

VideoStream

usbwebcamera.py

```
import os
import sys
import time
import math
import getopt
import numpy as np
import cv2
import threading
import subprocess
from collections import deque

from lock_manager import Lock_Manager
from util import Util

class UsbWebCamera(threading.Thread):
    def __init__(self, video_source=1, source=None, do_record=True,
do_display=True, do_add_contours=True, do_add_target=False):
        threading.Thread.__init__(self)

        self.name = self.__class__.__name__
        self.archive =
os.path.join(os.path.dirname(os.path.realpath(__file__)), 'archive')

        self.writer = None
        self.current_frame = None

        self.codec = cv2.VideoWriter_fourcc('M','J', 'P', 'G')
        self.OBSERVER_LENGTH = 5 # Time in seconds to be observed for motion
        self.threshold = 15

        self.CAMERA_SOURCE = video_source
        self.REMAIN_RECORDING_FILES = 3 # 10이상 부터 삭제 후 저장
        self.do_display = do_display
        self.do_record = do_record
        self.do_add_contours = do_add_contours
        self.do_add_target = do_add_target
        self.current_file = None
        self.source = cv2.VideoCapture(source) if source is not None else
self.init_camera()

        self.fps = self.find_fps(self.source)
        self.height, self.width = self.get_dimensions(self.source)
        Util.log(self.name, "Initializing usb camera class with video_source=")
```

```

+ str(self.CAMERA_SOURCE))
        Util.log(self.name, "width: {" + str(self.width) + "}, height :
{" + str(self.height) + "}")

        self.lock_manager = Lock_Manager("motion")

def __del__(self):
    # Release camera
    self.source.release()

    # Close all windows
    cv2.destroyAllWindows()

    # Remove lock if exists
    self.lock_manager.remove()

def get_frame(self):
    """
    Return the current frame

    @return bytes
    """
    return self.frame_to_jpg(self.current_frame) if self.current_frame is
not None else None

def frame_to_jpg(self, frame):
    """
    Convert video frame to jpg

    @param array frame
    @return bytes
    """
    ret, jpeg = cv2.imencode('.jpg', self.current_frame)
    return jpeg.tobytes()

def get_dimensions(self, source):
    """
    Determine height and width of the video source

    @return tuple(int, int)
    """
    frame = cv2.cvtColor(source.read()[1], cv2.COLOR_RGB2GRAY)
    return frame.shape[0: 2]

def find_fps(self, source):
    """
    Determine frames per second of the video source

    @param video source
    @return int

```

```

"""
Util.log(self.name, "Determining FPS...")

# How many frames to capture
num_frames = 120

# Start time
start = time.time()

# Grab a few frames
for i in range(0, num_frames):
    ret, frame = source.read()

# End time
end = time.time()

# Calculate frames per second
fps = int(math.floor(num_frames / (end - start)))
Util.log(self.name, "Setting FPS to " + str(fps))

return fps

def init_camera(self):
    """
    Start the camera

    @return cv2.VideoCapture
    """
    # Init camera
    camera = cv2.VideoCapture(self.CAMERA_SOURCE)
    #camera.set(3, 320)
    #camera.set(4, 240)

    # Wait half a second for light adjustment
    time.sleep(0.5)

    return camera

def start_recording(self):
    """
    Setup the recorder
    """

    self.current_file = self.archive + "/" + self.detected_at + "-usb.avi"

    Util.log(self.name, "Motion detected! Recording...")

    # Set path and FPS
    self.writer = cv2.VideoWriter(self.current_file, self.codec, self.fps,
(self.width, self.height))

```

```

def stop_recording(self):
    """
    Reset values to default
    """
    self.writer = None
    self.current_file = None
    self.detected_at = None

def convert_to_mp4(self, path):
    """
    Convert video file to mp4 using ffmpeg

    @param string path
    """
    try:
        Util.log(self.name, "Converting video...")
        destination = os.path.splitext(path)[0] + '.mp4'
        cmd = 'ffmpeg -i "{}" "{}" 2> /dev/null && rm
("{}"'.format(path, destination, path)
        #cmd = 'for i in ' + self.archive + '/*.avi; do ffmpeg -i "$i"
"${i%.*}.mp4" 2> /dev/null && rm "$i"; done'
        p = subprocess.Popen(cmd, shell=True)
        (output, err) = p.communicate()

    except subprocess.CalledProcessError:
        Util.log(self.name, "Error converting video")

def run(self):
    """
    Main worker
    """
    observer = deque(maxlen=self.fps * self.OBSERVER_LENGTH)
    previous_frame = None

    while True:
        # Grab a frame
        (grabbed, self.current_frame) = self.source.read()

        # End of feed
        if not grabbed:
            break

        # Gray frame
        frame_gray = cv2.cvtColor(self.current_frame,
cv2.COLOR_BGR2GRAY)

        # Blur frame
        frame_blur = cv2.GaussianBlur(frame_gray, (21, 21), 0)

