ViTrack: Efficient Tracking on the Edge for Commodity Video Surveillance Systems

Linsong Cheng, Jiliang Wang
School of Software and TNList
Tsinghua University, China
{chengls14,jiliangwang}@mails.tsinghua.edu.cn

***Abstract*—Nowadays, video surveillance systems are widely deployed in various places, e.g., schools, parks, airport, roads, etc. However, existing video surveillance systems are far from full utilization and have very limited capability due to high computation overhead, unknown camera information and variant video quality. In this work, we present ViTrack, a framework for efficient multi-video tracking using computation resource on the edge for commodity video surveillance system. In the heart of ViTrack lies a two layer spatial/temporal compressive target detection method to significantly reduce the computation overhead by combining videos from multiple cameras. Further, ViTrack derives the video relationship and camera information even in absence of camera location, direction, etc. To address variant video quality and missing targets, ViTrack leverages a Markov Model based approach to efficiently recover missing information and finally derive the complete trajectory. We implement ViTrack on a real deployed video surveillance system with 110 cameras. The experiment results demonstrate that ViTrack can provide efficient trajectory tracking with processing time 45x less than the existing approach. For 110 video cameras, ViTrack can run on a Dell OptiPlex 390 computer to track given targets in real time. We believe ViTrack can enable practical video analysis for widely deployed commodity video surveillance systems.**

I. Introduction

The global video surveillance market has grown at a high speed in recent years. According to the latest market research report [1], video surveillance market is expected to worth USD 71.28 Billion by 2022, at an estimated CAGR (compound annual growth rate) of 16.56%. Video surveillance markets have developed very fast due to the rapid development of urbanization construction. Nowadays, video surveillance systems are widely deployed in large buildings, public places, parks, roads, airports, etc.

Despite of the wide deployments, most video surveillance systems are still very under-utilized and support very limited functionalities. For example, most systems mainly support video collection, video storage or basic video analysis for event detection, etc. Moreover, existing systems often have very limited computation resource on the edge. Due to the large amount of video and high processing overhead, functionalities like trajectory tracking which require video analysis for multiple cameras are not supported. Meanwhile, uploading the video to online data center or offloading the computation to cloud is not practical considering the bandwidth and privacy issues.

Therefore, efficient video analysis using computation resource on the edge is very important considering the large number of deployments. For example, event tracking on the edge is a demanding function in practice. It can reveal the trajectory of a particular interested object by combining videos from multiple cameras. However, such a function is still not supported on most commodity video surveillance systems.

This work is also motivated by a practical demand in a real video surveillance system deployed in Tangkou Town, Huangshan City, China. Huangshan is one of the most famous mountains and a 5A (i.e. the highest grade) nature park in China. Tangkou Town is located at the south gate of the Huangshan nature park, which receives a large number of tourists (3.18 million/year) every year with a peak rate 50 thousand people per day [2]. Due to a high management pressure, a large video surveillance system with 110 cameras is deployed to facilitate the management. However, they still spend a lot of manpower watching videos to track events of their interest, e.g., track unlicensed cars that pick up passengers illegally.

To address those practical requirements, the key is to extract useful information, e.g., trajectory tracking, on video surveillance system using the computation resource on the edge. Given a rule (the plate number of car, the characteristics of a person) for a target, the target trajectory from *a number of cameras* can be revealed locally. Such a function is required in many applications, e.g., searching for the trajectory of a particular car, person, hit-and-run thieves and missing children.

We find that practical tracking on the edge faces several fundamental challenges: (1) large data size and limited resource on the edge, (2) inevitable video information missing, and (3) limited or unknown camera and physical information. First, it is time-consuming and resource-consuming to process the data. For basic target detection, practice experience shows that it takes more than 100 hours for a computer with i7-3687U CPU to process a 1-hour video. Video compression can reduce the size when the content is relative static. However, this may not work for scenarios like a road with continuous traffic. Second, there is inevitable video information missing. Solely relying visual analysis on a single camera is not sufficient. Videos may not be clear considering the high number of existing low quality cameras and physical constraints (e.g., fog weather). Even for high definition videos, object may be masked in practice. Third, the physical relationship among cameras is not clear. The precise location may not be available for existing cameras. Even the location can be obtained, different cameras may have different directions, height, etc., leading to a mismatch between camera locations and their monitoring area on the trajectory. For example, two closely located cameras may point to two completely different locations.

To address those challenges, we propose ViTrack, an efficient trajectory tracking system on the edge for commodity surveillance system. ViTrack mainly has the following components: (1) A Two-Layer spatial and temporal Compressive target Detection (TLCD) method to significantly reduce the computation overhead. Therefore, ViTrack only needs to process a very small part of video frames (< 1/10) to recover the trajectory. (2) A probabilistic based method to reconstruct the relationship among videos from different cameras while not requiring the precise location of each camera. (3) A Markov based trajectory inference method to derive event trajectory while recovering the missing information, e.g., undetected and unclear objects, etc.

We implement ViTrack on a video surveillance system with 110 cameras and evaluate it with extensive experiments. The results show that ViTrack can effectively and efficiently provide trajectory tracking. The processing time can be reduced by more than 45x compared with the existing event detection method. For a system with 110 cameras, the data can be processed on the edge with a Dell OptiPlex 390 computer.

The contributions of this work are as follows.

- We propose the design of ViTrack, a practical system for efficient tracking on the edge for commodity video surveillance systems.
- We propose a two layer spatial and temporal compressive target Detection to significantly reduce the overhead. We address practical challenges in compressive detection such as detection matrix and representation basis design.
- We propose a camera relationship construction method and derive the complete trajectory while recovering missing information.
- We implement ViTrack and evaluate its performance on a practically deployed video surveillance system with 110 cameras. The evaluation results demonstrate the effectiveness of ViTrack.

It should be emphasized that video analysis and image processing such as event/object detection is not the focus of our work. We can use any existing event/object detection algorithm. We focus more on tracking based on a video surveillance system.

II. Problem Definition

Considering a video surveillance system with M cameras, the video data of each contains N frames. We denote the *appearance matrix* ($M \times N$) of a given target as $X = \{x_{ij}, i = 1,2,\ldots,M, j = 1,2,\ldots,N\}$ as in Figure 1, where $x_{ij} = 1$ means the target appears in the jth video frame of the ith camera and otherwise $x_{ij} = 0$.

