AcTrak: Controlling a Steerable Surveillance Camera using Reinforcement Learning

ABDULRAHMAN FAHIM, EVANGELOS PAPALEXAKIS, SRIKANTH V. KRISHNAMURTHY, and AMIT K. ROY CHOWDHURY, University of California, Riverside, USA
LANCE KAPLAN, DEVCOM Army Research Laboratory, USA
TAREK ABDELZAHER, University of Illinois at Urbana Champaign, USA

Steerable cameras that can be controlled via a network, to retrieve telemetries of interest have become popular. In this paper, we develop a framework called AcTrak, to automate a camera's motion to appropriately switch between (a) zoom ins on existing targets in a scene to track their activities, and (b) zoom out to search for new targets arriving to the area of interest. Specifically, we seek to achieve a good trade-off between the two tasks, i.e., we want to ensure that new targets are observed by the camera before they leave the scene, while also zooming in on existing targets frequently enough to monitor their activities. There exist prior control algorithms for steering cameras to optimize certain objectives; however, to the best of our knowledge, none have considered this problem, and do not perform well when target activity tracking is required. AcTrak automatically controls the camera's PTZ configurations using **reinforcement learning (RL)**, to select the best camera position given the current state. Via simulations using real datasets, we show that AcTrak detects newly arriving targets *30%* faster than a non-adaptive baseline and rarely misses targets, unlike the baseline which can miss up to 5% of the targets. We also implement AcTrak to control a real camera and demonstrate that in comparison with the baseline, it acquires about *2*× more high resolution images of targets.

CCS Concepts: • **Computer systems organization** → **Sensor networks**; *Robotic control;*

Additional Key Words and Phrases: Reinforcement learning, Zoom-in tracking, PTZ cameras, Steerable cameras, camera control, surveillance systems

ACM Reference format:
Abdulrahman Fahim, Evangelos Papalexakis, Srikanth V. Krishnamurthy, Amit K. Roy Chowdhury, Lance Kaplan, and Tarek Abdelzaher. 2023. AcTrak: Controlling a Steerable Surveillance Camera using Reinforcement Learning. *ACM Trans. Cyber-Phys. Syst.* 7, 2, Article 14 (April 2023), 27 pages.
https://doi.org/10.1145/3585316

This work was partially supported by the DEVCOM Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053 and W911NF-17-2-0196. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the DEVCOM Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work was also partially supported by the NSF CPS grant 1544969 and NSF CNS grant 2038817. Amit Roy-Chowdhury was partially supported from ONR grant N00014-19-1-2264.
Authors' addresses: A. Fahim, E. Papalexakis, S. V. Krishnamurthy, and A. K. Roy Chowdhury, University of California, Riverside, Riverside, California, USA, 92521; emails: afahi002@ucr.edu, {epapalex, krish}@cs.ucr.edu, amitrc@ece.ucr.edu; L. Kaplan, DEVCOM Army Research Laboratory, USA; email: lance.m.kaplan.civ@army.mil; T. Abdelzaher, University of Illinois at Urbana Champaign, USA; email: zaher@illinois.edu.

2378-962X/2023/04-ART14 $15.00
https://doi.org/10.1145/3585316

1 INTRODUCTION

Networked surveillance cameras are becoming ubiquitous. It is estimated that around one billion surveillance cameras are currently installed globally [1–3]. A standard surveillance camera has a fixed and limited view of the scene of interest depending on its orientation and the width of its lens. In many applications however, there is a need for a complete coverage of the scenes of interest (e.g., trespass prevention) and leaving parts of the scene uncovered can lead to security breaches. In addition, in many applications, high resolution/ fine grained images of suspicious targets or objects are crucially needed. **Pan-Tilt-Zoom (PTZ)** cameras have been considered a suitable solution satisfying these demands [4, 5]. Such cameras have the ability to maneuver around, covering the entire scene unlike standard surveillance cameras. The "zoom" feature offered by such cameras, allows the camera to zoom in to acquire high resolution images of the targets and objects of interest, and yet provide wide view coverage when needed (via a zoom out). Most PTZ cameras can be controlled remotely over a network to satisfy various surveillance requirements, which opens up opportunities for building reliable and autonomous surveillance systems [4, 6]. One fundamental, unexplored question that arises in this case is: in an evolving scene, how to automatically control a PTZ camera to facilitate applications where there will be a need to (a) zoom out to identify any new target quickly when it enters the scene of interest, and (b) frequently zoom in on existing targets to *be able to monitor their fine-grained activities* via high resolution images. For example, we envision scenarios with a need to capture prohibited activities (e.g., eating/drinking or photographing in museums, and smoking in shopping centers). Another application relates to deployments in shopping centers, to track target shopping interests i.e., what products customers are looking at, in addition to catching shoplifting activities. Such information would aid data analytics to understand customer behavior and/or even guide re-organizing product placements across aisles (e.g., distribute popular products to enable Covid-19 social distancing needs).

Challenges: The control algorithm requires a good balance between the actions of zooming in and out. A simple example is shown in Figure 1. If we focus on target zoom-ins, we may miss out on arriving targets in the lower left corner and a continuous zoom out does not shed light on target activities. Achieving this balance by changing the camera's views is challenging due to two delay components associated with camera motion: mechanical camera movements incurred due to motion from one position (e.g., zoom out) to another (e.g., zoom in) and computational and networking latencies of processing and retrieving the captured frames. Therefore, a careful orchestration is needed for managing the frequency and patterns of zoom ins among targets especially in the case when several targets exist in the scene, while ensuring that we do not miss new targets.

Related Work: Multi-objective surveillance problems have been studied widely in static and steerable multi camera systems [7]. Cameras usually co-ordinate to achieve one or more objectives such as scene coverage, opportunistic acquisition of zoom-in images of targets [8], tracking [9–12], or power efficient surveillance [13]. However, in our work we consider a single camera that balances the trade-off between two conflicting objectives (Figure 1), and previous work on multi-camera solutions cannot satisfy these objectives in a single camera setup. The closest work to ours is [8], where the authors consider a multi camera system that (1) fully covers the physical scene all times. and (2) opportunistically captures high resolution shots of existing targets. This is different from ours in two aspects. First, the physical scene coverage constraints cannot be used in our setup (when a single camera is used), because when the camera zooms-in on a target, only a partial view of the scene is obtained (violating the physical scene coverage constraints). If for cost constraints (multiple camera deployments are more expensive) the full physical scene cannot be covered, one has to manage the zoom-in shots of targets without sacrificing coverage of new targets arriving to the scene; we consider this specific case in our work. Second, we consider continuous zoom-in tracking of existing targets for activity recognition, unlike the work in [8], where the focus is on

Fig. 1. Zoomed out (left) and zoomed-in (right) views. The zoomed out view enables the detection of new targets as they enter the scene. However, it is insufficient for inferring target activities. The zoomed in view facilitates inferring target activities (i.e., a target reads a book) but only partially covers the scene. AcTrak balances zooming in and out such that target activities are captured, while ensuring that arriving targets are detected quickly when they step in to the scene.

opportunistic acquisition of high resolution shots. More discussion is provided later (Section 5) on why a single camera solution can be desirable, and we tackle this important problem in this work. While there is prior work on controlling the PTZ of steerable cameras (e.g., [14, 15]) in a single camera setup, they have different objectives from ours; importantly, none consider changes in the target population in the scene (new arrivals) and most do not consider arbitrary motions of targets. For example, the authors in [15] propose an approach to orient the camera to zoom in on various (but fixed and pre-determined) chosen locations. Their approach allows the tuning of how frequently these locations are to be visited, but leaves other locations uncovered. Thus, this method is ineffective when tracking targets activities outside the selected locations is required. We argue that the camera should focus on activities rather than location coverage (since coverage of unoccupied locations yields little or no information).

Contributions: In this paper, we design and implement AcTrak to control a PTZ camera such that it (a) quickly identifies dynamically arriving targets to a scene, and (b) with appropriate frequency (target specific), obtains high resolution images of each target in the scene to capture their activities at fine time scales.[1] Importantly, AcTrak accounts for the associated latencies when determining the camera movement.

We formulate the problem as **Markov decision process (MDP)**, where a **reinforcement learning (RL)** agent dynamically learns, and thereby determines the best PTZ configuration given the current situation. This works in an online fashion i.e., it continuously tunes the camera PTZ dynamics to cope with the evolving scene. Importantly, it tries to minimize the latencies incurred in changing the camera views, by choosing visitation patterns in an informed way. Furthermore, it does not require expensive computations to derive its next PTZ configuration during runtime (a single feed forward neural network operation is needed).

In brief, our key contributions in this paper are as follows:

(1) We design AcTrak, a framework for the PTZ control of a steerable camera. AcTrak is based on MDP, which selects the PTZ configuration that provides the highest utility (discussed later) in terms of a trade-off between the goals of balancing rapid acquisition of new targets and obtaining fine grained information about existing targets, while minimizing latency penalties due to camera motion.

[1]Note that we do not store biometric features of targets (for privacy reasons), but rather use high resolution shots for activity recognition.

(2) Typically, RL agents need heavyweight training to deliver high accuracy [16]. While a creation of the requisite, a huge number of training instances is possible on fast machines, it is time-prohibitive in our scenario since camera mechanical movements and networking and computational latencies can be in the order of seconds. Hence, instead, we develop a simulator to mimic the camera, target movements, and other dynamics to enable training and we deploy the trained agent during test time.
(3) We showcase the merits of AcTrak via both simulations and real world experiments with an implementation on an off-the-shelf PTZ camera. In our real world experiments AcTrak outperforms a non-adaptive baseline by acquiring *2* × more high resolution images of targets. Our simulations using public datasets show that our agent detects new targets that arrive to the scene *30%* faster than a state of the art baseline.

2 OUR RL BASED CONTROL FRAMEWORK

In this section, we first describe the trade-offs between the functionalities we seek to achieve (zoom outs to find new targets and zoom in to track current targets). Next, we provide an overview of AcTrak, and then describe it in detail.

2.1 Functionalities and Associated Trade-offs

Scene coverage. The camera can zoom out to cover the entire scene (to find new targets) and we call this the *coverage tour*. Invoking unnecessary coverage tours at high frequencies will reduce the time for (frequency of) acquiring high resolution images of existing targets. Instead, if the camera undervalues coverage tours, new targets may leave unseen.

Zoom-ins. Upon identifying a target, AcTrak marks its location so that the camera can zoom in on it again in the near future, to track its activity; this obviates the need to zoom out each time in order to find that object. To acquire the high resolution image, an appropriate PTZ configuration is chosen to direct the camera at the last known (updated) location of the target. Since targets can be located at different distances from the camera, each target requires a different zoom level denoted as Z_j, where j is the target's index. Target locations are updated during coverage tours and the zoomed in visitations. A location becomes outdated when the target moves outside its previous zoom-in view, which depends on the target's motion (e.g., walking, jogging). The camera must obtain an updated location before zooming in on the target.

Managing zoom-ins of multiple targets. AcTrak's control algorithm is driven by two objectives: (A) if a target has not been visited recently, the highest priority should be given to that target, and (B) try to maximize the number of targets that can be seen with the same camera beam or field of view (this is dictated by the camera's orientation and targets' proximity to each other). The second objective saves time via visits to cover multiple targets (instead of multiple different visits to cover those targets). An auxiliary benefit not explicitly considered is that it can detect target group interactions (e.g., when multiple people meet as a group).

