

Panoptes: Servicing Multiple Applications Simultaneously using Steerable Cameras

Shubham Jain
WINLAB, Rutgers University
shubhamj@winlab.rutgers.edu

Marco Gruteser
WINLAB, Rutgers University
gruteser@winlab.rutgers.edu

Viet Nguyen
WINLAB, Rutgers University
viet.nguyen@rutgers.edu

Paramvir Bahl
Microsoft Research
bahl@microsoft.com

ABSTRACT

Steerable surveillance cameras offer a unique opportunity to support multiple vision applications simultaneously. However, state-of-art camera systems do not support this as they are often limited to one application per camera. We believe that we should break the one-to-one binding between the steerable camera and the application. By doing this we can quickly move the camera to a new view needed to support a different vision application. When done well, the scheduling algorithm can support a larger number of applications over an existing network of surveillance cameras. With this in mind we developed *Panoptes*, a technique that virtualizes a camera view and presents a different fixed view to different applications. A scheduler uses camera controls to move the camera appropriately providing the expected view for each application in a timely manner, minimizing the impact on application performance. Experiments with a live camera setup demonstrate that Panoptes can support multiple applications, capturing up to 80% more events of interest in a wide scene, compared to a fixed view camera.

CCS CONCEPTS

•Computer systems organization →Sensors and actuators; Real-time system architecture; •Information systems →Mobile information processing systems;

KEYWORDS

Camera, Scheduling, Virtual views, Smart infrastructure, Video analytics, Pan-Tilt-Zoom

ACM Reference format:

Shubham Jain, Viet Nguyen, Marco Gruteser, and Paramvir Bahl. 2017. Panoptes: Servicing Multiple Applications Simultaneously using Steerable Cameras. In *Proceedings of The 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, Pittsburgh, PA USA, April 2017 (IPSN 2017)*, 12 pages.
DOI: <http://dx.doi.org/10.1145/3055031.3055085>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IPSN 2017, Pittsburgh, PA USA

© 2017 ACM. 978-1-4503-4890-4/17/04...\$15.00

DOI: <http://dx.doi.org/10.1145/3055031.3055085>

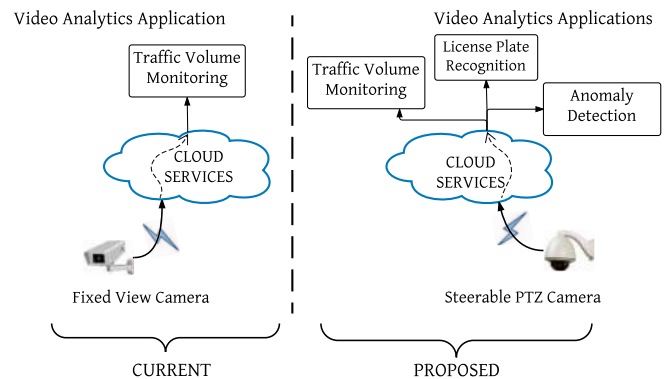


Figure 1: Cloud analytics platform for cameras.

1 INTRODUCTION

Surveillance cameras are becoming ubiquitous. The United Kingdom houses five million surveillance cameras, many of them in London, which has one surveillance camera for every 11 people [1, 2]. Networked cameras cover key areas of highways, they are mounted on adaptive traffic signal systems for traffic light control, and they are increasingly deployed in our offices and homes (e.g., Nest-Cam [3]). Many of these cameras can be remotely controlled over the Internet, feeding live streams to a distant cloud. This then, creates an opportunity to harness such cameras for additional vision analytics applications that gather data far beyond the original intended purpose of a deployed camera.

Outdoor surveillance cameras can be used for a wide range of analytic tasks, such as traffic volume measurements, behavior mapping, amber alert scanning, pedestrian monitoring, and traffic violations. Some of them are shown in Fig 2. Similarly, security cameras can be used for pet overseeing, security monitoring, baby activity etc. The ability to add analytics apps to existing camera feeds could contribute much to satisfy the data demands of future envisioned smart home and smart city applications. We therefore envision a cloud-based platform that allows adding many diverse analytics applications to existing cameras and support them in a concurrent manner.

Supporting multiple simultaneous fixed view analytics applications on the same camera often creates challenges because their view and image requirements tend to differ. Hitherto cameras are often installed with a specific application in mind and their view

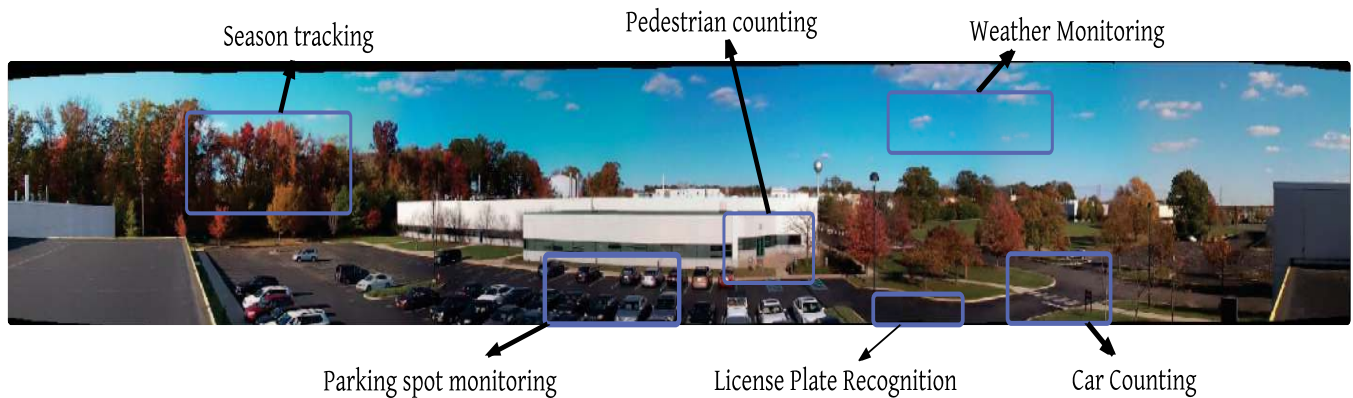


Figure 2: A 180 degree panorama from our PTZ camera, displaying the scene and regions of interest for potential applications.

(i.e., position, orientation) is carefully adjusted to fit this application, as shown on the left in Fig 1. Technology trends are leading to more flexible, steerable camera designs, enabling traffic operators to steer them occasionally to look for other events of interest. Even though such cameras could in theory satisfy all applications, letting applications directly issue steering commands would likely lead to conflicts. Moreover, such steering would need to keep up with steering and network latencies.

We propose Panoptes,¹ a view virtualization system for (electronically or mechanically) steerable cameras. We develop a mobility-aware scheduling algorithm for steering control, thus enabling multiple applications to be supported simultaneously on a single camera, as in Fig 1. Each application can specify its view requirements (e.g., orientation, resolution, zoom) and the system provides it with a view that meets them. We refer to these views as virtual views. The goal of virtual view abstraction is to make steering changes transparent to applications. This is based on the insight that temporary steering away of a camera can be easily masked by replaying the previous virtual view image, if no significant change occurs in the view at that time.

To know when to steer, our proposed system learns mobility patterns in its view and predicts when motion or change is likely in each virtual view. Traffic scenarios, for example, often exhibit fairly regular, constricted motion patterns (a car’s movement follows the roadway, which allow predicting when the car will enter a region of interest). We benefit from the continuous form of this motion to learn mobility patterns in the scene for scheduling steering actions.

