TS-Net: Landmark Localization from Event- Based Cameras Time Surfaces

Alon Shavit

School of Electrical Engineering, Faculty of Engineering, Tel-Aviv University, Ramat Aviv 69978, Israel.
alonshavit@mail.tau.ac.il

We present a deep learning solution for automatic localization and tracking from event based cameras time surfaces (TS). We suggest improvements to the known UNET architecture, in order to increase accuracy and generalization capabilities on TS data. A key aspect of our approach is adaptively combining heat- map loss with coordinate loss by using a differential 'softArgMax' layer. In addition, we propose a novel technique for automatic detection of outlier tracking points. We present results on a synthetic data set which we create, which simulates time surfaces images. Our framework successfully tracks 3D key- points from 2D simulated TS images, while providing <1 pixel accuracy.

***Index Terms*—Deep Learning, 3D Tracking, Event Based Cameras.**

I. INTRODUCTION

Event- based cameras (EBC) are asynchronous sensors, which measure brightness changes for each pixel within a frame. This principle of operation, which mimics the neural architecture of the eye, has made some researches to call those devices "silicon retina" [*21*]. In other words, while standard cameras measure absolute brightness, EBC are triggered by a change in brightness magnitude. When triggered, the output is a stream of events.

Each event consists of time tag, the location (x,y) and the sign of the brightness change, sometimes represented by t_k x_k and p_k accordingly, for the k^{th} event . Hence, events are caused by moving objects, which create brightness gradient in the area of dynamic edge pixels [*1*]. As a result, static scenes without conspicuous luminance changes are no likely to produce a notable stream of events.

EBC have several characteristics which make them a suitable solution for problems which are common among standard frame-based imaging sensors. Firstly, they are capable of supplying very high dynamic range (HDR) imaging (140 dB vs. 60 dB in standard cameras [*22*]). Fig. 1 for example, demonstrates the benefit of using HDR in a scenario of a car driving out of a tunnel. Secondly, EBC monitor the brightness changes using an analog circuit. This principle leads to a very low latency (less than 1 millisecond [*1*]). Finally, since power is only used to process changing pixels, EBC have very low power consumption (usually less than 100mW [*23*]).

Fig. 1. Standard camera (left) verses a reconstructed image from EBC (right) [22]

However, due to their paradigm shift in acquisition of visual information, EBC have posed the challenge of designing novel methods to represent and process the extracted events data. There are two common representations, which are used by most recent algorithms – time surface (TS) and motion compensated event image (MCEI). TS is a 2D map where each pixel stores the timestamp of the last event at that pixel [*25*],[*26*]. Thus, the constructed image gray levels represent the motion history for each pixel.

On the Contrary, MCEI depends not only on events, but also on some motion hypothesis. This method measures how well a group of events (over some

time interval) fits a candidate motion, and selects the hypothesis which yields the best fit. The resulted image is a map of edges, caused by the event stream during some time interval [*27*],[*28*].

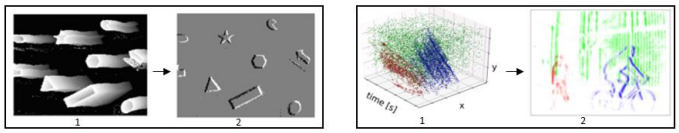


Fig. 2. ***(a)*** *Time surface (a2) is created over the last 33 ms from a stream of events (a1). For a1, brightness represents time, from past (dark) to present (bright).* [7]

(b) *Motion compensated image (b2) is created using a motion hypothesis for 3 classes of segmented events from(b1).* [27]

A notable observation is that for both methods (TS and MCEI), the output can be considered as **contour representation** of the moving objects. This important insight is be very useful in this work, since we could rely on it in order to approximate the output of EBC, after some primary preprocessing.

Although EBC have become commercially available only since 2008 [*24*], their benefits led to a big interest from the automotive industry. This new technology has narrowed the technical gaps for applications like steering prediction [*16*] and pedestrians' detection [*20*], which are crucial capabilities in any autonomous vehicle development process. For all those tasks, the capability of precisely tracking the location of an object is a fundamental purpose to enable the development of autonomous vehicles.

However, existing methodologies assume 2D motion model, and does not supply a suitable solution for situations when an object dramatically changes its 3D orientation. More generally, there is no existing solution which enables to track the 3D to 2D projection of some key points within a 3D object.

In this work, we propose a novel deep learning solution, specially designed to use 2D data from event based cameras, in order to track several 3D points. We aim to construct a new building block in the field of EBC processing, which would allow the development of many further algorithms which are required to use 3D data within a 2D processing pipeline.

Overall, this paper makes the following contributions:

- A simulation technique to create an approximated EBC time surfaces dataset. We show how the dataset is used to train a DNN to track key points within an object.
- A deep learning network which is highly optimized for TS data. We show that the network provides very accurate (<1 pixel) point-level tracking capability. In addition, we show how to perform suitable data preprocessing, including using binary representation - which is not a typical input to DNN.
- A novel training technique (AWM), which changes the loss function during the training time in order to increase the overall accuracy and precision. The method adaptively combines heat map loss with coordinate loss. First, we show how to perform a differential ArgMax operation (softArgMax) for 2D TS. Next, we suggest a method to adaptively combine the softArgMax loss after a primarily heatmap training. We analyze the suggested method, and show that this training flow creates a fully differentiable model with very accurate positioning capability.
- We present a novel method for automatic detection of outlier tracking points directly from the network output, and show how it leverages the unique characteristics of the AWM training technique.

II. RELATED WORK

In this section, we review several works regarding event based cameras tracking, available EBC datasets and the connection between EBC data and contour-based representation. Finally, works about

3D tracking on conventional frame cameras are reviewed for reference.

A. Tracking Algorithms for Event Based Cameras

Early works suggested some methods to track simple shapes (like lines [*30*] or circles [*31*]) by using event based cameras. For this task, each incoming event is assigned to a given blob, and some mathematical model like GMM [*29*] is then used to update the tracking parameters. However, those methods are limited for basic 2D shapes and require tuning many parameters manually in order to achieve sufficient performance.