Scheduling visitations across multiple targets have to be carefully orchestrated in order to minimize the overall latency associated with changing the PTZ configuration to switch targets; recall that this is a latency expensive operation. For example in Figures 2(A) and 2(B), there are two distant clusters, each containing targets in close proximity. If the camera alternates between targets from the two clusters (as in Figure 2(A)), the overall time for making a visit to all targets will be large due to wasteful, repeatedly large PTZ configuration changes. However, if it visits the targets within one cluster before it moves to the other cluster (as in Figure 2(B)), this overall time will be much smaller (small changes to the PTZ configuration at each step). *Thus, the proximity of targets has to considered while zooming in on different targets.*

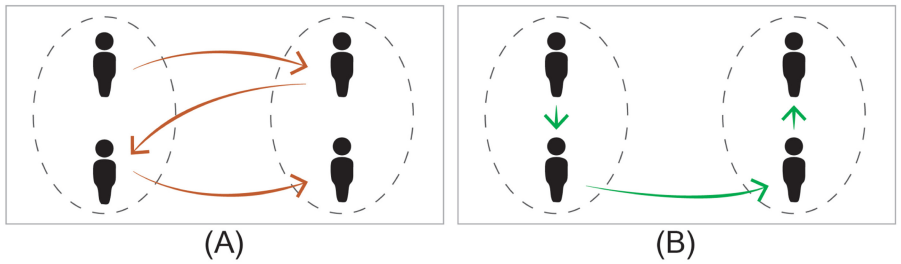

Fig. 2. Zoom-In Scenarios. In Figures (A) and (B), we show different camera strategies of zooming-in on existing targets (Note the arrows). The overall time to visit all targets of the strategy in Figure (A) is higher in comparison with the scenario in Figure (B) due to the camera's longer (wasteful) moves.

Continuing with the example, assume that there are only two targets within a cluster and there is a time budget λ s, associated with visiting these two targets. The camera can stay with the first target for $\lambda/2$ s, and move on to the second target and stay with the latter for remaining $\lambda/2$ s. A better strategy is to alternate between the targets, staying with each for a small period each time. In the first strategy, the camera stays with a target for a longer time collecting continuous (and more) high resolution shots; however these shots are unlikely to yield new information and the other target is not monitored for a big time gap ($\lambda/2$ s). In the second strategy, fewer high resolution shots of each target are obtained due to additional PTZ changes, but the time gap for which a target is not monitored is smaller. Further, each visit is more more likely to capture new activity (e.g., due to a new posture). As discussed later, our framework can be tuned to trade-off between the two strategies.

Overview. As discussed earlier, we seek to balance two tasks: (a) tuning the frequency of coverage tours and (b) managing zoom-ins across the multiple targets in the scene. `AcTrak` seeks to avoids wasteful PTZ configuration changes to the extent possible (i.e., zooming out when no new targets have arrived and zooming-in on obsolete locations of current targets which have moved). RL has been shown to be effective in applications needing adaptive parameter tuning (e.g., [17]), which is what we need to achieve the balance we seek. Our trained RL agent determines the next visit while accounting for the impact of this decision on future decisions, via a simple and quick feed forward operation, depending on the current state of the environment (determined by for example, clustering of targets, application needs, etc.).

To learn the appropriate frequency of coverage tours we monitor: (1) *target arrivals*: if no target appears within the current period between coverage tours, the camera learns to decrease the frequency of coverage tours, and (2) *new target locations*: if new targets have moved significantly from the coverage boundaries (have not been detected for a while), the camera infers that it needs to increase the coverage tour frequency. With regards to managing zoom ins, we seek to efficiently determine the next visit while accounting for the impact of this decision on future decisions. A trained RL agent determines this via a simple and quick feed forward operation, depending on the current state of the environment.

In an evolving scene, given what is observed by the camera beam, `AcTrak`'s control algorithm seeks to determine a PTZ configuration that maximizes a given utility that captures our goal. Then, the camera changes its current PTZ configuration to that new PTZ configuration. The PTZ configuration is adapted over a sequence of steps (time epochs) denoted as $k \in [1, \infty)$. We can model this problem as a Markov decision process (MDP), wherein we seek to make a decision with regards to an action relating to a PTZ configuration change, towards either a coverage tour or a specific zoom in, based on the utility. This utility is the long term reward (discussed later) accrued by a

Table 1. Key Notation

Symbol	Description
k	A time step in the discrete set of time steps
O_k	Set of new targets appear in the scene at time k
O_k	# of new targets appear in the scene at time k
$\mathcal{N}_k$	Set of targets exist in the scene at time step k
C_k	Camera's zoom magnification level at step k
Z_j	Zoom level requirement for target j
T_j	Target j
τ	A threshold on the time gap between two consequent visitations for the same target
d	A threshold on the target's displacement two consequent visitations for the same target.
t_j^{base}	The last time the camera is rewarded when it zoomed on target j
l_j^{base}	Target's location at which the camera was last rewarded when it zoomed in on the target j
V_j^k	# of zoom in visits to T_j up to step k
$v(T_j)$	Visual features of target j
γ	Discount on the future rewards
ϵ	Exploration rate to balance exploration exploitation tradeoff

Q-learning agent. In an offline phase, the agent learns a camera control policy that is to be deployed in the online phase to maximize the given utility. Our approach is consistent with previous work [18–20]. While alternatively, the problem can be formulated as a semi-MDP that captures steps (i.e., camera PTZ change) of different lengths corresponding to different distances of movement [21], we choose the MDP formulation due to its simplicity and success with previous work (where steps of varying length are also present [18]).

2.2 Problem Formulation

AcTrak is geared towards a single camera system monitoring a dynamic scene with dynamic target arrivals and departures. As discussed, it considers discrete time epochs $k \in [1, \infty)$ over which it continously adapts the PTZ configuration based on scene evolution. It tracks targets that exist in the scene and saves them in a list. We note that this list may not exactly reflect the ground truth because the camera may only have a partial view of the scene at several of these steps. At each step k, new targets may arrive and may be observed by the camera; we denote those targets as O_k and the size of the set of these new objects as O_k. Note that it is possible for the camera to find new targets in a zoom-in view (e.g., while zooming in on a door to observe a target that is leaving, a new target may arrive). We denote a set of targets that exist in the scene at step k as $\mathcal{N}_k$; this set includes targets that previously existed at step $k-1$ and O_k; formally, $\mathcal{N}_k \leftarrow \mathcal{N}_{k-1} \cup O_k$. If the action at step k is a coverage tour, we may remove targets $\in \mathcal{N}_{k-1}$ that do not appear in the scene (i.e., those targets that have left the scene). The maximum number of targets that can exist in the scene is assumed to be N. The key notations is shown in Table 1.

We collect features about existing targets (e.g., locations, time stamps at which they were last observed) which we use to compose the state in our MDP formulation (discussed next).

State. The state s_k describes the environment at the kth step of the evolving scene and is defined by the following extracted features from the environment:

(1) *The camera PTZ at step k*; the Pan and tilt ranges are $(0, 360)^\circ$, and $(-90, 90)^\circ$, respectively, and the zoom ranges from 0 to $m\times$. We denote camera's zoom magnification level at step k as C_k.
(2) *Location vectors*: We add three vectors (each of size N) to describe the targets' locations in terms of their associated PTZ configurations: (a) a vector with the most recently updated locations of all targets at step k, (b) the penultimate location where each target was observed, and (c) the *location base*, which is the location of a target at which the system was last "positively rewarded" (rewards are discussed later) for observing that target.
(3) *Time vectors*: We add three vectors (each of size N) that capture the following time features for each target. Specifically, we add the time difference between the time instance at step k and (a), (b) and (c), respectively: (a) the times when the targets were last observed in a zoom-in mode, (b) the times when the camera acquired the penultimate location for each target, and (c) the time when the agent got a positive reward for each target.
(4) *Coverage tour latency*: We also add a feature that includes the time difference between the time recorded at the k^{th} step and the time of the last coverage tour.
(5) *Number of visitations:* Finally, we include a vector of size of N that contains the number of visitations that the camera has made to each of the targets up to step k.

Action. An action a_k is performed at each state and leads to a new PTZ configuration which causes a new state to be composed.

There are a discrete set of $N + 1$ possible actions. These correspond to visiting a specific target (recall that the maximum number of targets is N), or a coverage tour to determine if new targets have arrived.

Rewards. Upon transiting from a state s_k to state s_{k+1} via action a_k, the agent accrues an immediate reward, r_k. The *projected cumulative reward* at step, k, is computed as:

$$R = r_k + \sum_{c=1}^{\infty} \gamma^c r_{k+c} \tag{1}$$

where, $\gamma \in [0, 1)$ is a discount factor for the future rewards.

Policy. The policy π selects an action (the next PTZ configuration) at each given state that maximizes the projected cumulative reward for current and future actions. This is defined formally as follows.

$$\pi(s_k) = \arg\max_a \mathbb{E}[R|s_k, a, \pi] \tag{2}$$

where a is the action selected from the action space and $\mathbb{E}[R]$ is the expected value of the cumulative reward.

2.3 Design of the Immediate Reward

We define the immediate reward, r_k, when an action a_k is taken in state s_k to be:

$$r_k = PR_k - NR_k \tag{3}$$

In the above, we decompose the immediate reward into two parts, a **positive reward (PR)** and a **negative reward (NR)**, that together drive the trade-off between zooming in and out.

Positive rewards (PR_k): The agent gets rewarded if it discovers new targets regardless of the camera's zoom (this is captured in the first term in the equation below) or successfully revisits an

existing target in a zoom in mode if certain conditions are satisfied (captured in the second term in the equation). Formally, this is expressed as:

$$PR_k = \sum_{j=1}^{O_k} \alpha_1 \mathbf{1}(T_j \notin \mathcal{N}_{k-1}) + \rho^{V_j^k} \alpha_2 \{\mathbf{1}(T_j \in \mathcal{N}_{k-1}) \quad \mathbf{1}(C_k \geq Z_j)\mathbf{1}(t - t_j^{base} \geq \tau, |l_j - l_j^{base}| \geq d)\} \tag{4}$$

where, T_j represent the targets captured with the camera's PTZ at the kth step. $\mathbf{1}(.)$ is an indicator function that is equal to 1 if at least one of its argument conditions holds true, and 0 otherwise. We capture the prior existence of a target in the scene with the indicator $\mathbf{1}(T_j \notin \mathcal{N}_{k-1}) = 1 - \mathbf{1}(T_j \in \mathcal{N}_{k-1})$ as follows:

$$\mathbf{1}(T_j \notin \mathcal{N}_{k-1}) = \mathbf{1}\left(\min_{i \in \mathcal{N}_{k-1}} [|v(T_j) - v(T_i)|] \leq \sigma\right) \tag{5}$$

where, $v(T_j)$ represent the visual features of target T_j and σ is a pre-defined threshold; the target was in the scene previously if the visual features match those of a target seen at step $k-1$.

