https://doi.org/10.2352/ISSN.2470-1173.2017.10.IMAWM-160

Creating the World's Largest Real-Time Camera Network

Ryan Dailey, Ahmed S. Kaseb, Chandler Brown, Sam Jenkins, Sam Yellin, Fengjian Pan, Yung-Hsiang Lu; Purdue University; West Lafayette, Indiana, USA

Abstract

Millions of cameras are openly connected to the Internet for a variety of purposes. This paper takes advantage of this resource to gather visual data. This camera data could be used for a myriad of purposes by solving two problems. (i) The network camera image data needs context to solve real world problems. (ii) While contextual data is available, it is not centrally aggregated. The goal is to make it easy to leverage the vast amount of network cameras.

The database allows users to aggregate camera data from over 119,000 network camera sources all across the globe in real time. This paper explains how to collect publicly available information from network cameras. The paper describes how to analyze websites to retrieve relevant information about the cameras and to calculate the refresh rates of the cameras.

Introduction

In recent years, the prices of electronic sensing devices have been falling rapidly. Among all sensing devices, image sensors (i.e., cameras) are special because the same sensors can capture a wide range of information. Cameras can be used to monitor traffic flow, view wildlife, detect intruders, or determine weather conditions. Millions of network cameras are connected to the Internet for a variety of purposes. Nearly two hundred million network cameras have been deployed [1]. The real-time visual data can be used in many applications, such as emergency responses. Some of these data streams are publicly available without password protection. As researchers gain the ability to collect large amounts of visual data about the world, the true potential of data-driven research is recognized. With the emergence of new machine-learning technologies [2–4], a wealth of previously untapped potential for large-scale visual data analysis has surfaced. In 2020, 75% of mobile traffic will be video [5] and 82% of IP traffic will be video [6]. Visual data is arguably the "biggest" of big data because one HD video camera can produce data orders of magnitude faster than text data. Computer vision is becoming one of the central data-analysis techniques driving big-data research. Visual data (image and video) is special because of the versatility and the rich information it provides. A single image may reveal many different types of information. For example, a family's vacation photograph also provides information about the weather. This paper uses "network cameras" for the cameras that are always connected to networks and have fixed locations; some network cameras may be PTZ (pan, tilt, and zoom). By this definition, network cameras do not include cameras in mobile phones.

Despite the large amount of real-time data publicly available, two major problems inhibit the true potential of analyzing the real-time data from network cameras. The first problem is the need for contextual information. Context could include the camera's location, refresh rate, whether it is indoor or outdoor, and so on. This contextual information is called metadata and can be helpful for data analytics. Metadata can be useful for identifying the cameras for specific purposes, for example, traffic cameras for studying urban transportation. The second problem is the wide range of protocols used to retrieve data from network cameras. Different brands of network cameras need different retrieval methods (for example, different paths for HTTP GET commands). Many organizations (such as departments of transportation in different cities) aggregate the data streams from multiple cameras. There is no standard as to how the information is aggregated and thus, there is no standard method for retrieving data from different sources.

This paper presents a system that adds context information to network cameras and retrieves data from large numbers of publicly available network cameras. The system solves the two problems that inhibit use of network cameras for large-scale data aggregation by providing techniques to build a uniform network camera database. The database, created using HTML parsing and web automation tools, includes the location information about each camera. In addition to the information collected using web parsing methods, tools have been developed to aggregate additional information about each camera. This paper describes methods to obtain information such as the frame rate and the resolution of the network cameras.

Evaluation of the proposed methods considers the amount of data collected and the reliability of the data. The validity of this data was determined by developing methods to eliminate static images from the database and validating the frame rate analysis of the network cameras using datasets with known frame rates. The database currently contains information from over 119,000 cameras located in 162 countries around the world.

This paper has the following contributions:

1. This paper describes how to construct one of the largest (perhaps the largest) camera network in the world.
2. The paper explains how to use HTML web parsing to pull data from websites that aggregate data from network cameras.
3. The system retrieves data from network cameras in websites using XML HTTP Requests.
4. Due to the large number of network cameras, it is necessary to frequently detect cameras that are disconnected or no longer being updated.
5. The paper describes how to determine the refresh rates of cameras posted to webservers.

Finding network cameras and the metadata about these cameras is part of the Purdue CAM2 (Continuous Analysis of Many CAMeras) project [7]. CAM2 is a computing platform for analyzing real-time data from network cameras. CAM2 provides real-time data from network cameras, has a run-time system for executing analysis programs, and manages computing resources.

Real-Time Visual Data from the Internet

Millions of network cameras have been deployed worldwide. Many of them have restricted accesses, protected by passwords or within private networks. Some of them make the data publicly available on the Internet. Even though the data is public, there is no central repository through which visual data from many different sources can be retrieved. If the visual data could be easily obtained and analyzed, many time-sensitive problems could be solved more easily.