Tracking by using complex shapes representation has been demonstrated by using feature based methods. Some articles, like [*2*], suggested using local edge patterns which are represented as points-sets, while incoming events are assigned to each feature by clustering techniques such as ICP [*32*]. Others ([*3*],[*5*]) suggested to detect edge patters on a regular frame, and then track them using events. A group of complex shapes as faces or the human body, were suggested to be tracked using "history based" shape models [*33*], where different elements are linked by springs [*34*]. Some approaches ([*7*],[*6*]) even suggested variations for the classical Harris [*58*] or FAST [*57*] corner detection, by operating on time surfaces images. However, those solutions required a lot of feature engineering in order to select appropriate edge descriptors and some motion model to accurately track them during time. In addition, none of those techniques are capable of dealing with 3D orientation updates which change the appearance of the tracked object.

As we can see, the problem of tracking in 6 degrees of freedom using event based cameras is challenging. An algorithm to track the camera pose was presented in [*35*]. It assumes that events are only generated by the closest map lines of events to the camera. This map is then tracked to provide points to camera calibration methods like [*36*]. For tracking, a non-linear optimization of the "event to line" projection error is made for each frame. However, those methods only compute the camera trajectory, rather than an object's pose. To overcome this limitation, [*37*] proposed a probabilistic filter with a robust likelihood function, which is based on a densities mixture model. Different approach is suggested by [*38*], which suggested nonlinear optimization of the photometric error between brightness increment images and their prediction given the scene map. However, none of the above is able to estimate the location of points which are hidden to the camera. In addition, there is no connection between the tracking points to some known locations within an object. Those two limitations make it impossible to precisely locate the object's points in the 3D world, without the need of physically rotating the camera. In this work, we show that this capability can be achieved using our method.

In recent years, artificial neural networks (ANN) have become the industry standard for solving complex computer vision tasks. Yet, works about event based cameras tracking ANN algorithms are not common. Some works showed the ability to use convolutional neural networks (CNN) to process event based data for semantic [*40*] and motion [*9*] segmentation. On [*16*], the possibility of estimating the steering angle of a driving vehicle by using event camera is demonstrated. The capability of estimating pixel- wise optical flow using CNN was demonstrated in [*39*]. While most of those methods use time surfaces information and CNN to produce their output, they all share some fundamental limitations; Firstly, none of them is capable of dealing with location estimation of hidden key points which have no direct line of sight through the camera. Secondly, all method do not provide any 3D information regarding the observed scene. This limitation was partially improved in [*41*], which added the capability of estimating the depth by combining 3 different images: time surface image, positive event image and negative events image.

However, the problem of tracking a point in a 3D object which changes its orientation during time, by using a **single** TS image remained unsolved. The purpose of this paper is to overcome this gap, by

using a single time surface image to estimate the location of any 3D points in any possible orientation of the object, without any requirements of having internal features.

B. Available Data Bases of Event Based Cameras

Any deep learning solution requires data for the development and the evaluation process. However, we have not found an existing data set which is suitable for our needs. On [*9*], a dataset of motion segmentation is provided. This dataset consists of ground truth data which divides pixels into sub-groups according to their motion. However, it has no information regarding the location of each pixel in the real world.

Alternative data set is presented in [*20*], which introduced an event-based detection dataset for automotive. It provides a good solution for detection tasks, which is not very useful in our specific case.

Another interesting approach is provided by [*19*], which suggests to simulate EBC data by using DNN in order to use tagged data from existing, regular cameras datasets. While this approach seems to be suitable for detection and segmentation tasks, it is missing the connection to the 3D to 2D projection. In other words, it is a good approach for scenarios where the object barely changes its 3D orientation, which is not a correct assumption in our case. In addition, the requirement of using another CNN just for the data creation requires much more pre-processing computational resources.

C. Event based data and contour representation

Despite of the difficulties of [*19*] , which were mentioned above, a very interesting observation is revealed when carefully observing its results - the output seems to be the **contour** of the moving objects. A demonstration for this claim can be seen in *Fig. 3*. Moreover, it seems like other resources support it as well. A classical method for event based tracking by using contours was suggested in [*18*], which showed that generative contours can be used instead of classical internal features such as corners. Another approach of using contours instead of features is described in [*11*], which showed how to use contours to resolve the occlusion problem for augmented reality applications. Finally, [*13*] deals with the problem of contour motion estimation for event based cameras. It shows that event based TS image could be interpreted as "contour motion".

Summing it all up, it seems that there exist evidences which support the claim that time surface representation of event camera can by approximate by using the object contours.

Fig. 3. Left: regular cameras. Right: Simulated events cameras [19]

D. Tracking Methods & Neural Networks Building Blocks

There is a rich literature regarding 3D tracking and pose estimation from a single regular camera image. Classical methods like [*42*], and more recent work such as [*43*], [*44*] build a mathematical model to recognize 3D areas of interest from a 2D image. However, we shall now focus on convolutional neural networks based methods, which have become the preferred method for most recent 3D tracking and pose estimation tasks.

Some CNN based methods, such as [*8*], [*45*] and [*46*] predict 3D pose directly from a single regular image. Those methods are usually based on discretizing the pose space into bins. Then, a network is trained to solve a pose classification problem. A particular interesting approach has been presented in [*14*], which showed that a self- supervised viewpoint learning can be done by using a generative adversarial network (GAN). However, in our case we want to track a specific group of points rather than estimating the global orientation of an object. Hence, we shall now review methods which are more related to this use case.

One of the first works about 2D localization of 3D points from a single regular image was presented in [*47*]. In this work, a method of generating heat maps

by running an image though multiple convolutional layers is presented. Others have tackled this problem in similar ways with some modifications. [*10*] and [*12*] suggested some variants to the network structure, such as placing multiple hourglass modules together ([*10*]) or adding a residual block ([*12*]).

More recent works suggested further improvements to the hourglass – based architecture. In [*15*], a technique to improve landmark localization in images from partially annotated datasets is presented. In [*17*], a solution for combining several 2D outputs into one 3D reconstructed model is shown.

