

# 3rd Person Dash-Camera

Personal Traffic Camera Footage Archival Tool

Andrew Johnston - [andrewjohnston@gatech.edu](mailto:andrewjohnston@gatech.edu)

Lewey C. Wilson III - [lwilson67@gatech.edu](mailto:lwilson67@gatech.edu)

GitHub Repository: <https://github.com/andrewjohnston99/3PDC>

Presentation Video: [https://youtu.be/qA0\\_2JSGRgg](https://youtu.be/qA0_2JSGRgg)

Project Report

CS 4635 - Introduction to Enterprise Computing

Spring 2021

# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>Motivation</b>	<b>3</b>
<b>Related Work</b>	<b>4</b>
CameraFTP	4
<b>Changes From Our Original Proposal</b>	<b>5</b>
<b>Features and Completed Objectives</b>	<b>6</b>
Driver Location Tracking and Predictive Camera Selection	6
Traffic Camera Video and Image Retrieval	6
Footage Archival	7
Rolling Deletion	7
<b>System Architecture</b>	<b>8</b>
Hardware	8
Software	9
Diagram	10
<b>Implementation</b>	<b>11</b>
Traffic Camera Video and Image Retrieval	11
Footage Archival and Organization	12
Scheduling, Sessions, and Multithreading	14
Session Handler	15
<b>Evaluation</b>	<b>17</b>
<b>Challenges</b>	<b>18</b>
Computing limitations of the Raspberry Pi	18
Fine-tuning the predictive camera selection algorithm	19
Diagnosing smartphone-side issues	19
<b>Additional Applications of 3PDC</b>	<b>20</b>
Bus and Semi-trailer Truck Fleet Monitoring	20
Police Chase Tracking	21
<b>Future Work</b>	<b>21</b>
<b>Bibliography</b>	<b>23</b>

## Motivation

The city of Atlanta is notorious for its congested and chaotic traffic, being ranked the tenth most-congested city in the United States [6]. In addition to this, in 2018, there were roughly 400,000 motor vehicle crashes across the state, resulting in over 1,500 deaths and many more injuries [1]. In the aftermath of these accidents, the victims are often faced with significant medical expenses, and if one party is at fault, the situation might have to be resolved in court. In this case, both parties will want to gather as much evidence as they can to strengthen their argument and prove that the other is at fault.

While dash-cameras are a great way to archive accidents that arise from Atlanta's unique traffic, not every driver uses one, and sometimes a different angle is necessary to fully investigate what caused an accident. Other times, it might be helpful to understand the state of traffic in the lead-up to an accident and any other external factors that could have contributed to the outcome. The government of Georgia has a publicly-accessible network of traffic cameras (<http://www.511ga.org/>), with both live feeds and live “snapshots”, but the way the state provides this service makes it nearly unusable for gathering information about an accident after the fact. The



Figure 1. Screenshot from [www.511ga.org](http://www.511ga.org)

problem is that the video and pictures from these cameras are not archived for any significant amount of time. Videos can only be viewed in real time and snapshots are only visible for 10-20 minutes before they are overwritten with the latest conditions [5]. This time period is far too short for victims or their representatives to access the footage before it is lost, leaving them frustrated and confused about how they will gather the necessary evidence to make their case [1]. If they are lucky, there might be eyewitnesses or private business surveillance that captured the accident, but these alternatives are not always available or reliable [2]. This problem caused us to evaluate the need for a short-term archival system for this data that is already being generated

and available to the public. We have developed this kind of archival tool to help drivers better handle personal injury lawsuits and insurance claims by giving them local access to this crucial data for a longer amount of time than the Georgia Department of Transportation makes it available.

## Related Work

### CameraFTP



Figure 2. Camera FTP Logo

CameraFTP is a cloud surveillance and recording service provider. This service allows individuals, organizations, and governments to upload their security camera footage to CameraFTP's database for storage and backup so they can view it later from any device. If the customer desires, CameraFTP can publish live footage and recordings for public viewing [4]. They recommend their service to organizations that manage traffic cameras because it is easier and more to upload and publish their footage with CameraFTP instead of attempting to host it themselves [4]. One of the reasons the Georgia Department of Transportation cites for their inability to save traffic footage is that storing it may be cost-prohibitive [3]. Additionally, the state itself mainly utilizes the cameras to understand current traffic flow and publish real-time reports on traffic rather than serve as an archive of public records.

