TIPAngle: Traffic Tracking at City Scale by Pose Estimation of Pan and Tilt Intersection Cameras

Shreehari Jagadeesha
SCAI, Arizona State University

Edward Andert
SCAI, Arizona State University

Aviral Shrivastava
SCAI, Arizona State University

***Abstract*—Modern cities have hundreds to thousands of traffic cameras distributed across them, many of them with the capability to pan and tilt, but very often these pan and tilt cameras either do not have angle sensors or do not provide camera orientation feedback. This makes it difficult to robustly track traffic using these cameras. Several methods to automatically detect the camera pose have been proposed in literature, with the most popular and robust being deep learning-based approaches. However, they are compute intensive, require large amounts of training data, and generally cannot be run on embedded devices. In this paper, we propose TIPAngle – a Siamese neural network, lightweight training, and a highly optimized inference mechanism and toolset to estimate camera pose and thereby improve traffic tracking even when operators change the pose of the traffic cameras. TIPAngle is 18.45x times faster and 3x more accurate in determining the angle of a camera frame than a ResNet-18 based approach. We deploy TIPAngle to a Raspberry Pi CPU and show that processing an image takes an average of .057s, equating to a frequency of about 17Hz on an embedded device.**

***Index Terms*—Camera Pose Estimation, Traffic Tracking, Pan Tilt Traffic Camera, Siamese Neural Network**

I. Introduction

Major metropolitan districts around the world use significant amounts of Pan Tilt Traffic Cameras (PTTCs) to control and modulate traffic and locate emergencies. PTTC traffic cameras allow city officials to monitor traffic incidents by aiming a camera rather than sending someone to the incident. This remote incident monitoring allows the evaluation of a fast and proper response from traffic incidents ranging from traffic, to road debris, and accidents. The many PTTCs distributed across the city provide a great opportunity for computer vision and machine learning techniques to automatically track vehicles, detect traffic incidents and efficiently manage traffic on a city-wide scale [1], [2], [3]. To do any of these, an important step is to track the vehicles in the camera field of view (FOV). This is usually done by transforming the positions of objects from the camera frame to a global frame. However, this requires precise knowledge of the camera pose, which is easy for fixed cameras, but much more difficult for PTTC cameras that can move throughout the day without notice. Nearly all PTTCs do not have orientation sensors, and even the few that have the sensors do not provide this information as image metadata that can be easily consumed by algorithms. The addition of an external IMU is not adequate solution as they are not precise enough for angle measurements at long range. The problem is further complicated by the fact that covering intersections adequately requires cameras in different poses, and traffic camera operators consistently change the camera angles for a better view or even for temporary observation of say, construction on the side of the street. As a result, an accurate mechanism to detect the camera pose in real-time on an embedded systems is needed.

There have been many methods proposed for determining the pose of a camera, ranging from SLAM to deep learning based methods. Although SLAM-based approaches can be effective [4], they suffer from accumulation of pose drift and require high frame rates and no skips in the camera feed as is moves, which cannot be guaranteed. Another popular method of tracking the pose of PTTC traffic cameras is vanishing point estimation techniques, which use vehicle movements or road lines to calibrate the camera using a vanishing point [5], [6]. These methods are great for calibration of PTTC cameras without context, but they are not very efficient in tracking pose when the roads and lane lines are not straight and need a calibration point in order to know the reference angle to transpose into a global map [6]. They also do not work well for our application of tracking vehicles on surface streets where there are typically many intersecting roads that may not be straight. Recent approaches use deep learning to estimate the pose of a camera relative to other objects [7], [8], [9] – and these are the most closely related work to ours, but as we show, they require a lot of training data and are compute-hungry and thus cannot run on an embedded system.

In this paper, we propose **TIPAngle** – an approach based on Siamese neural networks (SNN) [10], [11] to automatically detect the pose of the traffic cameras in real time. **TIPAngle** is accurate even with very little training, and is computationally efficient during inference, which enables it to run in real-time on embedded platforms.