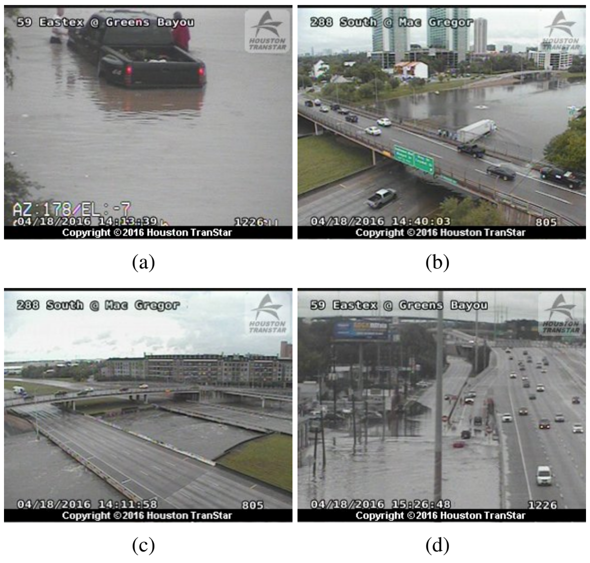


Figure 1. *Flooding in Houston on April 18, 2016*

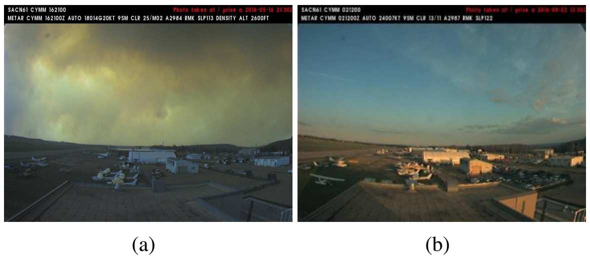


Figure 2. *(a) Smoky sky on May 16, 2016 in Canada due to wildfire (b) Clear sky on August 2, 2016 taken by the same camera*

Real-time visual data could be valuable in emergency responses. Figure 1 shows four images captured by the traffic cameras [8] in Houston during the flooding on April 18, 2016. Figure 2 shows two images taken by the same camera. One image was taken on May 16, 2016 when the sky was covered by smoke from a wildfire. The other image shows a clear sky on August 2, 2016 after the fire stopped. These two examples illustrate the potential for using network cameras for emergency responses. Network cameras require no human efforts for making emergency calls; thus, network cameras could help injured people even when they cannot make phone calls. If network cameras have high refresh rates, the visual data can be used to observe the development of an event.

This paper classifies network cameras into two types: IP cameras and Non-IP cameras. IP cameras have individual IP (Internet Protocol) addresses and anyone on the Internet can communicate with the cameras directly (some cameras may have password protection). IP cameras usually have built-in web servers and can respond to HTTP GET requests. Non-IP cameras do not have individual IP addresses and are not directly accessible on the Internet. Usually, the data streams from Non-IP cameras are aggregated into file servers and are accessible through websites. Many websites aggregate visual data from multiple cameras. Figure 3 (a) shows the website that displays the locations of traffic cameras in New York City [9]. Figure 3 (b) shows one snapshot from a traffic camera. Transportation officials can monitor whether an accident occurs on these streets. Figure 3 (c) is a website that allows viewers to see snow conditions in ski resorts [10]. Figure 3 (d) is a snapshot from a camera in a ski resort. Figure 3 (e) is a website through which viewers can observe weather conditions [11] and Figure 3 (f) is a snapshot. Figure 3 (g) shows a website that provides real-time views of tourist attractions [12] and Figure 3 (h) is a snapshot of Vancouver. The true potential of the real-time data from network cameras can be exploited more easily if the data is aggregated into a central repository.

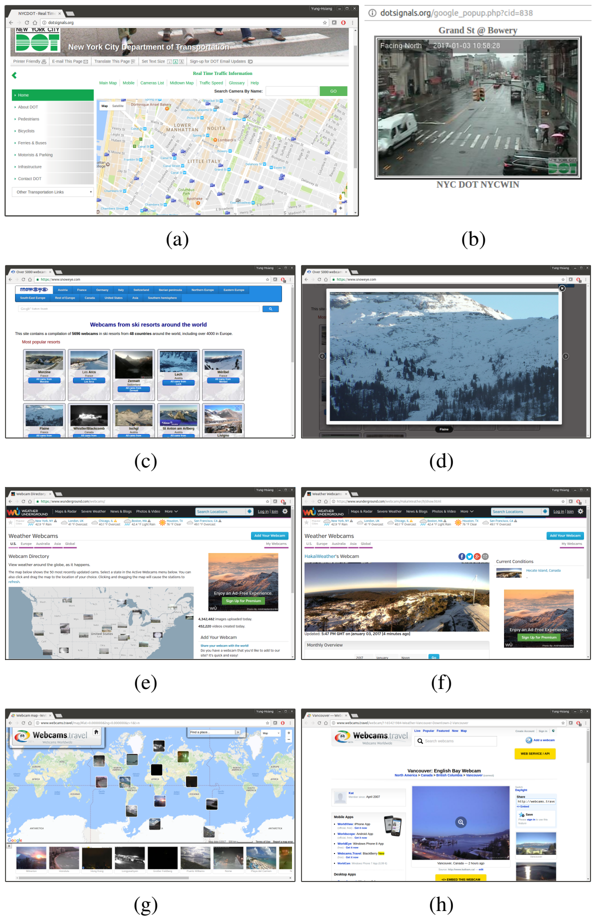

New York City Department of Transportation
dotsignals.org/google_popup.php?cid=838
Grand St @ Bowery
NYC DOT NYCWIN
(a)
(b)
Webcams from ski resorts around the world
(c)
(d)
Weather Webcams
(e)
(f)
Webcams.travel
Vancouver: English Bay Webcam
(g)
(h)

Figure 3. *Examples of websites that aggregate data from network cameras*

The four examples in Figure 3 show wide variations of these aggregation sites. Some websites provide precise locations and orientations of the cameras. Some others provide only approximate locations (such as the names of the cities). Some websites publish the application programming interfaces (API) to retrieve the data, while the others do not. Some websites assign individual URLs to the data streams; thus, knowing the URLs can retrieve the real-time data directly. Some other sites make the URLs reflect the time when the data is acquired; using an URL with a past timestamp obtains a old image. These websites have different refresh rates: some update the data every second (or multiple times per second); some others update every few minutes or every few hours. The CAM2 team has been building a camera database which can present a uniform interface to retrieve data from different sources. This database hides the heterogeneity of the differences so that the same program can utilize data from multiple sources.