## Changes From Our Original Proposal

Over the course of the semester, the intended direction and use case for our tool (originally Atlanta Traffic Log, now 3rd Person Dash-Camera) saw a few significant changes. Originally, our project was going to constantly monitor a large set of traffic cameras in the Atlanta area and periodically retrieve images from them for later viewing. If no image could be retrieved from the camera's JPG link, then our tool would attempt to take a frame from the RTSP live stream and save that instead. The use case for such a tool was that, in the event of an accident, users could then look at the saved images from any nearby cameras for the purposes of gathering evidence that can be used in personal injury lawsuits and insurance claims. While the idea of gathering evidence is still a guiding principle for the final version of our project, we found that constantly downloading still images from hundreds of cameras could be seen as a waste of resources since the tool runs locally on a Raspberry Pi. Additionally, due to the periodic nature of the stills, the

likelihood that the exact moments before, during, or after a crash are captured is very slim. Instead of compiling low-detail image data from hundreds of cameras and hoping that the user can be seen, we decided to collect continuously updated video data only from the cameras



Figure 3. On the left is traffic camera footage retrieved by 3PDC (target car is highlighted). On the right is dash camera footage from the target car.

that are likely to see the user. Using location data from the user's smartphone, the 3rd Person Dash-Camera determines which traffic cameras are within a close proximity to the user and saves their video streams as the user drives by. With the utilization of location data, the addition of a predictive traffic camera-searching algorithm, and a shift in focus to capturing videos instead of single images added complexity to the original

project, the end result provides the user with significantly more useful footage. In the event that evidence is needed after an accident, 3PDC can arm the user with full motion video from traffic cameras available along the path of their entire trip, in hopes that the accident and the traffic conditions leading up to it are able to be captured.

## Features and Completed Objectives

The 3rd Person Dash-Camera is a tool that captures footage of the user's driving by utilizing Georgia's network of internet-enabled traffic cameras. While the user is driving, 3PDC uses location data from their smartphone to check for nearby traffic cameras that they are likely to pass by. As they approach a traffic camera, 3PDC connects to its corresponding video stream and begins recording. Once the recording is finished, the footage is saved locally on a USB drive connected to the server that 3PDC is running on. The following features were implemented to achieve these objectives:

*[The implementation of these features will be elaborated upon in the [Implementation](#) section of this report.]*

### Driver Location Tracking and Predictive Camera Selection

3rd Person Dash-Camera runs on a server (explained in [System Architecture](#)) in your home and communicates with Google Maps' "Share My Location" feature running on your smartphone. Using this location data and past location data in order to approximate speed and estimate distance traveled, along with a GeoJSON containing all of Georgia's traffic cameras, 3PDC frequently checks for and identifies cameras in close proximity to the user. This allows the tool to focus on only capturing footage from cameras that are most likely to see the user as they approach them.

### Traffic Camera Video and Image Retrieval

The cameras used by the Georgia Department of Transportation record traffic in two formats: all cams periodically capture and upload still images and a large portion of

them also support real-time video streams. Based on the location of the driver and the output of the camera selection algorithm, the footage retrieval script attempts to establish a connection with each selected camera's RTSP stream. If a connection is successfully established, the script captures frames for a specified amount of time and saves the resulting video as an AVI file. The streaming video from GDOT cameras are real-time, while the images are only updated every few minutes making them less likely to see the user as they are driving. Not all cameras have video streams, and the ones that do are occasionally inaccessible. If the attempt to establish a connection with the RTSP stream fails, however, the script will default to the image and save that instead.

## Footage Archival

After the video and images are retrieved, they will be stored locally on a high-capacity USB drive connected to the Raspberry Pi. The file structure is organized in such a way that the user can follow their trip chronologically. Each driving session is given its own directory. Within that directory are directories sorted by date and within those are "event" directories. Events represent the occurrences of the location script polling the user's location and selecting nearby cameras. Since multiple cameras can be selected in one polling event, each event has its own directory. Within the event directories are folders labelled with the name of the camera, and within those is the time stamped video or image taken from that camera.

## Rolling Deletion

In order to manage storage space, the date-sorted directories and all of their contents are deleted after a certain interval of time. This is similar to rolling storage on a physical dash-camera where the memory card constantly overwrites the old footage, only in the case a time to keep alive is used to prevent wear on the drive from constantly being full.

# System Architecture

## Hardware

In our project we utilized a Raspberry Pi 3B+ as our server. This choice was made for a number of reasons. First, the Pi utilizes an ARM processor which allows for power efficiency to reduce the cost of running 24/7. The Pi is also surprisingly powerful for its wattage with a full quad-core processor coupled with 1GB of memory. Additionally, ARM processors are becoming more common in the datacenter, so utilizing the architecture made for a perfect learning opportunity. Another benefit of using a Pi is that the hardware is relatively inexpensive so it has a low barrier to entry, and it is also standardized hardware unlike most x86 based servers, which allows for easy replication of the project on known working hardware. Finally, as college students resources can be tight, and while a Pi 4 would yield much more performance while retaining the other benefits listed, we both happened to have the 3B+ model from a previous class which made the choice simple.