A reward of $\alpha_1 \in [0, 1]$ is accrued if the target is new ($T_j \notin \mathcal{N}_{k-1}$), regardless of the zoom configuration of the camera. A reward of $\rho^{V_j^k} \alpha_2$ is obtained when the camera zooms in on a target but with different conditions (discussed next); ρ is a factor $\in (0, 1]$ and V_j^k is the number of zoom in visits that are made by the camera to obtain the fine grained features of target T_j. This means that for each repeated visit to the same target, (if $\rho < 1$) the camera receives a lower reward than in its previous visits. This ensures that the camera visits other targets with fewer high resolution images, rather than revisiting a target repeatedly. The reward is accrued if the conditions 1, 2, and at least one of 3(a) or 3(b) are satisfied:

(1) The observed target has been seen in the prior step i.e., ($T_j \in \mathcal{N}_{k-1}$).
(2) C_k, is larger than the required magnification level for obtaining the fine grained features of the target T_j, (referred to as Z_j as discussed in Section 2.1).
(3)(a) The time gap ($t - t_j^{base}$), between two consequent zoomed in visitations is larger than a threshold, τ, where t_j^{base} is the last time the camera is rewarded when it zoomed in on the target and t is the time instance at the kth step. Note that we make the assertion $t_j^{base} = t$ when the camera visits a target and gets a reward of $\rho^{V_j^k} \alpha_2$.
(b) The target's displacement between two consequent visitations is larger than a threshold, d, where l_j^{base} is the target's location at which the camera was last rewarded when it zoomed in on the target and l is the location of the target at the k-th step. Similarly, we make $l_j^{base} = l_j$ when the camera visits a target and gets a reward of $\rho^{V_j^k} \alpha_2$.

The last condition helps control the time gap between acquiring two consequent images of the same target. For example, if the user is interested in receiving a continuous sequence of a target's images (video), τ can be set to zero. Similarly, if the user is only interested in tracking a target's activities when its location changes by a certain distance, d can be set to that value.

Negative Rewards (NR_k): The positive rewards do not guarantee that targets are left without being visited (the camera can stick with only one target and still get rewarded). Hence, next we introduce negative rewards (penalty) that coerce the agent to visit targets that have not been visited for a while. Specifically, this penalty (NR) increases in proportion to the time the camera was away from the target (not zooming in on the target) and is given by $\beta \sum_{i=1}^{\mathcal{N}_{k-1}} (t - t_i)$, where t_i is the last time when the camera zoomed in on target T_i, and t is the time of the k-th step.

Note that α_1, α_2 and $\beta \in [0,1]$, are *coefficients* that are selected to balance the positive and negative rewards.

2.4 Learning the Camera Control Policy

We find the optimal policy π^* that selects the best action in each state towards maximizing the accumulated reward, R. Since the problem is stateful (i.e., the action is selected based on the state), we use a Q-learning approach [22] as a basis, due to its relevance and ease of deployment.

The value of $E[R|s_k, a, \pi]$ is denoted as $Q(s_k, a)$, where the Q-value is updated by the following:

$$Q_{k+1}(s_k, a_k) = (1 - \eta)Q_k(s_k, a_k) + \eta \left(r_k + \gamma \max_a Q_k(s_{k+1}, a) \right) \tag{6}$$

where $\eta \in (0, 1]$ is the learning rate and $Q_k(s_k, a_k)$ is value of the state action pair at step k (the Q-value is learned by updates at each step). The optimal value of the accumulated reward at step k that is achieved by taking the action a_k, is given by $Q^*(s_k, a_k)$ and is computed using the well known Bellman equation as follows [23]:

$$Q^*(s_k, a_k) = r_k + \gamma \max_a Q^*(s_{k+1}, a) \tag{7}$$

Our model has a finite action space but an extremely large state space, since the targets can exist in any location in the scene. If we have L (quantized) locations where the N targets can exist, the complexity of the state space is $O(L^N)$. Thus, creating a table with the Q values of every state action pair combination is prohibitive. Thus, we use a neural network to approximate the Q value for a given state action pair [16]. The neural network is trained based on previously assigned rewards, and predicts the reward (similar to regression) when a new state action pair is encountered.

For reducing complexity and uncertainty we use a **Double Deep Q Network (DDQN)** [24], which uses two neural networks to provide an approximate Q value for a state action pair; one is used for action selection and the other for action evaluation. We also use the duelling architecture from [25] to enhance the performance of our model. The interested reader can find details in [25] and [24].

We use the mean square error between the output of the neural network and the Q value as the loss function to be minimized. We apply an ϵ greedy exploration policy, where a random action is selected at a given state with probability ϵ while the action that currently maximizes the reward is chosen with probability $1 - \epsilon$; ϵ, the exploration rate, is tuned to balance exploration and exploitation.

3 REALIZING ACTRAK IN PRACTICE

The implementation of our RL framework is on a real camera platform, but a simulator is used for training prior to online deployment. Specifically, we implement our system on Avipas HD PTZ camera (Model: Av- 1080w) [4]. The camera allows pans and tilts with ranges ± 135 and ± 35 degrees, respectively. The camera has a 10× optical zoom capability, quantized over 33 zoom levels. We have developed all our code using Python3 on an Apple Macpro machine, and both the camera and machine are part of wired local network where the machine acquires the frames for processing and controls camera's movement. Our model is capable of adapting to variations in computational, communication, and mechanical latencies that could be specific to the system and the setting.

3.1 System Setup

The camera frames undergo the the following pre-processing steps to compose the RL-state. A pre-trained tiny-Yolo model (YOLOV3 [26]) is used to detect targets in the frames (yielding bounding boxes in terms of frame coordinates) [26, 27]. While tiny-Yolo can detect targets in the processed frame, it cannot determine if the observed targets in the current processed frame have been observed before. This is important for composing the RL state (as discussed in Section 2.2). For that purpose, we crop each target's region from the frame and feed it to the *humanReid* module, which

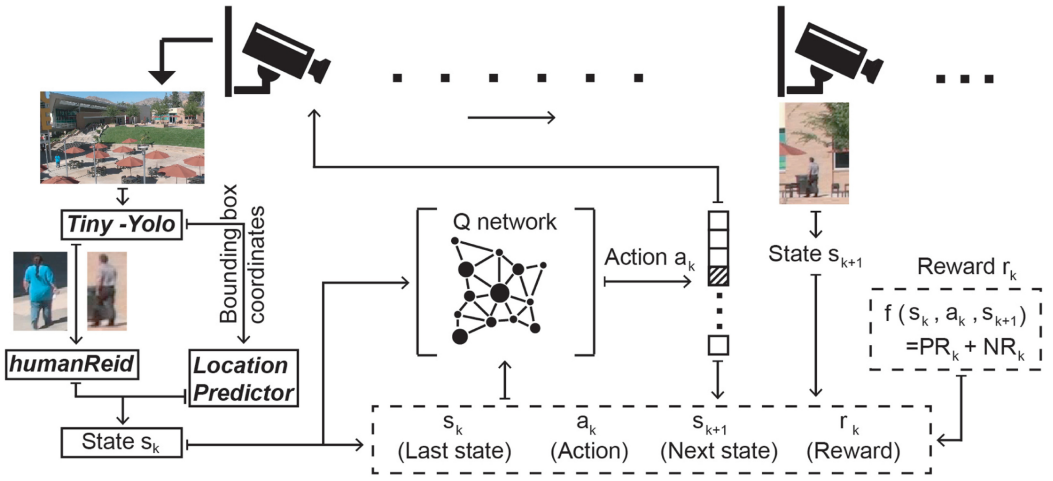

Fig. 3. A high level depiction of AcTrak.

associates the observed target with previously seen targets.[2] Although we obtain the targets' locations in the frame coordinates (using tiny-Yolo), the associated PTZ configuration that makes the target appear in the middle of the frame, with the required zoom level Z_j, is not known. We use the *Location Predictor* module to estimate this. Finally, using the outputs of the two modules, we compute the step rewards and compute the RL-state as discussed in Section 2.2. A depiction of AcTrak is in Figure 3.

3.1.1 humanReid. **humanReid (human re-identification module)** associates current targets with previously seen targets. Human re-identification is very challenging in itself [28, 29], and is beyond the scope of this work. For ease of experimentation, our targets wear *distinctive* colored apparel to enable the use a light weight pre-trained humanReid deep learning architecture, mobileNet [30]. We further tune the model by training it with traces of targets in different colors viz., red, blue, green, and black. We collect 24 one minute videos where each target walks in an area of interest with these colored shirts, and we randomly sample frames of those traces to train the model.

3.1.2 Location Predictor. The PTZ configuration required by the camera to observe the target in a zoomed in view cannot be accurately obtained from the bounding box dimensions in frame coordinates. This is because the depth relating to the target (how far is the target from the camera) is not known. The depth can be obtained using monocular depth estimation approaches [31–33] that estimate the depth of each pixel in the captured frame; however, associating the depth information with the required PTZ configuration to zoom on targets given the camera's intrinsic parameters such as focal length and distortion is not straight forward [34]. A better solution would be camera calibration, a technique via which we can learn the position and the orientation of the camera with respect to the world's 3D coordinates and its intrinsic parameters such as focal length and distortion [35]. However, camera calibration is sophisticated and may not be accurate because of optical lens distortion that may exist in PTZ cameras [15, 36]. Our aim is to make our solution generic and usable by non expert users with a simple pre-setup process.

[2]Technically, we can re-train a new model from scratch that performs the two tasks simultaneously (identifying targets from the raw frame and associating identified targets in the current frame with previously observed targets). However, we choose to use off the shelf trained models and architectures for ease of use, given that these existing models serve their intended purpose.

The approximation we make to support flexibility and generality by trading off accuracy is to estimate the PTZ configuration for an object based on the size of the detected bounding box surrounding that target (detected by tiny-Yolo): the larger the area of the bounding box the more likely the target is closer to the camera. To infer targets locations' in PTZ coordinates, we use a Decision Tree Regressor [37] that takes as input the camera's current PTZ and the bounding box frame coordinates and outputs the PTZ coordinates for the target. We point out that this approach has sufficient accuracy for our purpose but it is not extremely accurate for two reasons. First, the size of the bounding box is sensitive to the size of the person (e.g., children have smaller bounding boxes than adults even when their depth in the scene is similar). Second, if the size of the bounding box is not accurately determined (due to mis-detection from tiny-Yolo), the estimated location is also affected. As will be discussed, in our real camera and simulation experiments, we were not affected by such mis-detections that may have been caused by tiny-Yolo. Note that estimating the depth is orthogonal to our contributions and our method (i.e., controlling a camera using reinforcement learning) can work in scenes where camera calibration is computed; in other words, it can be combined with a camera calibration method.

3.2 Simulator

Creating a large number of training instances with a PTZ camera where each mechanical movement induces high delay (up to 3s in our camera), is inconvenient and prohibitive. For example, training our agent with ≈ 800K+ camera movements requires more than 18 days (assuming many targets and training round the clock and that each movement takes 2 seconds). In areas with sparse pedestrians, training will take much longer. Thus, we build a simulator that mimics targets and camera movements to accelerate the learning process. With our simulator, the training takes less than 18 hours as is the norm in such experiments.

3.2.1 Composing Training Datasets. We evaluate our model via simulations with public datasets, and real camera experiments for which we collect a training dataset (discussed in Section 4).