Network Camera Database

The camera database is constructed by retrieving camera information from the websites discussed above. A unique parsing script is created for each individual website because the structure of each site is different. The parsing scripts are written in Python and use HTML parsing and web automation tools. The time it takes to develop a parsing script can vary greatly depending on the structure and variation of the site, the number of cameras, and the amount of data provided on the site.

Analyze Camera Aggregation Websites

Parsing scripts take advantage of several Python modules and APIs. Two of the most commonly used tools are the HTML parsing module Selenium [13] and BeautifulSoup4 [14]. Both tools come with advantages and limitations. Often, the limitations of one tool is mitigated by a combination of the two tools.

Selenium is a browser automation tool that supports Firefox and Chrome. For parsing camera data, Selenium has several advantages. First, it is easy to debug. During the execution of the script, Selenium renders a browser window and simulates the user interaction with the website. The user can see exactly what is happening as the code executes. Selenium executes all the JavaScript on a page load, which is one major advantage when compared to BeautifulSoup4. Selenium allows access to webpage elements by Xpath. Xpath is a way of navigating the hierarchy of the HTML syntax. The main disadvantage of Selenium is its speed. It has to render all the pages fully in the browser. Selenium also has stability issues and can crash when the user attempts to navigate between pages before a page is fully loaded. Another issue is that browser compatibility limits the cross-platform usability of the script. Often, different browser versions will load pages at different speeds, and Internet connections and other variables can have large effects on a script's stability.

BeautifulSoup4 (BS4) is another tool used for parsing websites for camera data. BS4 uses a different approach: Instead of fully rendering the page in a browser, BS4 downloads the page source and parses the HTML into a Python object. BS4 does not fully render the page, so it is faster than Selenium. Selenium can access information only in the current fully rendered page; BS4 can store multiple HTML pages, each in its own Python object. BS4 does not simulate user interactions, such as clicks or keyboard presses; thus, BS4 is faster than Selenium. BS4 also has limitations because it doesn't render the entire page. This often causes problems when loading pages that have HTML page elements with Javascript. On many websites, JavaScript is responsible for loading lists and populating camera pop-ups. Many websites have cameras organized in maps requiring users to navigate an interactive map and click on camera links. BS4 is unable to extract the camera information from these websites.

Website-parsing scripts that take advantage of both Selenium and BS4 are often the best option. Selenium can be used to load the webpage in a headless browser, such as PhantomJS [15]. PhantomJS does not need to fully render the page. After the page is rendered, the HTML source can be sent to BS4 scripts to extract information. This method is faster and more reliable.

CAM2 also uses the Google Geolocation API [16] to obtain the information about cameras' locations. Some websites provide latitude and longitude information. It is possible to look up the address (city, county, state, country) from such information. Some websites do not provide location information with sufficient precision. These cameras' locations are marked as "approximate" in the camera database.

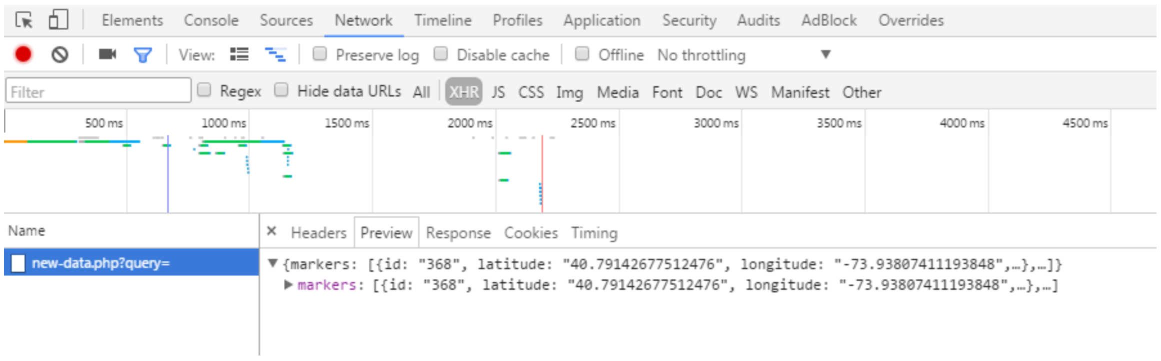
Find Camera Aggregation Websites

Searching "network cameras" on the Internet may return results that include vendors of cameras and network cameras with publicly available Non-IP camera streams (i.e., on aggregation websites). Before adding a camera into the CAM2 database, it is necessary to determine whether the site updates the visual data frequently. Right now, this decision is made manually. To improve the success rate of such searches, the CAM2 team currently focuses on websites that aggregate traffic cameras from governments, for example, the New York City Department of Transportation website [9]. The website has a pop-up window for each snapshot as shown in Figure 5 (a). There is HTML text above and below the image data. Using this URL could hinder the retrieval of the camera snapshot, so the path to the raw image URL is needed. Most URLs that link directly to the image data end with an image file extension, as in Figure 5 (b). Many websites follow a convention for each camera's image path.