Through analyzing the practical video data, we find there are four characteristics for appearance matrix X:

C1: The size of N (e.g., tens of thousands) is usually much larger than the size of M, i.e., $N \gg M$.

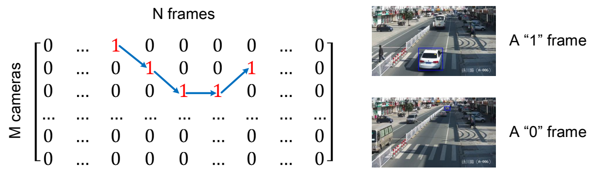


Fig. 1: Appearance information matrix X.

C2: If there is "1" element in a column, the adjacent columns contain "1" element with a high probability. This is because if a target is detected in a frame, the probability that it is detected in adjacent frames is high.

C3: Each column contains only a very limited number of "1" elements, since a target usually only appears under one or a very limited number of cameras at the same time.

C4: There are many all "0" columns and X is sparse.

A sampling policy α is given by sequence of frame index and camera index: $(I,J)^{\alpha} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_n, j_n)\}$ where $I \in \{1,2,\ldots,M\}$ and $J \in \{1,2,\ldots,N\}$. Applying α to X, we can obtain a sampling sequence $X^{\alpha} = \{x_{i_1 j_1}, x_{i_2 j_2}, \ldots, x_{i_n j_n}\}$. A recovery policy β is used to produce estimates of the original appearance matrix $\hat{X}^{\beta} = \{\hat{x}_{ij}, i = 1,2,\ldots,M, j = 1,2,\ldots,N\}$ based on the sampling sequence X^{α}, where $\hat{x}_{ij} = x_{ij}$ if $(i,j) \in (I,J)^{\alpha}$ and otherwise $\hat{x}_{ij} = \hat{x}^{\beta}_{ij}(X^{\alpha})$ for a certain estimation function $\hat{x}^{\beta}_{ij}(\cdot)$.

Our object is to find the best sampling and recovery policies to minimize the estimation error:

$$\begin{aligned} &\min \, Err(X, \hat{X}^{\beta}) \\ &s.t. \quad \frac{n}{N \times M} \le \lambda, \end{aligned}$$

where $Err(\cdot)$ is an error measurement function and λ is the requirement for sampling rate. Based on the recovered appearance matrix $\hat{X}^{\beta}$, we can then derive the trajectory of a given target as shown in Figure 1.

III. System Overview

The overall system architecture is shown in Figure 2. To reduce the computation overhead, we first design a two-layer spatial and temporal compressive target detection method as shown in Figure 3 based on the characteristics of the video surveillance system. Then we construct fine-grained camera relationship using a Markov Model. Last, we propose a trajectory inference algorithm using TLCD's results and the Markov Model. Therefore, the design of ViTrack consists of four major components.

Spatial compressive detection: First, ViTrack divides the original appearance matrix into a collection of sub-matrices as shown in Figure 3. A sub-matrix usually only contains data for all cameras in a short time period (i.e., a small number of columns, e.g., 25 in our implementation, which is referred to as a time cell). Here, we call a sub-matrix as a spatial matrix. Then according to a measurement matrix, we obtain a small samples of video frames. In each spatial matrix, we recognize given targets in sampled video frames using a pre-defined

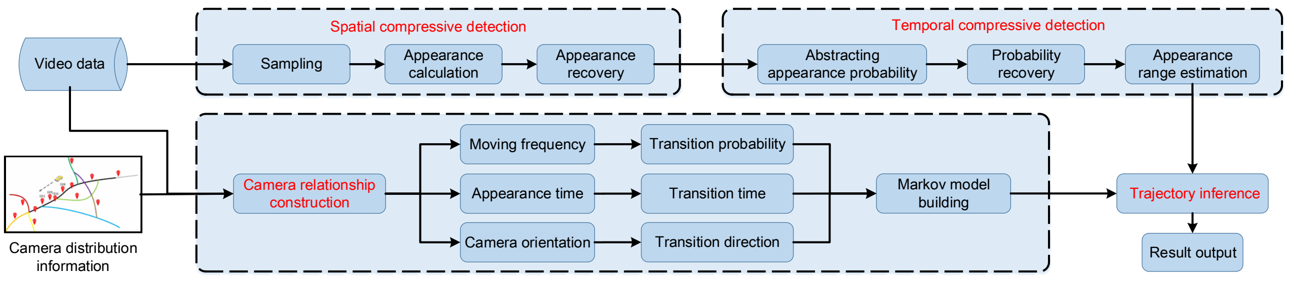
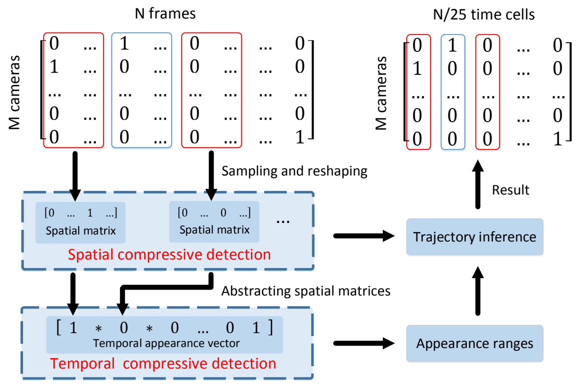


Fig. 2: System architecture



N frames
M cameras
N/25 time cells
Sampling and reshaping
Spatial matrix
Spatial compressive detection
Abstracting spatial matrices
Temporal appearance vector
Temporal compressive detection
Result
Trajectory inference
Appearance ranges

Fig. 3: Two-layer compressive target detection.

object recognition function. Last, by carefully designing the representation matrix, we recover each spatial matrix based on the recognition results of sampled frames (Section IV-A). This process is called spatial compressive detection (SCD).

Temporal compressive detection: To further reduce the overhead, not all spatial matrices are calculated and recovered. In this step, we design a measurement matrix to determine which spatial matrices should be sampled and processed in SCD. We calculate an abstracted appearance value for each sampled spatial matrix (Section IV-B1). Then we can obtain a vector, namely temporal appearance vector. Last, the temporal appearance vector can be efficiently recovered by carefully designing the measurement matrix and representation matrix (Section IV-B). This process is called temporal compressive detection (TCD).

