write to files. Release the frame."""
    if self.frame is None:
        self._enteredFrame = False
        return

    if self._framesElapsed == 0:
        self._startTime = time.time()
    else:
        timeElapsed = time.time() - self._startTime
        self._fpsEstimate = self._framesElapsed / timeElapsed
    self._framesElapsed += 1

    if self.previewWindowManager is not None:
        if self.shouldMirrorPreview:
            mirroredFrame = numpy.fliplr(self._frame).copy()
            self.previewWindowManager.show(mirroredFrame)
        else:
            self.previewWindowManager.show(self._frame)

    if self.isWritingImage:
        cv2.imwrite(self._imageFilename, self._frame)
        self._imageFilename = None

    self._writeVideoFrame()

    self._frame = None
    self._enteredFrame = False
```

```
def writeImage(self, filename):
    """Write the next exited frame to an image file."""
    self._imageFilename = filename

def startWritingVideo(self, filename, encoding=cv2.VideoWriter_fourcc('I', '4', '2', '0')):
    """Start writing exited frames to a video file."""
    self._videoFilename = filename
    self._videoEncoding = encoding

def stopWritingVideo(self):
    """Stop writing exited frames to a video file."""
    self._videoFilename = None
    self._videoEncoding = None
    self._videoWriter = None

def _writeVideoFrame(self):
    if not self.isWritingVideo:
        return

    if self._videoWriter is None:
        fps = self._capture.get(cv2.CAP_PROP_FPS)
        if fps == 0.0:
            if self._framesElapsed < 20:
                return
            else:
                fps = self._fpsEstimate

        size = (int(self._capture.get(cv2.CAP_PROP_FRAME_WIDTH)),
                int(self._capture.get(cv2.CAP_PROP_FRAME_HEIGHT)))

        self._videoWriter = cv2.VideoWriter(self._videoFilename,
                                            self._videoEncoding, fps, size)
    self._videoWriter.write(self._frame)
```

Analysis:

- What are these additional functions' intended functions? Provide a quick discussion on how they work by analyzing the parameters passed to the functions.

  The additional functions in the CaptureManager class serve various purposes related to video frame capture, display, and storage.

  enteredFrame(self):
  Intended Functionality: This function is used to indicate that a new frame is about to be captured. It asserts that the function has not been called before for the current frame. It then attempts to grab a frame from the capture source (e.g., camera).
  Parameters:
  self: The instance of the CaptureManager class.
  exitFrame(self):
  Intended Functionality: This function is called when the processing of the current frame is complete. It calculates the frames per second (FPS) estimate, updates the frame count, displays the frame if a preview window manager is provided, and writes the frame to an image or video file if enabled.
  Parameters:
  self: The instance of the CaptureManager class.

writeImage(self, filename):
Intended Functionality: Sets the filename for saving the next captured frame as an image.
Parameters:
self: The instance of the CaptureManager class.
filename: The name of the image file to be written.
startWritingVideo(self, filename, encoding=cv2.VideoWriter_fourcc('I', '4', '2', '0')):
Intended Functionality: Sets the filename and video encoding for saving the next frames as a video. The default encoding is set to 'I420' (YUV planar format).
Parameters:
self: The instance of the CaptureManager class.
filename: The name of the video file to be written.
encoding: The encoding format for the video (default is 'I420').
stopWritingVideo(self):
Intended Functionality: Stops writing video frames. Resets video-related attributes.
Parameters:
self: The instance of the CaptureManager class.
_writeVideoFrame(self):
Intended Functionality: Writes the current frame to the video file if video writing is enabled. It initializes the video writer if it hasn't been created yet.
Parameters:
self: The instance of the CaptureManager class.
These functions work together to manage the capture, processing, and storage of video frames, providing a higher-level interface for interacting with the OpenCV capture functionality.

```python
class WindowManager(object):
    def __init__(self, windowName, keypressCallback=None):
        self.keypressCallback = keypressCallback
        self._windowName = windowName
        self._isWindowCreated = False

    @property
    def isWindowCreated(self):
        return self._isWindowCreated

    def createWindow(self):
        cv2.namedWindow(self._windowName)
        self._isWindowCreated = True

    def show(self, frame):
        cv2.imshow(self._windowName, frame)

    def destroyWindow(self):
        cv2.destroyWindow(self._windowName)
        self._isWindowCreated = False

    def processEvents(self):
        keycode = cv2.waitKey(1)
        if self.keypressCallback is not None and keycode != -1:
            keycode &= 0xFF
            self.keypressCallback(keycode)
```

Analysis:

- What are the identifiable drawbacks for this implementation?
  Lack of error handling, limited configurability and limited support for multiple windows

- Based on those identifiable drawbacks, what solutions can you implement to solve them ?
  Try using try-except blocks to gracefully handle errors and provide meaningful messages, Adding parameters to the createWindow method to allow users to configure window properties and If multiple windows are required, you could modify the class to handle a list of window names, and adapt the methods accordingly.