To summarize, we make the following contributions:

- a virtual view abstraction for sharing a camera among multiple applications with fixed view requirements.
- a mobility-aware scheduling algorithm that anticipates object velocity, network latency, and steering speed before steering the camera to the application specific view (camera position). The scheduler maximizes the number of applications served while minimizing the number of events missed.

¹Panoptes, meaning all-seeing, is named after the 100-eyed Argus Panoptes in Greek mythology.

We evaluated the system with a total of 2870 hours of video collected over 5 months and demonstrate that Panoptes can support multiple applications on a steerable camera, capturing up to 80% more motion events, compared to commonplace fixed view cameras.

2 RELATED WORK

The space of steerable cameras predominantly focuses on tracking moving objects, especially people [4, 5]. These cameras often follow one target or rotate through preset positions to capture more targets. However, the camera view is tightly bound to one specific application. Ilie et al [6] proposed an active camera control technique, which explores the space of camera configurations and cluster regions of interest. This clustering is done based simply on proximity and does not take expected motion into account.

The most closely related work is MultiSense [7], which also seeks to support multiple applications on a steerable camera. MultiSense, however, addresses a different application domain: applications that have been designed for steerable cameras and seek to steer the cameras themselves. It interleaves steering requests from these applications. The system then focuses on resolving conflicts between commands issued by different applications, and allocating the sensor resource. MultiSense implemented an application independent weight-based fair sharing scheme for PTZ cameras. In contrast, we focus on accommodating multiple fixed-view applications on a steering camera, which we believe represents the vast majority of vision analytics applications. Fixed-view applications were usually developed for regular cameras and do not issue steering commands. Keeping this in mind, Panoptes adds a layer of abstraction and conceals steering actions from the applications. This allows us to use any conventional vision algorithms in a plug-and-play manner. It also provides an opportunity for the system to steer camera to maximize capture of motion.

Panoptes also proposes view virtualization, which generates candidate camera views that can sometimes support multiple applications simultaneously by accommodating regions of interest of different applications in one camera view. MultiSense, in comparison, can only accommodate one application at a time through virtual sensor and multiplexing, where each camera position allows capture for one application.

Many important contributions in vision analytics come from the computer vision community, that has long propounded detection and recognition algorithms in surveilled environments. Automated vehicle tracking and classification [8, 9], anomaly detection [10], pedestrian tracking [11], face recognition [12] have all been widely explored and enhanced over the years. However, these algorithms are often designed for unchanging camera views, and their performance is severely affected if the camera moves. Panoptes enables state-of-art vision algorithms to run concurrently on a single camera, and provides them with expected unchanging views through the concept of view virtualization.

There are also multiple works on multi-camera surveillance networks. However, they usually coordinate to support only one application, such as tracking people [13], identifying people [14, 15], or object-tracking [16]. Qureshi et al. [17] suggest greedy algorithms such as weighted round robin to distribute targets to different cameras. Yao et al. [18] propose an adaptive camera assignment algorithm for assigning resources to objects for a uniformly distributed computational load. For omnidirectional cameras, Chen et al. [19] quantitatively formulate the effectiveness of a camera in observing the tracked object and use this metric to decide hand-off between cameras. Panoptes is different in utilizing motion patterns to schedule diverse applications.

3 BACKGROUND

Many modern cameras are steerable, either through a mechanical or an electronic mechanism. Mechanically steerable cameras, more commonly known as Pan-Tilt-Zoom (PTZ) cameras, as shown in Fig 3, are frequently used in the surveillance realm. They use motors to rotate the camera along the horizontal and vertical axes (pan and tilt), and adjust the zoom lens, therefore allowing optical zoom. However, they can only observe events in one direction at a time. Mobility-aware scheduling can thus schedule when and where the camera is pointed at any time.

Intriguingly, image sensor resolution (e.g., hundreds of megapixels [20]) and processing advances have led to the availability of omnidirectional, or 360° cameras. The lens and image sensor in some of these cameras are designed to cover a wide view, frequently a 360° view through a fisheye lens, with no ability to zoom. These cameras place much higher requirements on the quality of the lens and it is difficult to match the quality of an optical zoom in motorized PTZ cameras.

They are not electronically steerable and thus mobility-awareness may not be very useful to these cameras. Although popularly, 360° cameras are constructed by mounting multiple cameras on a rig, pointing in various directions, where each has the ability to zoom. These are, in-effect, electronically steerable cameras. They do not require any moving parts, which simplifies construction, deployment, and eliminates mechanical wear. They also offer near-zero camera steering latency since



Figure 3: Mechanically steerable cameras' 3 degrees of freedom.

there is no need to wait until the motor has moved the camera and overall one can expect such designs to lower costs and significantly increase the usage of steerable cameras.

By virtualizing views from the panoramic scene for each application, a 360° camera can contrive electronic steering by extracting and transmitting only the virtual views over the network, thus making the system bandwidth efficient. Mobility-awareness can enable the 360° system to control the precise camera needed for conducting specific operations such as zoom and input resolution for an application.

For a readily deployable and practical system, we prototype and demonstrate our system on existing mechanically steerable PTZ cameras. However, the proposed work is generic in its applicability to both, mechanically (PTZ) and electronically (360) steerable cameras.

Existing control standards. While steerable cameras are usually manually steered via a joystick by the operator, they do offer standard network APIs that are also suitable for automatic control. The Open Network Video Interface Forum, or ONVIF [21], is a global open industry forum that standardizes communication between IP-based security products, and has defined a standard for controlling steerable cameras. ONVIF is built upon Web service standards. It uses Simple Object Access Protocol (SOAP) for message transfer, WSDL (Web Service Definition Language) for describing the service and XML for data description syntax. Some manufacturers also provide their proprietary SDK (e.g. [22]) for controlling additional camera parameters. However, neither ONVIF [23] nor the proprietary APIs automate camera steering to facilitate support for multiple concurrent applications.

Analytics Use Cases. While there are many potential use-cases of analytics with steerable cameras spanning residential, commercial, and public settings, we consider a concrete example in a traffic related-setting.

Traffic Volume Monitoring is useful for traffic and parking planning, traffic light timing, and real-time traffic or parking information services. Typically car counting algorithms [8, 24, 25] track key features along the entry point and the exit point of a route [26].

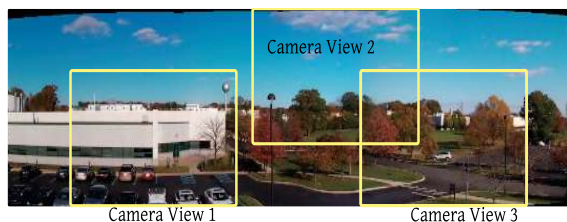
License Plate Scanning is increasingly deployed for applications such as toll booth payment, amber alert scanning, speed monitoring systems etc. Typical LPR vision algorithms scan the image frame to locate a license plate and runs character recognition techniques [27, 28] on the small region within the detected plate.

No Turn on Red prevents drivers from turning right when the traffic signal is red. It is a violation increasingly being caught by the use of auto-enforcement cameras. This application need not run continuously, but only when the traffic light is red.

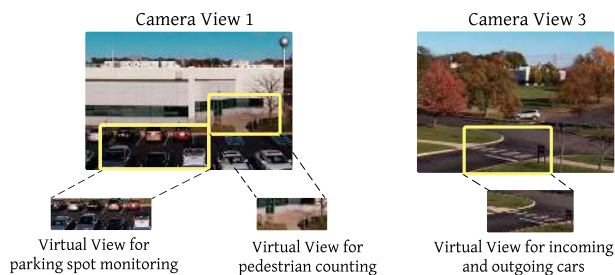
While such vision algorithms can seemingly be run in parallel on the same camera-feed, several challenges exist in practice. We discuss them next.