If the URL to the image data can be found, the next step is to determine if location information is also provided on the website. The information may appear in different places on the webpage. In Figure 5 (a), the HTML text at the top of the page provides the location information. For Figure 3 (d), the only location information provided is the name of the ski resort. It is important to review the overall structure of the website and determine what tools are necessary to obtain the relevant information. Some websites have easily identifiable HTML tags or an XML or JSON file associated with the camera data. If the website has all the camera data stored in one file, then adding the cameras is simple and can be done by parsing the JSON or XML file associated with the image data.

Obtain Location Information

If a website uses the Google Maps API showing the locations of cameras, the information may be obtained by analyzing the JSON or XML file. It is possible using the Chrome Developer Tools to view XML HTTP Requests (XHR), as shown in Figure 4. Some websites load many different XHR files; some sites load data from several JSON files into one map. If the JSON file containing the image data can be found, Python JSON mod-



Elements Console Sources Network Timeline Profiles Application Security Audits AdBlock Overrides

Preserve log Disable cache Offline No throttling

Regex Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Name: new-data.php?query=

Headers Preview Response Cookies Timing

```
▼{markers: [{id: "368", latitude: "40.79142677512476", longitude: "-73.93807411193848",…},…]}
  ▶markers: [{id: "368", latitude: "40.79142677512476", longitude: "-73.93807411193848",…},…]
```

Figure 4. *Finding XHR data using chrome developer tools*

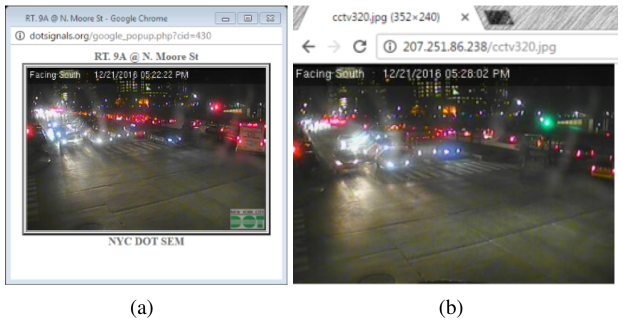

RT. 9A @ N. Moore St - Google Chrome
dotsignals.org/google_popup.php?cid=430
RT. 9A @ N. Moore St
Facing South 12/21/2016 05:22:22 PM
NYC DOT SEM
(a)
cctv320.jpg (352×240)
207.251.86.238/cctv320.jpg
Facing South 12/21/2016 05:28:02 PM
(b)

Figure 5. *Identifying the URL of the raw camera image data*

ule [17] is used to parse the JSON data and retrieve the location information. In the snippet of JSON code below, the latitude, longitude, and camera ID can be identified. If this information can be matched with the URL of the camera image stream, then all the necessary information has been obtained from the website and it can be added to the CAM2 database.

```
”id”:”745”,”latitude”:”40.743982”,
”longitude”:”-73.717583”,
”title”:”imagescamera1.png”,
”icon”:”imagescamera1.png”,
”content”:”Union tpke @ Little Neck Pkwy”
```

To match the URL of the image with the location information, the tools refer back to the page the image was displayed on, as shown in Figure 3. In this case, the ID shown in the JSON file can be matched with the description of the image in the pop-up window to retrieve the URL of the image. The tools can match the exact geographic coordinates of the camera in the JSON file to the image data shown in Figure 6.

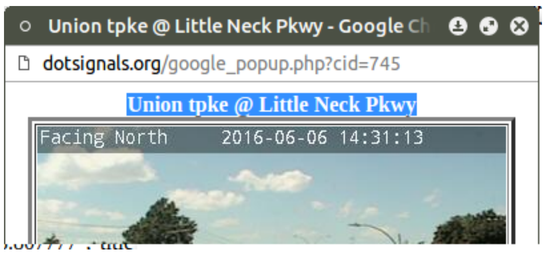

Union tpke @ Little Neck Pkwy - Google Ch
dotsignals.org/google_popup.php?cid=745
Union tpke @ Little Neck Pkwy
Facing North 2016-06-06 14:31:13

Figure 6. *Matching the JSON data to the image data*

If a website does not use JSON or XML to load the camera data onto a map, then other methods must be used to retrieve the location information. These methods are often less precise. Parsing the HTML pages can be more difficult because each page has unique structures. Fully understanding the directory tree of a site can expedite the process. For instance, the structure of the Alberta Department of Transportation website is outlined in Figure 7.