```
class Cameo(object):
    def __init__(self):
        self._windowManager = WindowManager('Cameo', self.onkeypress)
        self._captureManager = CaptureManager(cv2.VideoCapture(0), self._windowManager,

    def run(self):
        """Run the main loop."""
        self._windowManager.createWindow()
        while self._windowManager.isWindowCreated:
            self._captureManager.enterFrame()
            frame = self._captureManager.frame
            # TODO: Filter the frame
            self._captureManager.exitFrame()
            self._windowManager.processEvents()

    def onkeypress(self, keycode):
        """Handle a keypress.
        space -> Take a screenshot.
        tab -> Start/stop recording a screencast.
        escape -> Quit.
        """
        if keycode == 32:  # space
            self._captureManager.writeImage('screenshot.png')
        elif keycode == 9:  # tab
            if not self._captureManager.isWritingVideo:
                self._captureManager.startWritingVideo('screencast.avi')
            else:
                self._captureManager.stopWritingVideo()
        elif keycode == 27:  # escape
            self._windowManager.destroyWindow()

if __name__ == "__main__":
    Cameo().run()
```

Implementation:

- Run the Application; make sure that you have live camera feed

- What happens when you take a screenshot
    - It makes a png image in the folder
- Change the shouldMirrorPreview in the initialization of the CaptureManager class to false, take a screenshot again and show what happened

Supplementary Activity

This supplementary activity assumes that you have fully complied with the previous activities for image processing (including all functions created in the previous module that was tested on GrumpyCat -- and eventually your own images).

Part 1: Modify your cameo.py such that the following commented lines should appear in your own code.

```python
import cv2
#import Filters
from managers import WindowManager, CaptureManager

class Cameo(object):
    def __init__(self):
        self._windowManager = WindowManager('Cameo', self.onKeypress)
        self._captureManager = CaptureManager(cv2.VideoCapture(0), self._windowManager,

    def run(self):
        self._windowManager.createWindow()
        while self._windowManager.isWindowCreated:
            self._captureManager.enteredFrame()
            frame = self._captureManager.frame

            #self._curveFilter = filtersBGRPotraCurveFilter()

            self._captureManager.exitFrame()
            self._windowManager.processEvents()

    def run(self):
        self._windowManager.createWindow()
        while self._windowManager.createWindow():
            self._windowManager.isWindowCreated:
            frame = self._captureManager.frame
            #filters.strokeEdges(frame, frame)
            #self._curveFilter.apply(frame, frame)
            self._captureManager.exitFrame()
            self._windowManager.processEvents()
```

Detail the necessary steps taken to apply all the filters previously created to the live video feed captured for the supplementary activity. Show screenshot for each.

Part 2: Implement canny edge detection and contour detection to the live video feed. Make sure that the necessary functions/packages/modules are used. Provide an extensive analysis of your output supported by screenshots.

```
    def applyCannyEdgeDetection(self):
        if self._frame is not None:
            gray_frame = cv2.cvtColor(self._frame, cv2.COLOR_BGR2GRAY)
            edges = cv2.Canny(gray_frame, 50, 150)
            self._frame = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

class EdgeDetectionManager(object):
    def __init__(self, captureManager):
        self._captureManager = captureManager

    def processFrame(self):
        self._captureManager.enteredFrame()
        frame = self._captureManager.frame