```

```

        # If there's no previous frame, us the current one
        if previous_frame is None:
            previous_frame = frame_blur
            continue

        # Delta frame
        delta_frame = cv2.absdiff(previous_frame, frame_blur)

        # Threshold frame
        threshold_frame = cv2.threshold(delta_frame, 15, 255,
cv2.THRESH_BINARY)[1]

        # Dilate the thresholded image to fill in holes
        kernel = np.ones((5, 5), np.uint8)
        dilated_frame = cv2.dilate(threshold_frame, kernel,
iterations=4)

        # Find difference in percent
        res = dilated_frame.astype(np.uint8)
        movement = (np.count_nonzero(res) * 100) / res.size

        # Add movement percentage to observer
        observer.append(movement)

        if self.do_add_contours or self.do_add_target:
            self.current_frame, targets =
self.add_contours(self.current_frame, dilated_frame)

            if self.do_add_target:
                self.current_frame =
self.add_target(self.current_frame, targets)

        if self.do_record and self.detected(sum([x > self.threshold
for x in observer]) > 0):
            if not self.recording():
                self.start_recording()

            self.writer.write(self.current_frame)
        elif self.recording():
            # Delete Old files
            self.delete()

            # Convert
            self.convert_to_mp4(self.current_file)

            # Reset all
            self.stop_recording()

        Util.log(self.name, "Observing...")

```

```

        # Set blurred frame as new previous frame
        previous_frame = frame_blur

        # Display
        if self.do_display:
            cv2.imshow("Current frame:", self.current_frame)

        # Exit on 'q'
        key = cv2.waitKey(1) & 0xFF

        if key == ord('q'):
            break

    def delete(self):
        """
        delete mic data to a mp4 file.
        @param list data
        """
        count = 0
        Util.log(self.name, "Delete USB Cam video...")

        file_list = sorted(os.listdir(self.archive), reverse=True)
        for filename in [file for file in file_list if
file.endswith("usb.mp4")]:
            if not filename.startswith('.'):
                type = self.get_type(filename)
                if type == "video":
                    count = count + 1
                    if self.REMAIN_RECORDING_FILES < count:
                        Util.log(self.name, "Delete USB video
filename=" + filename + ", type=" + type + ", count=" + str(count))
                        os.remove(self.archive + "/" +
filename)

    def get_type(self, filename):
        name, extension = os.path.splitext(filename)
        return 'video' if extension == '.mp4' else 'video' if extension ==
'.avi' else 'audio' if extension == '.wav' else 'audio' if extension == '.mp3' else
'photo'

    def add_contours(self, raw_frame, dilated_frame):
        """
        Add contours to frame

        @param array raw_frame
        @param array dilated_frame
        @return tuple(array, list)
        """
        # Find contours on thresholded image
        _, contours, nada =

```

```

cv2.findContours(dilated_frame.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Make coutour frame
    contour_frame = raw_frame.copy()

    # Target contours
    targets = []

    # Loop over the contour
    for c in contours:
        # If the contour is too small, ignore it
        if cv2.contourArea(c) < 500:
            # Make sure this has a less than sign, not an html
escape
                continue

        # Contour data
        M = cv2.moments(c)
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        x, y, w, h = cv2.boundingRect(c)
        rx = x + int(w / 2)
        ry = y + int(h / 2)
        ca = cv2.contourArea(c)

        # plot contours
        cv2.drawContours(contour_frame, [c], 0, (0, 0, 255), 2)
        cv2.rectangle(contour_frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.circle(contour_frame, (cx, cy), 2, (0, 0, 255), 2)
        cv2.circle(contour_frame, (rx, ry), 2, (0, 255, 0), 2)

        # save target contours
        targets.append((rx, ry, ca))

    return contour_frame, targets

def add_target(self, raw_frame, targets):
    """
    Add crosshairs to frame

    @param array raw_frame
    @param list targets
    @return array
    """
    # Make target
    area = sum([x[2] for x in targets])
    mx = 0
    my = 0

    if targets:

```

```

        for x, y, a in targets:
            mx += x
            my += y
        mx = int(round(mx / len(targets), 0))
        my = int(round(my / len(targets), 0))

    # Plot target
    tr = 50
    target_frame = raw_frame.copy()

    if targets:
        cv2.circle(target_frame, (mx, my), tr, (0, 0, 255, 0), 2)
        cv2.line(target_frame, (mx - tr, my), (mx + tr, my), (0, 0,
255, 0), 2)
        cv2.line(target_frame, (mx, my - tr), (mx, my + tr), (0, 0,
255, 0), 2)

    return target_frame

def detected(self, has_motion):
    """
    Check if this or another detector detected something

    @param boolean has_motion
    @return boolean
    """
    if has_motion:
        self.lock_manager.set()
    else:
        self.lock_manager.remove()

    self.detected_at = self.lock_manager.get_lock_time()

    return self.detected_at is not None

def recording(self):
    """
    Check if currently recording

    @return boolean
    """
    return self.writer is not None

if __name__ == "__main__":
    args = sys.argv[1:]
    source = None
    do_display = False

    try:
        opts, args = getopt.getopt(args, "hs:d", ["source=", "display"])

```



```

except getopt.GetoptError:
    print('python3 motion_detector.py -s <source> [-d]')
    sys.exit(2)

for opt, arg in opts:
    if opt == '-h':
        print('python3 motion_detector.py -s <source> [-d]')
        sys.exit()
    elif opt in ("-s", "--source"):
        source = arg.strip()
    elif opt in ("-d", "--display"):
        do_display = True

if source is not None:
    print('Input: ', source)
else:
    print('Input: Camera')

if source is not None and not os.path.isfile(source):
    print(str(source) + " does not exist")
else:
    md = Motion_Detector(source=source, do_display=do_display,
do_add_contours=True)
    md.start()

```

piwebcamera.py

```

import os
import sys
import time
import math
import getopt
import numpy as np
import cv2
import threading
import subprocess
from collections import deque

from lock_manager import Lock_Manager
from util import Util

class PiWebCamera(threading.Thread):
    def __init__(self, video_source=0, source=None, do_record=True,
do_display=True, do_add_contours=True, do_add_target=False):
        threading.Thread.__init__(self)

        self.name = self.__class__.__name__
        self.archive =

```

```

os.path.join(os.path.dirname(os.path.realpath(__file__)), 'archive')

        self.writer = None
        self.current_frame = None

        self.codec = cv2.VideoWriter_fourcc('M', 'J', 'P', 'G')
        self.OBSERVER_LENGTH = 5 # Time in seconds to be observed for motion
        self.threshold = 15

        self.CAMERA_SOURCE = video_source
        self.REMAIN_RECORDING_FILES = 3 # 10이상 부터 삭제 후 저장
        self.do_display = do_display
        self.do_record = do_record
        self.do_add_contours = do_add_contours
        self.do_add_target = do_add_target
        self.current_file = None

        self.source = cv2.VideoCapture(source) if source is not None else
self.init_camera()

        self.fps = self.find_fps(self.source)
        self.height, self.width = self.get_dimensions(self.source)
        Util.log(self.name, "Initializing pi camera class with video_source="
+ str(self.CAMERA_SOURCE))
        Util.log(self.name, "width: {"+str(self.width)+"}, height :
{"+str(self.height)+"}")

        self.lock_manager = Lock_Manager("motion")

def __del__(self):
    # Release camera
    self.source.release()

    # Close all windows
    cv2.destroyAllWindows()

    # Remove lock if exists
    self.lock_manager.remove()

def get_frame(self):
    """
    Return the current frame

    @return bytes
    """
    return self.frame_to_jpg(self.current_frame) if self.current_frame is
not None else None

def frame_to_jpg(self, frame):
    """

```

```

        Convert video frame to jpg

        @param array frame
        @return bytes
        """
        ret, jpeg = cv2.imencode('.jpg', self.current_frame)
        return jpeg.tobytes()

def get_dimensions(self, source):
    """
    Determine height and width of the video source

    @return tuple(int, int)
    """
    frame = cv2.cvtColor(source.read()[1], cv2.COLOR_RGB2GRAY)
    return frame.shape[0: 2]

def find_fps(self, source):
    """
    Determine frames per second of the video source

    @param video source
    @return int
    """
    Util.log(self.name, "Determining FPS...")

    # How many frames to capture
    num_frames = 120

    # Start time
    start = time.time()

    # Grab a few frames
    for i in range(0, num_frames):
        ret, frame = source.read()

    # End time
    end = time.time()

    # Calculate frames per second
    fps = int(math.floor(num_frames / (end - start)))
    Util.log(self.name, "Setting FPS to " + str(fps))

    return fps

def init_camera(self):
    """
    Start the camera

    @return cv2.VideoCapture