In terms of storage, since video and photo data can add up quickly we opted to add “external storage” to the Pi in the form of a 128GB USB Flash Drive. Similar to a SAN (Storage Attached Network) in enterprise where servers utilize mapped storage for files and other resources, we again saw this as a learning opportunity, and decided to go this direction, only with our implementation being over USB 2.0 rather than a local network. Our test server was also equipped with a Class 10 32GB microSD card as the boot drive. This allowed for plenty of storage and reasonable storage latency for the OS.

The hardware for the end-user in our testing was an Android Smartphone, specifically a Google Pixel 4a running Android 11, operating on Verizon’s 4G LTE Network. Again, this choice stems from what we already own. The benefit of using Google Maps is that it is cross platform, so while our testing was done on Android there is little reason to believe that there would be an issue running on iOS.



## Software

Our project utilizes the “Raspberry Pi OS with Desktop” distribution based on Kernel version 5.10. The OS was selected as it was the latest version at the start of the project, is the officially supported OS for the Pi, and is a variant of Linux. Our programming language of choice for the project was Python 3, specifically version 3.7.3 32 bit as it came pre-installed with the OS. All written code including the session scheduler, session handler, location polling, and archive cleanup functions are implemented in Python. In order to support the complex requirements of the project, our project has a long list of dependencies. This includes the following:

- locationsharinglib - an unofficial library to interface with Google Maps Location Sharing
- numpy - a math library used for high level math, helpful for certain calculations
- pandas - a data manipulation and analysis library, useful for storing data frames such as when working with sets of coordinates and attributes
- requests - a library used for making http requests, helpful for fetching content from webpages
- sklearn - a library of machine learning functions, useful for doing math with coordinates
- opencv-python - a library used for programmatically recording and saving video and image files

Another requirement, though not a Python package is ATLAS (sudo apt-get install libatlas-base-dev) [8]. Described as Auto Tuned Linear Algebra Software, ATLAS is necessary for OpenCV to function on the Pi. ATLAS implements Basic Linear Algebra Subroutines, which apparently do not exist in Raspberry Pi OS by default and as such must be manually installed. In addition to installing ATLAS, in order to support browsing the footage over LAN, our project also utilizes Samba (sudo apt install samba samba-common-bin) to allow the Pi to act as a NAS.