4 CAMERA ANALYTICS CHALLENGES

Cloud computing services can easily provide processing resources for many analytics applications, but the use of multiple analytics applications on the same steerable camera feed is challenging due to the following factors.



(a) Panoramic view of partial camera coverage. Camera can only observe a small area at a time, known as the Camera View.



(b) Extracting Virtual Views from different camera views.

Figure 4: Virtual View abstraction from scene.

Sensitivity to view changes. The setup of many existing vision systems is view-specific. Even slight view changes through steering could bring defined regions of interest such as the inboxes and outboxes in traffic volume monitoring out of alignment with the roadway.

Conflicting view requirements. Vision applications are frequently interested in different parts of the panoramic view. Volume monitoring requires a wider view that contains a path from an inbox to the outbox. License plate recognition requires a zoomed view that focuses on area where the license plate of a vehicle is likely to be oriented towards the camera. In many cases, these requirements conflict, meaning that they cannot be achieved with one camera orientation and zoom level.

Slow camera tuning speeds. Mechanically steerable cameras have an inherent latency due to the time taken by the camera to pan, tilt or zoom. Most modern cameras pan at a speed of $100^\circ/\text{second}$, and capture images during the transitions, but for the most part, vision applications cannot use these blurry frames.

Variable network latency. Steering commands from a remote location will reach the camera after a variable and potentially significant delay, particular for the emerging LTE connected cameras. LTE measurements show average latencies of 70ms but frequent spikes to 200ms and beyond [29].

5 SYSTEM DESIGN

We propose a system-level approach to the aforementioned challenges, that aims to support multiple applications simultaneously by steering the camera towards expected motion. Rather than exposing camera steering control and associated latencies to each application, the system makes steering transparent to applications: it aims to present an unchanged virtual view to each application even when the camera has moved. By creating separate unchanging virtual views for each application, it addresses application's sensitivity to view changes. The system then exploits mobility awareness to manage conflicts as well as mask network and steering latencies. This means that the system learns where to expect mobility and seeks to steer the camera in time to observe motion events.

View Virtualization. A *Virtual View* is an application-specific abstraction of the camera view. It is defined by

$$VV = \{priority, ROI, R, fr\} \quad (1)$$

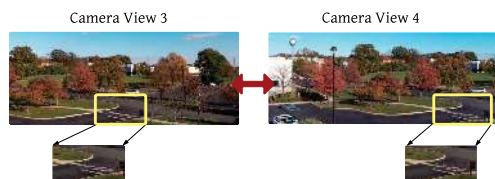


Figure 5: Virtual View mapping across camera views for car counting application.

Traditionally, one camera is bound to one application. Panoptes associates every application to a corresponding virtual view, defined by its requirements at initialization. It is assigned a priority and a region of interest in the scene, its *ROI*. The application also specifies a required resolution, *R*, and an acceptable frame rate, *fr*.

Typically, the virtual view for an analytics application is simply the view that the camera would be steered to by the operator if this application were running in isolation. Many vision applications only use a smaller part of the camera view, a region of interest (*ROI*). For those applications, a virtual view can be defined as a rectangular subregion of a camera view. Such views are virtual because the actual camera position may not be completely aligned with this view. If the actual view encompasses the virtual view, the system simply extracts the virtual view region from the actual camera view and presents this to the application (as long as the resolution requirements can be met), shown in Fig 4b. As a camera pans or tilts, the system updates the region to be extracted so that the extracted virtual view remains unchanged, as demonstrated in Fig 5.

Virtualization becomes more challenging when several virtual views do not completely fit in a camera view (or the zoom factor cannot satisfy resolution requirements). In this case, the system scheduler seeks to intelligently steer the camera to still maximize the capture of events of interest in all views. This is based on predicting when events (motion) occurs in a camera view. Since there exists a risk that events of interest will be missed, application developers can select whether this process should be transparent to the vision application or whether the application should be notified when the virtual view is not actively monitored by the system. The first option is important to support legacy applications. The

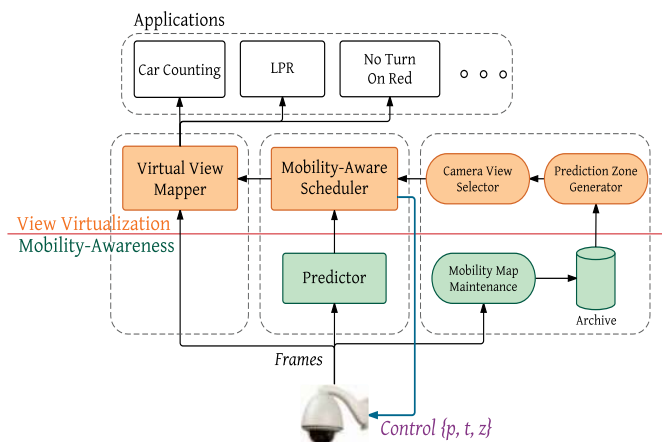


Figure 6: System Overview.

second allows more sophisticated applications better estimate the uncertainty inherent in their analytics results.

Scheduling Views with Motion Prediction. Often, applications are interested in video sequences where motion occurs in the view. In most scenes, the degree of motion determines how frequently the view needs to be updated. For example, if a camera monitors a parking lot, it could turn away without any loss of information, when no vehicle or person is approaching. Any missing static parts of the video can be easily reconstructed from a single image or a short video sequence. This insight then allows the system to choose what part of the scene a camera must capture, when there are no simultaneous motion events across those views. If the system can accurately predict when a moving entity will enter a virtual view, it can steer the camera just in time to capture this motion event. We therefore propose that cameras learn mobility patterns in their scene and use this awareness to predict motion events.

System Overview. While our proposed camera control system works for both, mechanically and electronically steerable cameras, we build our prototype using PTZ cameras as they are a large part of the existing infrastructure across countries. At different camera positions, these cameras view different regions of the scene. We call these *camera views*, rendered on a panoramic view in Fig 4a. Each camera view may have one or more virtual views for different applications, as illustrated in Fig 4b. Our system consists of two primary components: *View Virtualization* and *Mobility Awareness*. The key processing steps are depicted in Fig 6. The *virtual view mapper* extracts an application’s virtual view from the current camera view and passes it to the analytics application.

Meanwhile the *predictor* computes location estimates for moving entities in the scene and provides these predictions to the *scheduler*. The scheduler steers the camera to maximize motion capture. The virtual view mapper obtains the current camera position from the scheduler to accurately map raw frames to virtual views for applications. During the initialization phase, the system learns common trajectories from the scene and stores them as a mobility map, providing information on motion distribution in the scene. Mobility map is used by the *Prediction Zone Generator* to identify

prediction zones corresponding to virtual views. This information is then passed on to the *Camera View Selector* for determining candidate camera views, and aid in mobility-aware scheduling. We discuss the details of these components in the following sections.

6 MOBILITY AWARENESS

In this section, we present the details of mobility-awareness in steerable cameras.

6.1 Learning Mobility Patterns

Most environments have common motion patterns, specific to the scene being observed. We discuss below how we learn these patterns.

6.1.1 Sensing motion paths. The tracking module tracks all active entities in the scene and learns their trajectories. This is performed during the initialization phase, and continuously updated during camera operation. Each trajectory is initialized by the initial position of the object. This position is determined by detecting foreground blobs in the image sequence. The detected blobs are tracked using a Kalman Filter. The efficiency of a Kalman filter in estimating trajectories of multiple moving targets has well been established in the vision community [8, 30], and it enables us to handle brief occlusions. We can further improve this by replacing it with more sophisticated tracking schemes. These techniques exploit the spatio-temporal context information along with motion constraints, significantly promoting the robustness of tracking in the presence of longer occlusions. The detected trajectories are archived by the system.