However, **all the works above use different assumptions compared to ours**. First, they all designed the network to deal with regular, frame-based images, which consists of many internal features. Hence, most the evaluation and the development phases are made by using images of faces or the human body. Secondly, they all use RGB images, which contain information from three color channels. Furthermore, they all assume that the majority of the key points have a direct line of sight to the camera. Finally, none of them suggest methods for outliers' detection. While our solution adapts some concepts from those works, we improve crucial components in the network and the training process, in order to enable: (a) using TS images by considering only the object contour without relying on any internal features, (b) using binary images with only one channel, (c) location estimation of hidden points and (d) automatic detection of outliers in each frame.

III. METHODS

This section describes the methods we used to create the 3D to 2D ground truth data base. Then, we discuss the preprocessing considerations which increases the trainability and accuracy of our solution. Next, the network architecture is described with focus on elements which improve the "global attention" of the network. We then described our innovative loss and adaptive weighting training process (AWM). Finally, we suggest a novel method for automatic outlier detection.

A. 3D to 2D Data Creation

In order to create the data base, we used several 3D cars models. Each model consists of 3D locations, sometimes called "points cloud". To simulate a rotation in a 3D space, several steps were handled.

First, we defined the 3D rotation range around each primary axe of the model. The three primary axes were defined to be the object's principal axis of inertia, which were calculated according to [*48*]. The reference point for the rotation process was set to be the object's center of gravity. Next, we calculated the 3D rotation matrix around the reference point, by a matrix multiplication of three independent rotation matrices (one matrix per each principle axe of inertia). Finally, we added a 3D translation and scaling by performing another multiplication. Overall, the 3D transformation matrix R_j is the overall 4x4 transformation matrix of the j^{th} transformed sample. The matrix is given by Eq. 1, where $R_{j,i}$ is the 3D rotation matrix around the i^{th} moment of inertia and T_j, S_j are 4x4 translation and scaling matrices, accordingly.

Eq. 1: $$R_j = S_j T_j R_{j,2} R_{j,1} R_{j,0}$$

The next step is creating the 2D ground truth (GT) locations for some selected points which their location is given in the 3D world. To produce this data, we need to make some assumptions regarding the camera properties and its location within a 3D world. For this work, we simulated a 512x512 events camera, which is located in a constant location. Then, we calculated its intrinsic and extrinsic transformations. The extrinsic matrix, E, is a 4x4 matrix, while the intrinsic matrix, N, is a 3x4 matrix which makes the 3D to 2D projection. Overall, we get the full 3D to 2D transformation matrix F_j of the j^{th} simulative sample, which is given by Eq. 2:

Eq. 2: $F_j = NER_j$

Now, the 2D location of any 3D coordinate within the simulated object is given by *Eq. 3*:

Eq. 3: $a_{k,j} = F_j b_{k,j}$

Where $a_{k,j}$ is the 2D location of the k[th] point, under the j[th] 3D to 2D transformation of point $b_{k,j}$. Note that the points are given in homogenous coordinates, so the final output vector $a_{k,j}$ is finally normalized by its last component. The process above produces *n* ground truth 2D projections (j=1,2…n) for *m* 3D locations of points within the object (k=1,2…m).

To generate images samples for training, we have applied this process over 500,000 randomly sampled 3D points to get their 2D locations. Next, we created an approximated depth map by counting the number of 3D points which are mapped into the same 2D pixel, and gave each 2D pixel a relative grey level. Finally, we used morphological operations to construct the final 2D time surface image. A graphical demonstration of the output is shown on the top row of *Fig. 4*.

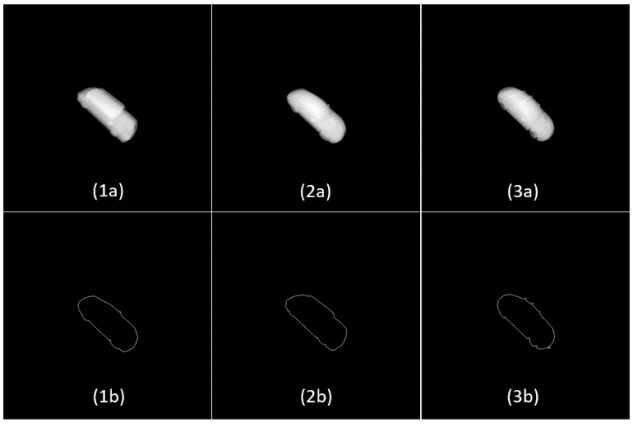


Fig. 4. Depth map (up) and simulated TS frames (down) for different 3D modules

Note that the entire data generation process is entirely handled without using any external rendering engine. The purpose of this choice is to simulate event based data which is created pixel-wise, just like our suggested process. In addition, it enables better control of the entire creation process.

B. Pre- Processing Pipeline

After the previous stage, we now have a group of *mn* 2D coordinates $C = \{a_{k,j}\}$ and a group of *n* 2D images $S = \{I_j\}$. We now describe a four steps pre-processing pipeline which is made before inserting data to the network.

The first pre-processing step is image binarization. In general, a time surface image from event cameras is not a binary image. However, we suggest to use binary images for several reasons. First, it narrows the gap between simulation and reality by cancelling the integration time and the specific camera model dependency. In addition, it cancels the dependency of the movement direction. Hence, unlike traditional time surface, the binary image of an object which moves to the right is identical to the binary image for the same object when it moves to the left. Another advantages is that binarization enables better support for pruning and quantization optimizations when assembling the solution into real- time devices.

Next step in the pipeline is simulating the contour nature of the event based cameras, as we discussed in the introduction. To create it, we use a simple canny edge detector which is applied over the binary images. This process simulates the physical nature of event based cameras, which tends to create events in the edges of moving objects. A graphical demonstration of the output is shown on the bottom row of *Fig. 4*.