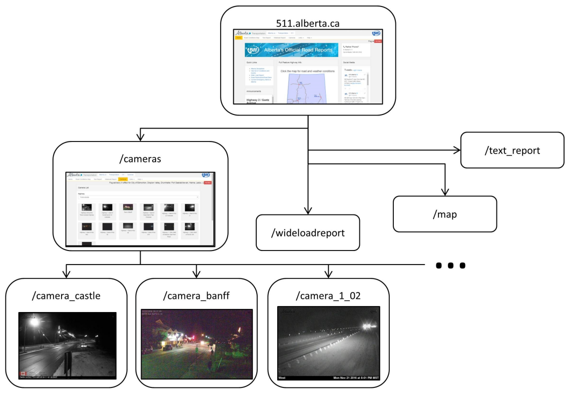
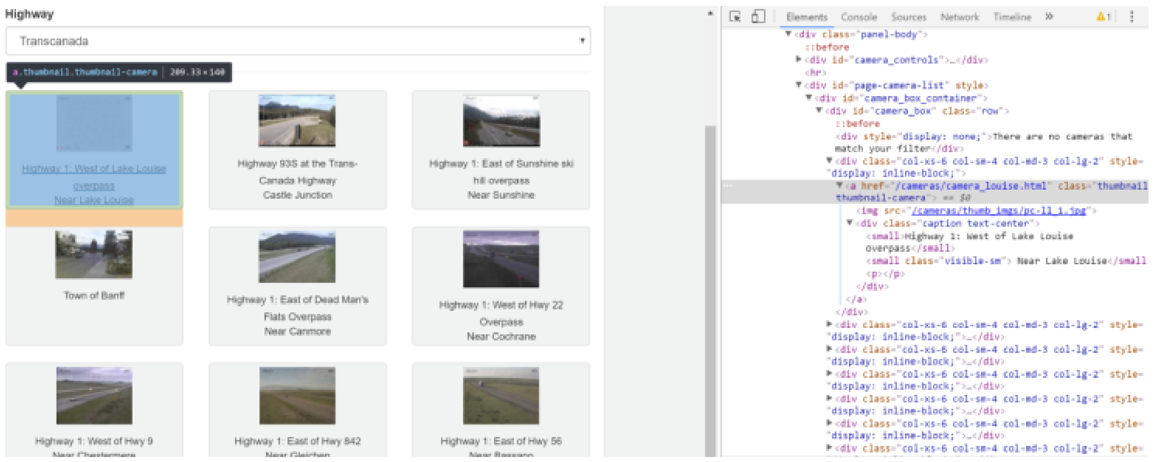


Figure 7. *Map of 511.alberta.ca website structure*

In addition to understanding the overall directory organization of the website, the HTML of each page containing relevant information must be reviewed. The hierarchical structure of HTML can make it difficult to locate desired information within the webpage. To navigate through the page structure in Figure 7, links must be identified between each page within the HTML page. Figure 8 is an example of how the Google Chrome Developer Tools may be used to identify page structure and determine how to navigate between webpages. The highlighted portion of the HTML source code contains the link to the camera page. Each page must be individually loaded to find the source URL of the camera image data.

After the HTML hierarchy of the website is understood and the tags containing pertinent information are identified, a Python script written in Selenium or BS4 navigates the website and pulls the relevant information. Once the information has been obtained from the website via the parsing of HTML or JSON files, the information from the entire website can easily be added to the CAM2 database.


Highway
Transcanada
a.thumbnail.thumbnail-camera | 209.33 × 140
Highway 1: West of Lake Louise overpass
Near Lake Louise
Highway 93S at the Trans-Canada Highway
Castle Junction
Highway 1: East of Sunshine ski hill overpass
Near Sunshine
Town of Banff
Highway 1: East of Dead Man's Flats Overpass
Near Canmore
Highway 1: West of Hwy 22 Overpass
Near Cochrane
Highway 1: West of Hwy 9
Near Chestermere
Highway 1: East of Hwy 842
Near Gleichen
Highway 1: East of Hwy 56
Near Bassano
Elements
Console
Sources
Network
Timeline

```
▼<div class="panel-body">
   ::before
  ▶<div id="camera_controls">…</div>
   <hr>
  ▼<div id="page-camera-list" style>
    ▼<div id="camera_box_container">
      ▼<div id="camera_box" class="row">
         ::before
         <div style="display: none;">There are no cameras that
         match your filter</div>
        ▼<div class="col-xs-6 col-sm-4 col-md-3 col-lg-2" style=
        "display: inline-block;">
         ▼<a href="/cameras/camera_louise.html" class="thumbnail
         thumbnail-camera"> == $0
             <img src="/cameras/thumb_imgs/pc-11_1.jpg">
           ▼<div class="caption text-center">
               <small>Highway 1: West of Lake Louise
               overpass</small>
               <small class="visible-sm"> Near Lake Louise</small
               <p></p>
             </div>
           </a>
         </div>
        ▶<div class="col-xs-6 col-sm-4 col-md-3 col-lg-2" style=
        "display: inline-block;">…</div>
        ▶<div class="col-xs-6 col-sm-4 col-md-3 col-lg-2" style=
        "display: inline-block;">…</div>
        ▶<div class="col-xs-6 col-sm-4 col-md-3 col-lg-2" style=
        "display: inline-block;">…</div>
        ▶<div class="col-xs-6 col-sm-4 col-md-3 col-lg-2" style=
        "display: inline-block;">…</div>
        ▶<div class="col-xs-6 col-sm-4 col-md-3 col-lg-2" style=
        "display: inline-block;">…</div>
        ▶<div class="col-xs-6 col-sm-4 col-md-3 col-lg-2" style=
```

Figure 8. *Identifying the HTML structure of a webpage*

Collecting Additional Metadata

After the essential information of each camera is obtained, several tools can be used to collect additional metadata about each camera. The information includes frame resolution (number of pixels), the refresh rate, and the amount of change in different frames to estimate the space needed for storage. This metadata can be collected by analyzing snapshots from each camera.