6.1.2 Mobility Maps. We define a *mobility map* as a two dimensional distribution of motion in the scene. Fig 8b shows the mobility map generated over a scene. A scene is defined as the maximum possible panoramic view of the camera. We create the mobility map by dividing the entire scene in discrete cells. Each pixel in the camera image belongs to one cell in the grid. We empirically chose square cells of sixty pixels each. By probing into the history of trajectories, we assign each cell a *motion quotient*, Q_m , which quantifies how much of the motion in the scene transpires in that cell. For any cell k , it is defined as:

$$Q_m(k) = \frac{\text{Motion events in the cell } k}{\text{Motion events in the scene}} \quad (2)$$

Here, motion events correspond to frames with moving objects. Our objective is to maximize motion capture from the scene, and this metric helps us distinguish between regions of high motion events from those with relatively fewer motion events. The *ROI* for any virtual view is defined as the set of cells in that virtual view. Additionally, every virtual view also has a motion probability attribute, p_m . It is the sum of the motion quotients of all cells in the virtual view, and is given by:

$$p_m = \sum_{k \in C} Q_m(k) \quad (3)$$

where C is the set of cells in that virtual view.

6.1.3 Motion Prediction. Camera settings, such as pan-tilt-zoom cannot be changed instantaneously. To capture motion in a virtual view, it is important to have prior information of ensuing motion

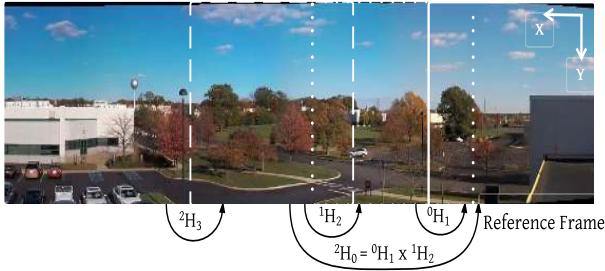


Figure 7: Coordinate frame mapping using homography.

events, to adjust camera settings for a future time instant. The predictor is implemented as a decision tree based on the following spatio-temporal features: the current location of the object, speed, and heading. Instead of looking at an entire sequence, we only consider the speed and heading in the last transition, i.e. from the previous cell to the current cell. For every position sample obtained from the tracking module, the predictor probes the mobility map database for all trajectories that match the three features. We calculate speed as the number of pixels traversed by the object from one frame to the next. For those trajectories that most closely match the criteria, the predictor looks up the cell where majority of the matched tracks appeared after time t_{la} . We call this the lookahead time. The resultant cell number is returned to the scheduler as the predicted position. If the predicted cell belongs to a virtual view, the scheduler acts accordingly. This is an empirical data-driven model and is independent of the scene. Thus it can be applied to any traffic environment. In scenarios with slower objects, such as people, where motion is less regular, Panoptes could be adapted by shortening the look ahead time.

6.2 Identifying Prediction Zones

We introduce prediction zones as substitute views for a camera to monitor. In the event where a virtual view is not in the camera’s view but a prediction zone is, a motion event in the prediction zone can trigger camera control. Prediction zone is defined as the subset of cells capable of predicting motion in a virtual view. Inclusion of prediction zones in a camera view enables the camera to capture motion even for virtual views that may not be in the current camera view. We use the lookahead time, t_{la} for predicting whether an object will be in a virtual view at the end of that time. Analogously, any cell where an object was t_{la} seconds before appearing in the virtual view, is likely to trigger prediction for that virtual view. Typically, the approach area leading up to a virtual view constitutes its prediction zone. For each virtual view, we deduce the prediction zones from the history of trajectories. We construct a View Table, where every virtual view is associated with its probability of motion p_m , and prediction zones PZ :

$$\text{View Table} : VV \rightarrow PZ, p_m \tag{4}$$

Each prediction zone also has a motion probability $p_{z,m}$, computed as in Equation 3.

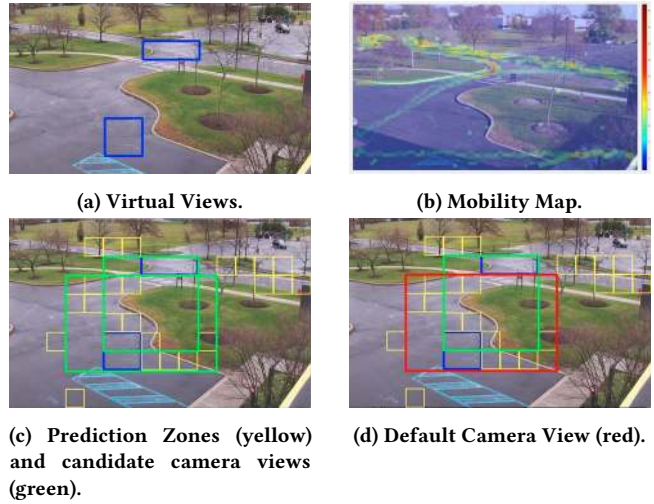


Figure 8: Camera View Selection steps (Snapshots from our implementation).

6.3 Mapping Virtual Views

To ensure that applications are transparent to camera steering, the system must successfully map virtual views from a camera view to any other camera view as the camera steers. While coordinate frame mapping is one of the oldest ideas in computer vision, previous proposals either (i) map the scene to camera coordinates, or (ii) require complete camera calibration. The camera coordinate system alone cannot achieve this mapping, specially for mechanically steerable cameras with moving parts. Minor manufacturing defects can cause errors that accumulate over time and can gravely affect system performance. Moreover, steerable cameras are equipped with a wide-angle lens that suffers from high lens distortion, which varies from camera to camera. For a truly scalable surveillance system, calibrating each camera or relying on the numbers provided by the manufacturer is not only an inconvenience, but also impractical.

In contrast, we take advantage of homography [31] to map the entire scene to a global reference frame. This approach works for all steerable cameras, irrespective of the manufacturer, and scales easily. Homography is the projective transformation from one image to another, and can be used efficiently to compute camera rotation, and relate one camera view to other camera views. We use the view from a predefined position of the camera as the reference frame and map all points in subsequent frames to the reference frame, frame 0. Homography matrices, 3×3 matrix H , can be concatenated to relate points in current frame to points in the reference frame. 0H_n is the concatenation of intermediate homographies between frame n and frame 0, given by

$${}^0H_n = {}^0H_1 \times {}^1H_2 \times \dots \times {}^{n-1}H_n. \tag{5}$$

Fig 7 demonstrates this mapping from one frame to another. We compute the homographies during camera initialization by using the SURF [32] feature detector to detect keypoints in frames. By using features from the scene, the system can adaptively update

```

Data: Res - Camera View Resolution
Data: n - Number of applications
Result: CamView[] - Candidate Camera Views
begin
  VV ← createVirtualViews(n)
  foreach v ∈ VV do
    candidateView.add(v)
    for i ← 2 to n do
      B = getBoundingBox(v, VV(i))
      if B.dimensions < Res then
        candidateView.add(VV(i))
        v ← B
      else
        PZ = getPredictionZones(VV(i))
        foreach pz ∈ PZ do
          Bpz = getBoundingBox(v, pz) if
            Bpz.dimensions < Res then
              candidateView.add(pz)
              v ← Bpz
            end
          end
        end
      end
    end
  CamView.add(candidateView)
end

```

Algorithm 1: Finding Candidate Camera Views.

```

Data: CamView[] - Candidate Camera Views
tla - Prediction Look Ahead Time
begin
  CamViewcurrent ← CamView(0)
  foreach frame do
    S ← getCurrentObjectPosition(frame)
    foreach s ∈ S do
      | future[s] = predict(s, tla)
    end
    destination = maxPriority(future)
    cv = getCameraView(destination)
    CamViewcurrent ← cv
    startTimer();
    if timerEnd then
      | CamViewcurrent ← CamViewnext
    end
  end
end

```

Algorithm 2: Mobility-Aware Scheduling.

homographies and accurately map points from any view to the reference frame, eliminating any errors accumulated over time.