The third pre-processing step is heat- map generation. Given the j[th] transformation, we have a group of *m* 2D coordinates. For each coordinate within the group $a_{k,j}$ we create a heat- map image, which has the same shape as its corresponding object's image I_j. This heat- map image represents the ground truth 2D location of the 3D coordinate. To create this image, we first set all its pixels to zero, except of the pixels within the area of the ground truth coordinate. We noticed that creating an image

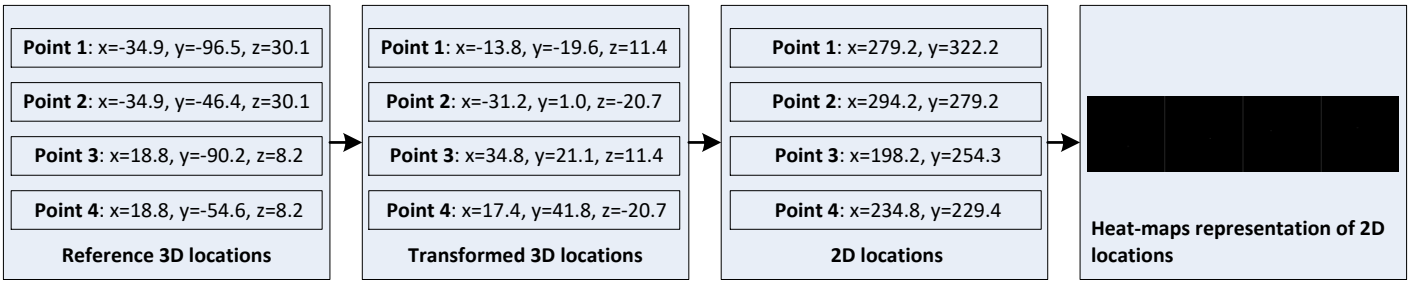


Fig. 5. Ground truth (GT) creation flow

with only one non-zero pixel (which is located in the coordinate according to the ground truth) has led to very slow learning process, which sometimes would not coverage at all. Instead, as proposed by [*47*], we used a blurring kernel to create the heat map. For this purpose, we used a Gaussian kernel of shape (17,17) and $\sigma = 1$. A demonstration of the creation flow is shown by *Fig. 5*.

Next, we apply data augmentation. This step is very important since it encourages learning process which is invariant to translation, rotation and scaling. To create the augmentation, we generate an affine transformation T, each time we insert a certain sample into the network. Then, T is applied to the input image and the ground truth coordinates.

The final pre- processing stage is data normalization. All the data is normalized to the range 0 to 1 to achieve dynamic range consistency.

C. Network Architecture

In this session we describe the primary modules of our deep learning architecture, in a "top down" approach. We start with a general overview, and then focus in our solutions to enhance the capability of dealing with time- surfaces frames from event based cameras.

The input to the network is a binary, one channel tensor of simulated TS image, as we described in the previous session. For each sample, the network is trained to output m-channels heat- map tensor, where the location of maximum value within each channel represents the predicted location of the k^{th} landmark.

We use the "UNET" network [*49*], which is a common CNN architecture for applying deep learning for sparse image data [*50*],[*51*],[*52*]. We build upon previous work and improve the original architecture by leveraging recent advancement in CNN architecture design, to achieve a solution which supplies superior performance for the simulated time surface data.

Our first modification is integrating squeeze and excitation (SE) blocks within the UNET architecture. SE module [*53*] improves "channel attention" by applying three steps over the input data: squeeze, excitation and scaling. In our work, we slightly modified the architecture which is described in the original paper. In the "squeeze" step, we apply global average pooling over the input channels. That way, we transfer an input with shape (H,W,C) into an output of shape (1,1,C). Next, in "excitation" step we apply four stages, where 'f' is a factoring parameter:

a. 2D convolution with (1x1xC/f) kernel and (1x1xC/f) bias vector.
b. ReLU activation.
c. 2D convolution with (1x1xC) kernel and (1x1xC) bias vector.
d. Sigmoid activation

Finally, we multiply the input with the output of the excitation step to get the final output of the module. The multiplication is done channel- wise, so the final output has the same shape as the input. The entire flow is described in *Fig. 6*. In our implementation we use f=16.

The SE block has several advantages for time surfaces data. Firstly, it extracts global information rather than conventional convolutions which basically look for local features. This capability is particularly important when dealing with TS data, which has no internal features, and is characterizes by its "global contour shape". In addition, when

localizing several key points, one can obtain that they do have some structural connectivity, i.e. the location of each point is correlated with the location of the others. Hence, the channels attention obtained by the SE block seems to profit from this insight. Finally, it was recently shown [*54*] that SE block enables better generalization capability across different datasets, which is an important capability when using simulated data.

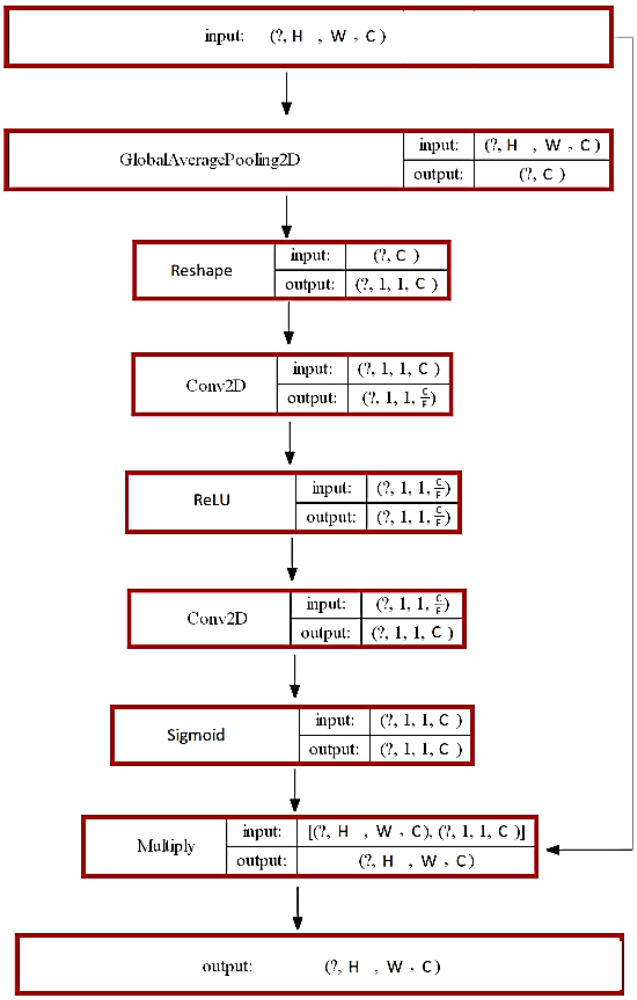


Fig. 6. SE block in our implementation – a flow diagram. 'F' is a scaling hyper- parameter.