The primary contributions of this paper are:

- A novel Siamese Neural Network (SNN)-based approach to determine the pose of off-the-shelf pan tilt traffic cameras (PTTC) that lack an encoder or IMU. The approach is accurate even with little training data.
- A grid-based training data structure combined with a gradient descent-based optimization that minimizes inference steps so that the inference can be run in real time onboard embedded processors.

To demonstrate the effectiveness of **TIPAngle**, we collect pictures with different pan and tilt angles from a vantage point similar to traffic cameras and divide them into training, validation, and testing sets. We train **TIPAngle** and a state-of-the-art Resnet-18 based approach seen in [12] on the test data set. Our results show that **TIPAngle** is 18.45x times faster at determining the angle of a camera frame than [12]. **TIPAngle** is also 3x more accurate than [12] when measuring the RMSE of the predicted angle vs. the measured angle. We deploy **TIPAngle** to a Raspberry Pi and show that processing a single frame can be done in $.057s$ equating to a frequency of about 17Hz. Finally, we show that **TIPAngle** coupled with a YoloX-based traffic tracking pipeline and show the effectiveness of the tracking transposed onto a global coordinate frame.

II. Related Work

There are a multitude of methods to track the orientation of PTTC cameras using estimation of vanishing point(s) by using the lane lines or road marking [13]. These vanishing point methods can get an accurate results in many cases. however, in the case of high traffic arterial roadways, most approaches fail due to lack of visibility of lane lines. To solve this, Dubska *et. al* [6] propose a method which uses the vehicles traveling in the camera FOV as indicators of the lanes and calculates the vanishing point using the result. This approach doesn't work well when roads aren't straight and especially not in the case of an intersection on surface streets and it doesn't converge on a solution fast enough. Other vanishing point based vehicle tracking techniques improve vehicle tracking but do not return pan tilt angles to transpose the data into a global frame [14].

Simultaneous Localization and Mapping (SLAM) can also be used for camera pose estimation. However, a main challenge with SLAM is the drift over time especially as vehicles move in and out of the camera frames. Zang *et. al* develop a SLAM method for traffic cameras that eliminates drift, however it relies on a 3D map of the city [15]. Lu *et. al* showcase a method for using SLAM to predict pose of a camera at sporting events by subtracting out players; however, as the authors state, the system performance suffers due to obstructions and the latency numbers are not published, despite being run on a very powerful machine [16]. Del *et. al* showcase a method that can recover for PTTC cameras but it can run at only 17FPS on a 240p camera data on a very powerful machine. SLAM approaches are not performant enough to scale to hundreds of cameras and, furthermore, are not robust to frame rate skips while the camera is moving [4]. While it is clear that SLAM is effective when video frames are sequential and frame-rate is high, in a PTTC environment where frame-rate can be very choppy due to transport delay, we need a pose estimator that doesn't suffer from long recovery times.

Deep learning approaches like ResNet and PoseNet have also been widely used to solve camera pose estimation [17]. The main drawbacks of using deep neural network (DNN) architectures is that they require extensive labeled data and are compute-heavy [18]. High-performance GPUs are typically necessary to train these models and are also often needed to run them in real time, which does not scale well to hundreds of PTTCs [18].

Siamese Neural Networks (SNNs) have been used for relative camera pose estimation with results better than DNNs [10]. Li *et. al* proposed a method to use SNN to estimate the pose of various object using open source datasets [11]. Yu *et. al* used an SNN to estimate the pose of various VGA connectors so that a robot arm can connect them (visual servoing) [19]. Although none of these applications use the SNN to detect the pose of the camera itself, they show the prowess of SNNs for detecting pose more accurately than competing methods such as PoseNet and RelNet. In this paper we adopt an SNN architecture to accurately and efficiently estimate PTTC pose.