Public datasets: We use two datsets (described later) with ground truths of targets' locations (described by bounding box coordinates). This obviates the need for the object detection model or the humanReid model; our *Location Predictor* module uses the ground truths of targets' locations to associate them with PTZ configurations. When zooming in on a target in a dataset, we pick an image patch that coincides with the bounding box of the target. We associate the PTZ configuration latencies associated with moving from the acquisition of one image patch to another. We acknowledge that there is an implicit assumption that the cameras capturing those videos are similar to ours (similar overall delays exist), but since both our platform and these are based on a real world camera, we believe that this assumption is realistic.

Real camera dataset: We collect training videos from the area of interest with the maximum zoom out and label the ground truth target locations for all frames, which can be obtained using the models from tiny-yolo, *humanReid* and *Location Predictor*.

3.2.2 Accounting for AcTrak*'s Mechanical and Computational Delays.* Accounting for the delays experienced in live experiments while training the agent in the simulator is important to ensure that the simulator emulates reality. Otherwise the discrepancy between the real system and the simulator leads to poor performance during run time (test time).

Modeling camera's mechanical delays: The time it takes the camera to change its PTZ configuration depends on the displacement magnitude and the speeds of the camera's mechanical control motors across its axes. We observe that the camera motor does not move with a constant speed, but there is an acceleration component that makes the prediction of the time taken hard.

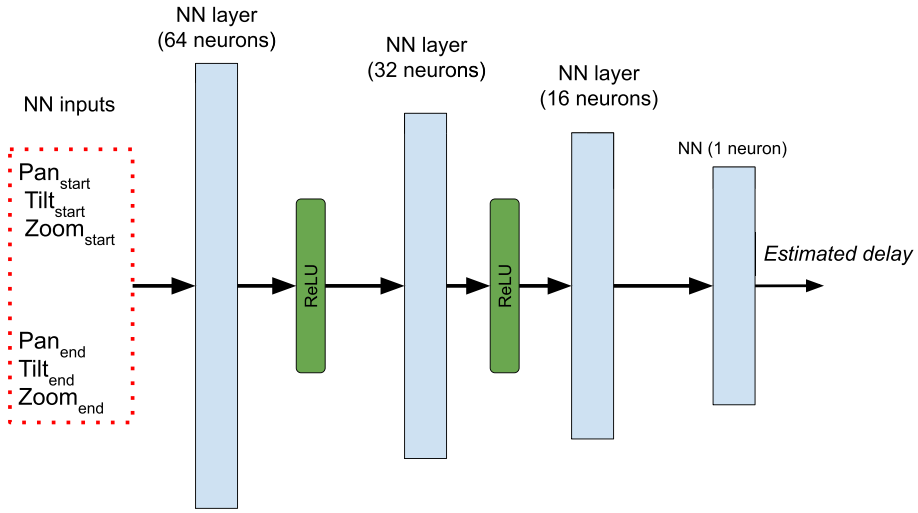

Fig. 4. The neural network architecture of the *Delay Estimator* model.

We also observe that the camera's average speed along an axis (e.g., the pan axis) changes even with the same displacement magnitude, if there is simultaneous movement on another axis (e.g., tilt). Besides mechanical delays, there are other factors such as camera response times and network communication latency that contribute to the overall latency. To deal with these challenges, we use a machine learning predictor that takes as an input the starting and ending PTZ configurations, and predicts the time it takes for the displacement (after accounting for all sources of latency). We have collected delay traces from random changes in the PTZ configurations and used these to train the neural network. We use a simple three multilayer perceptron model (shown in Figure 4) and a loss function that captures the normalized difference between the true and estimated latencies; more formally, the loss function is $|\frac{time-\overline{time}}{time}|$. Our model has a mean normalized absolute error of 4% and a mean absolute error of 40 ms. We denote this model as *Delay Estimator*.

Modeling computational delays: In real world experiments, we run a *tiny-yolo* module to detect targets in the scene. To emulate edge device computations, we run all computations on a standard computer (with no GPU). Because of that and due to the large frame size of the HD camera, we observe that the computational latency of tiny-yolo is very large (up to 3s in some cases), which causes significant latency. To account for this delay, we run 1,000 tiny-yolo queries and record their response times. The response time of 95% of the queries are between 1.8s and 2.5s. We use the average (2.15s) and add a random value in the range of ±0.35s when processing each frame, so that the model can account for computational latencies that occur in real deployments.

Modeling camera focus delays: We also experience a delay during the process of the camera focusing on a target. Specifically, we observe that images collected even when the camera has slight motion are blurry and cannot identify observed targets. Hence, we stabilize the camera at its selected location for 0.2s to ensure that no mechanical noises affect the quality of captured images. We account for this latency in the training as well.

3.2.3 Training the RL Agent. We train `AcTrak` with multiple episodes where each episode is a training video that is made available to the simulator. A flowchart of the training process is shown in Figure 5. The frame is processed to compose the RL state that is made available to the agent. The agent then selects an action either based on its learned policy or by randomly selecting

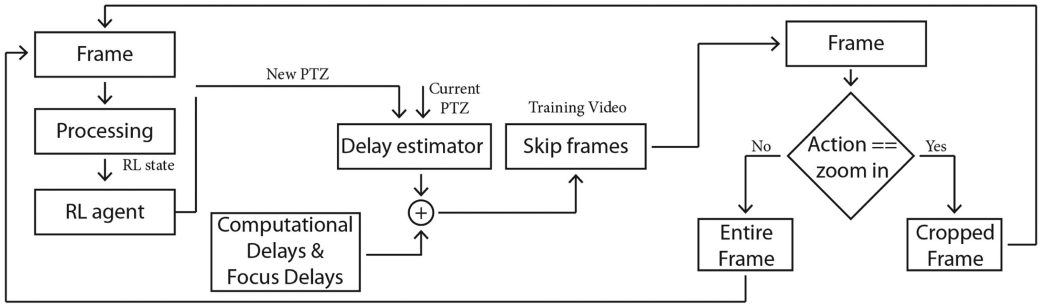

Fig. 5. A given frame is processed to compose the state as described in Section 3.1, upon which the agent selects an action (selecting the new PTZ). We compute the latencies associating with PTZ change. Subsequently, we skip a number of frames that correspond to the computed latency and use the first retrieved frame. If the selected action is zoom-in, a cropped frame is passed to the frame processor. Otherwise, the entire frame is passed.

an action from the action space depending on the value of ϵ (RL exploration rate). We assume that the camera does not acquire images of the scene while it is processing a frame and changing its PTZ configuration. For computing the mechanical latencies, the new PTZ associated with the selection of the action is now provided, along with the current PTZ, to the *Delay Estimator*, which returns the estimated time (latency incurred) that the camera would need to make such a move. To account for the delays, we skip a number of frames that correspond to the latency obtained from *Delay Estimator* and other latencies, and feed the first retrieved frame to to the object detection system for the subsequent processing.

If the selected action is a zoom out, then the entire frame is considered and the agent can observe all the targets in the scene and record their updated locations. With a zoom in, a cropped image of the frame that makes "the ratio of the size of the bounding box surrounding the target to the cropped image" equal to a pre-selected value, denoted as M, and centered around the target's last observed location, is provided to the object detection system. If the system detects a target(s), it compares its features with those in the existing list of targets, and if there is a match, the system updates the target's most recent location. Otherwise, it adds the newly identified target(s) to the list of the existing targets. Next, the system computes the rewards as described in Equation 3. A new state is created with the most recent locations of the targets, and the system continues in the same fashion. During test time, we set the exploration rate to zero so that the agent relies solely on its learned policy.

Differences between the trained agent with the live camera and that with the public datasets: When processing public datasets, both in training and testing, we exclude the computational and camera focus latencies so that we can observe the performance under the effect of the mechanical latencies solely. In real camera experiments, we used all the associated latencies while training the agent so that it can work in live experiments and showcase the performance in this realistic setup.

4 EVALUATIONS

In this section, we present results from simulations, and from real experiments on our camera platform, to showcase the effectiveness of AcTrak.

Baselines: We consider the following baselines:

(1) *Standard tour*: We propose this simple baseline that makes the camera *greedily* hover only across the hot spots (areas of interest). We select four hot spots (the most heavily populated

locations) using the method proposed in [15]. Note that we did an exhaustive search to find the locations which have the highest populations per unit time; we also find via a search of a plurality of dwelling times that the best value for this parameter is 1 s.

(2) *Panoptes* [15]: This is similar to the Standard tour but uses a machine learning model to predict targets' mobility. The model takes as input a target's location and predicts its new hotspot location after a "look ahead" time period. If Panoptes predicts a new hotspot location for a target, it re-schedules the camera tour accordingly (details in [15]). The look ahead times depend on the hotspot locations, and are computed by measuring the average time targets in the dataset take, to move from one hotspot location to another. The mechanical delays for the camera to switch from one location to the other are accounted for while computing the look ahead time periods.

(3) *Tracking greedy (greedyB)*: Inspired by Panoptes and Standard tour, this is a greedy algorithm which, instead of hovering over hot spot locations, sequentially hovers over targets' last observed locations and zooms out to discover if new targets have arrived. This baseline uses the same system setup and pre-processing modules as that of `AcTrak` (see Section 3). We add rule based enhancements to the previous baselines for better performance i.e., if the camera views a target with an incorrect zoom, it zooms in further for the higher resolution image. Since the targets can appear anywhere and not just at hotspots, and move arbitrarily, we do not try to perform look ahead predictions; we show that greedyB already outperforms Panoptes later.

Metrics: We consider the following evaluation metrics.

- *Time gap between a target's entry and its discovery (TG)*: The difference between the time the camera first observes a target and the time of the target's entry to the scene.
- *Percentage of unseen targets (UT)*: The fraction of targets that enter the scene but leave unseen by the camera.
- *Percentage of targets with zero zooms-ins (ZZI)*: The fraction of targets observed by the camera via a zoom out but for which, the camera never acquires a zoom-in image.
- *Number of visits per target (NumV)*: The number of high resolution shots acquired for each target normalized by the total time the target spends in the scene (# of visits/second).
- *Maximum time away from target (MTA)*: The maximum gap for which the target's activity was not monitored (i.e., the maximum time between any two consecutive visits to the same target or the time between when the last high resolution target image was acquired and when the target left the scene).

Model: Our model is implemented using Keras (TensorFlow based platform) and it is trained on a Tesla 100 GPU. As discussed in Section 2.2, we use the DDQN approach [24], where two neural networks with identical architectures but with different weights (updated while learning), are used to compute the Q-values of the state action pairs. The neural network architecture is described in Figure 6. We set the discount factor, γ, to be 0.975 and the exploration rate, ϵ, decays from 1 to 0 at a rate of 0.00001. The learning rate of the neural network is 0.00001. We use the concept of experience replay [16], where we train the agent on its past experiences. We store the actions, states, and rewards from the last 30,000 camera transitions in memory, where we sample from to train the agent. We show the learning curves of `AcTrak` in terms of the average accrued rewards as a function of number of steps taken by the agent in Figure 7.

