GitHub Repository Link

https://github.coventry.ac.uk/5062CEM/STUDENTID_IPA

recogniser.py

```
"""
The Object Recogniser class. This class will consist of the methods that will be used to
detect an object from a given frame.

For example, we will use my staff identification card as a training image to be detected in
the camera whenever it is shown.
"""
import time

import cv2
import numpy


class Recogniser:
    """
    The class initializer is used to set the feature detector of the object recognition class.
        There
    are two types of detectors used:

        * BRISK - Provide an explanation of BRISK
        * SURF -  Provide an explanation of SURF
    """

    def __init__(self, detector):
        if detector == "BRISK":
            # The default parameters for BRISK are used, i.e. 30, 3, 1.0. These can be changed
            # accordingly to increase (or decrease) the performance of the feature detector.
            self.feature_detector = cv2.BRISK_create(30, 3, 1.0)
        elif detector == "SURF":
            # The default parameters for SURF are used, i.e. 100, 4, 3, FALSE and FALSE. These
                can
            # be changed accordingly to increase (or decrease) the performance of the feature
                detector.
            self.feature_detector = cv2.xfeatures2d.SURF_create(100, 4, 3, False, False)
        else:
            # If there was no feature detector set, then it will raise an exception and the
                application
            # will no longer continue.
            raise Exception("No Feature Detector")

    """
    This function is used to detect the key areas of interest, otherwise known as keypoints,
        form an
    image or a frame grabbed from the camera or video.
    """

    def get_keypoints(self, _img):
        tmp_keypoints = self.feature_detector.detect(_img)
        return tmp_keypoints

    """
    This function is used to extract the descriptor from an image, or frame grabbed from the
        camera
    or video. It will use the key areas of interest, otherwise known as keypoints, to form this
```

```
    descriptor.
    """

    def get_descriptor(self, _img, _keypoints):
        tmp_keypoints, tmp_descriptor = self.feature_detector.compute(_img, _keypoints)
        return tmp_keypoints, tmp_descriptor

    """
    This function will match the descriptors that were extracted from the image, an a frame
        grabbed
    from the camera or video. Depending upon the type of feature detector used, there may be
        two types
    of descriptors extracted. This means that their may be two types of 'matchers', BruteForce
        for those
    that are an 8-bit integer, or Flann for those that are 32-bit floats.

    The matching process uses the k-nearest-neighbour algorithm, with k set as 2. These
        initial matches
    are then refined using a ratio test (set to 0.7) to filter out the 'good matches'. These
        matches are
    then used to determine whether the object has been found in the image or frame grabbed
        from the camera
    or video.
    """

    @staticmethod
    def match(_d1, _d2):
        matcher = None
        if _d1.dtype == "uint8":
            # BRISK
            matcher = cv2.DescriptorMatcher.create(cv2.DescriptorMatcher_BRUTEFORCE)
        elif _d1.dtype == "float32":
            # SURF
            matcher = cv2.DescriptorMatcher.create(cv2.DescriptorMatcher_FLANNBASED)

        knn_matches = matcher.knnMatch(_d1, _d2, 2)
        ratio_thresh = 0.7
        good_matches = []
        for i in range(len(knn_matches)):
            if knn_matches[i][0].distance < ratio_thresh * knn_matches[i][1].distance:
                good_matches.append(knn_matches[i][0])
        return good_matches

    """
    This function will calculate the FPS in which the camera, or video file is running at.
        This is purely
    for debugging purposes only; and ensures that I can see my object recogniser is running in
        a real-time
    constraint. For example, if my camera feed is 30FPS, then I expect it to continue running
        at 30FPS.
    """

    @staticmethod
    def calculate_fps(_frame_number, _time):
        return int(_frame_number // (time.time() - _time))
```

```
    """
    This function will detect the object from the frame grabbed from the camera or the video.
    """

    @staticmethod
    def detect_object(_training_image, _training_keypoints, _frame_keypoints,
        _filtered_matches):
        tmp_object = numpy.float32(
            [_training_keypoints[m.queryIdx].pt for m in _filtered_matches]
        )

        tmp_frame = numpy.float32(
            [_frame_keypoints[m.trainIdx].pt for m in _filtered_matches]
        )

        try:
            homography, mask = cv2.findHomography(tmp_object, tmp_frame, cv2.RANSAC, 3)
        except cv2.error:
            return None

        height, width = _training_image.shape[:2]
        boundaries = numpy.float32([
            [0, 0],
            [0, height],
            [width, height],
            [width, 0]
        ]).reshape(-1, 1, 2)

        try:
            boundaries = numpy.int32(cv2.perspectiveTransform(boundaries, homography))
            x, y = 0, 0
            for a in boundaries:
                for b in a:
                    x += b[0]
                    y += b[1]
        except cv2.error as e:
            print(Exception(e))
            return None
        except UnboundLocalError:
            print(Exception(UnboundLocalError.__str__))
            return None
        return boundaries
```

main.py

```
import time
import cv2
from recogniser import Recogniser

# Create an instance of the class using the SURF feature detector
obj_rec = Recogniser("SURF")

# Load the training image that we want to use to detect
training_img = cv2.imread("img.png", cv2.IMREAD_REDUCED_COLOR_2)

# Grab some key areas of interest from the training image
training_keypoints = obj_rec.get_keypoints(training_img)

# Extract a descriptor from the training image using the key areas of interest
training_keypoints, training_descriptor = obj_rec.get_descriptor(training_img,
    training_keypoints)

# Set the video capture method to use our in-built webcam
cap = cv2.VideoCapture(0)

# Check whether we have opened the camera or not.
if cap.isOpened() is False:
    raise Exception("The camera is already open.")

# Change the camera settings to 1280x720 (720p) resolution
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

# Get the time when we begun the application
start_time = time.time()
frame_number = 0

# Use an infinite loop to grab frames from the webcam, we can break this later on
while True:
    # Read a frame from the camera, and a return value on whether it is grabbing a frame
    ret, frame = cap.read()
    # If the frame is none (i.e. empty) then we can throw an error
    if frame is None:
        raise Exception("Error reading from the camera.")

    # Find key areas of interest from the frame of the camera
    keypoints = obj_rec.get_keypoints(frame)
    # Generate a descriptor from the key areas of interest
    keypoints, descriptor = obj_rec.get_descriptor(frame, keypoints)
    # Perform a match between the descriptor of the training image and the frame to determine
        if
    # the object can be found.
    matches = obj_rec.match(training_descriptor, descriptor)

    # Get the detected boundaries of the object
    detected_boundaries = obj_rec.detect_object(training_img, training_keypoints, keypoints,
        matches)

    # Draw the detected boundaries onto the frame
```

```
cv2.polylines(frame, [detected_boundaries], True, (0, 255, 0), 1, cv2.LINE_AA, 0)

# Put the FPS in the top-left corner of the image
frame_number += 1
cv2.putText(frame, str(obj_rec.calculate_fps(frame_number, start_time)), (0, 15),
        cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1, cv2.LINE_AA)

# Displays just the camera, with a bounding box around the detected image.
cv2.imshow("Window", frame)

# Sets a wait key for one second, and listens for ESC key to break the while loop
if cv2.waitKey(1) == 27:
    # Releases the camera when the while loop has ended
    cap.release()
    # Destroys any windows that were created
    cv2.destroyAllWindows()
    # Now lets break
    break
```