7 PANOPTES CAMERA CONTROL

The camera control system needs to make one key decision: where to look at any given time. We use the mobility awareness concepts presented in Section 6 for camera control.

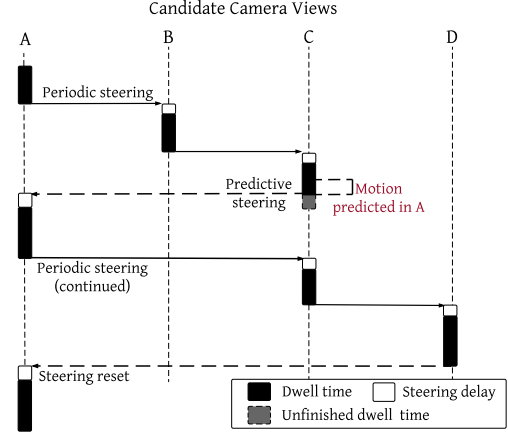


Figure 9: Sample timeline for events under mobility-aware scheduling.

7.1 Camera View Selection

It is not always best to aim the camera at one virtual view, because the requirements of a virtual view can often be satisfied while also simultaneously keeping other virtual views in the camera view. A camera view refers to a particular pose or orientation of the camera that views a part of the entire available scene, as shown in Fig 4a, and is given by:

$$CV = \{p, t, z, v \subseteq V\} \quad (6)$$

where p, t, z are the pan, tilt, zoom values for that camera pose, and V is the set of all virtual views.

We compute candidate camera views, where each is an optimal combination of one or more virtual views and/or prediction zones that fit the camera's field of view (FOV). An application's virtual view is said to fit a camera view if its bounding box is in the camera's FOV and the zoom setting of the camera is the same as that required by the application. The camera view selection technique, as also seen in Algorithm 1, takes a greedy approach and iterates over all virtual views. In each iteration, it selects a virtual view and attempts to fit it with all the other virtual views one by one. If the subset of virtual views under consideration fits into the camera FOV, their bounding box now serves as a temporary view, and the algorithm goes on to find other virtual views that fit the camera FOV with the temporary view. For any virtual view that does not fit, the system tries to include as many of its prediction zones as possible. The algorithm outputs candidate camera views. Fig 8c shows snapshots from our implementation, where the virtual views are in blue, the prediction zones in yellow and the candidate camera views in green. It might seem intuitive to select a camera view that supports the largest number of applications (i.e. fits the largest number of virtual views), but we noticed that often a large number of virtual views can exhibit far less motion events.

Given the set of candidate camera views, we define the importance I for each camera view based on motion probabilities of the virtual views in it and priorities of the associated applications.

$$I = \sum_{i \in S} Pr(VV_i) * [p_m(VV_i) + pz_m(VV_j)] \quad (7)$$

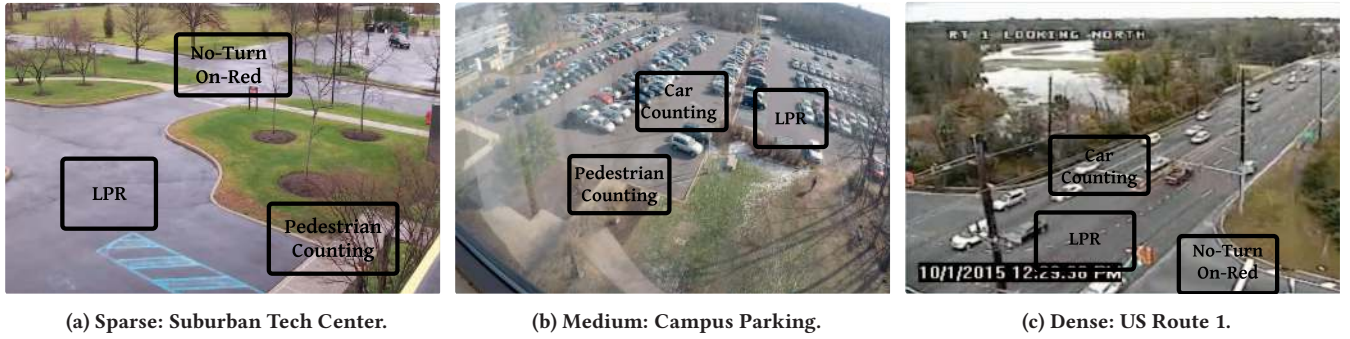


Figure 10: Test environments with different motion densities.

where, S is the set of virtual views in the camera view, $Pr(VV_i)$ is the priority of the application i , $p_m(VV_i)$ is the motion probability of Virtual View i , as defined in Equation 3, and $pz_m(VV_j)$ is the motion probability of prediction zones of virtual view j , where $j \notin S$. This metric accounts for application priority as well as expected motion in the corresponding virtual view, by weighing probability of motion with application priority.

From all the candidate camera views, the system selects the one with the maximum important I as the *default camera view*. Fig 8d marks the chosen default camera view in red. The scheduler starts at the default camera view and scans other camera views in decreasing order of Importance.

7.2 Mobility-Aware Scheduling

The Panoptes mobility-aware scheduler scans candidate camera views periodically, and predicts future positions for objects in its view. When the predicted position belongs to a virtual view, the camera is steered to the view containing that virtual view. The dwell time at each camera view is directly proportional to its importance, I . The scheduler always prioritizes predictions for a higher priority application, and captures them first.

Keeping in mind that most applications are interested in motion, we define η as motion capture efficiency. For each application, the scheduler aims to maximize:

$$\text{Max.} \quad \frac{\text{Motion events captured}}{\text{Actual motion events}} \quad (8)$$

The scheduler can efficiently predict motion in any virtual view due to the presence of prediction zones in candidate camera views. The camera uses these prediction as triggers to look away from the current camera view, and capture events in the predicted view.

Fig 9 shows a sample sequence diagram for scheduling camera views in Panoptes. The camera starts at the default view, and then periodically steers to the other views. Prediction for a virtual view triggers the camera to fall out of scanning order and steer to the view containing the virtual view.

Traditional scheduling schemes cannot be applied to camera view scheduling domain. In a time-based fair-share scheduling scheme, often the camera ends up looking at no motion in one view while missing motion in another. A priority-only scheduling scheme is inefficient because high priority events may be less frequent, such as ‘no-turn-on-red’, and with this scheme the camera spends

most of its time capturing no motion for the high priority view and missing other relevant motion events in the scene.

7.3 Steering and Network Latency

In a real-time scenario, high network latency can cause the control signal to the camera to be delayed, which in turn can lead to failure in capturing motion events. In addition, camera steering latency also has to be taken into account. As the candidate camera views are pre-known, we can assume a constant speed between these views. Our PTZ camera [33] offers a speed of $400^\circ/\text{second}$ between presets. We account for the steering latency by predicting object location for a future time, t_{la} , which is dependent on the camera steering speed. If the camera has zero steering latency, one should ideally, be able to switch from one view to another in no time. However, even with zero steering latency, events will be missed when prediction cannot be made from the current camera view, due to the absence of prediction zones. The control algorithm adapts by predicting for $t_{la} \approx 1$ second. When a prediction occurs for a virtual view, the corresponding camera view is scheduled for time $t_{la} - \Delta t$ later, where Δt is the steering delay.