Another improvement we made to the original "UNET" is combining residual blocks with skip connections [*55*]. We used the ResNet-18 backbone, with some modifications to handle the different tensor shapes through the network. To combine the residual blocks with the SE blocks, we applied the SE block over the output of the residual block as proposed by [*53*]. A demonstration of this integration is shown in *Fig. 7*.

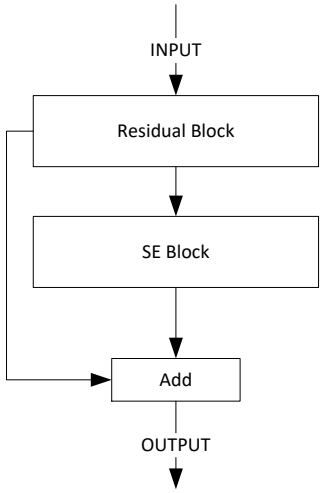


Fig. 7. Combining SE block within Res-Net architecture

Finally, unlike the original "UNET", we use transpose blocks to perform up- sampling. By doing it, we train the network to learn the optimal up-sampling kernel rather than using predefined interpolation method.

At the bottom of the network, we get a [H,W,m] tensor, where 'm' is the total number of key points which we want to estimate their location. Then, we use the Sigmoid activation function to force an output with a dynamic range between 0 to 1. The channels within this tensor are the predicted heat-maps, which represent the 2D location of each key point.

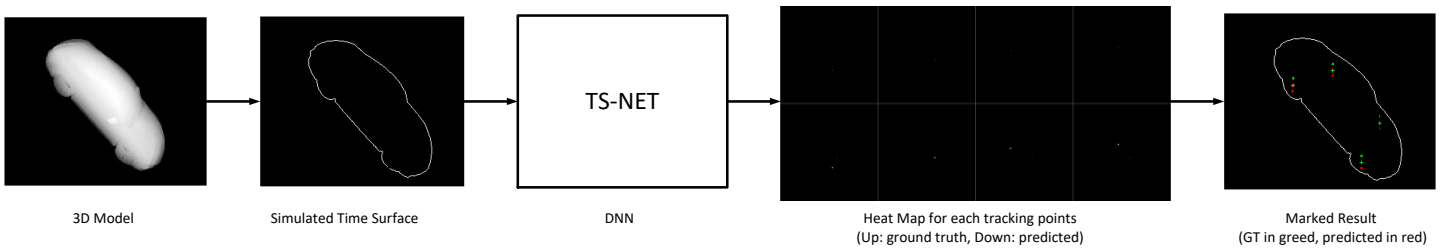


Fig. 8. TS-Net input to output schematic flow

The flow could have stopped here, since we got the desired heat- maps. However, we further improve the performance by leveraging the fact that we do know the ground truth X,Y coordinate of each heat map (since we have created each heat map during the pre-processing stage). The obvious solution is to use an ArgMax layer which gets a heat map as its input. However, since ArgMax is not a differential operation, we use an approximation called SoftArgMax:

$$\text{Eq. 4: } M(I) =: \sum_{i=0}^{WH-1} [range(I) softMax(\beta I)]_i$$

Where:

- I is a 1-D vector of shape [(W x H) x 1]^T elements.
- M(I) is a scaler approximation for argMax(I).
- β is an extending factor scaler. In our implementation we use $\beta = 150$.
- $range(I)$ is the vector [0,1,2…WxH-1]^T
- $softMax(x)$ is the function $\sigma(\bar{X})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$

The extending factor is required to get better spatial separation between close values. Note that the expression returns an approximation of a single coordinate. Since we deal with 2D image, we define the following:

$$\text{Eq. 5: } \begin{cases} H_x = Flatten(H) \\ H_y = Flatten(H^T) \end{cases}$$

Where:

- H is a 2D heat map image of shape (W x H)
- $Flatten$(x) takes a 2D tensor and turns it to 1D tensor by rows- stack operation

Eventually, we get three different output from the network:

- H – [W,H,m] shaped heat- map tensor
- M(H_x) – [m] shaped softArgMax tensor for the x direction
- M(H_y) – [m] shaped softArgMax tensor for the y direction

The complete network architecture appears in **Error! Reference source not found.**.

D. Loss Functions & Adaptive Weighting Method (AWM)

To train the network, we use three different loss functions. The first loss, L_{HM} is defined to be the mean square error (MSE) between the output heat map H of the network to the ground truth heat map H_{GT}. The two other losses, L_X and L_Y are coordinate loss. Those losses are computed by comparing the output of the softArgMax layer to the known ground truth coordinate of each tracking point, for axes 'x' and 'y', respectively. Then, the mean square errors, MSE[M(H_x), X_{GT}] and MSE[M(H_Y), Y_{GT}] are calculated for each axe, independently. A graphical illustration for this process is given in *Fig. 9*.