Camera relationship construction: To recover those unsampled spatial matrices based on temporal appearance vector, we leverage the camera relationship in this step. However, the original camera distribution information may not be available. Even when the location distribution can be obtained, the direction, monitoring area, height of different cameras are different, leading to a mismatch between the cameras' locations. We build a Markov Model using the video data to construct fine-grained camera relationship (Section IV-C2). It should be emphasized that for a video surveillance system, camera relationship only needs to be constructed once.

Trajectory inference: The last component is the trajectory inference component. Trajectory inference uses TLCD's results to infer the appearance information of the unsampled spatial matrices based on camera relationship (Section IV-D). Finally, ViTrack obtains the complete trajectory of given targets and outputs the result.

IV. ViTrack Design

We first briefly introduce the principle of compressive sensing. For a sparse signal given by the vector x of size N, based on the $n\times N$ ($n \ll N$) measurement matrix Φ, we can obtain the samples $y_{n\times 1} = \Phi x$. Thus the signal x can be recovered from y if x is sufficiently sparse, subject to some pre-conditions on Φ that we will discuss below. If x is not sparse, it can often be sparsely represented in an alternative domain by a $N\times N$ representation basis Ψ. Specially, x can be written as $x = \Psi s$, where s is a sparse $N\times 1$ coefficient vector. The measurement vector can be written as:

$$y = \Phi\Psi s. \quad (1)$$

The problem is to recover s given measurement y and matrices Φ and Ψ. Then, we reconstruct the original signal using $x = \Psi s$. Equation 1 is an under-determined linear system as $n \ll N$. Finding the solution to this ill-conditioned problem has been extensively studied in recent years. A mainstream approach is to solve the l_1 norm minimization problem (also known as Basis Pursuit(BP) [3]):

$$\min_{s\in R^N} \parallel s \parallel_1 \quad s.t. \quad y = \Phi\Psi s \quad (2)$$

which can be solved using algorithms such as IRWLS [4], OMP [5] and LP (linear programming) [6].

A. Spatial compressive detection

1) Sampling: For a spatial matrix, we reshape it into an appearance vector S of size V_p ($V_p = F \times M$, F is the number of columns), by joining all the rows successively. The measurement matrix Φ_p ($U_p \times V_p$ where $U_p < V_p$) specifies a sampling policy: a "1" element in the (u,v) position ($1 \le u \le U_p, 1 \le v \le V_p$) if the u-th sampling is taken at the v-th value of S.

In order to reduce the sampled video frames, we design two kinds of measurement schedules which only contain one "1" element in any row and at most one "1" element in any column, and "0" everywhere else. The first is a periodic sampling schedule where samplings are taken every $\lfloor \frac{V_p}{U_p} \rfloor$ frames, which is referred to as the *periodic schedule* (PS). The second is a random sampling schedule generated using certain probability distribution with an average sampling rate of $\frac{U_p}{V_p}$, which is referred to as the *random schedule* (RS). Because of the characteristics C2 and C3, the "1" elements of S are usually clustered together. Therefore, PS is preferable than RS in most cases.

2) Appearance calculation: For all sampled video frames, ViTrack uses the existing object recognition methods to recognize the given target and calculate the appearance information $y = \Phi_p S$.

3) Recovering: In this stage, we select a good representation basis Ψ_p to recover S from sampling results y.

There are two major requirements for the design of Ψ_p: (1) its corresponding inverse has to sufficiently sparsify S, and (2) it must be sufficiently incoherent with Φ_p. As each column contains one "1" element at most, our measurement matrix Φ_p is sparse, which is different from the common measurement matrixes in CS literature (e.g the Gaussian measurement matrix which is very dense with virtually no 0-entries). Thus the incoherence between the measurement matrix and the representation basis poses a challenge for the design of Ψ_p. We find that the appearance information vector S is relatively smooth and slow changing in most frames. Thus S can be sparsely represented through the difference between two adjacent values. We use the following differentiation matrix to get the projection D_p of S:

$$M_p = \begin{bmatrix} -1 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 1 & \dots & 0 & 0 \\ 0 & 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & 1 \\ 0 & 0 & 0 & \dots & 0 & -\gamma \end{bmatrix}$$

where the last element γ $(0 < \gamma < 1)$ ensures that M_p is invertible. $D_p = M_p S$, so S is sparsely represented in the M_p domain as $S = M_p^{-1} D_p$ with the representation basis $\Psi_p = M_p^{-1}$. The incoherence of our measurement matrix Φ_p and representation basis Ψ_p satisfies the requirements of CS literature [7]. Therefore, as in Eq. (2), we can recover D_p according to $y = \Phi_p \Psi_p D_p$ and obtain the recovery result $\hat{S}$ of S according to $S = \Psi_p D_p$.

Figure 4 shows the recognition results and recovery results of a 1000×1 vector S, which contains the appearance information of 25 video frames for 40 cameras in a time cell. Figure 4 (a) shows the recognized appearance information of all 1000 video frames. Figure 4 (b) shows the recovery results of SCD according to the recognition results of sampled frames at a sample rate of 20%. Comparing the recognition results with the recovery results, SCD recovers most of appearance information with a small number of video frames (20%). Further, we compare the details in Figure 4 (c) and (d). We can find that SCD achieves better results when the curve is smoother. In Section V-A1, we conduct more experiments to evaluate the effectiveness of SCD.

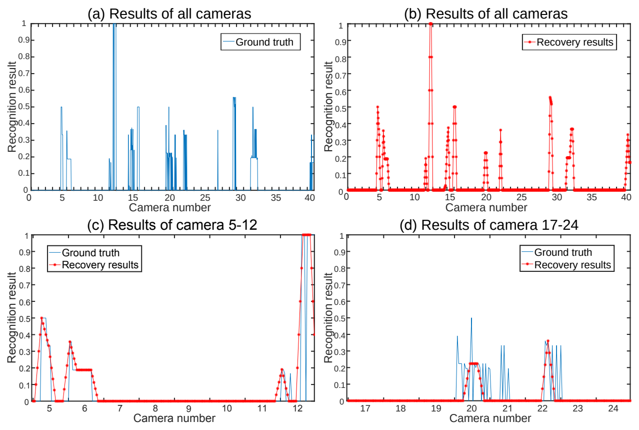


Fig. 4: Recognition results and SCD's recovery results

B. Temporal compressive detection

SCD significantly reduces the computation overhead for recovering the appearance information in a time cell. We further design temporal compressive detection (TCD) to reduce the overall computation overhead.