```

```

"""
# Init camera
camera = cv2.VideoCapture(self.CAMERA_SOURCE)
#camera.set(3, 320)
#camera.set(4, 240)

# Wait half a second for light adjustment
time.sleep(0.5)

return camera

def start_recording(self):
    """
    Setup the recorder
    """

    self.current_file = self.archive + "/" + self.detected_at + "-pic.avi"

    Util.log(self.name, "Motion detected! Recording...")

    # Set path and FPS
    self.writer = cv2.VideoWriter(self.current_file, self.codec, self.fps,
(self.width, self.height))

def stop_recording(self):
    """
    Reset values to default
    """
    self.writer = None
    self.current_file = None
    self.detected_at = None

def convert_to_mp4(self, path):
    """
    Convert video file to mp4 using ffmpeg

    @param string path
    """
    try:
        Util.log(self.name, "Converting video...")
        destination = os.path.splitext(path)[0] + '.mp4'
        cmd = 'ffmpeg -i "{}" "{}" 2> /dev/null && rm
("{}").format(path, destination, path)
        #cmd = 'for i in ' + self.archive + '/*.avi; do ffmpeg -i "$i"
"${i%.*}.mp4" 2> /dev/null && rm "$i"; done'
        p = subprocess.Popen(cmd, shell=True)
        (output, err) = p.communicate()

    except subprocess.CallProcessError:
        Util.log(self.name, "Error converting video")

```

```

def run(self):
    """
    Main worker
    """
    observer = deque(maxlen=self.fps * self.OBSERVER_LENGTH)
    previous_frame = None

    while True:
        # Grab a frame
        (grabbed, self.current_frame) = self.source.read()

        # End of feed
        if not grabbed:
            break

        # Gray frame
        frame_gray = cv2.cvtColor(self.current_frame,
cv2.COLOR_BGR2GRAY)

        # Blur frame
        frame_blur = cv2.GaussianBlur(frame_gray, (21, 21), 0)

        # If there's no previous frame, us the current one
        if previous_frame is None:
            previous_frame = frame_blur
            continue

        # Delta frame
        delta_frame = cv2.absdiff(previous_frame, frame_blur)

        # Threshold frame
        threshold_frame = cv2.threshold(delta_frame, 15, 255,
cv2.THRESH_BINARY)[1]

        # Dilate the thresholded image to fill in holes
        kernel = np.ones((5, 5), np.uint8)
        dilated_frame = cv2.dilate(threshold_frame, kernel,
iterations=4)

        # Find difference in percent
        res = dilated_frame.astype(np.uint8)
        movement = (np.count_nonzero(res) * 100) / res.size

        # Add movement percentage to observer
        observer.append(movement)

        if self.do_add_contours or self.do_add_target:
            self.current_frame, targets =
self.add_contours(self.current_frame, dilated_frame)

```

```

        if self.do_add_target:
            self.current_frame =
self.add_target(self.current_frame, targets)

        if self.do_record and self.detected(sum([x > self.threshold
for x in observer])) > 0):

            if not self.recording():
                self.start_recording()

            self.writer.write(self.current_frame)
elif self.recording():
    # Delete Old files
    self.delete()

    # Convert
    self.convert_to_mp4(self.current_file)

    # Reset all
    self.stop_recording()

    Util.log(self.name, "Observing...")

    # Set blurred frame as new previous frame
    previous_frame = frame_blur

    # Display
    if self.do_display:
        cv2.imshow("Current frame:", self.current_frame)

    # Exit on 'q'
    key = cv2.waitKey(1) & 0xFF

    if key == ord('q'):
        break

def delete(self):
    """
    delete mic data to a mp4 file.
    @param list data
    """
    count = 0
    Util.log(self.name, "Delete PI Cam video...")

    file_list = sorted(os.listdir(self.archive), reverse=True)
    for filename in [file for file in file_list if
file.endswith("pic.mp4")]:
        if not filename.startswith('.'):
            type = self.get_type(filename)
            if type == "video":