4.1 Evaluations with Datasets

Setup: For our dataset based evaluations we use the Virat [38] and Zara datasets [39]. Zara consists of two videos that are taken from a birds eye view in front of a shopping mall, where most of the

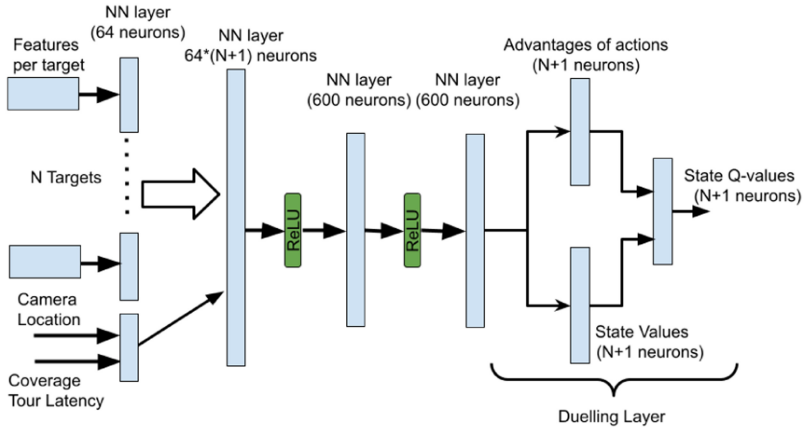

Fig. 6. AcTrak Model Architecture: The neural network consists of multiple layers as shown. First, for each target, a vector composed of its collected features (i.e., features related to timeliness, location, and number of visitations as described in Section 2.2) is fed to a NN layer of size 64. The camera location and time coverage tour latency (both features are part of the state) are concatenated and fed to an NN layer of size 64. Subsequently, the outputs are concatenated into a layer of size 64 * (N+1) with a ReLU activation function. Subsequently, the output is fed to two subsequent neural layers followed by the RL duelling layer.

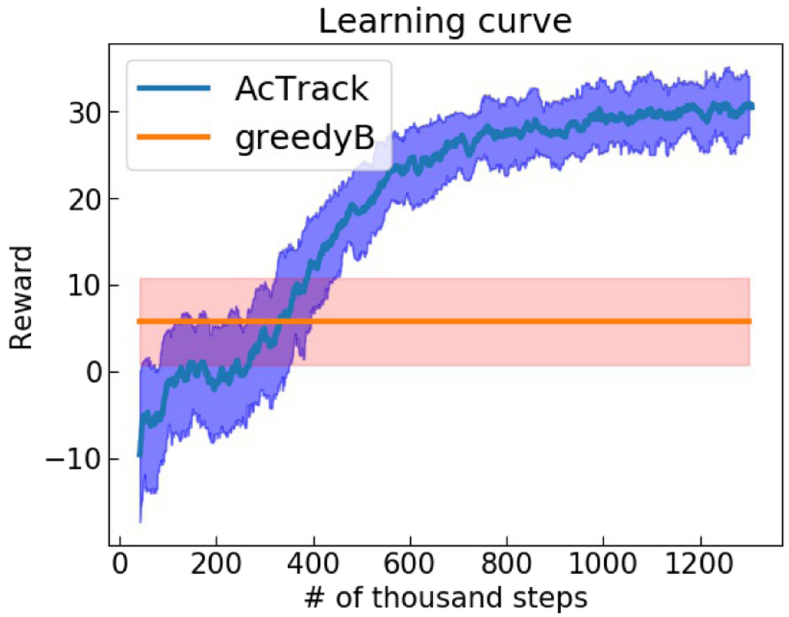

Fig. 7. Average reward accrued by the AcTrak agent and *greedyB* (with 10% and 90% confidence intervals) as a function of # of steps taken. This is collected with the Zara dataset.

targets are seen almost with the same depth with respect to the camera. The duration of the first video is 7 minutes and the second is 6 minutes. The videos have footages of large crowds (148 and 204 pedestrians) with different motion speeds and arbitrary entry and egress times. We use the first video for training and the second for testing. Virat contains 28 videos of various durations. The

Table 2. Baselines Evaluations (Zara Dataset)

Metric	TG	UT	ZZI	MTA
Standard tour	5.24s	59%	0%	98s
Panoptes [15]	5.56s	52%	0%	91s
greedyB	1.79s	0.89%	16%	8.1s
AcTrak	1.25s	0.09%	8.1%	6.2s

footages are from a camera looking at the scene from an inclined angle where targets are viewed with different zooms. Experiments with Virat show that AcTrak works with different camera view angles and heights. Our model is trained with the 20 shortest videos and tests are on the other 8 longer videos (to evaluate AcTrak in long term surveillance).

Targets perform different activities in the scene (e.g., walking, standing at a STOP sign); thus, the overall times they spend in the scene vary. Hence, in each dataset, we sort the targets according to their speeds in the scene. We estimate a target's speed by measuring the target's overall displacement divided by the overall time the target spent in the scene. We report the results on 50% of the fastest and slowest targets, individually to show the merits of our method (they are denoted as *fast targets* and *slow targets*, respectively).

Tuning reward function coefficients: The coefficients are selected based on synthetically collected validation data from the training instances, where we instrument the targets in the training dataset by changing their arrival times to the scene. We tune the coefficients with the objective of minimizing the number of unseen targets and configure AcTrak so as to acquire high resolution single shots (the exact coefficients values are defined within the experiment descriptions corresponding the specific dataset). We also report other results (with other coefficients) where we favour the acquisition of continuous high resolution images of targets (video) at the cost of less frequent coverage tours (shown in Table 4).

Baseline comparisons: We mentioned earlier that the *tour* mode incorporated in both Standard tour and Panoptes [15] fail when the goal is to track activities of targets that may span the entire scene of interest (locations not limited to hotspots). To show this, we evaluate their performance using the Zara dataset and show the results Table 2. The two most important metrics that we are interested in are UT and MTA, because the goal is to ensure discovery of targets that appear in the scene (measured by UT), and track their activities frequently with small time gaps (captured by MTA). Clearly Tracking greedy (*greedyB*) outperforms the other two baselines with regards to both these metrics. This is because many targets do not appear in the locations of interest (i.e., the hot spots). Furthermore, many targets may arrive at these hot spots and leave while the camera is pointed at another hot spot. Although Standard tour and Panoptes are superior in other metrics, the big gap in UT, TG, and MTA make them unsuitable in achieving our dual objectives. Since they are outperformed by Tracking Greedy, we exclude them from the following discussions and use Tracking Greedy (with the label *greedyB*).

We note that adding a prediction module to *greedyB*, to predict or account for targets' future locations is difficult because it is hard to know when and where targets could be, in our scenarios. A simple prediction model similar to the one used in Panoptes [15] cannot be applied because, unlike Panoptes, the target can exist in any location of the scene, which makes the prediction uncertainty high; we have run experiments and observe that it does not make *greedyB* any better (not shown due to space constraints) and even makes it worse many times. Coming up with a more sophisticated location prediction method in such scenarios is difficult. This is because the problem of what and when to observe data (i.e., targets' locations) in order to maximize a long term objective (e.g., reliable mobile target location prediction) is a hard problem and is referred

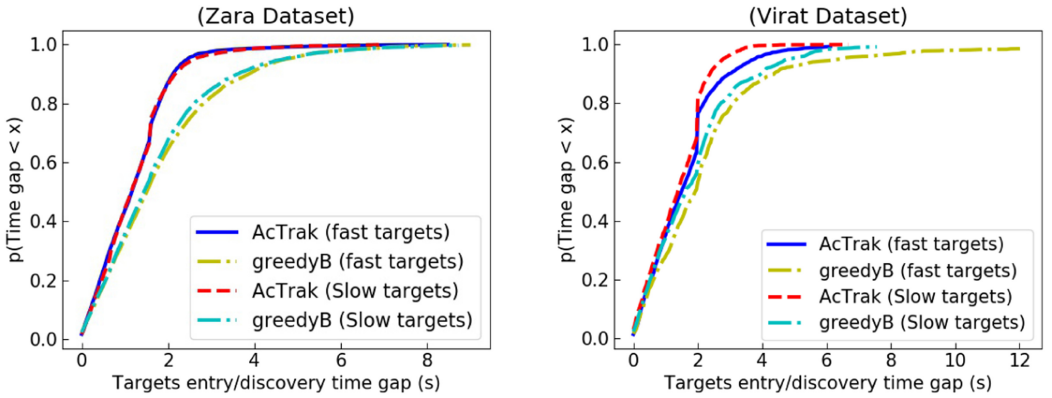

Fig. 8. CDF of the time gap between target's entry and discovery for Zara (left) and Virat datasets.

to as Active sensing [40, 41]. While there are existing solutions that tackle similar problems in different contexts, none, to the best of our knowledge, has tackled this problem before. The closest is called PatchDrop [41], and we discuss in (Section 6) why it cannot be utilized in our scenario. We point out here that in fact, AcTrak is an deep RL based method that does this task implicitly and outperforms all the baselines (as shown next).

Experiments with Zara: In these experiments, we set α_1 and α_2 to be 0.15 and 0.08, respectively. We set τ to be 2s (50 frames), β to be 0.00035 and ρ to be 0.85. We first evaluate AcTrak in terms of the **cumulative distribution function (CDF)** of the time gap between the arrival of targets to the scene and when they are first discovered by the camera. As shown in Figure 8, 90% of the arriving targets are discovered within 2s, while *greedyB* takes around 4s to do so, for both fast and slow targets. The average time gap with AcTrak is 1.25s for both slow and fast targets, whereas the *greedyB*'s time gap is 1.76s and 1.82s for slow and fast targets, respectively. This is because AcTrak adapts its zooming out frequency with the expected arrival rate leading to fast capture of targets that step into the scene. *On average,* AcTrak *detects new targets in the scene 30% faster than greedyB.* As shown in Table 2, due to this fast detection of arriving targets, our agent rarely misses targets that arrive to the scene.

From the observed targets, AcTrak captures high resolution images of $\approx$ 92% of the targets while *greedyB* does so for $\approx$ 84% of the targets. This is because AcTrak prioritizes visits to targets that were not visited before, in addition to smart scheduling of its zoom-ins to capture as many targets as possible without wasteful PTZ changes.

In Figure 9, we evaluate the maximum time gap between two consecutive zoom-ins on the same target. AcTrak has higher gains in the case of fast targets in comparison with the case of slow targets. This is because, with *greedyB* fast targets' locations observed by the camera become outdated faster than those of the slower targets; hence, when the camera visits a target at its previous location, it does not observe the (fast) target (leading to a wasteful zoom-in visit). AcTrak avoids wasteful zoom-ins by observing target displacements and adapting target visits accordingly.

We finally evaluate the number of visits for each target as shown in Figure 10. Both AcTrak and *greedyB* relatively, exhibit the same performance. The key difference between our agent's visits and *greedyB*'s visits is that the agent's visits are distributed over time whereas *greedyB* visits are concentrated over the same time periods leading to big time gaps where certain targets' activities are not monitored (this can be verified from Figure 9).

Experiments with Virat: For this dataset, we tune the values of the coefficients α_1, and α_2 to be 0.3, and 0.06, respectively. We set τ to be 4s (100 frames), β to be 0.00025, and ρ to be 0.75.