We take advantage of an important fact that our target tracking problem only needs to know the coarse-grained appearance information (if our target appears under any camera in a time cell) instead of the appearance information of all video frames. Therefore, we abstract each spatial appearance vector to a value, which describes the appearance probability in a time cell. Then we can obtain a temporal appearance vector T of size V_t $(V_t = N/F)$ of all time cells. By designing a measurement matrix, we only calculate the temporal appearance value in sampled time cells and then recover the overall temporal appearance vector.

1) Abstracting appearance probability: The first step is to calculate the abstracted temporal appearance value using the recovery results $\hat{S}$ of SCD. $\hat{S}$ contains $V_p = M \times F$ elements, and we denote it as $\hat{S} = \{\hat{s}_1, \hat{s}_2, \dots, \hat{s}_{(j-1)\times F+i} \dots, \hat{s}_{V_p}\} (1 \le i \le F, 1 \le j \le M)$. For F video frames of camera j in a time cell, we calculate the appearance probability a_j with the following rules:

- With more "1" values in F video frames' recognition results, a_j is higher. Specially, when the number of "1" values is more than a threshold β, we set $a_j = 1$.
- a_j is larger when the average recognition result is higher.
- We focus on recognition results' non-zero values to estimate the appearance probability. Our target may only appear in part of a time cell (e.g., it comes into the camera's monitoring area somewhere inside a time cell).

For camera j, we define its appearance probability based on the above three rules as:

$$a_k = \begin{cases} (\gamma + \frac{(1-\gamma)\times n_h}{\beta}) \times \frac{Sum(j)}{n_l}, & n_h < \beta \\ 1, & n_h \ge \beta. \end{cases} \quad (3)$$

where $Sum(j) = \sum_{i=(j-1)\times F+1}^{j\times F} \hat{s}_i$ is the sum of camera j's recovery results, n_h is the number of "1" values and n_l is the number of non-zero values. γ is a parameter between 0 and 1, which is used to judge if the target appears according to the calculated appearance probability.

Then we can choose the maximum of all cameras' appearance probability $A = \{a_1, a_2, \dots, a_M\}$ as the temporal appearance value of our target in this time cell.

2) Sampling and recovering: As in SCD, we need to choose an appropriate measurement matrix Φ_t for sampling and a good incoherent representation basis Ψ_t for recovering.

We also consider two kinds of measurement schedules (PS and RS) for Φ_t. Unlike the vector S in SCD, the appearance probability vector T is sporadic. A periodic schedule may miss appearance measurements. Therefore, we prefer a random schedule to obtain enough appearance information for recovering.

Compared with SCD, it is more challenging for TCD to design a good representation basis. The main reason is the appearance probability vector T is sporadic, which makes the former differentiation matrix M_p and recovery policy useless. Intuitively, if a target is detected in a time cell, it may disappear for a certain time period due to the monitoring area gap among cameras. A target may appear in consecutive frames for a certain time cell with a high probability, while in adjacent time cells with a low probability.

We first design an accumulation matrix M_t:

$$M_t = \begin{bmatrix} 1 & 1 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 1 & 1 & \dots & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 1 \end{bmatrix}$$

For the kth row of M_t, if $k - \lceil \frac{W}{2} \rceil \le j < k + \lceil \frac{W}{2} \rceil \cap 1 \le j \le N$, $Mt_{kj} = 1$. Otherwise, $Mt_{kj} = 0$. W is the accumulation window size. The goal of M_t is to project the discrete T into a smoother vector D_t, where $D_t = M_t T$. Accordingly, the original T can be represented in the M_t domain as $T = M_t^{-1} D_t$ and we get the representation basis $\Psi_t = M_t^{-1}$. We recover D_t according to $y = \Phi_t \Psi_t D_t$ and obtain the recovery result $\hat{T}$ of T.

D_t is sparse (the amount of non-zero elements is below 10%) which satisfies the first requirement of representation basis design. According to [7], [8], we examine the incoherence between a representation basis Ψ and a measurement matrix Φ. Projecting each row of Φ onto the space spanned by the columns of Ψ we get:

$$\zeta_j = (\Psi^T \Psi)^{-1} \Psi^T \phi_j^T \tag{4}$$

where ϕ_j is the jth row of Φ and ζ_j is the vector of coefficients corresponding to its projection on the space spanned by the columns of Ψ. A measure of the incoherence is defined as:

$$I(\Phi, \Psi) = \min_{j=1,\dots,N} [\sum_{i=1}^{N} 1\{\rho_i^j \neq 0\}] \in [1, N] \tag{5}$$

where ρ_i^j is the ith entry of vector ζ_j and $1\{A\}$ is the indicator function: it is 1 when A is true and 0 otherwise.

TABLE I: Incoherences of (Φ_p, Ψ_p) and (Φ_t, Ψ_t)

N	**I(Φ_p, Ψ_p)**	**I(Φ_t, Ψ_t)**
64	63	63
128	125	126
256	255	252
512	512	510
1024	1024	1024
2048	2048	2048

In Table I, I(Φ_p, Ψ_p) is calculated in [7] and I(Φ_t, Ψ_t) is calculated according to Equation 5. It shows the measurement matrix and representation basis of both of SCD and TCD own a high incoherence.

When W is bigger than the average transition time between adjacent cameras, most time cells in actual appearance ranges will obtain a high appearance probability through M_t's projection. So the non-zero ranges of the recovered temporal appearance vector $\hat{T}$ are similar with those of the actual temporal appearance vector T and we can use $\hat{T}$ to obtain the appearance ranges of given targets in time domain.

C. Camera relationship construction

1) Constructing camera relationship: Till now, we have obtained the target's appearance information in sampled time cells and its appearance ranges. Next, we construct the camera relationship for recovering appearance information in unsampled time cells. Figure 5 shows camera distribution diagram of Tangkou Town's video surveillance system.