III. Our Approach – **TIPAngle**

A. Network Architecture

TIPAngle uses a Siamese Neural Network (SNN) to detect camera pose, specifically the pan and tilt angle. SNNs are a type of network architecture that consists of two identical relatively small subnetworks, usually a Convolutional Neural Net (CNN), Artificial Neural Networks (ANN), or Recurrent Neural Networks (RNN) configured in the same way. Parameter updates are mirrored across both subnetworks and they seek to maximize the similarity between the output feature vectors of the two branches. This framework has been used successfully in weakly supervised metric learning [20] and object pose estimation [10], [11]. A characteristic of SNNs is their ability to be trained on very little data, which is very important for our traffic camera application.

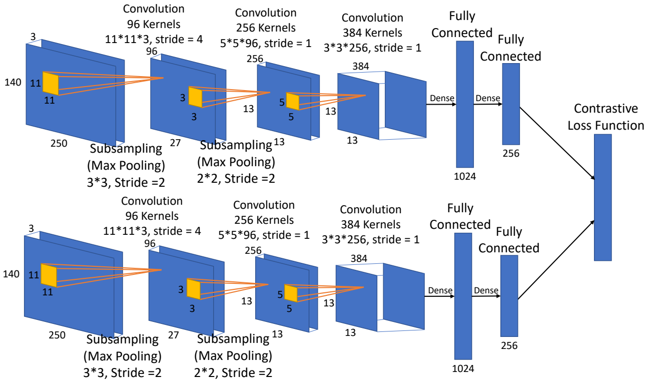

Fig. 1. **TIPAngle** uses a Siamese Neural Network architecture with a contrastive loss function. The goal of training is to learn embeddings that are close-by for images that have similar pose.

Figure 1 shows our SNN architecture. It has two arms, each having 3 convolutional layers and 2 max pooling layers and 2 linear layers. Finally, we generate a 256-value vector, which is the embedding of the input image to the arm. Each image is flattened into a 1D vector by concatenating pixel values to derive a vector of pertinent features for the calculation of Euclidean distance. The length of these vectors corresponds to the total number of dimensions, which is equivalent to the number of RGB pixels within the image. We use a CNN architecture inspired by [21]. Table I lists the filter sizes for convolution and pooling layers as $H \times W \times D$, where H is the height, W is the width and D is the depth of the corresponding filter. Stride denotes the distance between the application of filters for convolution and pooling operation. ReLU are used as the activation function to the output of all convolutional and fully connected layers throughout the network. There is no padding for any layer, and the Local Response Normalization is used to generalize learned features.

TABLE I
SIAMESE NEURAL NETWORK LAYER DETAILS.

#	Layer Type	Kernel	Stride	Input Channels	Output Channels
1	Convolutional	11x11	4	1	96
2	Max Pooling	3x3	2	-	-
3	Convolutional	5x5	1	96	256
4	Max Pooling	2x2	2	-	-
5	Convolutional	3x3	1	256	384
6	Linear	-	-	384	1024
7	Linear	-	-	1024	256

B. Loss Function

The metric is similarity and not probability. This is we chose the contrastive loss function in our application of SNN, shown in equation 1, where D_w is the Euclidean distance between the embeddings generated from the sister networks. Y is the similarity label, which is 0 when both images have the same orientation, and 1 if different.

$$\mathcal{L} = (1 - Y)\frac{1}{2}\left(D_W\right)^2 + (Y)\frac{1}{2}\left\{\max\left(0, m - D_W\right)\right\}^2 \quad (1)$$

If the input image samples are similar ($Y = 0$), then we minimize the term (D_W^2) that corresponds to their Euclidean distance, and when the input images are dissimilar ($Y = 1$), then we minimize the term $max(0, m - D_W)^2$ that is equivalent to maximizing their euclidean distance until some limit margin m. If m is set too small, dissimilar pairs may not be pushed far enough apart, leading to an overlap between similar and dissimilar pairs in the embedding space, making the network unable to distinguish images effectively. In contrast, if m is set too large, the network may try to push dissimilar pairs farther than necessary, leading to wasted effort and slower convergence. In practice, m is set between 0.5 and 2.0. In our case we have set m to 2.0 for effective convergence as a smaller m started resulting in overlapping of distinguished images as similar ones.