Calculate Frame Rates

To determine a camera's frame rate, a tool detects the changes between two adjacent frames. This may take from several seconds (for a camera with a high refresh rate) to a few hours (for a camera with a low refresh rate). The results can also vary significantly due to network delays or the aggregation sites being too busy.

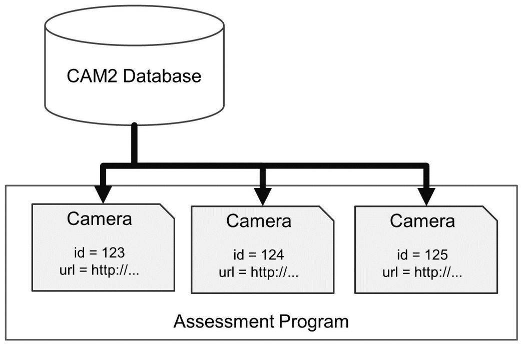


Figure 9. *Camera information is loaded from the CAM2 database to a Python object in the frame rate assessment program*

Figure 9 shows the first step in the frame rate aggregation process. The program obtains the information about a list of cameras from the database and retrieves snapshots from these cameras. The time to download one image from each camera could vary and affect the accuracy of the calculation of frame rates. Also, this program goes through the list sequentially. To improve accuracy, cameras are moved from the cameras list to a separate list called activeCameras if the response time is sufficiently short. Only snapshots from cameras in the activeCameras list will be

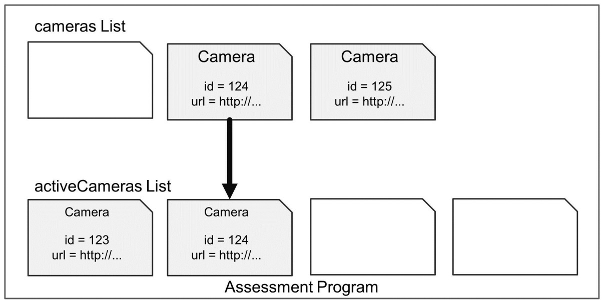


Figure 10. *The camera list is populated with cameras from the database until it has reached capacity*

downloaded and compared to estimate the refresh rates, as illustrated in Figure 10.

The program makes an HTTP request to the web server hosting the camera image data. Figure 11 shows the first image downloaded for each network camera. This file is called the *Reference Image*. If the request is successful, the program moves the camera to the activeCameras list. The program begins downloading more images via another HTTP GET request. Figure 12 shows how the next image (called the *Starting Image*) is downloaded.

The Python file comparison library *filecmp* is used to compare the *Reference Image* with the *Starting Image*. It compares only the checksum of the two image files. If the two image files are identical, the program will continue to download new *Starting Image* files for that camera until it finds the image file on the provider server has changed (i.e., the *Reference Image* and the *Starting Image* are not identical). Once a *Starting Image* has been downloaded and it is different from the *Reference Image*, the program will cease getting *Starting Images* for that camera. The time that this change occurred is recorded by the program and it will then begin to get the final image needed for a successful frame rate analysis.

The final image necessary to determine the frame rate is called the *Ending Image*. The *Ending Image* is found using a method identical to the way the *Starting Image* was found, except the *Ending Image* is compared to the *Starting Image* rather than the *Reference Image*. Once the *Ending Image* has been deter-

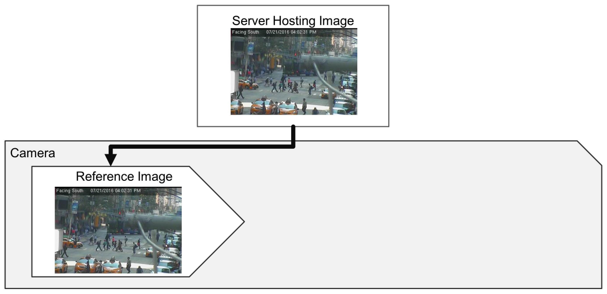


Figure 11. *Using and HTTP GET request, the first image (Reference Image) is downloaded from the network camera's web server*

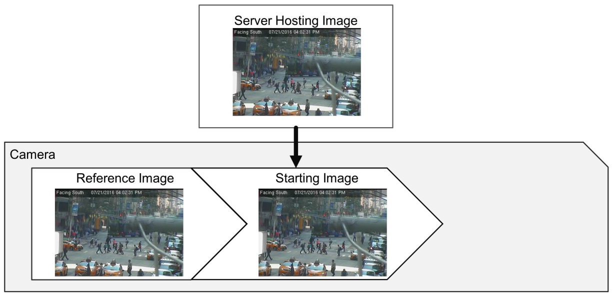


Figure 12. *The second image (Starting Image) is downloaded from the network camera's server using an HTTP GET request*

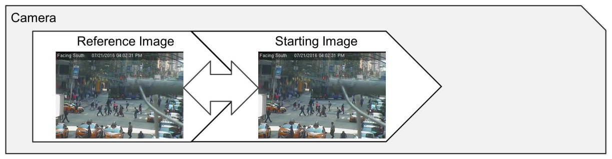


Figure 13. *The Starting Image is compared to the Reference Image using the Python filecmp module*

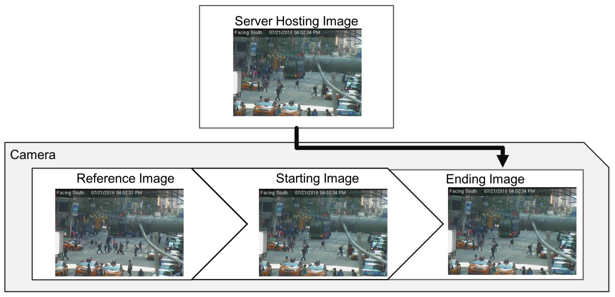


Figure 14. *The Ending Image is downloaded in the same way as the Starting Image and the Reference Image*

mined, the frame rate of the network camera can be determined by subtracting the timestamp of the *Ending Image*s from the timestamp of the *Starting Image*.