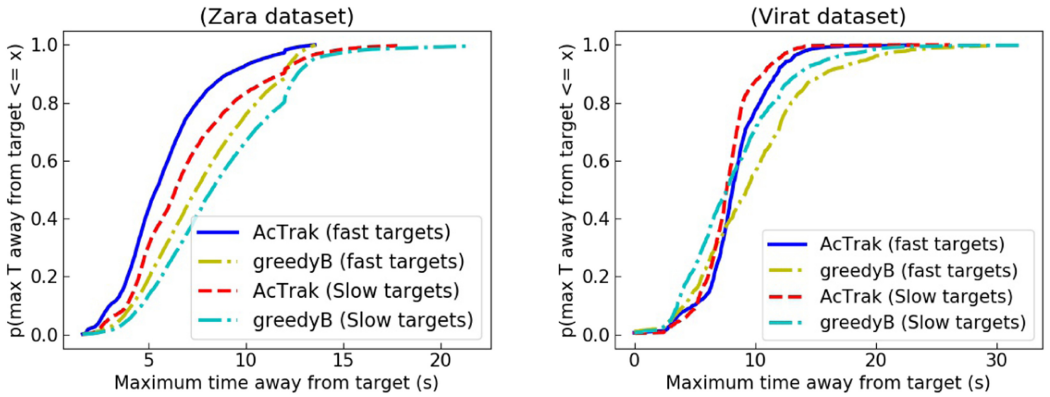

Fig. 9. CDF of the maximum time away from target for Zara (left) and Virat datasets.

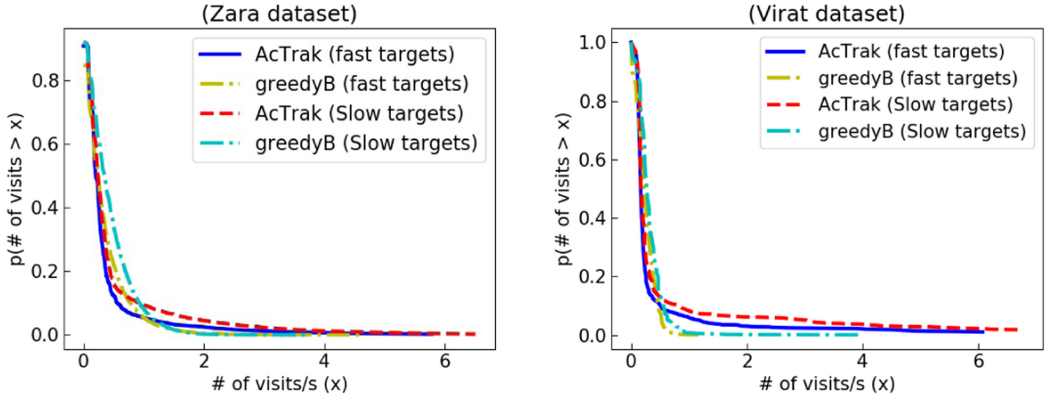

Fig. 10. Complementary CDF of target's number of visits/s for Zara (left) and Virat datasets.

Table 3. Virat Dataset Results

Methods	TG	UT	ZZI	MTA
AcTrak	1.49s	1.1%	0.027%	8s
greedyB	2.3s	1.3%	0.11%	9s

As shown in Table 3, the percentage of unseen targets is relatively higher in comparison with the other dataset (but we still outperform *greedyB*). This is because the dataset has mostly videos with random start and end times, and so some targets may appear in the scene in only a very few frames; they appear in the scene right before the video ends or they leave just after the video starts. With the other metrics, we observe a similar trend with this dataset (compared with the Zara dataset) but with lower gains. This is because the camera is positioned with an inclined angle with respect to the scene of interest where targets can move deeper in the scene (their distances to the camera increase) but target's displacement in terms of x and y coordinates (translated to pan and tilt) does not change significantly. Thus, the camera does not lose track of targets easily, as is the case with the Zara dataset (even if the zoom magnification is not met when the camera visits a target, it can potentially know its updated location). The plots associated with the results of this dataset are Figures 8, 10, and 9. We observe that in cases with sparse target arrivals, *greedyB* zooms

Table 4. Continuous High Resolution Shots (Video) Results

Methods	**TG**	**UT**	**ZZI**	**MTA**	**TCVA**
AcTrak	1.95s	1%	18%	9s	1.2s
greedyB	3.3s	5.4%	24%	15.8s	1.1s

Table 5. Performance with Varying Crowdedness

Arrival Rate (λ)	Method	TG	UT	ZZI	MTA
λ_{small}	AcTrak	1.15s	0%	0%	5.2s
	greedyB	1.3s	0%	6%	6.7s
λ_{medium}	AcTrak	1.4s	0%	3%	7.6s
	greedyB	1.9s	0%	11%	9.7s
λ_{large}	AcTrak	2.0s	0.3%	9%	10.1s
	greedyB	2.4s	0.8%	20%	11.5s

out unnecessarily wasting opportunities for acquiring high resolution images. AcTrak avoids these and does better in terms number of visits to each target as shown in Figure 9. Note that this effect is also seen in our in real camera experiments discussed later.

Understanding the variable τ: In this experiment, we tune the coefficients of the agent's reward function to bias it towards acquiring sequences of high resolution images of targets (videos) rather than single shots as in earlier experiments. Here, our aim is to showcase the merits of AcTrak in performing different tasks and its flexibility in tuning the trade-off between zoom ins and coverage tours for different applications. The key parameter that we tune towards this is τ; unlike previously, we now select a small value to cause the agent to acquire a sequence of images (video) of targets. We run again the prior experiment on Zara, but with a different set of reward coefficients. We set α_1 and α_2 to be 0.0 and 0.025, respectively. We set τ to be 0.08s (2 frames), β to be 0.00025, and ρ to be 1. We use the same baseline (*greedyB*) but we vary its dwelling time, so that it can acquire a sequence of images of the same target. With regards to this experiment, we report the average time of continuous video acquired per target, denoted as *TCVA*. The results are in Table 4. As shown, AcTrak misses 1% of the targets while *greedyB* misses 5.4% of the targets.

4.2 AcTrak Performance with Change in "Crowdedness" in the Scene

In many practical scenarios, the number of targets (referred to as crowdedness [42]) in the scene changes over the day, even in the same scene. We show the performance of AcTrak with varying crowdedness. We consider a single scene associated with the Zara dataset but we create synthetic data where we tune the arrival rates of targets to create varying crowdednesss. In particular, we tune target arrival rates (i.e., number of targets per minute) in accordance with a Poisson distribution, with mean arrival rates of $\lambda_{small}, \lambda_{medium}$ and λ_{large} per minute; these values are 2, 5, and 10, respectively. Targets' entry and exit locations into and from the scene are selected randomly and the targets speeds are chosen randomly from three different speeds equal to the 25% and 50% and 75% percentile of target speeds in Zara dataset. We evaluate AcTrak using the trained model on the Zara dataset on this setup and the results are reported in Table 5. As shown, as the arrival rate increases, it becomes harder for the camera to visit targets frequently. AcTrak outperforms *greedyB* with respect to all metrics of interest. For example, in a scenario where the target's arrival rate is tuned to be λ_{large} (more frequent arrivals), the percentage of observed targets with zero zoom-ins (ZZI) is 20% when greedyB is used. However, while using AcTrak, the ZZI is only 9%.

Table 6. Results of Real World Experiments

Methods	TG	UT	ZZI	MTA
AcTrak	5	0%	0%	25s
greedyB	7	0%	0%	33s

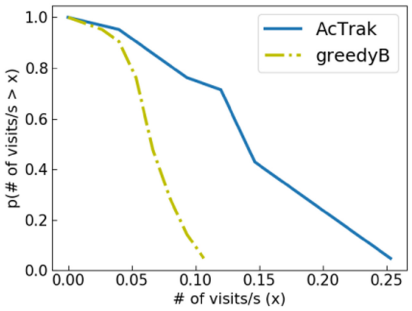

Fig. 11. Live camera: frequency of visits per target. (*NumV*).

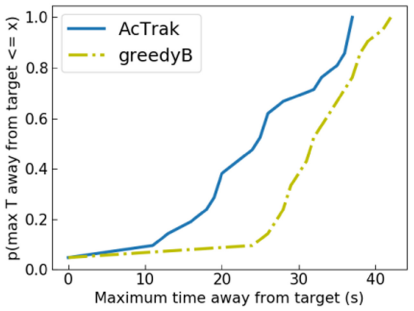

Fig. 12. Live camera: maximum time camera is away from a target. (*MTA*).

4.3 Real World Experiments

We showcase our approach in an IRB approved, real-world setting where volunteers walk randomly in a scene on interest. Figure 1 (left) shows the scene at which the real experiments are conducted. To train our agent, we have collected a total of 12 traces of individual random walks (each ranges from 90 seconds to 150 seconds), and we obtain their PTZ coordinates using our *Location Estimator* module. In our simulator (used for training), we further instrument those targets to vary their arrival times over different execution runs, and create many different possible target interactions to enable the agent to learn how to adapt to different scenarios and conditions. For this experiment, we set up the reward coefficients (α_1, α_2, β and τ) to be 0.15, 0.1, 0.00001, 8s (200 frames), respectively.

At test time, we have four volunteers that were asked to move randomly in the scene of interest. We select their entry locations and time instances arbitrarily for each new experiment. We have repeated the experiment five times using our trained agent, and five times using the baseline algorithm). Each experiment lasts for 3 minutes. Our model on average obtains 38 zoomed in shots (per experiment).

Quantitative results. The results from the real camera experiments are shown in Figures 11 and 12 and Table 6. Because of higher PTZ change latencies (associated delays with tiny-Yolo), we observe that the baseline makes many more wasteful moves (zooms on outdated locations of targets or zooms in on a target with a zoom level lower than the required zoom). We also observe that due to the sparsity of arrivals (only four targets over a long period), the baseline zooms out more often than necessary. Our agent outperforms the baseline by a big margin in terms # of acquired high resolution shots. In particular, on average, the agent obtains *2x more high resolution images of targets.*

Case studies: Next we present two microscopic case studies of both algorithms from our real deployment (see Figures 13 and 14).

Unnecessary zoom-outs (Figure 13): In this scenario, a single person appears alone in the scene and stays for some long time (around 50s). *greedyB* reacts to this by alternating between zoom ins on the target and zoom outs to continuously check for new targets. However, AcTrak, learns upon

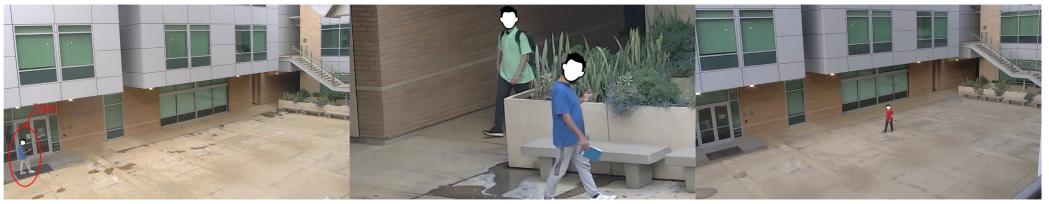

Fig. 13. Snapshots from the surveillance videos obtained by AcTrak and the baseline (we hide targets' faces for privacy reasons). In the first image, a single target is observed via a zoom out view (blue shirt at bottom left near door). AcTrak zooms-out much less frequently than *greedyB*, learning that the scenario does not change (no new targets arriving). The second image shows that AcTrak while zooming in on the first target, discovers a new target that appears in the scene (light green shirt). The third image shows *greedyB*'s behavior in a similar scenario; it zooms out much more than necessary. There was no other target stepping in to the scene, and instead of zooming in on the red target it zoomed out; this leads to much fewer high resolution images per target, thereby potentially missing activities.