C. Training

The goal of training **TIPAngle** training is to make the embeddings of similar image pairs (images that have the same pan and tilt angles), and the embeddings of dissimilar image pairs (images that do not have the same pan and tilt angles) [22]. To do this, we pick two random images from our training dataset and input them to the two arms of the SNN, and compute the contrastive loss function (1) based on the label Y that we provide (0 when both the images have the same orientation, and 1 if different). This is a differentiable loss function so that gradient descent and network weight optimization can take place. Back propagation is used to update the weights of different layers. Table II shows all the relevant training parameters.

TABLE II
TRAINING PARAMETERS

Parameters	Values
Activation Function	ReLU
Learning Rate	0.0005
Optimizer	Adam Optimizer
Loss Function	Contrastive Loss
Loss Margin	2.0
Batch Size	64
Epoch Count	100

The learning rate is chosen empirically for cautious and yet efficient convergence. The Adam optimizer recognized for its adaptive learning rate mechanism is chosen, since the Adam optimizer dynamically customizes learning rates for individual parameters. This dynamicity inherently expedites convergence, which is particularly important when navigating gradients that exhibit variance across the network. Furthermore, the operational parameters, batch size and epoch count) are set at 64 and 100 respectively.

D. Architecture Exploration

We initially started with a minimalist SNN architecture with just two convolutional layers and one Max Pooling Layer for camera pose estimation. The focus was on prioritizing simplicity and ease of computation while maintaining an optimal level of performance. Although the model exhibited computational efficiency, it became evident that its capacity to capture nuances within the data was restricted. To address these limitations, the architecture was then expanded to the configuration presented in Table I, Table II and figure1. This enhanced version integrated additional max-pooling and convolutional layers, increasing its depth and potential to discern more complex features.

Convolutional Layer 1, which features an 11x11 kernel size and a stride of 4, the architecture captures large, foundational features such as road and building structures. The subsequent integration of Max Pooling as layer 2, with a 3x3 pooling size and a stride of 2, increases the network's ability in extracting such large features. Layer 3 as a Convolutional Layer, with a 5x5 kernel size and stride of 1, further refines feature extraction, particularly those of intermediate sizes, like traffic lights and sidewalks. Layer 5, operating with a 2x2 Max Pooling and a stride of 2, is added further for dimensionality reduction. Using an odd-sized filter in Convolutional Layers ensures that all the preceding layer pixels are symmetrically positioned around the output pixel. This symmetry is crucial because without it, we would need to address distortions that can occur when using an even-sized kernel. The usage of Rectified Linear Unit (ReLU) activation function introduces non-linearity while mitigating vanishing gradient issues. By setting negative values to zero, ReLU facilitates efficient gradient flow during back propagation.

E. Fast Inference

The inference problem in **TIPAngle** is that, given an image, determine its pan and tilt angles. This is typically done in

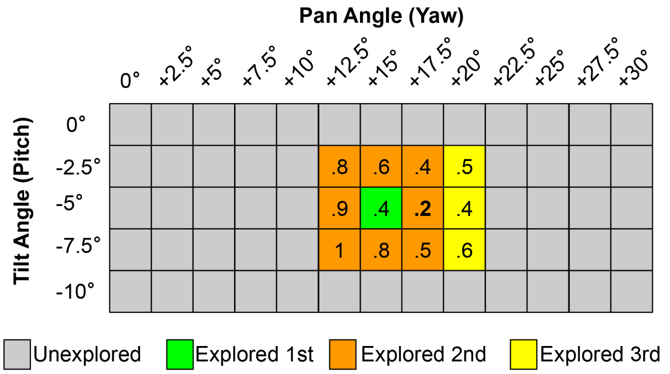

Fig. 2. Gradient descent shown in Algorithm 1 is run on a camera frame that is 2.5 degrees from the original. In this case, the gradient descent algorithm explores 12 locations, running inferences in 3 rounds until it terminates fining the lowest value, .2.