Evaluation

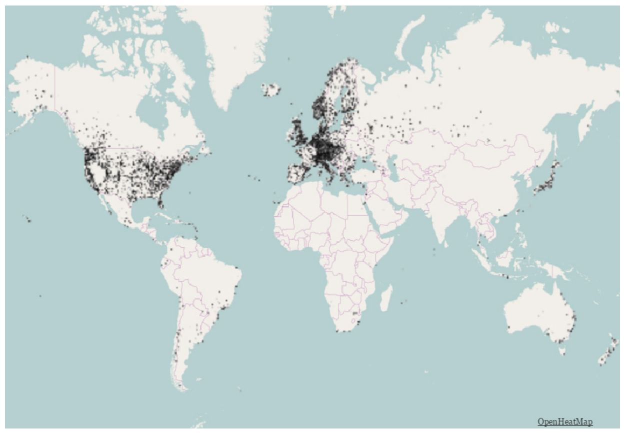


Figure 15. *Locations of network cameras in CAM2 database*

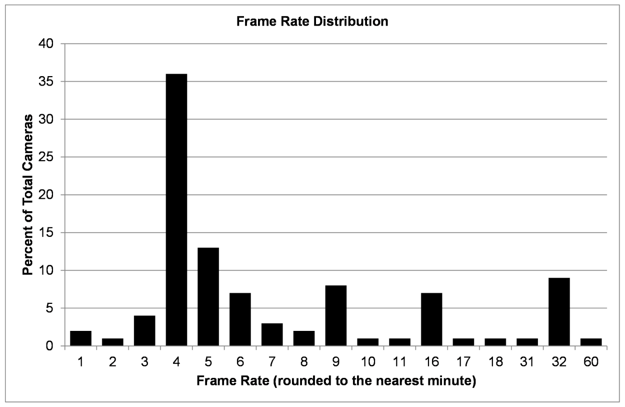


Figure 16. *Percent of cameras with successfully assessed frame rates rounded to the nearest whole minute*

The map in Figure 15 displays the locations of the cameras in CAM2. Currently, CAM2 contains over 119,000 cameras located in 162 countries around the world; approximately 53.8% of them are in the United States. Figure 16 shows the distributions of refresh rates.

The CAM2 system has been able to capture images from a wide range of events and natural disasters including the Houston flooding disaster pictured in Figure 1 and the Alberta wildfire in Figure 2. These images show the capacity of the CAM2 camera database to capture image data from both remote and urban environments.

Figure 16 shows the results of analyzing the frame rates of several thousand US cameras from the CAM2 database. The data was collected for at least 4 runs of the frame rate assessment program over several weeks. The data shows that around 36% cameras update frames every 4 minutes.

The method may encounter high variations in the frame rates. Figure 17 plots the frame rate analysis data from four sites with network cameras. Each source has dozens of cameras, The

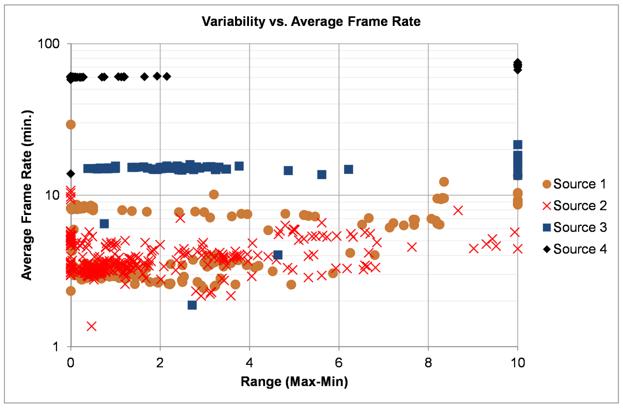


Figure 17. *Range vs. average frame rate across all assessments for each camera in four unique sources*

x-axis of the graph shows the range (difference between the maximum and minimum frame rate value found during all assessments for the camera). The y-axis is the average value for the frame rate found during all assessments.

Each site has a distinct grouping along the frame rate axis (y-axis). An estimate of the true frame rate for each source can be determined using Figure 17. From this figure, we can determine that the network cameras in Source 4 have a frame rate of about 60 minutes and that the cameras in Source 3 have a frame rate around 15 minutes. For Source 1, two distinct groups can be identified: one group around 8 minutes and another group around 2 minutes. Figure 18 shows another example of a network camera source with distinct groupings.