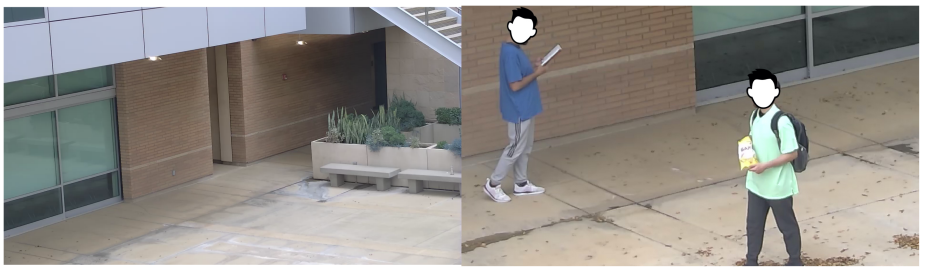

Fig. 14. Snapshots from the surveillance videos obtained by AcTrak and the baseline (we hide targets' faces for privacy reasons). The first picture shows a case where *greedyB* goes back to a target and finds the location empty because the target has already left its marked location. This is due to the computational and mechanical latencies and the unorganized patterns of zoom-in on targets (which *greedyB* does not account for). The final image shows that AcTrak is able to zoom in on the green shirt target when eating chips and at that point it also re-locates the blue shirt target.

zooming out and not discovering new targets; it then zooms in on the target for longer times and avoids wasteful zoom outs.

Adapting to high latency (Figure 14): We observe that the high latency makes *greedyB* zoom in on empty places because the targets have already left their marked locations (stale data). AcTrak's RL agent learns to tackle this issue and thereby avoids zooming in on outdated locations and performing unnecessary zoom outs. It does so by visiting fast moving targets more frequently, avoiding expensive moves and by opportunistically obtaining updated target locations while it is capturing other targets (other targets may appear in the background with lower zoom requirements).

5 DISCUSSION

Single camera vs multicamera: A single camera can address the multiple goals compared to using multiple cameras, but provides significant cost reductions. A quick search on Amazon.com [43] and Bhphotovideo.com [44] reveals that a two lens PTZ camera and supporting optical zoom is more than twice the price of a single camera. Thus, our approach can be desirable to small business owners who can just use one standalone camera instead of networking multiple cameras with associated issues such as synchronization and configuration issues, camera calibration issues

[45], incompatible software from different vendors, and so on. If two cameras are used, one can provide target locations at all times and the other can be used for acquisition of high resolution images. AcTrak can be used with the second camera to manage the frequency and patterns of zoom ins on targets. A study of how to harmonize multiple cameras for surveillance is left to future work.

AcTrak's computational overhead: We process the captured frames from the camera using popular models from computer vision viz. MiniYolo (identifying targets) and humanReid (associating observed targets with existing targets). Recall that the baseline, *greedyB*, uses the same preprocessing modules that are used by AcTrak. The only difference is in executing the trained RL agent that determines what action to take given a state. We run the 10,000 random queries using the trained model for Zara dataset on a regular Apple macpro machine, and find that the average response time in terms of determining the proper action, is 4.3ms which is negligible in comparison to the latencies incurred with the computer vision models themselves and the camera's mechanical latencies (which are of the order of seconds). Our expectation is that this time will not increase by three orders of magnitude, even if a less powerful computation machine was used.

Impact of the tiny-yolo's performance on the takeaways: AcTrak accounts for latencies due to various factors and this is a key reason why it outperforms *greedyB*. We show that AcTrak outperforms greedyB in various simulation and real camera setups where different latencies (e.g., mechanical and computational latencies) are accounted for. In the simulation setup, computational costs are ignored (including those relating to tiny-yolo) and only mechanical latencies are included; this causes the overall latencies to be much smaller than in the case of the real camera setup. This is similar to lowering some of the processing latencies in the real deployment (that might decrease from GPU usage). Based on these results, we do not expect that the take-aways from the real camera experiments will change in flavor, when we run tiny-yolo on GPU or when using a low resolution camera (lower processing delays).

AcTrak's performance when the runtime setup is different from the training setup: In this part, we discuss different forms of 'deviation' between training and test scenarios in the following list:

- AcTrak*'s trained model can adapt to crowdedness change:* We observe that the trained model can work even if crowdedness changes over the day with no need to retrain. To showcase this, we run a set of experiments, wherein we vary the arriving rates of targets to the scene of interest. We use the setup associated with the Zara dataset, but we create synthetic data with three different arrival rates. We show that the same exact same trained agent on the Zara dataset can work even if 'crowdedness' changes and can still outperform the baseline. We present these results in Table 5.
- AcTrak*'s transferability across different scenes:* AcTrak cannot transfer to scenes that are not part of the training. For example, the model trained on the Zara dataset may not work on Virat dataset and vice versa. This is because the nature of the scene varies. In the Zara dataset, all targets are viewed from a bird's eye view (e.g., all of them are approximately at the same distance); thus, there is not much variation in the zoom level required to zoom in on targets. In contrast, in the case in Virat, the camera is observing the scene from an angle such that the target distances from the camera vary significantly. Thus, the policy learned by the agent from the training samples from Zara dataset cannot be applied on Virat. In this work, we do not consider transfer learning (i.e., learning a global model with different scenes). This is beyond the scope of this work and requires significant new effort.

- AcTrak*'s performance may degrade if target profiles significantly change:* In cases where target profiles significantly change, one can expect the performance of AcTrak to degrade. For example, let us say AcTrak is trained on a dataset from a public park for kids. If the park is repurposed and now teenagers with skateboards and bikes can play in the park, the performance of AcTrack may degrade since the testing data/setup has completely changed from the training setup. In this case, retraining with the new target profiles may be necessary to boost the performance. We argue that in practice such dramatic changes in target profiles are uncommon. In scenes with various target profiles, the training dataset collected from the scene should cover those various profiles. Thus, AcTrack is expected to work. In conclusion, as long as the training video(s) have a good coverage of scenarios and target profiles expected to be present during test time, AcTrack is extremely effective.
- AcTrak *can be extended to scenes with dynamic target profiles by using an ensemble of models:* In cases where target profiles and dynamics change dramatically, AcTrack can be simply extended by using an ensemble of trained models, each tuned to the specific dynamics with specific target profiles. During run time, the camera uses the maximum zoom out to observe targets' movements (i.e., targets' profiles) for a small time period. It can then use a neural network which is trained to determine the model that best fits the profile - (it outputs the best model for the specific scenario). For example, consider a public street with walking pedestrians. In the uncommon event of protest/ parade, the model associated with such events (e.g., parade/ protest) can be deployed on the camera. We leave examining this possibility for the future along with the transferability of a trained model across different cameras.

Sensitivity of selected reward coefficients: Any RL Framework is sensitive to reward function coefficients (i.e., hyperparamaters) [46, 47]. The rewards need to be tuned with respect to the given setup. We give an example by considering the coefficient τ. To recall, an agent is given a positive reward for visiting the same target if the time gap between two consequent zoom-in is larger than τ. In the Virat dataset, target zoom requirements vary significantly and thus moving from one target to another incurs higher delays in comparison with Zara dataset where all targets have very similar zoom requirements. When selecting a smaller τ (=2s) similar to Zara dataset), we observe that the camera favors the continuous zoom ins of targets (video) and does not move quickly to different targets. The exercise suggests that using the Virat coefficients blindly, with the Zara dataset or vice versa causes the model to underperform, even to significant extents in some cases. One solution is to use the correct set of hyperparameters with each member of the ensemble from the previous paragraph; in other words, each model in the ensemble has its own set of hyperparameters appropriate for the scenario in which it is to be used. Thus, by using the ensemble, the hyperparameters are also properly changed as scenario dynamics evolve, and thus, can provide superior performance. These aspects are beyond the scope of this current paper, and will be investigated in future work.

6 RELATED WORK

There are works that model camera based tracking as an NP-hard, travelling salesman problem [14]. The problem is different from ours in two ways. First, it does not consider dynamic new target arrivals to the scene (where the camera has to capture and subsequently track them). Second, in our context, targets move arbitrarily with different velocities. Prior work such as [14], impose a pre-determined deadline within which a target has to be visited [14]. This assumption is unrealistic when there is unexpected mobility (the target will be missed). We assume no such deadlines; rather, dynamically changing deadlines are implicitly learnt and the camera avoids wasteful moves (e.g.,

when it zooms in on an expected target location, it does not find it). Further, AcTrak induces zoom outs with appropriate frequencies to detect new targets.

There is work in multi-camera networks on co-ordination among the cameras to achieve a particular goal such as target tracking [9–11] or scene coverage [48] or both [8, 49–51]. The key difference is that these works assume using multiple cameras to cover the entire scene, thus obviating the need for adaptation to the arrival rate of new targets. However, deploying additional cameras incurs cost, and if certain areas are sparsely populated deploying cameras to cover them always is wasteful (dual optical PTZ cameras are at least twice the price of mono optical PTZ cameras [6]). We consider a more cost effective single camera setting, wherein the frequency of zoom outs to cover the scene fully, are tuned in accordance with target arrival rates while zooming in at other times to acquire high resolution images for activity tracking. We point out that in [8, 50, 51], the focus is on only obtaining high resolution images of existing targets.

There is prior work on using RL to control a PTZ camera's to achieve a single simpler objective. In [52], the authors use RL to opportunistically zoom in on targets that satisfy certain conditions (e.g., a frontal pose available for face recognition). In [53], the authors use RL to rapidly tune the camera's PTZ to zoom in on a target with a required magnification level from a zoomed out view. However, unlike in AcTrak, they do not consider the problem of fine-grained tracking of multiple targets nor do they invoke zoom outs to capture new arrivals.

A recent work called PatchDrop formulates an RL approach wherein the goal is to select where and when to acquire high resolution data (patches) to train a model while preserving training accuracy [41]. This work is different from ours in two aspects. First, the work assumes the availability of lower resolution data at all times (the environment is completely observable). In contrast, we assume only a single camera and the environment to be only partially observable while zooming-in. Second, the work does not account for the delays associated with switching between low and high resolution data acquisitions (incurred by PTZ mechanical movements) and between different patches (different targets).

Very few efforts consider a realistic scenario setup like ours viz., the use of a single camera system with multiple objectives [15, 54]. In Panoptes [15], the authors propose a mobility-aware camera scheduling algorithm over a few pre-selected fixed locations (maximizing coverage in these locations only). In contrast, we consider mobile targets whose locations change arbitrarily (not tied to fixed locations). We show in the evaluation section that their approach is not suitable for target activity tracking.

7 CONCLUSIONS

In this paper, we design a framework, AcTrak, for the adaptive steering of a camera to zoom in on targets to track their activities, while appropriately zooming out to detect newly arriving targets to a scene. AcTrak incorporates an RL agent that selects the best PTZ configuration that achieves good trade-offs between the aforementioned goals. Via evaluations on two datasets, we show that AcTrak detects new targets 30% faster compared to the best current non-adaptive baseline we are aware of. Our real world experiments with volunteers show a *2*× improvement in terms of proper zoom ins of targets in the scene compared to the baseline.

APPENDIX

A VISUALISATION EXAMPLE FROM ZARA DATASET

We finally show in Figures 15 and 16 an example of a sequence of actions taken by *greedyB* and AcTrak in the same specific scenario, respectively. As shown, AcTrak movements are much more efficient.

