Image Capture Reference

Traffic Net

Stephen Kirby – 04/24/23

me@skirby.dev

Overview

This document details the practices and scripts used to collect and store images for TrafficNet-related projects in DCSL. These systems were developed between 2021-2023 in Python 3.8 / 3.9, please be aware that packages and practices might have changed since. I'll first detail what packages are used and avoided for reference, then explain the process and scripts, then talk about network-specifc details at the end. For questions or concerns, please email me.

Packages

Open-CV Python

This is the main image processing library used across TrafficNet. Images are represented as numpy arrays, making it fast and easy to use with machine learning models. I recommend becoming familiar with this library before changing the code. The cv2.VideoCapture object is what queries the RTSP URL.

Process

Most traffic cameras support some version of an RTSP protocol, allowing us to setup a video stream in most cases, or capture a single frame in some cases. In this collection method requires an authenticated user to request a video stream from an RTSP URL. The credentials, IP address, and camera-specific protocol comprise said URL, here are some examples:

- AXIS Cameras: rtsp://<username>:<user_password>@<camera_IP>/axis-media/media.amp
- CoHu Costar Cameras: rtsp://<username>:<user_password>@<camera_IP>/stream1

The ending or path for the stream can be found online. These both produce a video stream, which we'll pass to a cv2.VideoCapture object. If you're using a github repository to store this code, make sure you import your credentials from a local file outside of the repository for security. In the example below, I'm importing them from a config.py file in the same folder.

```python
import cv2  # opencv-python
from config import CAM_UNAME, CAM_PASS, CAM_IP

AXIS_RTSP_URL = f"rtsp://{CAM_UNAME}:{CAM_PASS}@{CAM_IP}/axis-media/media.amp"
COHU_RTSP_URL = f"rtsp://{CAM_UNAME}:{CAM_PASS}@{CAM_IP}/stream1"

cap = cv2.VideoCapture(AXIS_RTSP_URL)

while True:
    # Read
    ret, img = cap.read()
    if not ret: # Failed to capture frame
        break

    # do whatever with the frame here...
```

This is relatively common knowledge, just google “Capture video stream opencv python” to learn more. However, using the base class like this doesn’t grab the *latest* frame, just the next frame in a queue that fills up with images @ 60fps as soon as the stream is started. That’s why I’ve created the **CamCapture** class, detailed below, for incremental image capture.

Capture -> CamCapture

The capture.py file holds the CamCapture class, which is used to always get the latest frame from an RTSP stream. This is done by allowing the video capture to read in images at 60 frames per second, but always discarding the previous frame when a new frame arrives. This newest frame is held in the thread-safe queue and can be read in with the capture time using the same ‘.read()’ method as opencv. It also restarts the capture object when the connection is closed, and reattempts capture until the camera is back online. This should be an all-in-one capture solution for new cameras, so I have added a usage example below.

```
import cv2  # opencv-python
from capture import CamCapture
from config import CAM_UNAME, CAM_PASS, CAM_IP

AXIS_RTSP_URL = f"rtsp://{CAM_UNAME}:{CAM_PASS}@{CAM_IP}/axis-media/media.amp"
COHU_RTSP_URL = f"rtsp://{CAM_UNAME}:{CAM_PASS}@{CAM_IP}/stream1"

FPS = 10
cap = CamCapture(AXIS_RTSP_URL)

while True:
    # Get the datetime of capture and image as numpy array
    cap_time, img = cap.read()

    # do whatever with the frame here...

    time.sleep(1/FPS)
```

This should be all you need to set up a simple collection script, and I have made sure the capture.py file can be used in isolation. To get started, cv2.imsave(filepath, img) will save the file locally.

TDOT Collection

The Tuscaloosa VPN requires a TDOT account to connect. The protocol/client you should use is Palo Alto Networks GlobalProtect VPN. Note that no official client exists for Ubuntu. I installed my client from the University of Washington, but you can use the openconnect library. The authentication requires 2FA and should be set up with our current contact in TDOT. Note: your TDOT credentials are DIFFERENT from the camera credentials.

ALDOT Collection

ALDOT capture is significantly more complex and requires a bit of paperwork. Get with Dr. Atkison on setting up an account. You will go through many steps to set up a remote desktop viewer on the target device. Once you have done this, install the CamCapture code and your script on the remote machine and have it upload the captured images to a cloud storage solution. You will need administrator privileges on the device.