```

```

        count = count + 1
        if self.REMAIN_RECORDING_FILES < count:
            Util.log(self.name, "Delete PIC video
filename=" + filename + ", type=" + type + ", count=" + str(count))
            os.remove(self.archive + "/" +
filename)

    def get_type(self, filename):
        name, extension = os.path.splitext(filename)
        return 'video' if extension == '.mp4' else 'video' if extension ==
'.avi' else 'audio' if extension == '.wav' else 'audio' if extension == '.mp3' else
'photo'

    def add_contours(self, raw_frame, dilated_frame):
        """
        Add contours to frame

        @param array raw_frame
        @param array dilated_frame
        @return tuple(array, list)
        """
        # Find contours on thresholded image
        _, contours, nada =
cv2.findContours(dilated_frame.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Make coutour frame
        contour_frame = raw_frame.copy()

        # Target contours
        targets = []

        # Loop over the contour
        for c in contours:
            # If the contour is too small, ignore it
            if cv2.contourArea(c) < 500:
                # Make sure this has a less than sign, not an html
escape
                continue

            # Contour data
            M = cv2.moments(c)
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])
            x, y, w, h = cv2.boundingRect(c)
            rx = x + int(w / 2)
            ry = y + int(h / 2)
            ca = cv2.contourArea(c)

            # plot contours
            cv2.drawContours(contour_frame, [c], 0, (0, 0, 255), 2)

```

```

        cv2.rectangle(contour_frame, (x,y), (x+w,y+h), (0,255,0), 2)
        cv2.circle(contour_frame, (cx,cy), 2, (0,0,255), 2)
        cv2.circle(contour_frame, (rx,ry), 2, (0,255,0), 2)

        # save target contours
        targets.append((rx,ry,ca))

    return contour_frame, targets

def add_target(self, raw_frame, targets):
    """
    Add crosshairs to frame

    @param array raw_frame
    @param list targets
    @return array
    """
    # Make target
    area = sum([x[2] for x in targets])
    mx = 0
    my = 0

    if targets:
        for x, y, a in targets:
            mx += x
            my += y
        mx = int(round(mx / len(targets), 0))
        my = int(round(my / len(targets), 0))

    # Plot target
    tr = 50
    target_frame = raw_frame.copy()

    if targets:
        cv2.circle(target_frame, (mx, my), tr, (0, 0, 255, 0), 2)
        cv2.line(target_frame, (mx - tr, my), (mx + tr, my), (0, 0,
255, 0), 2)

        cv2.line(target_frame, (mx, my - tr), (mx, my + tr), (0, 0,
255, 0), 2)

    return target_frame

def detected(self, has_motion):
    """
    Check if this or another detector detected something

    @param boolean has_motion
    @return boolean
    """
    if has_motion:

```



```

        self.lock_manager.set()
    else:
        self.lock_manager.remove()

    self.detected_at = self.lock_manager.get_lock_time()

    return self.detected_at is not None

def recording(self):
    """
    Check if currently recording

    @return boolean
    """
    return self.writer is not None

if __name__ == "__main__":
    args = sys.argv[1:]
    source = None
    do_display = False

    try:
        opts, args = getopt.getopt(args, "hs:d", ["source=", "display"])
    except getopt.GetoptError:
        print('python3 motion_detector.py -s <source> [-d]')
        sys.exit(2)

    for opt, arg in opts:
        if opt == '-h':
            print('python3 motion_detector.py -s <source> [-d]')
            sys.exit()
        elif opt in ("-s", "--source"):
            source = arg.strip()
        elif opt in ("-d", "--display"):
            do_display = True

    if source is not None:
        print('Input: ', source)
    else:
        print('Input: Camera')

    if source is not None and not os.path.isfile(source):
        print(str(source) + " does not exist")
    else:
        md = Motion_Detector(source=source, do_display=do_display,
do_add_contours=True)
        md.start()

```

capture.py

```
import os
import cv2
import datetime, time
from pathlib import Path

archive_path = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'archive')

def capture_and_save(im):
    s = im.shape
    # Add a timestamp
    font = cv2.FONT_HERSHEY_SIMPLEX
    bottomLeftCornerOfText = (10,s[0]-10)
    fontScale = 1
    fontColor = (20,20,20)
    lineType = 2

    cv2.putText(im,datetime.datetime.now().isoformat().split(".")[0],bottomLeftCornerOfText,font,fontScale,fontColor, lineType)

    m = 0
    p = Path(archive_path)
    for imp in p.iterdir():
        if imp.suffix == ".png" and imp.stem != "last":
            num = imp.stem.split("_")[1]
            try:
                num = int(num)
                if num>m:
                    m = num
            except:
                print("Error reading image number for",str(imp))
    m +=1
    lp = Path(archive_path + "/last.png")
    if lp.exists() and lp.is_file():
        np = Path(archive_path + "/img_{0}.png".format(m))
        np.write_bytes(lp.read_bytes())
    cv2.imwrite(archive_path + "/last.png",im)

if __name__=="__main__":
    capture_and_save()
    print("done")
```

🔄Revision #2

★Created 1 June 2023 14:33:14 by Hyeon Su Ryu

✎Updated 1 June 2023 14:37:44 by Hyeon Su Ryu