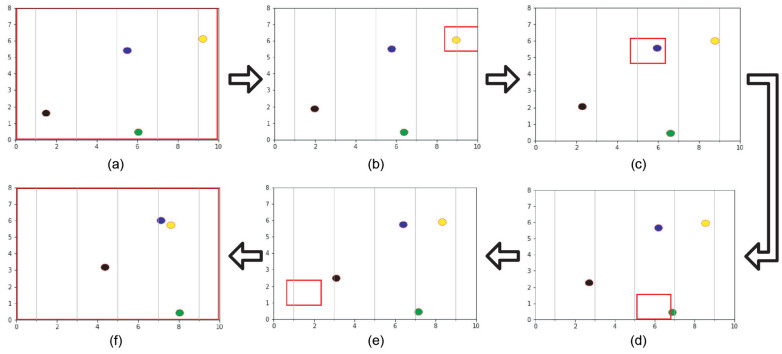

Fig. 15. *greedyB camera's movement:* We show a snapshot of a trace from Zara dataset, where four targets are moving in the scene, and the camera zooms-in to acquire high resolution images of targets. We use different colors for targets to distinguish them. The red rectangle is the view port of the camera (note that in zoom in action, the view port is small). The camera acquires updated locations of targets via the zoom out action in frame (a). It iterates over the targets sequentially while successfully acquiring high resolution shots of yellow and blue targets in frames (b) and (c), respectively. However, it zooms on outdated locations when it zooms on green and black targets due to their fast movement as shown in frames (d) and (e). Finally, it zooms out to obtain the updated locations of targets while also observing if new targets arrive.

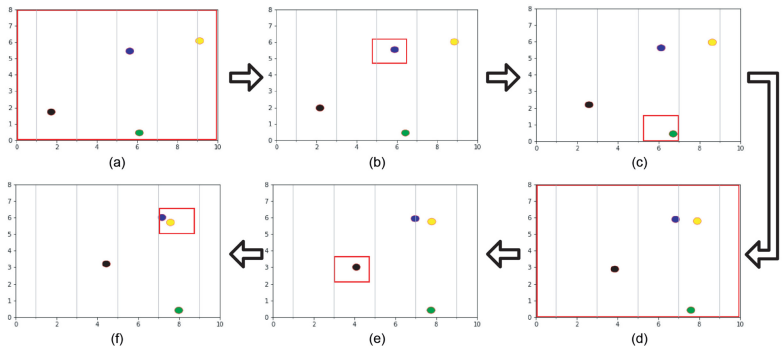

Fig. 16. AcTrak *camera's movement:* After acquiring the updated locations of targets in frame (a), it zooms in on blue and green targets in frames (b) and (c), respectively. Unlike the case with greedyb, Actrak zooms out to acquire updated locations of targets before zooming on black and yellow targets. After zooming on the black target, it zooms in on the yellow target while simultaneously observing the blue target (to avoid having a dedicated zoom in for the blue target).

REFERENCES

[1] The Verge. Dec, 2019. The US, like China, has about one surveillance camera for every four people, says report. https://www.theverge.com/2019/12/9/21002515/surveillance-cameras-globally-us-china-amount-citizens. (Dec, 2019). [Online; accessed March-30-2020].

[2] Andy Penfold. [n. d.]. How IoT is reshaping the future of Video Surveillance. https://www.securityandsafetythings.com/insights/iot-reshaping-future-surveillance.

[3] Sonali P. Gulve, Suchitra A. Khoje, and Prajakta Pardeshi. 2017. Implementation of IoT-based smart video surveillance system. In *Computational Intelligence in Data Mining*. Springer, 771–780.

[4] Avipas. [n. d.]. AViPAS model AV-1081 manual. https://e7aba150-670b-4b8b-9a25-311a84251d5f.filesusr.com/ugd/6b6a18_34ff6afc20914be9943327dc3f5a6211.pdf.

[5] Sony. [n. d.]. Remotely controlled PTZ color video camera with IP streaming. https://pro.sony/ue_US/products/ptz-network-cameras/srg-300se.

[6] alsecuritycamera. [n. d.]. Axis outdoor multi sensor IP security camera. https://www.a1securitycameras.com.
[7] Claudio Piciarelli, Lukas Esterle, Asif Khan, Bernhard Rinner, and Gian Luca Foresti. 2015. Dynamic reconfiguration in camera networks: A short survey. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 5 (2015), 965–977.
[8] Chong Ding, Jawadul H. Bappy, Jay A. Farrell, and Amit K. Roy-Chowdhury. 2016. Opportunistic image acquisition of individual and group activities in a distributed camera network. *IEEE Transactions on Circuits and Systems for Video Technology* 27, 3 (2016), 664–672.
[9] Faisal Z. Qureshi and Demetri Terzopoulos. 2009. Planning ahead for PTZ camera assignment and handoff. In *ICDSC*. IEEE, 1–8.
[10] 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* 114, 4 (2010), 463–474.
[11] 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* 114, 2 (2010), 179–197.
[12] Wiktor Starzyk and Faisal Z. Qureshi. 2011. Learning proactive control strategies for PTZ cameras. In *2011 Fifth ACM/IEEE International Conference on Distributed Smart Cameras*. IEEE, 1–6.
[13] Umair Ali Khan and Bernhard Rinner. 2014. Online learning of timeout policies for dynamic power management. *ACM Transactions on Embedded Computing Systems (TECS)* 13, 4 (2014), 1–25.
[14] Joao C. Neves and Hugo Proença. 2015. Dynamic camera scheduling for visual surveillance in crowded scenes using Markov random fields. In *AVSS*. IEEE, 1–6.
[15] Shubham Jain, Viet Nguyen, Marco Gruteser, and Paramvir Bahl. 2017. Panoptes: Servicing multiple applications simultaneously using steerable cameras. In *IPSN*. 119–130.
[16] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[17] Tianshu Wei, Yanzhi Wang, and Qi Zhu. 2017. Deep reinforcement learning for building HVAC control. In *DAC*.
[18] Shuyue Lan, Rameswar Panda, Qi Zhu, and Amit K. Roy-Chowdhury. 2018. FFNet: Video fast-forwarding via reinforcement learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6771–6780.
[19] Qi Wang, Jianmin Liu, Katia Jaffrès-Runser, Yongqing Wang, Chentao He, Cunzhuang Liu, and Yongjun Xu. 2021. INCdeep: Intelligent network coding with deep reinforcement learning. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 1–10.
[20] Stefan Schneider, Haydar Qarawlus, and Holger Karl. 2021. Distributed online service coordination using deep reinforcement learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 539–549.
[21] Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112, 1-2 (1999), 181–211.
[22] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3-4 (1992), 279–292.
[23] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.
[24] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.
[25] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).
[26] darknet. [n. d.]. YOLO: Real-Time Object Detection. https://pjreddie.com/darknet/yolo/. [Online; accessed August-10-2022].
[27] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An incremental improvement. *arXiv* (2018).
[28] Rahul Rama Varior, Mrinal Haloi, and Gang Wang. 2016. Gated siamese convolutional neural network architecture for human re-identification. In *ECCV*. Springer, 791–808.
[29] Shayan Modiri Assari, Haroon Idrees, and Mubarak Shah. 2016. Human re-identification in crowd videos using personal, social and environmental constraints. In *European Conference on Computer Vision*. Springer.
[30] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[31] Amlaan Bhoi. 2019. Monocular depth estimation: A survey. *arXiv preprint arXiv:1901.09402* (2019).
[32] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. 2018. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 340–349.

[33] Greire Payen de La Garanderie, Amir Atapour Abarghouei, and Toby P. Breckon. 2018. Eliminating the blind spot: Adapting 3D object detection and monocular depth estimation to 360 panoramic imagery. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 789–807.
[34] Robert McCraith, Lukas Neumann, and Andrea Vedaldi. 2020. Calibrating self-supervised monocular depth estimation. *arXiv preprint arXiv:2009.07714* (2020).
[35] Zhengyou Zhang. 2000. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 11 (2000), 1330–1334.
[36] Ziyan Wu and Richard J. Radke. 2012. Keeping a pan-tilt-zoom camera calibrated. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 8 (2012), 1994–2007.
[37] Harris Drucker. Improving regressors using boosting techniques. In *ICML*.
[38] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, J. K. Aggarwal, Hyungtae Lee, Larry Davis, et al. 2011. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR 2011*.
[39] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. 2007. Crowds by example. In *Computer Graphics Forum*. Wiley Online Library.
[40] Jinsung Yoon, William R. Zame, and Mihaela Van Der Schaar. 2018. Deep sensing: Active sensing using multi-directional recurrent neural networks. In *International Conference on Learning Representations*.
[41] Burak Uzkent and Stefano Ermon. 2020. Learning when and where to zoom with deep reinforcement learning. In *CVPR*. 12345–12354.
[42] Siyuan Liu, Yunhuai Liu, Lionel Ni, Minglu Li, and Jianping Fan. 2012. Detecting crowdedness spot in city transportation. *IEEE Transactions on Vehicular Technology* 62, 4 (2012), 1527–1539.
[43] Amazon.com. [n. d.]. Amazon.com. https://www.amazon.com/.
[44] bhphotovideo.com. [n. d.]. bhphotovideo.com. https://www.bhphotovideo.com/. [Online; accessed June-29-2021].
[45] Liming Song, Wenfu Wu, Junrong Guo, and Xiuhua Li. 2013. Survey on camera calibration technique. In *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, Vol. 2. IEEE, 389–392.
[46] Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. 2017. Hybrid reward architecture for reinforcement learning. *Advances in Neural Information Processing Systems* 30 (2017).
[47] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846* (2017).
[48] Zongjie Tu and Prabir Bhattacharya. 2013. Game-theoretic surveillance over arbitrary floor plan using a video camera network. *Signal, Image and Video Processing* 7, 4 (2013), 705–721.
[49] Niccolò Bisagno, Alberto Xamin, Francesco De Natale, Nicola Conci, and Bernhard Rinner. 2020. Dynamic camera reconfiguration with reinforcement learning and stochastic methods for crowd surveillance. *Sensors* 20, 17 (2020), 4691.
[50] Stefan Rudolph, Sarah Edenhofer, Sven Tomforde, and Jörg Hähner. 2014. Reinforcement learning for coverage optimization through PTZ camera alignment in highly dynamic environments. In *Proceedings of the International Conference on Distributed Smart Cameras*. 1–6.
[51] Paul Jasek and Bernard Abayowa. 2018. Visual sensor network reconfiguration with deep reinforcement learning. *arXiv preprint arXiv:1808.04287* (2018).
[52] Andrew D. Bagdanov, Alberto Del Bimbo, Walter Nunziati, and Federico Pernici. 2006. A reinforcement learning approach to active camera foveation. In *ACM International Workshop on Video Surveillance and Sensor Networks*.
[53] Dongchil Kim, Kyoungman Kim, and Sungjoo Park. 2019. Automatic PTZ camera control based on deep-Q network in video surveillance system. In *ICEIC*. IEEE.
[54] Navin K. Sharma, David E. Irwin, Prashant J. Shenoy, and Michael Zink. 2011. MultiSense: Fine-grained multiplexing for steerable camera sensor networks. In *ACM Conference on Multimedia Systems*.

Received 28 April 2022; revised 29 September 2022; accepted 20 February 2023