The total loss, L_{tot} is finally calculated by linearly combining the three losses:

$$\text{Eq. 6: } L_{tot} = \alpha L_{HM} + \beta L_X + \delta L_Y$$

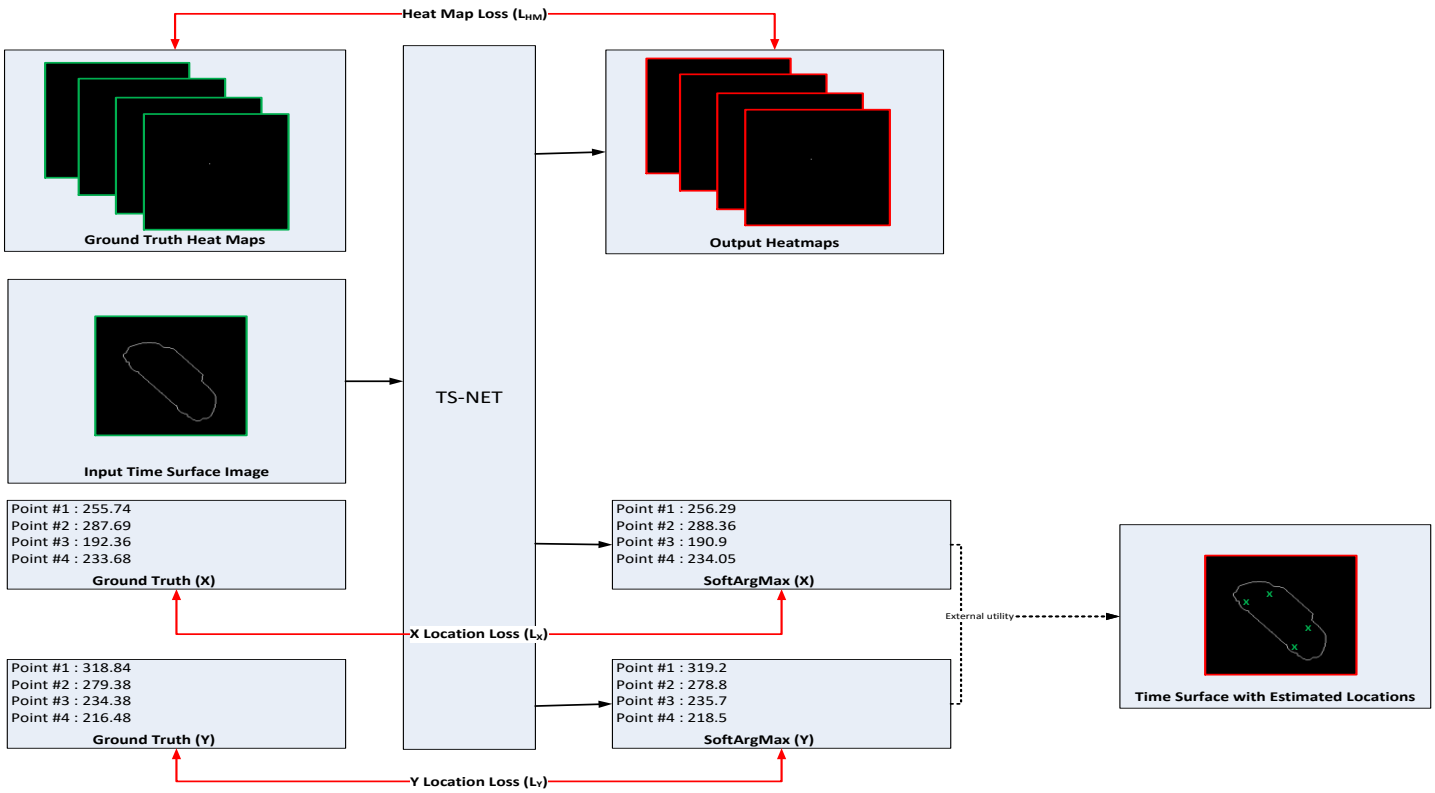


Fig. 9. TS-Net losses

The intuition behind Eq. 4 is that the argument of the maximum value can be approximated by the expectation of the input vector. However, this assumption is valid only for sufficiently leptokurtic input distribution. One way of increasing this property is by using the extending scaler β from Eq. 4. However, sometimes it is not enough since the input has no clear maxima which are "spatially concentrated" in a limited area. Hence, the SoftArgMax approximation is not valid for primary stages of the training process, when the heat- map output has not yet been stabilized enough to produce a leptokurtic probability distribution.

To overcome this problem, we suggest using an "adaptive weighting" method (AWM). This method changes the values of α, β, δ according to the progress of the training process. We begin by setting $\alpha = 1$ and $\beta, \delta = 0$ to encourage the creation of a primary heat map, while monitoring the values of L_X and L_Y. Then, when those values stopped decreasing, we decrease α and increase β, δ. We later show that AWM enables precise location estimation by overcome the problem of platykurtic input.

E. Automatic Outliers Detection

One of the major problems of deep learning tracking solutions is the abundance of a measurement for outliers detection and filtering. Unlike classical tracking methods, which provide some "score" for each tracking results (i.e. correlation coefficient etc.), the output of DNN is usually provided without any "confidence measurement".

This limitation is a major problem for tracking systems which are usually integrated into a close loop control system. Hence, even rare wrong results might lead to unstable tracking, if deep learning output is used without some classical temporal filtering and averaging.

We suggest a novel scoring method for automatic outliers detection, which leverages the properties of the AWM training method and the softArgMax layer.

The first step is to give a score for each point. Given an output heat map H of shape [W,H,m] (w=1..W,

h=1..H), we perform the following steps for each channel k (k=1..m):

a. Calculate the total heat- map energy $J_{k,total}$ by: $J_{k,total} = \sum_{w,h} H_{w,h}$
b. Find the x-y coordinate of the maximum, by using the SoftArgMax layer.
c. Delete the area around the maximal x-y coordinate, by using circle – shaped zero mask with a constant radii. The output of this stage is $H_{w,h,masked}$
d. Calculate the total heat- map energy $J_{k,relative}$ by: $J_{k,relative} = \sum_{w,h}(H_{w,h,masked} - H_{w,h})$
e. Calculate the score of point k by: $s_K = \frac{J_{k,relative}}{J_{k,total}}$

After the previous step, we get a group of *m* scores: $[s_1, s_2 \dots s_m]$. Then, in order to find the point which is most likely to be an outlier, we return outlier_index = argMin($[s_1, s_2 \dots s_m]$.

IV. EXPERIMENTS

A. Data

To create the data set, we used the process which was described in the previous section (parts A,B). Three different 3D models (*Fig. 10*) were used. Note that we have intentionally chosen modules which have different appearance. Hence, each simulated time surface is different compared to the other models (see Fig. 4).