## Diagram

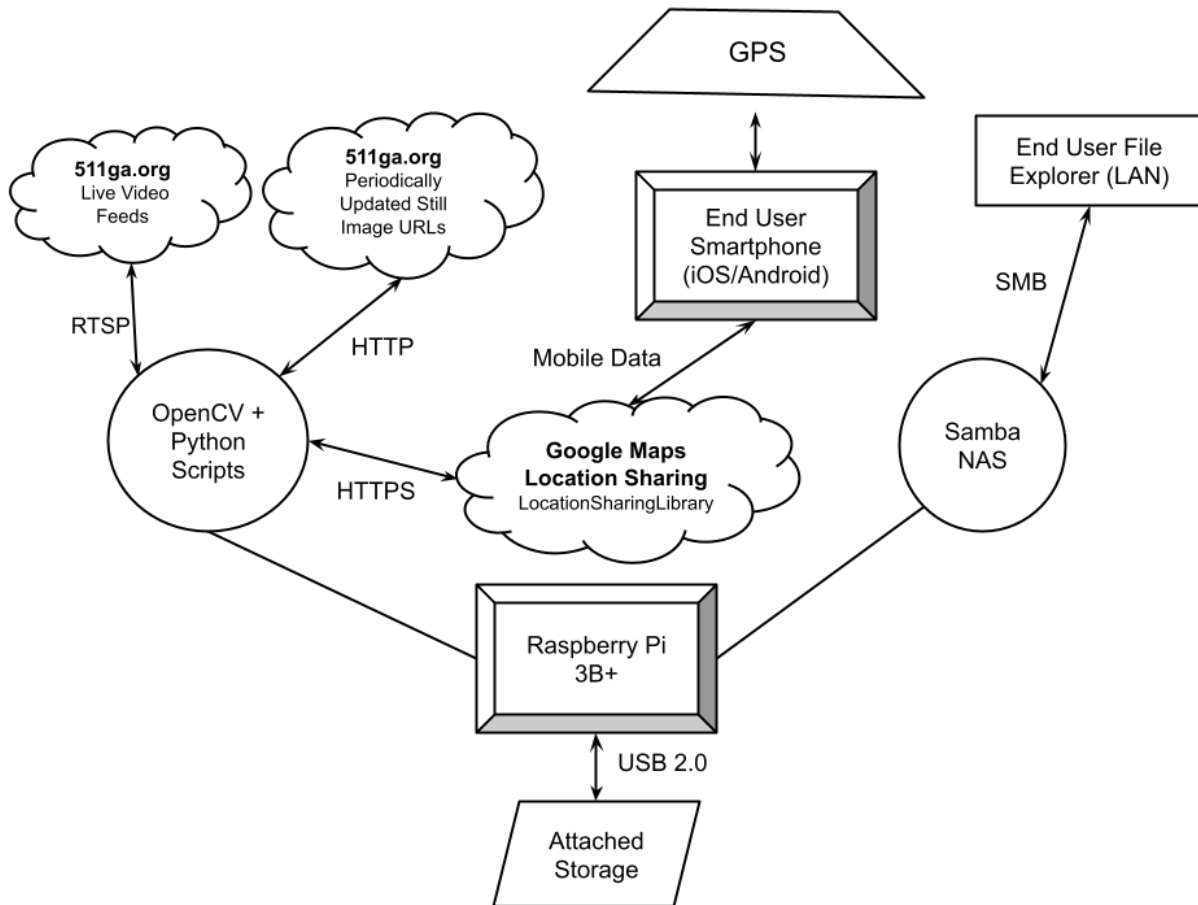


Figure 4. Architecture Diagram

Figure 4 displays our system architecture and the interactions between different components as detailed in the previous sections.

# Implementation

## Traffic Camera Video and Image Retrieval

The Georgia Department of Transportation's network of traffic cameras have two formats in which they may output their footage: RTSP live-streaming video and JPG images. Every camera in the network supports one or the other, and a large number of cameras support both. The cameras have a URL associated with each format that they support where the footage can be retrieved. 3PDC prioritizes the live streaming video because it is much more likely to capture the user's vehicle than the still images, which are updated about every ten to twenty minutes. However, if a camera does not support live streaming or the live stream is inaccessible, 3PDC will download the still image instead.

The video and image retrieval script is located in *OpenCVTest.py*. When the retrieval script is called, it needs to be supplied with the camera's RTSP and JPG links as well as a recording duration. It also needs a target directory and session number, which are used for directories the footage will be saved in (explained in Footage Archival and Organization). We utilize the OpenCV computer vision library for live streaming video capture and the Requests Python module to download the JPG images.

When the retrieval script is called it will attempt to establish a connection with the camera's live-stream using `cv2.VideoCapture()` and the RTSP link (Figure 5). If the connection is successful, an OpenCV VideoWriter data structure is initialized, and every new frame retrieved from the live stream is stored in it (Figure 6). The script continues to retrieve and store frames until the recording duration has elapsed. At that point, the connection to the live stream is terminated, and the footage is saved as an AVI video running at 10 frames per second.

```
22 #Establish a connection with the stream link
23 vcap = cv2.VideoCapture(stream)
```

Figure 5. Establishing connection with stream

```

77     #Save frames in a videowriter data structure
78     video = cv2.VideoWriter(filename, cv2.VideoWriter_fourcc(*'MJPG'), 10, size)

```

Figure 6. VideoWriter data structure

If the attempt to connect to the video stream fails, the script instead downloads the image from the JPG link supplied when the script was called. *Requests.get()* is used on the link and the image is then saved to the same directory that the video would have been saved to. This process can be seen in Figure 7.

```

62     #If the establishment of a connection to the video stream failed, retrieve the picture instead
63     if not vcap.isOpened():
64         print("Video File Cannot be Opened")
65         r = requests.get(url)
66         #Timestamp the pricture and save it
67         filenamePicture = finalpath + "/" + datetime.now().strftime("%Y%m%d-%H%M%S") + r".jpg"
68         print(filenamePicture)
69         with open(filenamePicture, "wb") as f:
70             f.write(r.content)

```

Figure 7. Downloading image from JPG link

## Footage Archival and Organization

We wanted to design a file structure for footage archival that made it easy to locate a specific clip by following the driver's usage of the 3PDC. The directories are structured as follows:

1. Driving Session (Outermost Directory)
2. Today's Date
3. Location Polling Event Number
4. Name of the camera being recorded
5. Video file labelled with date and time (Innermost)

The initial archive directory and the driving session directories are created by the session handler in *scheduler.py*. The initial *3PDCArchive* directory is created in the attached USB storage and the session directories are stored within it. Each session

directory is labelled with a *session\_ID* made up of the user's name and a timestamp of when the session was started. This process can be seen in Figure 8.