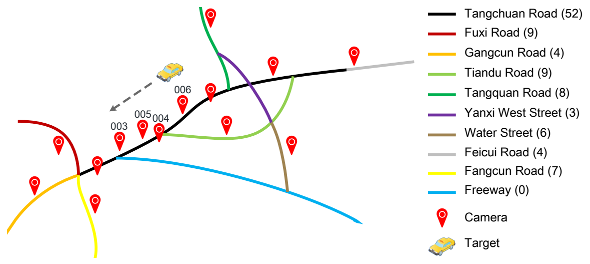


Fig. 5: Camera distribution diagram of Tangkou Town. The number behind each road represents its camera number

We propose a camera relationship construction method based on Markov Model. For each road, we select some targets from existing video data and obtain their trajectories. Using these trajectories, we focus on three kinds of camera relationship: (1) We use an $M \times M$ matrix F to store the transition frequency $F(A, B)$ between camera A and camera B. A bigger transition frequency implies a higher transition probability. (2) We use an $M \times M$ matrix E to store the average transition time between any two cameras. (3) For a transition from camera A to camera B, we use a $M \times M$ matrix G to store that whether the target is far away from camera A or not. Because an actual target keeps going straight with higher probability, this information can help ViTrack make a more precise inference. Lastly, we combine the above three kinds of information for camera relationship construction.

Algorithm 1 ForwardInference

```
Require: Known appearance time cells $I = (t_1, t_2, \dots, t_k)$, corresponding
    cameras $C = (c_1, c_2, \dots, t_k)$, and appearance range $(t_b, t_e)$
Ensure: Appearance time cells and corresponding cameras in $(t_b, t_e)$
 1: for each $t_i \in I$ & $t_i \in (t_b, t_e)$ do
 2:     repeat
 3:         if $t_i$ and $t_{i+1}$ are atomic then
 4:             break;
 5:         $Q \leftarrow$ getCandidateQueue();
 6:         for each $q \in Q$ do
 7:             $W_s \leftarrow$ setSearchingWindow();
 8:             if The target appears under camera $q$ in $W_s$ then
 9:                 addAppearanceInfo($q, t_q$);
10:                 Update $t_i$ break;
11:         end for
12:         if The target does not appear under any candidate camera then
13:             break;
14:     until $t_i > t_e$ or $t_i < t_b$
15: end for
```

2) Building Markov Model: **State definition.** We denote the k appearance time cells as $t_1, t_2, \dots, t_k$. The state $l_{t_i} = c_i$ denotes a target appears under camera c_i $(1 \le c_i \le M)$ in time cell t_i. The state $l_{t_{k+1}}$ of the target depends only on current state l_{t_k} not on states before:

$$P(l_{t_{k+1}} = c_{k+1} | l_{t_1} = c_1, l_{t_2} = c_2, \dots, l_{t_k} = c_k) = P(l_{t_{k+1}} = c_{k+1} | l_{t_k} = c_k). \quad (6)$$

Transition probability. From current time t_k to next time t_{k+1}, the transition probability from state $l_{t_{k+1}}$ to state l_{t_k} is denoted as:

$$Pr(l_{t_{k+1}} | l_{t_k}; t_k, t_{k+1}) = \frac{F(c_k, c_{k+1})}{\alpha \times \sum_{i=1}^{M} F(c_k, i)} \quad (7)$$

where $F(\cdot)$ is the transition frequency matrix, $\alpha = |c_{k+1} - c_k|$ is designed to ensure that a closer camera owns higher transition probability for the same transition frequency.

D. Trajectory inference

Lastly, ViTrack derives the complete trajectory using the above results.

1) Basic trajectory inference between two cameras: Assume a target appears under camera A in time cell t_i and under camera B in time cell t_{i+1}. The goal is to infer the real trajectory between t_i and t_{i+1}. We first define the atomic appearance of a target between t_i and t_{i+1}, if the target does not pass any other cameras. If $Pr(l_{t_{i+1}} | l_{t_i}; t_i, t_{i+1})$ is the highest transition probability for camera A and $t_{i+1} - t_i$ is approximately equal to $E(A, B)$, we consider those two appearances are atomic, denoted as $A \rightarrow B$. However, using the highest transition probability is not enough. In practice, two appearance may not be atomic as the target may be missed in some cameras. To infer the missing appearances, we not only need to obtain the cameras where the target appears, but also estimate the appearance time to verify our inference.

2) Trajectory inference algorithm: We propose a bi-directional trajectory inference algorithm to obtain the whole trajectory. For each appearance range, ViTrack performs a bi-directional inference (ForwardInference and BackInference). ForwardInference means the trajectory inference from camera A to camera B. As Algorithm 1 shows, from line 3 to 4, if the current detected appearance and the next detected appearance are atomic, the loop beaks. Otherwise, from line 5 to 11, we get the candidate queue and verify if the target appears under each candidate camera in the corresponding search window. If we find the target appears under a candidate camera, we continue the trajectory inference from this candidate camera. Otherwise, the loop breaks. Similarity, BackInference means the reverse inference from camera B to camera A. We evaluate the effectiveness of our trajectory inference algorithm in Section V-C2.

V. Evaluation

We implement ViTrack on a practical video surveillance system with 110 cameras deployed in Tangkou Town. Among those cameras, we remove some camera videos due to privacy issue. We make use of 84 cameras of video data to evaluate the accuracy and efficiency of ViTrack. ViTrack system runs on a Dell OptiPlex 390 with Intel i5 CPU and 8 GB memory. We believe such a computer resource can be easily achieved by most video surveillance systems on the edge.

We first process all the video frames to obtain the ground truth. Object recognition is not the focus of our work. We use the same object detection method and parameter settings when comparing ViTrack and the ground truth.

A. Spatial compressive detection accuracy

1) Accuracy for PS: First, we choose 40 time cells at regular intervals to conduct SCD. We use periodic schedule (PS) as Φ_s at a sampling rate of 20% which is equal to a sampling interval of 5. For each target in every time cell, we run 5 tests and evaluate SCD's accuracy in terms of detection ratio, which is defined as the total number of correctly detected targets divided by 5.