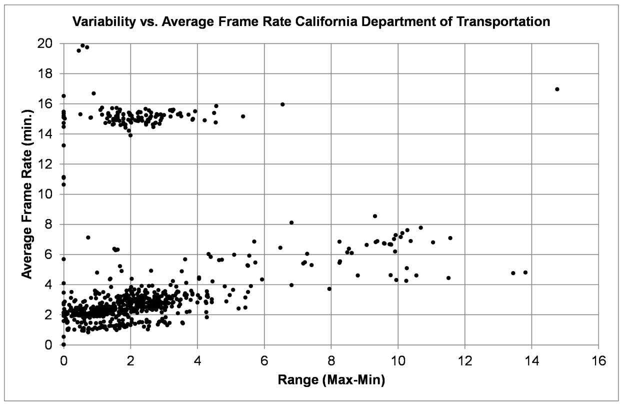


Figure 18. *Distribution of frame rate data for one department of transportation*

Figure 18 is similar to Figure 17 but this data is taken from frame rates of the traffic cameras from one department of transportation. Of the 1,117 cameras from this source, the frame rates of 753 (67%) were able to be accurately obtained using the frame rate analysis program. The frame rates for the other 300 cameras could not be found because not enough successful attempts were made to find the frame rate or no data could be retrieved from the camera. Two distinct groups of cameras can be identified from Figure 18. The first group has an average frame rate around 15 minutes and the second group has an average frame rate around 3 minutes. In Figure 19, we can see that the two groups each have distinctly different snapshots. Figure 19 (a) is a snapshot taken from the group with frame rates around 15 minutes and Figure 19 (b) is a snapshot from the group around 3 minutes.

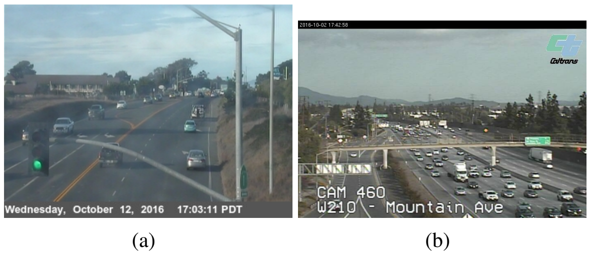


Figure 19. *Examples of the different cameras used by one department of transpiration website*

Conclusion

This paper presents the method to build a large (possibly the largest in the world) camera network using publicly available data from network cameras. The paper explains how to discover network cameras, retrieve relevant information about these cameras, and download visual data from these cameras. CAM2 currently has more than 119,000 network cameras and the number keeps growing. CAM2 is a collection of tools available for researchers. Interested readers may register at `https://cam2.ecn.purdue.edu/`.

References

[1] Niall Jenkins. 245 million video surveillance cameras installed globally in 2014. `https://technology.ihs.com/532501`, June 11 2015.

[2] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, July 2006.

[3] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, Aug 2013.

[4] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, Jan 2013.

[5] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html`, Feburary 2016.

[6] Cisco. Service Provider Forecasts and Trends. `http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html`.

[7] A. S. Kaseb, Y. Koh, E. Berry, K. McNulty, Y. H. Lu, and E. J. Delp. Multimedia content creation using global network cameras: The making of CAM2. In *IEEE Global Conference on Signal and Information Processing*, pages 15–18, Dec 2015. `https://cam2.ecn.purdue.edu/`.

[8] Houston Transtar. `http://www.houstontranstar.`

org/.
[9] New York City Department of Transportation. Real time traffic cameras. http://dotsignals.org/.
[10] Snoweye. Webcams from ski resorts around thCameras. https://www.snoweye.com/.
[11] Weather Underground. Weather Webcams. https://www.wunderground.com/webcams/.
[12] http://www.webcams.travel/.
[13] Selenium - web browser automation. http://www.seleniumhq.org/, 2016.
[14] Leonard Richardson. Beautiful soup. https://www.crummy.com/software/BeautifulSoup/, dec 2016.
[15] Ariya Hidayat. Phantomjs. http://phantomjs.org/, 2016.
[16] The google maps geolocation api — google maps geolocation api — google developers. https://developers.google.com/maps/documentation/geolocation/intro, nov 2016.
[17] Python Software Foundation. 18.2. json - json encoder and decoder - python 2.7.13 documentation. https://docs.python.org/2/library/json.html, dec 2016.

Author Biography

Ryan Dailey is a Junior in the Electrical and Computer Engineering department at Purdue University and is also the leader of the CAM2 Database Team.

Ahmed S. Kaseb is an assistant professor in the Computer Engineering Department, Faculty of Engineering, Cairo University. He obtained the Ph.D. in computer engineering from Purdue University in 2016. He obtained the M.S. and B.E. in computer engineering from Cairo University in 2013 and 2010 respectively. He is conducting research in Cloud Computing and Big Data in order to enable cost-effective large-scale real-time analysis of image and video streams from worldwide distributed network cameras.

Chandler Brown is an undergraduate student in Computer Engineering at Purdue University.

Sam Jenkins is an undergraduate student in Computer Engineering at Purdue University.

Sam Yellin is currently a Junior in Computer Engineering at Purdue University. He spent one year at UTC Aerospace through the Purdue co-op program and is currently an avionics software engineering intern at Blue Origin in Seattle, Washington.

Yung-Hsiang Lu is an associate professor in the School of Electrical and Computer Engineering and (by courtesy) the Department of Computer Science of Purdue University. He is an ACM distinguished scientist and ACM distinguished speaker. He is a member in the organizing committee of the IEEE Rebooting Computing Initiative. He is the lead organizer of Low-Power Image Recognition Challenge, the chair (2014-2016) of the Multimedia Communication Systems Interest Group in IEEE Multimedia Communications Technical Committee.

Acknowledgments

This project is supported in part by NSF ACI-1535108. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the National Science Foundation. We want to thank all organizations that provide the visual data for this project. This project receives cloud research credits from Amazon and Microsoft.