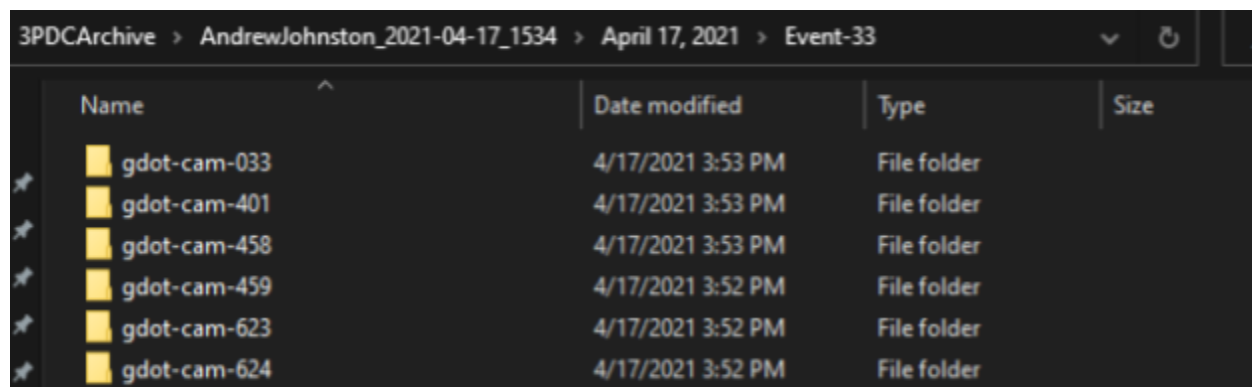
```

103     name = (person_full_name.replace(" ", "") + "_")
104     init_date = datetime.datetime.now()
105     session_ID = name + init_date.strftime("%Y-%m-%d_%H%M")
106
107     try:
108         path_string = r"/mnt/usbdrive/NAS/3PDCArchive/" + session_ID
109         os.mkdir(path_string)

```

Figure 8. Creation of 3PDCArchive and Session directories

The rest of the file structure is handled in the same script that handles video and image retrieval, *OpenCVTest.py*. When the recording script is called, it needs to be supplied with a target directory to save the footage in, which should be the session directory created earlier. Before it captures footage from the traffic camera's stream, this script creates a directory for today's date, within that it creates a directory with the event number (supplied by the session handler), and within that it creates a directory with the name of the camera being recorded. The captured footage is then saved in the directory for the camera it was retrieved from. This structure can be seen at the top of Figure 9.



Name	Date modified	Type	Size
gdot-cam-033	4/17/2021 3:53 PM	File folder	
gdot-cam-401	4/17/2021 3:53 PM	File folder	
gdot-cam-458	4/17/2021 3:53 PM	File folder	
gdot-cam-459	4/17/2021 3:52 PM	File folder	
gdot-cam-623	4/17/2021 3:52 PM	File folder	
gdot-cam-624	4/17/2021 3:52 PM	File folder	

Figure 9. Directories labelled with the name of the camera the clip was retrieved from.

Because of differing file structure and syntax between operating systems, there are two versions of the video & image retrieval script: Linux and Windows. The primary difference is the direction of slashes ("/" for Linux and "\" for Windows). The windows

version of the script is primarily used for testing, as the tool is intended to be run on a Raspberry Pi.

The deletion script is located in *cleanupRecordings.py*. It takes in a target directory that contains the sub-directories labelled by date and an integer value representing how many days before today to delete from (0 will delete today's recordings, 1 will delete yesterday's, etc). With this information it will search the target directory for any folders labelled with a date n-days before the current date and delete it along with all of its contents.

## Scheduling, Sessions, and Multithreading

Due to the nature of polling Google Maps for shared locations, 3PDC needs to constantly run in the background on the Pi. In order to facilitate this, our code utilizes a main "scheduler" which facilitates all tasks necessary for 3PDC to function, located in *scheduler.py*. The scheduler is designed to be run at boot so it is constantly ready to capture a session so long as the Pi is powered on and connected to the internet. When the scheduler first starts it does a few setup tasks including defining global variables for use by the threads, enabling and writing to a log file, and most importantly loading and transforming the GA 511 GeoJSON data. The GeoJSON file is either loaded from the GA 511 CDN endpoint or from local storage if the online copy is not available. This file defines each camera and includes metadata such as its coordinates, name, street address, and most importantly feed URLs. The scheduler initialization loads the GeoJSON and transforms it into a dataframe with only important information including the latitude, longitude, RTSP endpoint, and JPG endpoint. Once the initialization is done, the scheduler enters into a four-part state machine. In the first state, the scheduler listens for a session to start, checking every x seconds where x represents the check for broadcast polling interval defined in the config file. When it detects that location is being shared with it, and the user sharing their location matches the target user listed in the config file, it then exits the first state and moves on to the second state. In the second state, the session is initialized. The initialization includes defining and creating the path for the session folder and establishing an ID for the session. Now that

the session is initialized, the state machine proceeds to the third state which is the session handler. The session handler is explained in detail in the following section due to the sheer complexity of the stage. After a session finishes, the state machine enters the fourth and final state, which is responsible for resetting any global variables used by the session and preparing the scheduler to return to the initial state of checking for a location sharing session (check for broadcast). It is important to note that all states in the state machine are run as threads. A thread status variable is used to tell the state machine which state it is in. The benefit of this approach is that no one state can block the scheduler from running. Additionally, the scheduler is responsible for invoking the archive cleanup script. This is run separately from the state machine and is time based. When the scheduler detects that it is time for the cleanup script to run it then spins up a separate thread for it. Overall, the scheduler solution works very well for 3PDC and it also allows flexibility for supporting multiple users at the same time in the future if used on more powerful hardware since everything is isolated into its own thread.