SNNs by first generating the embedding of the image using the pre-trained SNN, compare it with the embeddings of all the existing images, and return the pan and tilt angle of the image by extrapolating from the four embedding closest to the input image. However, that is quite computationally intensive to do for large image sets. Instead, we note that most of the time PTTC cameras remain stationary. We use this insight to our advantage, by creating a modification to the inference step that strategically searches our organized image training set starting with the last known angle and moving outward in a gradient descent approach, shown in algorithm 1. This algorithm works on the principle that the fastest inference speedup is not to run inference at all when it is not necessary. In the case that the PTTC camera is at the same pose as the last frame, we will only run inference on nine images instead of N images. In the case that the PTTC camera has actually moved, the search space expands outwards until we find the global minima, the less movement of the PTTC the faster this is. Figure 2 shows a situation where the subsequent image in a sequence is 2.5 degrees away from the previous one. In this scenario, we first explore the closest position to the last angle and the surrounding 8 angles. If all the 8 explored around the last angle show at least a 2x weight gain from the last closest position, the algorithm immediately exits. However, in Figure 2, the location +17.5,-5 degrees showed a lower value, and thus we go on to explore the next 3 angles in that direction. Finally after exploring those, we find a global minima that is 2x smaller than all that surround it. Now we interpolate the nearest weights and come up with the actual angle estimate, and set the lowedt field as the first to be explored on the next camera frame. Even if we explore the entire search space, the worst-case result is the same as exploring all the images, which is what the brute force method was doing without this optimization. Algorithm 1 results on average in a 6.24x performance improvement over the SNN doing inference on all available frames for a 13x5 grid.

F. Traffic Tracking Using Camera Pose

We integrate the camera pose angle estimation with an object recognizer to track the traffic, outlined in algorithm 2. Using a pinhole camera model, we estimate the distance to each class of vehicles (car, truck, bus, etc.) based on the expected average height of the class, focal length of the camera, and pixels of height in the video frame (line 1). The yaw and pitch angle to the object is estimated based on the camera lens and pixel position of the bounding box center (line 1). With a distance and angle, we can move this to $< x, y, z >$ coordinates *w.r.t.* the camera pose (line 1). This detection result is then transformed based on the known GPS of the camera as well as the pose returned by **TIPAngle** (line 2-4). This results in a tracked object set with coordinate with respect to the global map frame. We then feed these coordinates into a multi-object tracker that maintains IDs, positions, and velocities using an EKF for the objects with respect to the global frame, seen in line 5 and 6.

Algorithm 1: Gradient Descent Algorithm

```
Input  : lastPose, imageGrid
Output: pose
smallestDistance ← ∞ localMinima ← None
  unexploredPositions[]
  unexploredPositions.append((gridPosition(lastPose),
  inferDistance(lastPose)))
while True do
    unexploredPositions.Sort(using [1])
    testGridPosition, testDistance =
      unexploredPositions[].pop()
    if testDistance < smallestDistance then
        localMinima ← testGridPosition
    if testDistance < (2*smallestDistance) then
        for neighbors of testGridPosition do
            unexploredPositions.append(gridPos(neighbor)
              , inferDistance(neighbor))
    if length(unexploredPositions) == 0 then
        return LocalMinima
```

Algorithm 2: Vehicle Tracking in Global Map Frame

```
Input  : image, YoloXRecognizer, TIPAngle,
         cameraGPSPosition, EKFTracker
Output: tracksInGlobalFrame
1 detections = YoloXRecognizer.recognize(image);
2 cameraToWorldTf.position = cameraGPSPosition;
3 cameraToWorldTf.rotation =
    TIPAngle.getPose(image);
4 detectionsInGF =
    detections.transformEuler(cameraToWorldTf);
5 EKFTracker.predict();
6 tracksInGlobalFrame =
    EKFTracker.update(detectionsInGF);
```

IV. Experimental Setup

We collected our dataset using a Canon 1300D camera on a tripod that was zeroed out in the roll direction. Tilt angle