More cameras are being installed in remote locations and relying on the internet to communicate the video streams. Offline storage of videos may not be affected greatly by this delay, but it becomes non-negligible for real-time camera control. A delayed control command to the camera could lead to the steering starting much later than intended, leading up to missed motion in the destination camera view. We consider the case of a wireless link under LTE latency to evaluate system performance under network delays.

8 TEST SCENARIOS

We evaluate our system across three environments with diverse traffic densities.

Sparse Traffic. We mounted an IP camera [33] on the roof of our building, in a suburban technology center, overlooking a street and a parking lot. We recorded the video stream from this camera 24/7 for over 5 months. Fig 10a shows the three example virtual views in the scene. Note that not all appear in the camera view at the same time.

Medium Traffic. We installed a GoPro Hero 4 camera on the 5th floor of a campus building, overlooking a huge parking lot, shown in Fig 10b. This data was collected over 2 months. GoPro cameras

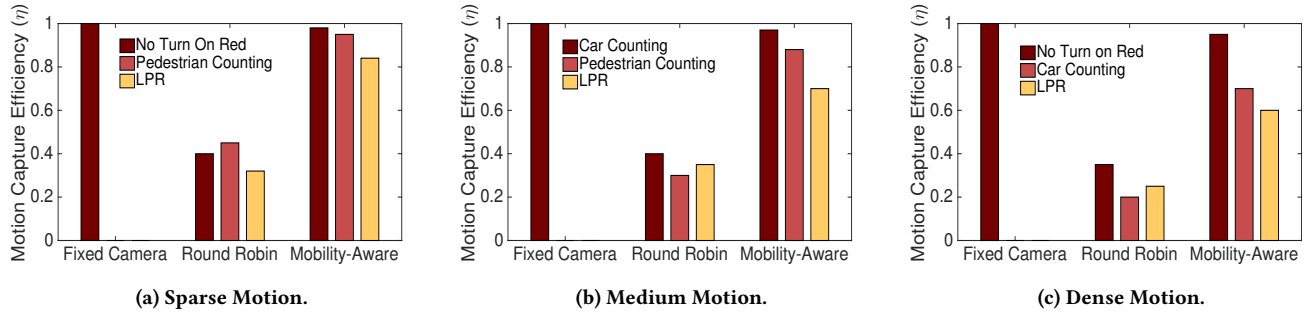


Figure 11: Comparison of mobility aware scheduling scheme with baseline schemes for camera view selection. Darker shade represents higher priority.

have a wide field of view (FOV) and in this setting it is able to capture the entire visible scene from the building. This parking lot is closest to the classroom building and hence the camera captures a large number of pedestrians walking on sidewalks and across the parking lot.

Dense Traffic. We collaborated with the New Jersey Department of Transportation (NJDOT) to obtain few days’ video feeds from their surveillance cameras at US Route 1. A snapshot from the video feed is shown in Fig 10c. This is a heavy vehicular traffic scenario, devoid of pedestrians. The rate of simultaneous motion events is extremely high in this environment.

9 IMPLEMENTATION

To ensure a hardware agnostic solution, we used the ONVIF [21] standard for controlling the camera. Our control algorithm is thus general enough to be realized on any ONVIF-compliant camera. We use OpenCV’s implementation for our tracking algorithm [34].

The PTZ camera used for our data collection is a Pelco Spectra Professional [33], as shown in Fig 12. It streams 1080p H.264 video data at 30 fps. We set up a desktop on the same network as the camera, and captured the video stream 24/7. All our code, including the stream capturing job, was written in C++ using OpenCV and libVLC [35] libraries. We set each candidate camera view as a preset.



Figure 12: PTZ camera installed on the roof for data collection.

Since active cameras typically offer faster pan-tilt speed between presets, this gives us the benefit of extremely low latency. For our camera, this speed was $400^\circ/\text{second}$. This implies that the camera can switch between any two views in less than a second. The motion anticipation, therefore, need only be one second ahead in time, allowing the camera ample time to switch to the required view to capture motion.

Camera Initialization. The initialization procedure starts right after camera setup to infer mobility patterns from the scene. During this phase, the system identifies a finite set of M discrete poses.

Each camera pose transforms to a unique combination of $\{p, t, z\}$. At any given time, the camera will have one of a finite number of states, defined by:

$$state_n = \{p, t, z, background, {}^0H_n, mobilityMap\} \quad (9)$$

where $state_n$ is the camera state for pose n and 0H_n is a running estimate of the homography matrix that maps the n^{th} pose to the 0^{th} pose. This state is different from camera view defined in Equation 6, as camera views are determined based on the virtual views and prediction zones, while initial states are used to scan the environment and learn the mobility patterns.

10 EVALUATION

In this section we evaluate Panoptes’ performance and other design choices. While the system can accommodate any application, we focus on four example applications: (i) continuous car counting, (ii) periodic no-turn-on-red, (iii) license plate scanning and (iv) pedestrian counting. We selected these applications based on the diversity of application frequency, camera pose requirements, and traffic environment. We use motion capture efficiency η from Eqn 8 for quantifying performance. To this end, we seek to answer the following questions:

- How does the mobility-aware scheduling compare to traditional scheduling schemes?
- How sensitive is Panoptes to steering and network latency?
- How well does the prediction work?
- What other factors affect system performance?

10.1 Mobility Aware Scheduling

We compare the proposed mobility-aware planning with fixed camera capture and scheduling choices from traditional domains. We carry out this evaluation across all three test environments. It is evident from Fig 11 that a fixed camera can only cater to one application. A priority-only scheme works the same way as a fixed-camera capture, and captures events only for the application with the highest priority. A time-based, or round robin, scheduling approach captures less than 40% events in each virtual view, across all environments. For implementing round robin, each virtual view was assigned a fixed slot before the camera is steered to the next task. Note that small changes in the duration of the slot do not affect the overall η .

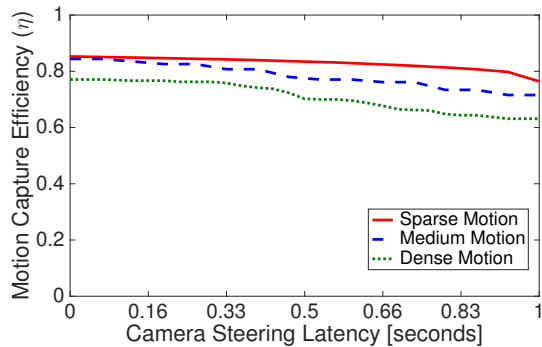


Figure 13: Sensitivity to steering delay.

In environments with sparse motion, as seen in Fig 11a, Panoptes is able to capture most events in each virtual view as simultaneous events are rare. The applications are listed in the order of decreasing priority. The assigned priorities are a heuristic choice and affect only the η for individual applications. As we move to medium motion scene in Fig 11b, the Car Counting application is still provided with the highest number of motion events, but the tradeoff becomes slightly noticeable in pedestrian counting and LPR. In dense motion environments, such as the highway we chose, simultaneous events are very frequent, making control decisions harder. Even in this case, Panoptes performs better than the the other two approaches.