## Session Handler

The session handler is responsible for handling the bulk of the functionality for 3PDC. Inside the handler, a session log is created and placed in the same directory as the footage, before then transitioning into a loop to check the location, calculate stats for the cam selection algorithm, and invoke camera recording. Within the handler the polling rate is increased to every 30 seconds. This interval was determined from extensive testing of how often the Location Sharing feature pushes out the user's location. Outside of a navigation session a user's location is updated every 2-3 minutes. Inside a navigation session, a user's location is updated every 20-30 seconds, therefore we chose the 30 second . The end user is encouraged to run a navigation session in conjunction with location sharing to ensure that the most accurate location data is conveyed to the Pi. Part of our camera selection algorithm originally adjusted the polling rate each round however we found that it was best to collect the data as quickly as possible in order to be able to use a smaller radius and prioritize the most relevant cams. Once a location is found, the handler checks to make sure it is not empty, and

proceeds to calculate the straight line distance traveled and use that with timestamps to get a speed. The straight line distance is calculated by using the Haversine Formula on the last latitude and longitude and the current latitude and longitude. The speed is simply this distance, calculated in miles, divided by the difference in the location timestamps in fractional hours. Using the approximated speed, a polling radius is defined by a function that anticipates the distance to be traveled and defines a radius that roughly covers that. This radius value is converted from miles to kilometers and then into a fractional value based on the radius of the earth. Utilizing Scikit-Learn's Ball Tree structure, all camera coordinates from the GeoJSON are placed inside. Using the radius function on the Ball Tree, a query with the current latitude, longitude, and radius value can be used in order to get the indices in the GeoJSON of the closest cameras to the current location (Figure 10). This function is crucial in finding the cams out of a large dataset.

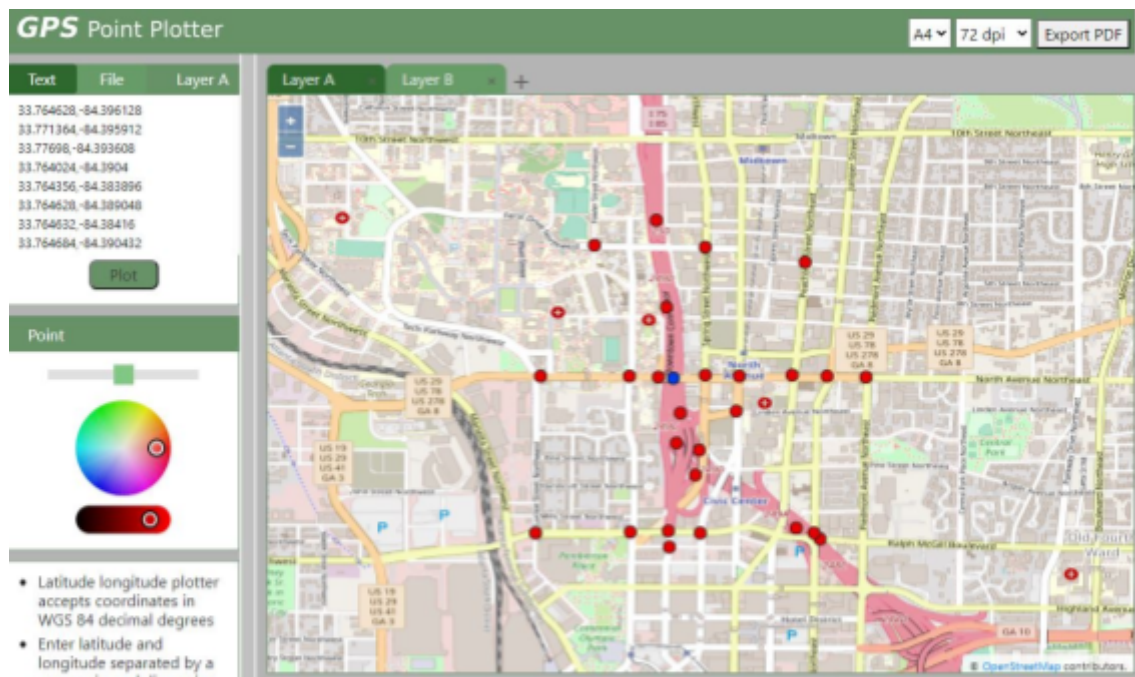


Figure 10. Output of testing Ball Tree on GeoJSON. Red dots are the cameras returned by the selection algorithm. The blue dot represents the supplied input coordinates. The radius in this image is set to 1 mile.