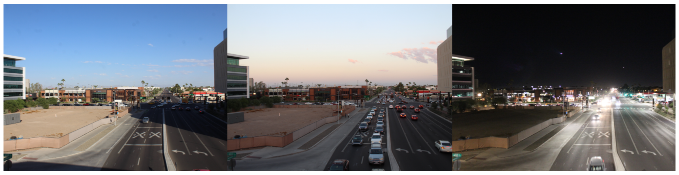

Fig. 3. Images Taken at Same Angle I.E., 0° Pan and -3° Tilt Angle During Day, Dusk and Night Scenarios

measurements were obtained using an IMU as an inclinometer while pan angle measurements were obtained using a tripod base angle measuring attachment, zeroed out to magnetic north. These images were collected from a vantage point on a bridge that mimics the height PTTC cameras are often positioned. The pan and tilt angles were varied with a step size of 2.5° for the dataset. The pan angle was taken in the range of 30° while the tilt was taken with a range of 10°. Images were collected systematically in 3 sets of lighting conditions, day, dusk and night scenario with each set consisting of 65 images. Figure 3 shows three images collected at the same pan and tilt angle during day, dusk and night. An additional 90 separate images are collected at randomly decided angles within the pan and tilt range for validation and as a ground truth-ed test set. Finally, for test purposes, multiple five minute long videos are taken where the camera is moved to a different pan and tilt angle four times within the video for a total of five contained angles. Pan/tilt angles and transition times in the video are recorded for ground truth.

Using this collected dataset, we set train both our SNN model and a lightweight state of the art model, ResNet. We then use the 90 images taken at random angles to provide an unbiased evaluation of a model fit on the training dataset on the predicted pan and tilt angles versus the measured. The training loss and the validation loss are about the same, which demonstrates the robustness and generality of the training.

Finally we use the recorded videos for testing the performance as these 12,000 frame long videos mimic the operation of a pan tilt camera that is moving often and are used to test our gradient descent algorithm in a realistic scenario.

V. Results

A. *TIPAngle Can Be Trained Quickly*

We trained the model for until convergence which occured at 100 epochs, using training rate of 0.0005, and minimum batch size of 64 using Tensorflow and Keras libraries. On a Core i5 10210U CPU, as shown in table III, the training of the SNN took 6 minutes and 48 seconds while ResNet-18 on the other hand took 37 minutes and 25 seconds to train using the same data. This shows the simplicity of the SNN approach compared to ResNet-18. ResNet-34 and 50 took significantly longer and are therefore not shown.

TABLE III
Training Time

Method	of Layers	Training Time (Core i5)
ResNet-18	18	2246s (37m, 25s)
VGG-16	16	408s (6m, 48s)

B. ***TIPAngle*** *Can Be Trained With Less Data*

In order to demonstrate the ability of the proposed system to accurately detect the pose of the PTTC, the Root Mean Squared Error (RMSE) values comparing the predicted and actual values of output angles of **TIPAngle** are shown compared to ResNet18. It can seen from the figure 4 that the RMSE is the lowest during the day. As the lighting decreases the RMSE value increases slightly. However, the proposed solution still performs well. Compared with ResNet18 it can clearly be seen that **TIPAngle** shows better results and is more than 3x as accurate using the same training data. Even the most accurate ResNet50 model is slightly more accurate, but runs far too slowly for our application.

C. ***TIPAngle*** *Inference is Fast*

Comparing execution times, we can see that the SNN approach is 4.48x faster than ResNet-18 in table IV. However, once we add in our gradient descent approach to make **TIPAngle**, the performance improvement over ResNet-18 increases to a whopping 28.56x improvement. This is because the vast majority of camera poses being unchanged when dealing with PTTC cameras, and thus most of the time we only need to search the immediate surrounding angles to confirm that the angle has not changed.