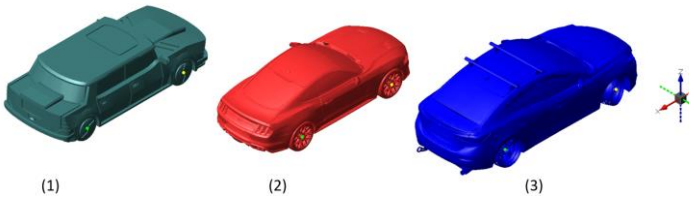


Fig. 10. The 3D models which were used to create data. Two of the four tracking points are marked on yellow (front right) and green (rear right)

As tracking points, we have selected four locations on the outer middle part of the cars' wheels (front left, front right, rear left and rear right wheel). Each point was given a number in the range 0 to 3, accordingly.

To simulate different 3D orientation, we used the parameters from Table 1 below:

	Minimum value	Maximum value	Resolution
x rotation	-50	50	4
y rotation	0	50	3.8
z rotation	40	140	4

Table 1: Transformation parameters for data creation. All numbers are given in degrees

Totally, we created a dataset of 8,125 simulated time surfaces samples, for each model. Some instances of different orientations are given in *Fig. 11*. Each data set was randomly divided into train (70%), validation (10%) and test (20%).

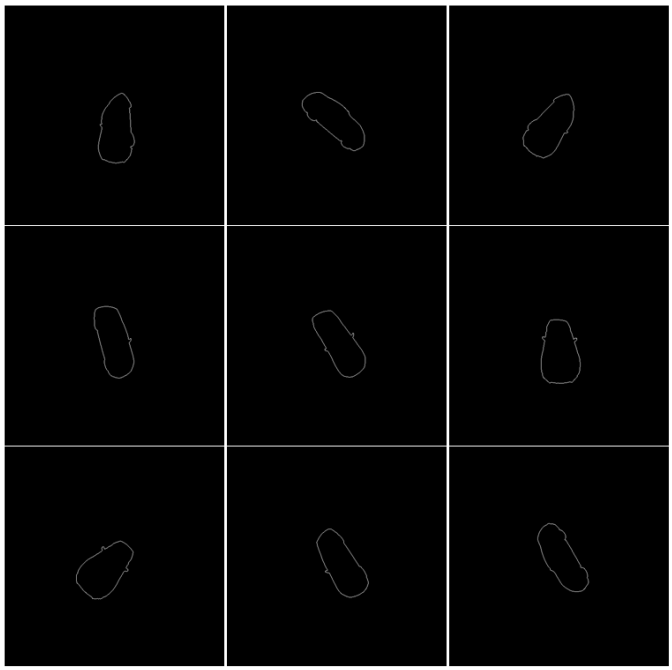


Fig. 11. Examples of simulated TS frames for different 3D positions (model #1)

B. Training Details

As part of the pre-processing, we have applied a similarity transformation over the simulated TS, as described in B, on the methods section. We used a transformation T, which is randomly created for each sample with the following parameters range:

- x translation (pixels) $\in$ [-5,5]

- y translation (pixels) $\in$ [-5,5]
- rotation angle (degrees) $\in$ [-10,10]
- magnification factor $\in$ [0.8,1.2]

We used Adam optimizer [56] with alternated decreasing learning rate. In addition, we have adaptively changed the values of α, β, δ according to our adaptive weighting method (AWM). The batch size was set to 16.

C. Evaluation Setup

We evaluated our solution by estimating the location of the 4 key points on the **test** part of the dataset. For the process, we created four different evaluation setups.

On the first two setups ("A","B") we trained the network on samples from model #1 or model #2, accordingly. Those setups demonstrate cases when there is only one possible type of object to track. On the third setup ("C") we have combined the data sets of model #1 and model #2, to create a unified dataset (of total 16,250 samples). This setup demonstrates a case when there are several known possibilities for the object's type. We would like to emphasize that for all those three setups, the network has never given the test samples during training.

On the last setup ("D") we trained the network on the unified dataset "C" which consists of samples of model #1 and model #2. Then, we evaluated the performance over **all samples** of model #3, which the network has never been trained on. Note that we chose 3D model #3 to have different external appearance compared to the first 2 models – i.e. it has different contour comparted to the two others. This setup represents a case when the object of interest is completely unknown, except of its type (a "car" in this case). In other words, it enables exploring the generalization capability of the network to other categories. A summary of the evaluation setups is given on Table 2.

	3D models for training	**3D models for evaluation**
A : 3D Model #1	1	1
B: 3D Model #2	2	2
C: 3D Models 1+2	1+2	1+2
D: Unseen 3D model	1+2	3

Table 2: Evaluation setups

In order to evaluate the performance of our solution, we compare the estimated x,y location of each tracking point against its ground truth (GT) location. Then, the accuracy statistics is calculated for the Euclidian distance between the predicted and the ground truth points.