10.2 Sensitivity to Latency

We analyze the effect of steering latency and network latency on Panoptes. Fig 13 shows the change in η with increasing steering delay. Cameras have varying rotation speeds for pan, tilt and zoom, which affects the motion captured from the scene. For cameras with low tuning delays, we can efficiently capture 85% of motion from the scene. As the steering delay increases, the motion capture efficiency drops less than 5% for environments with sparse motion, because of fewer concurrent motion events. In environments with medium or dense traffic, the decline in performance is more noticeable, and up to 10%.

In Fig 14, we examine the performance of a test video under no delay, network delay only, and steering and network delay. It is evident that when the system experiences no steering latency and network delays, very few events are missed. These are simultaneous events, or those in another view that did not trigger prediction. We used the app G-NetTrack Pro [36] to collect 4 days of LTE traces with a fixed Nexus 5 smartphone, to simulate stationary networked cameras.

Typically, the RTT is ≈ 70 ms with peaks of 200 ms [29]. We noticed similar values, and used the maximum ping values from the trace with the highest variation to evaluate our system. It can be noticed that our system compensates for peak network latency of up to 250 ms. Under the effect of steering latency, the overall motion capture drops to 80% because the camera spends more time now to switch between views. This trace was computed for a steering latency of 0.6 second, as that's the amount of time needed by the camera for switching between the furthest candidate views in our

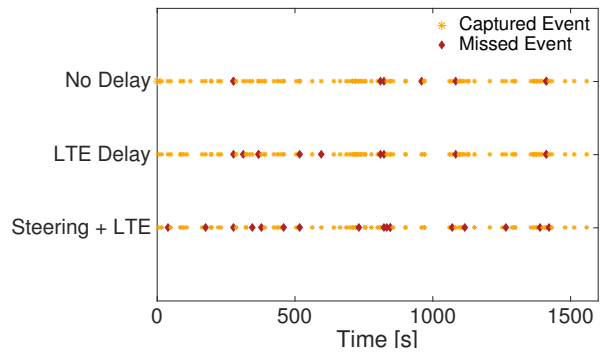


Figure 14: Effect of network and steering latency on a test video.

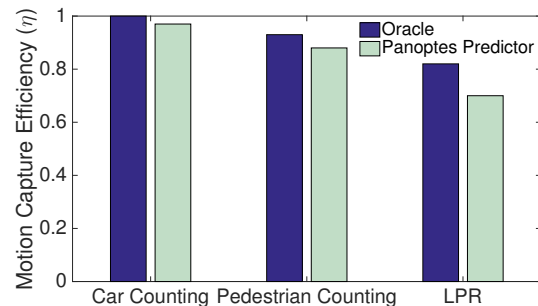


Figure 15: Comparison of Panoptes predictor with Oracle's perfect motion prediction. The events missed by the Oracle represent those targets not in the camera view to trigger a prediction.

scenario. The combined effect of steering and network latency corresponds to system performance of 72%.

10.3 Prediction Performance

We compare the overall motion capture efficiency using the Panoptes predictor to perfect motion awareness. Perfect motion awareness, or Oracle, translates to making accurate predictions for objects in camera view. It must be noted that if the object is not in the current camera view, even the Oracle cannot predict its future location. We compute η using the mobility aware scheduling, but instead of predicting the future location of an observed object, we probe the offline trace for the future location. Overall, the results in Fig 15 show that even with accurate predictions, some motion events will be missed due to their concurrency, and steering latency. While the Panoptes predictor is comparable to Oracle for the higher priority applications, it loses approximately 10% events for the lower priority application, and is able to capture only 70% of the LPR events compared to 80% η achieved by Oracle.

10.4 Scaling to Multiple Virtual Views

We also aim to identify the limit to the number of virtual views a single steerable camera can reasonably support. Generally speaking,

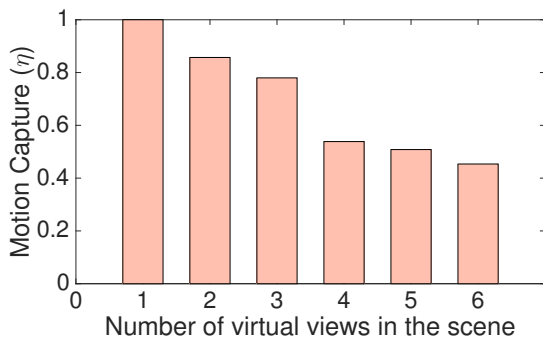


Figure 16: Adding more virtual views to the scene.

this number depends on how active the scene is. We added up to 6 virtual views to the medium motion scenario. We defined virtual views along various directions of motion and assigned equal probability to all. As seen in Fig 16, with increasing number of virtual views to support, the motion capture efficiency reduces from 100% for a single virtual view to 50% as we go up to 6 virtual views. We infer that a single steerable camera can support only a limited number of applications with reasonable motion capture. To sustain a large number of applications over a wider scene, multiple steerable cameras may be necessary. Multiple camera coordination remains an interesting topic for future work.

11 DISCUSSION AND CONCLUSION

We have presented the design, implementation and evaluation of Panoptes, a view virtualization and mobility-aware scheduling system for steerable networked cameras. Unlike previous work, our system focuses on supporting multiple applications through transparently steering between their corresponding views, while maintaining the abstraction of a fixed camera view for each application. Our results show that with motion prediction and mobility-aware scheduling, we can capture up to 80% more motion events than with a fixed camera in our test scenes. We derive these results by conducting experiments for four sample applications across three different traffic environments.

As more cameras are equipped with a high resolution 360° lens and compute capability, network and steering delays become insignificant. In this case mobility-aware scheduling may not apply. For 360° cameras composed of one or more wide angle lens or cameras, that can be optically zoomed, the proposed scheduling technique may still be useful. That being said, motorized cameras are already a large part of the existing infrastructure, deployed in millions across countries like China and Great Britain. They provide the much needed optical zoom, thus proving that they have their own benefits, and may not be completely replaced by 360° cameras.

While Panoptes does rely on position prediction for moving objects, we do not re-identify the object after steering and any captured motion is classified as the anticipated motion.

As part of the current implementation, the responsibility to distinguish between various objects in the view lies with the application, and Panoptes is not made aware of the specifics of the application. It can, however, be extended to implement and execute

a recognition algorithm, to distinguish between moving objects. This should enable it to schedule an application when corresponding motion is of interest to the application.

We have evaluated the system in day and night conditions, as well as in light snowy conditions. However, we excluded rainy scenarios as water droplets on the camera dome obscure the camera view and cause false tracks due to trickling. Other instances that are hard to predict by our current implementation are constant motion such as moving water (a river), and motion that is hard to predict (opening of a window or door). Changes in illumination also impact system performance.

The proposed scheme is generic in its applicability to other camera platforms, which includes but is not limited to security cameras at home, surveillance drones, and dashboard cameras. Additionally, it could be deployed in locations with an existing camera infrastructure, such as parking decks, supermarkets, and shopping malls. For scenarios where motion is less regular, the object position may still be predictable in a small time frame, and the proposed technique may still work relatively well. However, not all environments have predictable entities and our approach works best for more predictable objects across the scene.

Panoptes shows promise to allow cameras that were originally deployed for a single application to be simultaneously used for other analytics applications. Overall, we hope that this work demonstrates the broader uses of existing cameras, and how they can be enabled by automated camera control.

ACKNOWLEDGMENTS

The authors thank Jakub Kolodziejski for his invaluable help in camera setup, even on snowy days. This material is based in part upon work supported by a Microsoft Research internship and the National Science Foundation under Grant No CNS-1329939.