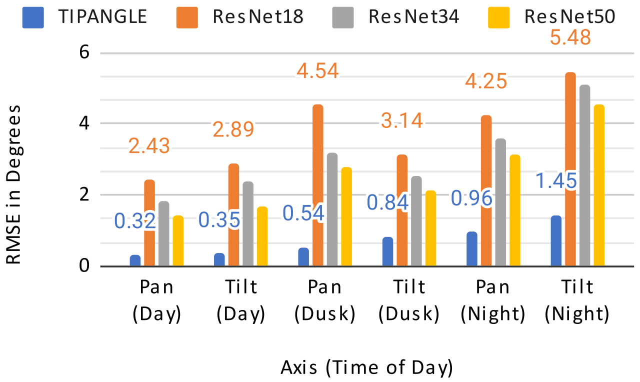

Fig. 4. Comparison of **TIPAngle** and Resnet18 RMSE vs. ground truth labels for test dataset consisting of 90 images, 30 captured at each of the three light level at randomized angles.

D. ***TIPAngle*** *Runs On Embedded Hardware in Real-Time*

Convolutional Neural Network models can be resource intensive, requiring large amounts of memory and computational power to train and use for inference. This can make it difficult to deploy these models on resource-constrained devices such as Raspberry Pi with limited computational power(Quad Core 1.2GHz Broadcom BCM2837 64bit CPU) and RAM (1.0 Gb). To overcome this limitation, we use separable convolution, pruning, and our gradient descent to reduce these model's resource requirements. This results in a .057s runtime on the Raspberry Pi which equates to 17Hz. Resnet-17 was significantly slower and even coupled with the improvement from algorithm 11 and did not break the 1 second barrier.

TABLE IV
PERFORMANCE COMPARISON

	Core i5	Raspberry Pi 3
ResNet-18	3.713s	19.2s
SNN	0.29s	0.36s
Pangle	0.031s	0.057s

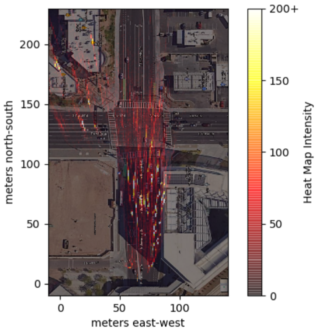

Fig. 5. Without **TIPAngle** there are 5 distinct patterns of the heat map due to the fact the camera moves 4 times during the test and it was not compensated for.

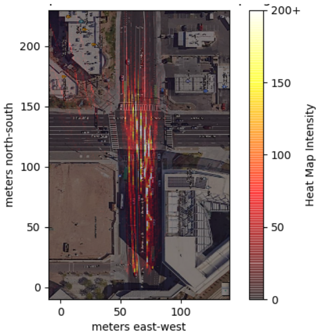

Fig. 6. With **TIPAngle** we can see that the heat map lines up with the expected center of the lanes with a much higher heat signature, despite the camera moving four times.

*E. **TIPAngle** achieves excellent traffic tracking*

In order to showcase the effectiveness of **TIPAngle**, we recorded a five minute videos with changes occurring at roughly 1 minute intervals where we change the pan and tile to a new unique angle. We use a YoloV4 vehicle tracker and transform the detections as a heat map onto a satellite image of the roadway the camera is roughly faced towards from the pose of the camera using two methods. Method one is using a fixed pose gathered from the initial angle the camera is set to, shown in figure 5. And method two is using **TIPAngle**, shown in figure 6. Each 1m by 1m square heat position represents the number of times a vehicle is detected within that position for the five minute long test. We can see that without **TIPAngle** we get five distinct sets of lines from where the camera was stable at a different pan angle. With **TIPAngle** we can see that the track heat map lines up well with the lanes and shows very little deviation despite the camera moving four times.

VI. CONCLUSION

In this paper, we present a lightweight, easy-to-train pan-tilt traffic camera pose estimation method based on a Siamese Neural Network architecture, which we call **TIPAngle**. **TIPAngle** can run inferences at a high frame rate for PTTC cameras distributed throughout a city, even those lacking precise IMUs or angle encoders. Using a sparse dataset and a ground truth method, we train both **TIPAngle** and a state-of-the-art method, ResNet18. Our tests consistently show that **TIPAngle** outperforms ResNet18. Specifically, **TIPAngle** is 18.45 times faster in determining the angle of a camera frame, while being three times more accurate than ResNet18 in measuring the RMSE of the predicted versus the measured angle. We deployed **TIPAngle** on a Raspberry Pi 3 and demonstrated that processing a single frame takes only 0.057 seconds, equivalent to a frequency of approximately 17Hz. In future work, we plan to analyze **TIPAngle**'s performance across multiple traffic cameras simultaneously and explore its deployment across a large set of PTTC cameras.