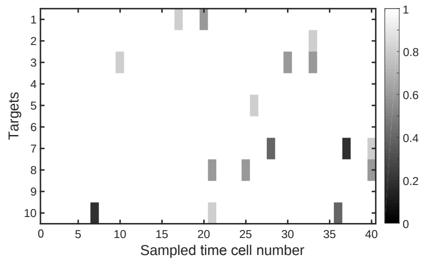


Fig. 8: Color map of SCD's detection ratio using PS.

Figure 8 shows the color map of SCD's detection ratio. The lighter areas represent higher detection rate. We can observe that SCD achieve a very high detection rate of all targets, i.e. 98.40% in average. Further, we evaluate SCD's accuracy only in time cells where the target does appear in the video surveillance system. For all 3055 tests, SCD's detection ratio is 85.63%, which proves SCD can provide efficient detection for the target's appearance.

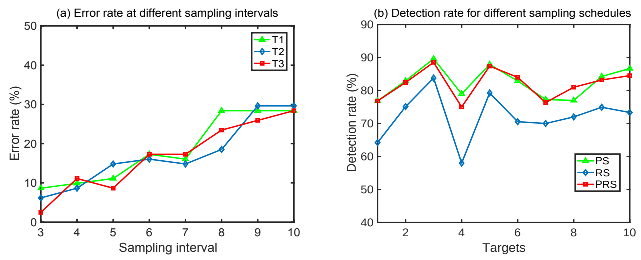


Fig. 6: Detection error ratio of SCD for different sampling intervals and sampling schedules

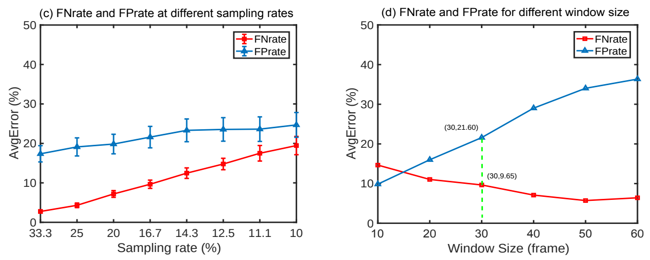


Fig. 7: FNrate and FPrate of TCD for different sampling rates and accumulation window sizes

2) Accuracy v.s. sampling rates: To determine the impact of different sampling rates on the accuracy of SCD, we perform three sets of experiments by changing the sampling interval from 3 to 10. Correspondingly, the sampling rate decreases from 33.33% to 10%. As shown in Figure 6 (a), SCD's error ratio roughly increases with the increasing of sampling intervals. The average error ratios are respectively less than 10% for interval 3 and 4, 20% for interval 5-7, and 30% for interval 8-10. Even if the sampling rate is decreased to 10%, SCD's detection ratio is still higher than 70%.

3) Accuracy v.s. different sampling schedules: Theoretically, PS is preferable than RS in most cases as mentioned in Section IV-A1. We conduct 3055 tests using three kinds of sampling schedules: PS, RS and PRS (a hybrid periodic-random schedule randomly changes the sampling position in each interval of PS). As Figure 6 (b) shows, we can see that RS has the lowest performance and the error rations of PS and PRS are close for all targets. Therefor, PS is a simple and effective schedule, which suits our SCD processing best.

B. Temporal compressive detection accuracy

1) Accuracy for RS: The goal of TCD is to obtain a target's appearance ranges. Thus we quantify TCD's accuracy in terms of true positive rates (TPrate), false positive rates (FPrate), true negative rates (TNrate) and false negative rates (FNrate). We set a threshold to determine if two appearance time cells belong to the same appearance range according to their difference. We can use the ground truth to obtain the actual appearance ranges. TPrate is defined as the total number of targets' appearance time cells in estimated appearance ranges divided by that in actual appearance ranges. Correspondingly, FNrate is equal to 1-TPrate. TNrate is defined as the total number of time cells not in estimated appearance ranges divided by that not in actual appearance ranges. Correspondingly, FPrate is equal to 1-TNrate. For each target, we perform TCD 10 times and evaluate TCD's accuracy in one hour. Because RS owns a higher incoherence with Ψ_t than PS, we use RS as Φ_t at a sampling rate 16.67%.

Figure 9 shows the average TPrate and TNrate of TCD for each target. TCD achieves a high TPrate of all targets, 90.35% in average. A small part of targets' TNrate (e.g., target 3) is low because the target frequently appears in the video surveillance system. It is harder for the target to estimate the time cells not in actual appearance ranges. However, a 78.40% TNrate in average still reduces almost 80% of unnecessary search ranges for trajectory inference.

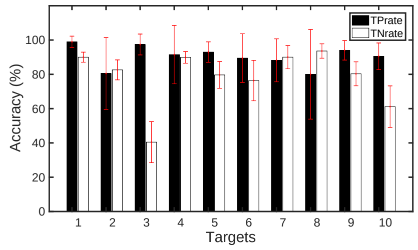


Fig. 9: TCD's TPrate and TNrate of 10 targets using RS.

2) Accuracy v.s. sampling rates: Through changing the sampling rates gradually from 33.33% to 10%, we perform the above tests to quantify the impact of different sampling rates on TCD's accuracy as Figure 7 (a) shows. The experimental results show that FNrate and FPrate gradually increase with the decreasing of sampling rates and FNrate owns a higher growth rate. When the sampling rate is more than 16.67%, FNrate is less than 10% and FPrate is less than 20%.

3) Accuracy v.s. accumulation window sizes: In theory, a higher accumulation window size W of SCD means that more time cells in actual appearance ranges own high appearance probability. We conduct TCD experiments through changing W gradually from 10 to 60 as Figure 7 (b) shows. With the increase of W, FNrate increases and FTrate decreases gradually, which coincides with our analysis. Therefore, we can adjust W to tradeoff FNrate and FTrate. The green dashed line shows an appropriate choice of W (30) for TCD, where FNrate is 9.65% and FPrate is 21.60%.