The cameras are also ordered from closest to farthest based on setting the ordered parameter to true, which allows the scheduler to get the x closest cams where x is the



minimum of either the max number of cams supported by the device as defined in the config file or the total number of cams returned if it is less than that. Once the handler has done this processing, it then invokes the camera recording script, making a thread for each camera to support concurrent recording, providing the camera name, session directory, RTSP and JPG URLs, and providing a duration to record, in this case the polling interval plus five additional seconds to allow for clip overlap as commonly used in physical dash-cameras. The handler then sleeps for the polling duration and repeats the process until it gets a none value for the current location, indicating that location sharing has stopped and the session is complete.

## Evaluation

The testing and evaluation of 3PDC took place in several phases. The early tests were performed using an Android emulator to run Google Maps. We would set the user's "mock location" in the emulator and monitor which cameras our algorithm selected and recorded. Before testing, we had expected that 3PDC would only be able to handle recording a few cameras (>5) simultaneously. These early tests demonstrated that it could simultaneously record more than we had anticipated, but when the location was set to an area with a very large number of cameras, 3PDC would call the recording script hundreds of times over the course of a few minutes. This was too much for the Raspberry Pi to handle, and recording tasks began failing. We would eventually set the maximum amount of cameras recorded in a single event to the 10 cameras closest to the user's position. This resulted in no errors for exceeding the maximum thread limit and seemed to decently saturate the processor on the Pi with a little bit of headroom left.



Figure 11. Driving past GDOT Traffic Camera #621 from the perspective of the driver (left) and the camera (right)

The next phase of testing involved real-world usage of the app. Andrew conducted several trial driving trips with 3PDC running, and the resulting footage was used to fine-tune the predictive camera selection algorithm so that the recordings were more likely to capture the car as it passed by (see Figure 10). During this phase of testing, we also discovered issues that can be caused by the smartphone (elaborated upon in [Challenges](#)). In short, if too many processes were running on the smartphone, it would stop sending properly formatted location data to 3PDC, causing silent errors in a dependency. After several iterations of the camera selection algorithm, we were able to get our project to frequently capture footage from the traffic cameras in which the car is visible.

The footage gathered by the tool during the final test drive was used to produce our demonstration video.

## Challenges

### Computing limitations of the Raspberry Pi

We needed to limit the number of cameras that can be recorded in a single event. Before the limit, when too many cameras were in one area, the Raspberry Pi could no longer successfully run the video retrieval script.

## Fine-tuning the predictive camera selection algorithm

We went through several iterations of the camera selection algorithm to account for the driver's speed and the location polling rate.

## Diagnosing smartphone-side issues

When too many processes were running on the smartphone, it would stop sending location data. This caused the location sharing library module to silently lock-up, causing the session to come to a halt while also leaving the scheduler stuck in a “dead” session-handler running state. In our case, as we were producing the demonstration video, the end user smartphone was simultaneously running Google Maps Locating Sharing, a Google Maps Driving Session, and the screen recorder built into Android 11, in addition to standard background tasks. Due to the intensity of the sun, the usage of GPS and cellular connectivity, and the processor intensive tasks, our hypothesis is that the phone started to disable non-crucial services as a way to prevent overheating. While we were unable to confirm this due to the lack of logs, on one testing trip when screen recording was not in use, we received a message indicating overheating as pictured in Figure 11. This particular session was with an outdoor temperature of 80°F, sunny weather, a black phone case mounted to the windshield, and no A/C active in the vehicle. While this error did not appear in the sessions that broke, when the screen recorder was active the phone was placed in “Do Not Disturb” to prevent recording sensitive notification data. It is likely that the overheating error message was silenced because of this. The issue only happened after around 15-20 minutes into a session when the screen recorder was active. 100% of the sessions with screen recording failed while 100% of the sessions without screen recording worked as expected. Our recommendation for users is to be mindful of the interior temperature of their vehicle,

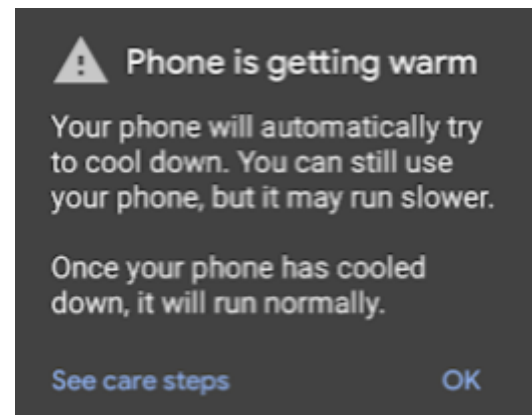


Figure 12. Android system temperature warning

especially if their device is a darker color and in direct sunlight, avoid running too many other tasks, and avoid using fast-charging or wireless charging.