REFERENCES

- [1] Telegraph 2013. One surveillance camera for every 11 people in Britain. <https://goo.gl/gXFmwN>. (2013).
- [2] BBC 2006. Britain is Surveillance Society. <http://news.bbc.co.uk/2/hi/uk/6108496.stm>. (2006).
- [3] Nest Cam 2013. Nest Cam. <https://goo.gl/I85twa>. (2013).
- [4] Cash J, Costello, Christopher P, Diehl, Amit Banerjee, and Hesky Fisher. 2004. Scheduling an Active Camera to Observe People. In *Proceedings of the ACM 2Nd International Workshop on Video Surveillance & Sensor Networks (VSSN '04)*. ACM, New York, NY, USA, 39–45. DOI: <http://dx.doi.org/10.1145/1026799.1026808>
- [5] J. C. Neves and H. Proena. 2015. Dynamic camera scheduling for visual surveillance in crowded scenes using Markov random fields. In *Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on*.
- [6] Adrian Ilie and Greg Welch. 2014. Online Control of Active Camera Networks for Computer Vision Tasks. *ACM Trans. Sen. Netw.*, Article 25 (2014), 40 pages. <http://doi.acm.org/10.1145/2530283>
- [7] Navin K. Sharma, David E. Irwin, Prashant J. Shenoy, and Michael Zink. 2011. MultiSense: Fine-grained Multiplexing for Steerable Camera Sensor Networks. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. ACM, New York, NY, USA, 23–34. DOI: <http://dx.doi.org/10.1145/1943552.1943556>
- [8] A. J. Lipton, H. Fujiyoshi, and R. S. Patil. 1998. Moving target classification and tracking from real-time video. In *IEEE WACV*. DOI: <http://dx.doi.org/10.1109/ACV.1998.732851>
- [9] C. Stauffer and W. E. L. Grimson. 1999. Adaptive background mixture models for real-time tracking. In *IEEE CVPR*. DOI: <http://dx.doi.org/10.1109/CVPR.1999.784637>
- [10] Vijay Mahadevan, Weixin Li, Viral Bhalodia, and Nuno Vasconcelos. 2010. Anomaly detection in crowded scenes. *IEEE CVPR (2010)*. DOI: <http://dx.doi.org/10.1109/CVPR.2010.5539872>

- [11] Paul Viola, Michael J. Jones, and Daniel Snow. 2005. Detecting Pedestrians Using Patterns of Motion and Appearance. *Int. Journal of Computer Vision* 2 (2005). DOI: <http://dx.doi.org/10.1007/s11263-005-6644-8>
- [12] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. 2004. Face recognition with local binary patterns. In *ECCV 2004*. Springer.
- [13] Nam T. Nguyen, Svetha Venkatesh, Geoff West, and Hung H. Bui. 2003. Multiple camera coordination in a surveillance system. *ACTA Automatica Sinica* (2003).
- [14] A. W. Senior, A. Hampapur, and M. Lu. 2005. Acquiring Multi-Scale Images by Pan-Tilt-Zoom Control and Automatic Multi-Camera Calibration. In *IEEE WACV/MOTIONS*. DOI: <http://dx.doi.org/10.1109/ACVMOT.2005.16>
- [15] C. J. Costello and I-Jeng Wang. 2005. Surveillance Camera Coordination Through Distributed Scheduling. In *CDC-ECC '05*. DOI: <http://dx.doi.org/10.1109/CDC.2005.1582368>
- [16] K. Nummiaro, E. Koller-Meier, T. Svoboda, D. Roth, and L. Van Gool. 2003. Color-Based Object Tracking in Multi-Camera Environments. *Lecture Notes in Comp. Sci.* (2003).
- [17] F. Z. Qureshi and D. Terzopoulos. 2009. Planning ahead for PTZ camera assignment and handoff. In *Distributed Smart Cameras, 2009. ICDCS 2009. Third ACM/IEEE International Conference on*. 1–8. DOI: <http://dx.doi.org/10.1109/ICDCS.2009.5289420>
- [18] Yi Yao, Chung-Hao Chen, Andreas Koschan, and Mongi Abidi. 2010. Adaptive online camera coordination for multi-camera multi-target surveillance. *Computer Vision and Image Understanding* (2010). DOI: <http://dx.doi.org/10.1016/j.cviu.2010.01.003>
- [19] Chung-Hao Chen, Yi Yao, David Page, Besma Abidi, Andreas Koschan, and Mongi Abidi. 2010. Camera handoff and placement for automated tracking systems with multiple omnidirectional cameras. *Computer Vision and Image Understanding* (2010). DOI: <http://dx.doi.org/10.1016/j.cviu.2009.04.004>
- [20] Canon. 2015. Canon goes resolution-crazy with a 250-megapixel sensor. <https://goo.gl/hKx7N>. (2015).
- [21] Onvif. 2016. ONVIF. <http://www.onvif.org/>. (2016).
- [22] Pelco. 2016. Pelco SDK. <https://goo.gl/pIICIZ>. (2016).
- [23] Onvif. 2016. ONVIF Video Specification. <https://goo.gl/W6B3nK>. (2016).
- [24] E. Bas, A. M. Tekalp, and F. S. Salman. 2007. Automatic Vehicle Counting from Video for Traffic Flow Analysis. In *Intelligent Vehicles Symp.* DOI: <http://dx.doi.org/10.1109/IVS.2007.4290146>
- [25] D. Beymer, P. McLauchlan, B. Coifman, and J. Malik. 1997. A real-time computer vision system for measuring traffic parameters. In *CVPR*. DOI: <http://dx.doi.org/10.1109/CVPR.1997.609371>
- [26] A. B. Chan, Zhang-Sheng John Liang, and N. Vasconcelos. 2008. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *CVPR*. DOI: <http://dx.doi.org/10.1109/CVPR.2008.4587569>
- [27] Shyang-Lih Chang, Li-Shien Chen, Yun-Chung Chung, and Sei-Wan Chen. 2004. Automatic license plate recognition. *IEEE Trans. Intell. Transp. Syst.* 1 (2004), 42–53. DOI: <http://dx.doi.org/10.1109/TITS.2004.825086>
- [28] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, I. D. Psoroulas, V. Loumos, and E. Kayafas. 2008. License Plate Recognition From Still Images and Video Sequences: A Survey. *IEEE Trans. Intell. Transp. Syst.* 3 (2008). DOI: <http://dx.doi.org/10.1109/TITS.2008.922938>
- [29] Ashkan Nikraves, Hongyi Yao, Shichang Xu, David Choffnes, and Z. Morley Mao. 2015. Mobilyzer: An Open Platform for Controllable Mobile Network Measurements (*MobiSys*). ACM, New York, NY, USA, 16. DOI: <http://dx.doi.org/10.1145/2742647.2742670>
- [30] Erik V Cuevas, Daniel Zaldivar, and Raul Rojas. 2005. Kalman filter for vision tracking. (2005).
- [31] R. I. Hartley and A. Zisserman. 2004. *Multiple View Geometry in Computer Vision* (second ed.). Cambridge University Press.
- [32] Herbert Bay, Tinne Tuytelaars, and Luc Gool. 2006. *ECCV 2006*. Chapter SURF: Speeded Up Robust Features. DOI: http://dx.doi.org/10.1007/11744023_32
- [33] Pelco. 2016. Pelco Spectra. <https://goo.gl/M991vQ>. (2016).
- [34] OpenCV. 2017. OpenCV Motion Analysis. (2017).
- [35] libvlc. 2016. libVLC. <https://wiki.videolan.org/LibVLC/>. (2016).
- [36] G-NetTrack. 2010. G-NetTrack Pro. <https://goo.gl/MHEQJ7>. (2010).