C. Effectiveness of trajectory inference

1) Constructed camera relationship: For each road, we choose 15 targets to construct camera relationship from the videos. According to Equation 7, we obtain Markov Model's transition probability matrix as Figure 10 (a) shows. The darker points represent higher transition probability. Because we number the cameras of all roads in sequence successively, we can find that most of points with high transition probability are close to the diagonal line. For each road, these points of middle cameras are closer to the diagonal line than cameras at the end. Figure 10 (b) shows the transition time matrix of any two cameras which own a non-zero transition probability. As

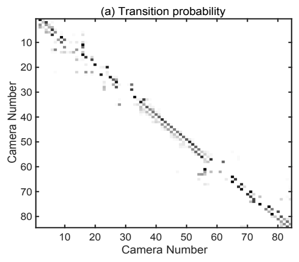


Fig. 10: Transition probability and time of Markov Model

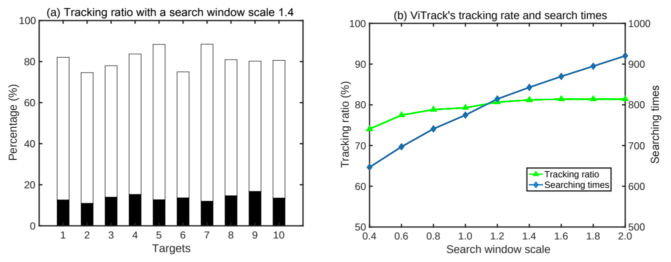


Fig. 11: Tracking ratio for different search window scales

the cameras with higher transition probability usually own less transition time, the color of these non-zero points in Figure 10 (a) and (b) compensate each other. The average transition time of any two cameras is 29.03s, which is close to the chosen accumulation window size W (30) in Section V-B3. When W is larger than the average transition time, most time cells in actual appearance ranges will obtain high appearance probability through projection of accumulation matrix. Therefore, TCD can achieve a high TPrate.

2) Effectiveness of trajectory inference: Using the results of SCD and TCD, we conduct 10 tests for each target to evaluate the effectiveness of trajectory inference in terms of tracking ratio and search times. The tracking ratio is defined as the total number of detected appearance time cells divided by that of actual appearance time cells. We define a search as detecting all video frames of a camera in a time cell. Then we calculate the number of searches of each trajectory inference. Figure 11 (a) shows the tracking ratios of all target. The black part of every target is the ratio of SCD' detected appearance time cells, 13.63% in average. This result is corresponding to TCD's sampling rate. The white part of every target is the ratio of trajectory inference's detected appearance time cells, 67.58% in average, which proves the effectiveness of trajectory inference. Averagely, ViTrack only samples one thirtieth of video frames, but achieves a tracking ratio of 81.20%.

To quantify the influence of different search window scales W_s, we repeat the above tests through changing W_s gradually from 0.4 times to 2.0 times of the corresponding transition time. Figure 11 (b) shows that when W_s is bigger than 1.4, the growth of tracking ratio becomes slow but the search times increase continually. Therefore, we choose 1.4 as an optimal search window scale, where ViTrack can achieve a tracking ratio of 81.20% with an average search times of 842.88.

D. Accuracy and efficiency of three tracking schemes

We choose three kinds of trajectory tracking schemes (4×4, 5×6, and 12×6) to evaluate ViTrack's accuracy and efficiency. A $A\times B$ scheme means that ViTrack performs SCD at a sampling rate of $1/A$ and TCD at a sampling rate of $1/B$. Accordingly, the overall sampling rate is $1/(A\times B)$.

1) Accuracy: We also use tracking ratio to evaluate ViTrack's accuracy as Figure 12 shows. 4×4 scheme achieves a high tracking ratio for all targets, 88.74% in average. Using 5×6 scheme, the tracking ratio of most of targets is more than 80% and the average is 81.20%. Using an extremely low

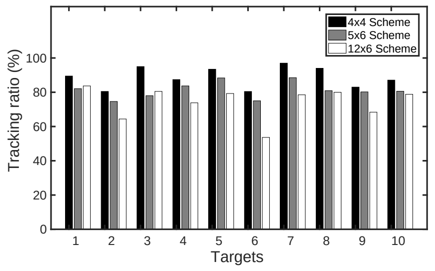


Fig. 12: Tracking ratio of different schemes for 10 targets.

TABLE II: Running time of different schemes

	COCH				COLOR			
	OR	SCD	TI	Over time	OR	SCD	TI	Over time
1×1	312.04	0	0	312.04	136.15	0	0	136.15
4×4	19.63	4.19	0.97	24.79	8.51	4.19	0.42	13.12
5×6	10.40	2.36	0.87	13.63	4.54	2.36	0.38	7.28
12×6	4.33	0.77	0.78	5.88	1.89	0.77	0.33	2.99

sampling rate of 1.39%, 12×6 scheme can still achieve an acceptable tracking ratio, 74.10% in average. Meanwhile, there are a small part of appearance time cells which are difficult to be detected. However, ViTrack can provide a part of the trajectory. Users can easily complete the rest trajectory based on ViTrack's results.

2) Overall running time: ViTrack's running time consists of three part: object recognition (OR) time, SCD time and trajectory inference (TI) time. The trajectory inference time is determined by search times. Because TCD is only performed once (about 10s) for a target tracking, we ignore its time cost. In addition, we use two kinds of object recognition methods to evaluate ViTrack: (1) CMCH which detects a target using both the color and character information, (2) COLOR which detects a target only using the color information.

Table II shows the overall running time of three schemes for 1 hour videos. 1×1 scheme is the ground truth which needs 312.04 hours using CMCH and 136.15 hours using COLOR. We have three observations. First, all three schemes significantly reduce the overall running time. For the CMCH method, ViTrack's three schemes reduce the processing time by 12.59×, 22.89× and 53.09× respectively compared with the ground true. For the COLOR method, three schemes reduce the processing time by 10.38×, 18.70× and 45.54× respectively. Second, most running time is spent in the object recognition of sampled video frames. This can be further improved by more efficient object recognition method. The time cost of trajectory inference is very small. Thus we can further increase the search window size to improve ViTrack's accuracy. Besides, SCD and object recognition are independent in every time cell, we can reduce the running time using multiple threads and computers. Using a Dell OptiPlex 390 desktop with four threads, ViTrack can adopt the 12×6 scheme for real-time trajectory tracking.