REFERENCES

[1] B. K. M *et al.*, "Road accident detection using machine learning," in *2021 International Conference on System, Computation, Automation and Networking (ICSCAN)*, 2021, pp. 1–5.

[2] M.-N. Chapel and T. Bouwmans, "Moving objects detection with a moving camera: A comprehensive review," *Computer science review*, vol. 38, p. 100310, 2020.

[3] S. M. Sunny *et al.*, "Image based automatic traffic surveillance system through number-plate identification and accident detection," in *2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, 2021, pp. 467–472.

[4] A. Del Bimbo, G. Lisanti, I. Masi, and F. Pernici, "Continuous recovery for real time pan tilt zoom localization and mapping," in *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2011, pp. 160–165.

[5] Y. Zheng and S. Peng, "A practical roadside camera calibration method based on least squares optimization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 831–843, 2013.

[6] M. Dubská *et al.*, "Fully automatic roadside camera calibration for traffic surveillance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 3, pp. 1162–1171, 2014.

[7] Y. Nakajima and H. Saito, "Robust camera pose estimation by viewpoint classification using deep learning," *Computational Visual Media*, vol. 3, pp. 189–198, 2017.

[8] Y. Shavit and R. Ferens, "Introduction to camera pose estimation with deep learning," *CoRR*, vol. abs/1907.05272, 2019. [Online]. Available: http://arxiv.org/abs/1907.05272

[9] M. M. Albakri *et al.*, "Traffic surveillance: Vehicle detection and pose estimation based on deep learning," *Przeglad Elektrotechniczny*, vol. 02/2023 Page no. 131, p. 4, 01 2023.

[10] I. Melekhov *et al.*, "Relative camera pose estimation using convolutional neural networks," in *Advanced Concepts for Intelligent Vision Systems: 18th International Conference, ACIVS 2017, Antwerp, Belgium, September 18-21, 2017, Proceedings 18*. Springer, 2017, pp. 675–687.

[11] Q. Li *et al.*, "Relative geometry-aware siamese neural network for 6dof camera relocalization," *Neurocomputing*, vol. 426, pp. 134–146, 2021.

[12] M. Xu *et al.*, "A critical analysis of image-based camera pose estimation techniques," *arXiv preprint arXiv:2201.05816*, 2022.

[13] K.-T. Song and J.-C. Tai, "Dynamic calibration of pan–tilt–zoom cameras for traffic monitoring," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 5, pp. 1091–1103, 2006.

[14] J. Sochor *et al.*, "Traffic surveillance camera calibration by 3d model bounding box alignment for accurate vehicle speed measurement," *Computer Vision and Image Understanding*, vol. 161, pp. 87–98, 2017.

[15] Y. Zhang *et al.*, "Bundle adjustment for monocular visual odometry based on detected traffic sign features," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 4350–4354.

[16] J. Lu, J. Chen, and J. J. Little, "Pan-tilt-zoom slam for sports videos," *arXiv preprint arXiv:1907.08816*, 2019.

[17] A. Kendall *et al.*, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2938–2946.

[18] B. Zohuri and M. Moghaddam, "Deep learning limitations and flaws," *Modern Approaches on Material Science*, vol. 2, 01 2020.

[19] C. Yu, Z. Cai, H. Pham, and Q.-C. Pham, "Siamese convolutional neural network for sub-millimeter-accurate camera pose estimation and visual servoing," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 935–941.

[20] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 539–546 vol. 1.

[21] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., 2012.

[22] A. J. Kumar *et al.*, "2023 7th international conference on computing methodologies and communication (iccmc)," 2023, pp. 251–256.