        # Apply edge detection
        self._captureManager.applyCannyEdgeDetection()

        self._captureManager.exitFrame()
```

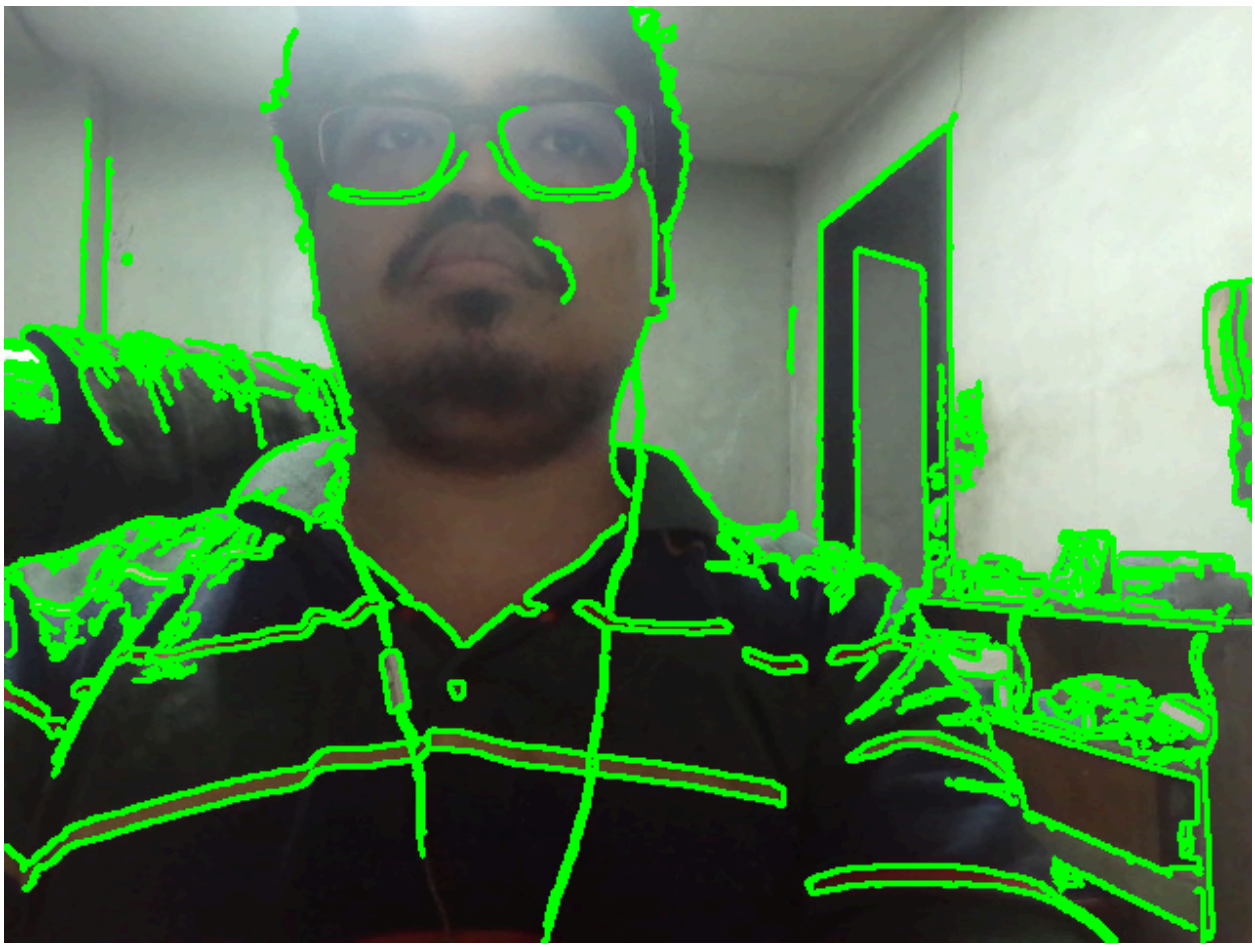
This part of the code in the manager file makes a def in the capture manager class to find any grays colors while the Class edge detection detects the edges from the frames in real time

```
def main():
    videoCapture = cv2.VideoCapture(0)  # Use the appropriate camera index or video file path
    captureManager = CaptureManager(videoCapture)
    windowManager = WindowManager('Live Video', keypressCallback=handleKey)

    edgeDetectionManager = EdgeDetectionManager(captureManager)

    windowManager.createWindow()

    while windowManager.isWindowCreated:
        captureManager.enteredFrame()
        frame = captureManager.frame

        # Process frame for edge detection
        edgeDetectionManager.processFrame()

        if frame is not None:
            windowManager.show(frame)

        windowManager.processEvents()

    windowManager.destroyWindow()
    videoCapture.release()

def handleKey(keycode):
    if keycode == 27:  # ESC key
        cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

This def main is for video capture the edges in real time from the def and class for it to actually work

```
def run(self):
    self._windowManager.createWindow()
    while self._windowManager.isWindowCreated:
        self._captureManager.enteredFrame()
        frame = self._captureManager.frame

        # Apply Canny edge detection
        edges = cv2.Canny(frame, 50, 150)

        # Find contours in the edges
        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Draw contours on the original frame
        cv2.drawContours(frame, contours, -1, (0, 255, 0), 2)

        self._captureManager.exitFrame()
        self._windowManager.processEvents()
```

This is to fully implement the classes and def for the edge detection to make it works properly



This is the final result

Part 3: Implement circle detection to your code. What should happen now is that whenever a flat circular object is held up to your camera, it should draw the circle on the image where the object is found. Show screenshots to support this outcome.

Conclusion:

In this exam I was very confused on a lot of things like how my webcam wasn't working but it was just that my code was wrong in some places but I was able to fix all of this in time. Making the windows for the webcam to take screenshots and videos was a nice thing to experience in making because I think this will be the foundation for future activities in scanning people and things to see what they are using shape and line detections to predict what items they are.

In the lessons I learned, I relearned how to create gui windows but more effectively and easier but for cameras in scanning objects. I also learned more effective ways in making detection for cameras and understanding how security cameras work because of the movements of objects and how they can be coded to record movement.