## Additional Applications of 3PDC

3rd Person Dash-Camera was designed primarily for individuals to obtain traffic camera footage of their driving in order to have evidence for an insurance claim or court case if an accident were to occur. However, this is not the only potential use case for 3PDC. This tool can be used in any situation where an individual or organization wants to utilize a state's network of traffic cameras to follow a vehicle. Additionally, given enough storage space, our tool can follow the same vehicle over long periods of time, as long as the vehicle can continue to supply location data. The following are potential applications for 3PDC beyond the use case we have demonstrated:

### Bus and Semi-trailer Truck Fleet Monitoring

Organizations that operate large fleets of busses or trucks (schools, universities, transit authorities, shipping companies, etc) can have their drivers connect their smartphones' location service or a dedicated location-sharing device belonging to the truck to an instance of 3PDC running on a system belonging to the organization or potentially a virtual machine running in the cloud. As these busses and trucks drive around the state (or the country, if support for other states' traffic camera networks is added), 3PDC will record the traffic cameras along their route. Aside from the benefits of having this data in the event of an accident, organizations can also use this data for driver accountability, by making sure that they are staying on their intended route and making good time. It can also be used to gain insight into the traffic conditions bus and truck drivers are encountering, which can be used to inform decisions about which route they should take to get to their destination more efficiently.

## Police Chase Tracking

Similar to the bus and truck fleet monitoring application, police departments can have their officers and vehicles send location data to 3PDC to keep track of police car movements. Since 3PDC tries to record cameras that are ahead of the driver, it could possibly be useful in police chases. However, there is another way it can be used to monitor police chases. The only information 3PDC requires in order to follow a driver is their coordinates. As a result, it can operate using coordinates supplied manually; this is how we did much of our early testing. A police department could create a simulated driver moving along a path that they expect the suspect in a police chase to follow, and supply those coordinates to 3PDC. The tool can then record cameras following this simulated route and give officers insight into the suspect's movements.

## Future Work

1. Vehicle ID in the retrieved footage
  - a. Our demonstration video has manually drawn boxes over the car in order to identify the vehicle as it is recorded by our tool. In a future version of this project, it would be good to implement AI vehicle identification into the tool itself. This would require an improvement in camera quality, likely a minimum of 720p resolution, however as camera technology becomes cheaper and high-bandwidth wireless networks become more widespread this could definitely be seen in the near future.
2. Adding support for other states' network of traffic cameras
  - a. Different states may have implemented their traffic camera network differently from Georgia, and therefore would not be compatible with 3PDC in its current version. Future revisions could add support for other states to the tool.
  - b. Alabama's Traffic Camera Network: [Cameras - ALGO Traffic](#)

3. Web server and web application to access video files remotely
  - a. The ability to access traffic footage remotely could allow 3PDC to be utilized by multiple users as a web service.
4. Cloud storage backup (off-site replication)
5. Dedicated app for better location integration and recording status integration
  - a. Right now, 3PDC relies on the user sharing their location with it using Google Maps' "Share My Location" feature, which has limitations regarding the granularity and polling frequency of location data. An app designed specifically to communicate with 3PDC could improve location functionality and give the driver a better understanding of the tool's status (recording, offline, out of storage space, etc).
6. In the far future, with 5G and unlimited data, the project could be reimagined as mobile app and operate entirely in the edge.

## Bibliography

1. Butler, Wooten, & Peak LLP “Can I Obtain Traffic Camera Video of a Car Accident in Georgia?” Feb. 6, 2020. Web. Available:  
<https://www.butlerwootenpeak.com/2020/02/can-i-obtain-traffic-camera-video-of-a-car-accident-in-georgia/>
2. HG Legal Resources “How to Gather Car Accident Evidence from Traffic and Security Cameras” Web. Available:  
<https://www.hg.org/legal-articles/how-to-gather-car-accident-evidence-from-traffic-and-security-cameras-50063>
3. Simon, Christopher “How to Get Traffic Camera Video in Georgia” Web. Available:  
<https://www.christophersimon.com/how-to-get-traffic-camera-video-in-georgia/>
4. CameraFTP “Public Traffic Cameras and Webcams” Web. Available:  
[https://www.cameraftp.com/CameraFTP/Features/Traffic\\_cameras.aspx](https://www.cameraftp.com/CameraFTP/Features/Traffic_cameras.aspx)
5. Georgia Department of Transportation “Georgia 511” Web. Available:  
<http://www.511ga.org/>
6. INRIX “INRIX 2019 Global Traffic Scorecard” Web. Available:  
<https://inrix.com/scorecard/>
7. Gwennap, Linley “Apple’s M1 Signals New Era Of Competition In PC Processors” *Forbes*. Dec. 4, 2020. Web. Available:  
<https://www.forbes.com/sites/linleygwennap/2020/12/04/apple-m1-processor-new-era-of-pc-design/>
8. Debian Packages, “Package: libatlas-base-dev (3.10.3-10 and others)” *Debian*. Web. Available: <https://packages.debian.org/sid/libatlas-base-dev>