VI. Related Work

A. Compressive sensing

Compressed sensing is a signal processing technique for efficiently acquiring and reconstructing a sparse signal. Recent advances in CS theory [9], [10] allow one to represent compressible/sparse signals with significantly fewer samples than required by the Nyquist sampling theorem. This technique has been used in channel coding [11], routing [8], data collection [12], [13], soil moisture sensing [7] and targets tracking in wireless sensing networks (WSNs) [14], [15]. Our work is inspired by those successful works of CS. ViTrack designs a two-layer spatial and temporal compressive target detection method to solve the challenge of insufficient computing resources on the edge.

B. Object recognition/tracking

Object recognition/tracking has been widely studied in image/video processing. A large of object (e.g., face, plate number, and landmark) recognition methods are proposed to improve the recognition accuracy and efficiency. Take the plate number for example, [16] using a Support Vector Machine (SVM) achieves an accuracy of 90% at an average recognition time of 52.11ms per frame. To enhance the robustness of multiscaled and skewed images under different lighting conditions, [17] using Deformable Part Models (DPM) and Structural Support Vector Machine (SSVM) achieves 96.03% detection accuracy at an average recognition time of 2800ms. ViTrack uses these existing object recognition methods.

Meanwhile, there emerge a large collection of target tracking approaches for video sequences (e.g., based on Particle Swarm Optimization [18], [19] and compressive sensing [20], [21]). However, different from these works, ViTrack solves the trajectory tracking problem of given targets on a video surveillance system. Those approaches are orthogonal to ViTrack and can be combined with ViTrack. The goal of ViTrack is to provide efficiently trajectory tracking for existing surveillance systems with multiple cameras using computation resource on the edge.

VII. Conclusion

In this paper, we propose ViTrack, an efficient trajectory tracking framework on the edge for commodity video surveillance systems. ViTrack designs a two-layer spatial and temporal compressive target detection to reduce the computation overhead and then leverages a Markov Model to derive the complete trajectory. We implement ViTrack on a real deployed video surveillance system with 110 cameras. The results demonstrate ViTrack can provide efficient trajectory tracking with processing time $45\times$ less than the existing approach. In future, we will apply and evaluate ViTrack to more video surveillance systems.

References

[1] "Video Surveillance Market Report," http://www.marketsandmarkets.com/PressReleases/global-video-surveillance-market.asp, 2016.

[2] "The tourist data of Mount Huang," http://www.tourmart.cn/news/china/2016-04-19/1149.html, 2016.

[3] D. L. Donoho, "For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution." Technology Report, 2004.

[4] I. F. Gorodnitsky and B. D. Rao, "Sparse signal reconstruction from limited data using FOCUSS: a re-weighted minimum norm algorithm," *IEEE Trans. Signal Processing*, vol. 45, no. 3, pp. 600–616, 1997.

[5] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[6] M. Pilanci, L. E. Ghaoui, and V. Chandrasekaran, "Recovery of sparse probability measures via convex programming," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2420–2428.

[7] X. Wu and M. Liu, "In-situ soil moisture sensing: measurement scheduling and estimation using compressive sensing," in *IPSN 2012, Beijing, China, April 16-19, 2012*, 2012, pp. 1–12.

[8] G. Quer, R. Masiero, D. Munaretto, M. Rossi, J. Widmer, and M. Zorzi, "On the interplay between routing and signal representation for compressive sensing in wireless sensor networks," in *Information Theory and Applications Workshop, 2009*. IEEE, 2009, pp. 206–215.

[9] D. L. Donoho, "Compressed sensing," *IEEE Trans. Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[10] E. J. Candès and T. Tao, "Near-optimal signal recovery from random projections: Universal encoding strategies?" *IEEE Trans. Information Theory*, vol. 52, no. 12, pp. 5406–5425, 2006.

[11] ——, "Decoding by linear programming," *IEEE Trans. Information Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[12] C. Luo, F. Wu, J. Sun, and C. W. Chen, "Compressive data gathering for large-scale wireless sensor networks," in *MOBICOM 2009, Beijing, China, September 20-25, 2009*, 2009, pp. 145–156.

[13] Z. T. Li, J. X. Xie, D. B. Tu, and Y. J. Choi, "Sparse signal recovery by stepwise subspace pursuit in compressed sensing," *International Journal of Distributed Sensor Networks,2013,(2013-8-4)*, vol. 2013, no. 1, pp. 945–948, 2013.

[14] Y. Zheng, N. Cao, T. Wimalajeewa, and P. K. Varshney, "Compressive sensing based probabilistic sensor management for target tracking in wireless sensor networks," *IEEE Trans. Signal Processing*, vol. 63, no. 22, pp. 6049–6060, 2015.

[15] J. Luo, Z. He, Y. Liu, J. Zha, and K. Li, "Energy confirmable overlapping target tracking based on compressive sensing in wireless sensor networks," *Ad Hoc & Sensor Wireless Networks*, vol. 32, no. 1-2, pp. 131–148, 2016.

[16] C. Arth, F. Limberger, and H. Bischof, "Real-time license plate recognition on an embedded dsp-platform," in *CVPR 2007, 18-23 June 2007, Minneapolis, Minnesota, USA*, 2007.

[17] Z. A. Shaikh, U. A. Khan, M. A. Rajput, and A. W. Memon, "Machine learning based number plate detection and recognition," in *ICPRAM 2016, Rome, Italy, February 24-26, 2016.*, 2016, pp. 327–333.

[18] C. Mollaret, F. Lerasle, I. Ferrané, and J. Pinquier, "A particle swarm optimization inspired tracker applied to visual tracking," in *ICIP 2014, Paris, France, October 27-30, 2014*, 2014, pp. 426–430.

[19] F. Sha, C. Bae, G. Liu, X. Zhao, Y. Y. Chung, and W. Yeh, "A categorized particle swarm optimization for object tracking," in *CEC 2015, Sendai, Japan, May 25-28, 2015*, 2015, pp. 2737–2744.

[20] H. Li, C. Shen, and Q. Shi, "Real-time visual tracking using compressive sensing," in *CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, 2011, pp. 1305–1312.

[21] Y. Wu, N. Jia, and J. Sun, "Real-time multi-scale tracking based on compressive sensing," *The Visual Computer*, vol. 31, no. 4, pp. 471–484, 2015.