D. Accuracy Analysis

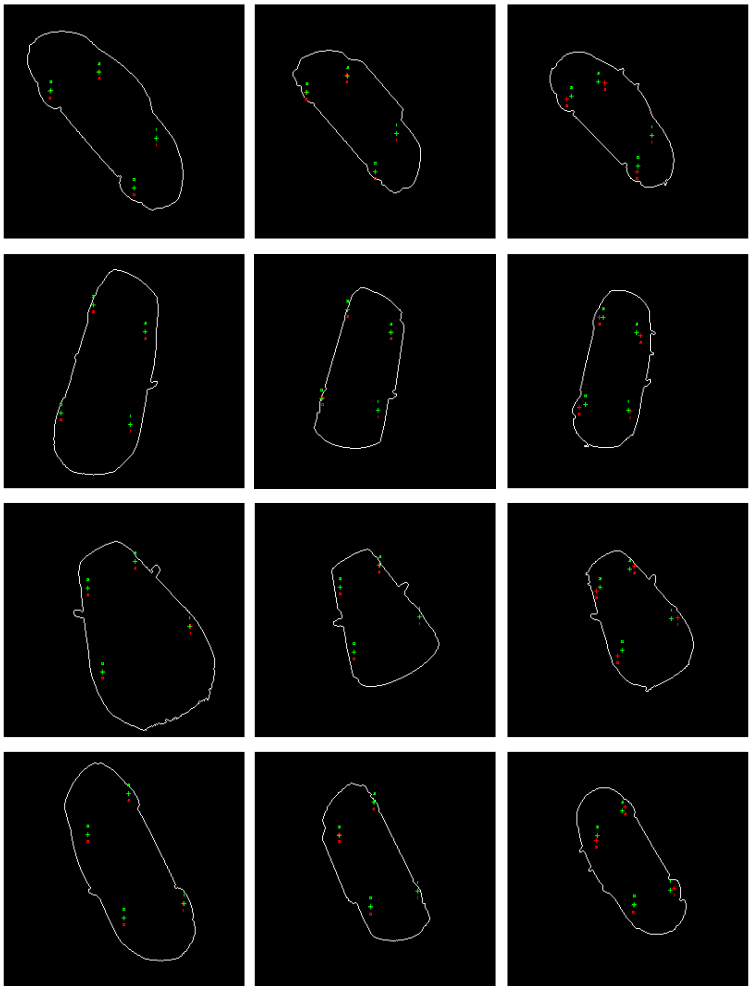
The mean and the standard deviation for the key points' Euclidian distances (GT vs. predicted) are given in Table 3 below:

Setup/ Statistics	**A : 3D Model #1**	**B: 3D Model #2**	**C: 3D Models 1+2**	**D: Unseen 3D model**
Mean	0.26	0.43	0.27	5.66
STD	0.45	0.53	0.46	1.74

Table 3: Accuracy statistics (in pixels) for various experimental setups

It can be seen that our method enables superior accuracy (~0.5 pixel) when the type of the 3D model is known. In addition, this accuracy is achieved even for a group of several 3D models, as in setup 'C'. This is very interesting property which suggests that the solution is capable to distinguish between different models, in order to select the unique features which are best suitable for a given input. In other words, it seems like the network learns to perform some classification step.

The results for setup 'D' are particularly interesting given the fact that the network was never trained on any samples from model #3. Despite this fact, it



(A) (B) (D)

Fig. 12. Output Examples of setups A,B,D for the same object pose. Predicted locations are red, while GT locations are green

seems that out solution learns to generalize a car according to the simulated time surface from models #1 and #2. Hence, given this generalization, it could use samples from **other** categories to estimate the location of key points on a "generic car". As for the accuracy, note that even though the mean Euclidian distance is relatively big (~5.5 pixels), its standard deviation is much smaller (~1.7 pixels). This is a positive property, since we usually interest to maintain the tracking stability, which is given by the STD, rather than knowing the exact "ground truth" location, which is given by the mean error.

Another interesting property to investigate is the statistical consistency for different points on interest. In other words, are there "preferred" points which yield better tracking accuracy? To answer this question, we have calculated the Euclidian error distance statistics for each point separately. The results are given in Table 4 and Table 5 below:

Setup/ Statistics for specific point		**A : 3D Model #1**	**B: 3D Model #2**	**C: 3D Models 1+2**	**D: Unseen 3D model**
0 (FL)	Mean	0.30	0.45	0.26	5.81
	STD	0.47	0.53	0.45	2.40
1 (FR)	Mean	0.27	0.40	0.27	4.89
	STD	0.45	0.51	0.45	2.21
2 (RL)	Mean	0.25	0.44	0.27	5.80
	STD	0.45	0.53	0.46	1.24
3 (RR)	Mean	0.24	0.44	0.26	6.15
	STD	0.44	0.53	0.45	1.11

Table 4: Accuracy statistics (in pixels) for different locations: front left (RL), front right (FR), rear left RL) and rear right (RR) wheels

Statistics/ Setup	**STD of mean over all points**	**STD of standard deviation over all points**
A : 3D Model #1	0.03	0.01
B: 3D Model #2	0.02	0.01
C: 3D Models 1+2	0.01	0.01
D: Unseen 3D model	0.54	0.66

Table 5: STD of statistical properties (in pixels) for different experimental setups

From Table 5, it can be seen that setups A-C have very low variance between the statistical properties of different points (STD is ~ 0.01-0.03 pixels). This is a desirable property for any tracking or landmark localization solution, since it shows that the accuracy has weak correlation with the selection of the landmark position.

However, we see that for setup D, we get higher STD (~ 0.54 – 0.66 pixels). This property suggests that for unseen 3D model there exist some

"preferable" tracking points, which yield lower statistical errors. In other words, we can say that some areas are "closer to the generic car" than others. Hence, points which are located within those "preferred" areas are expected to have better accuracy statistics.

E. Adaptive Weighting Method (AWM) Analysis

As we explained in previous section, all the experiments were handled by using our AWM for training. As a reminder, AWM adaptively increases the weight of softArgMax ('coordinate loss'), while decreasing the weight of the heat map loss.

In order to enable unbiased evaluation of the AWM method, we created a dedicated evaluation setup. At first, we trained the network by using only heat map loss until the loss stopped decreasing (i.e. the heat map loss as a function of training epoch became "flat"). Then, we saved the weights and stopped the training process. Next, we reinitiated two separated training sessions using the very same initial weights which were saved before.

For the first session ("Heatmap loss"), we set $\alpha = 1$ and $\beta, \delta = 0$. For the second session ("SoftArgMax loss") we set $\alpha = 0$ and $\beta, \delta = 1/2$. In addition, we reinitiated the process with a slightly higher learning rate than the "stop point", in order to encourage the network to "get out" from local minima. This setup enables us to fairly evaluate the effectiveness of combing softargmax (as in AWM), compared to the case when only heat map loss is used.

The results for this experiment are given in *Fig. 13*. In the two upper parts of the graph, it can be seen that using softArgMax has decreased the x and the y loss by ~ 30%. On the other hand, using heat map loss has not led to loss decreasing. Moreover, note that training with softArgMax is a much more stable process, with less "loss picks" compared to the alternative of training only with heat map loss.

The lower part of the graph is practically interesting

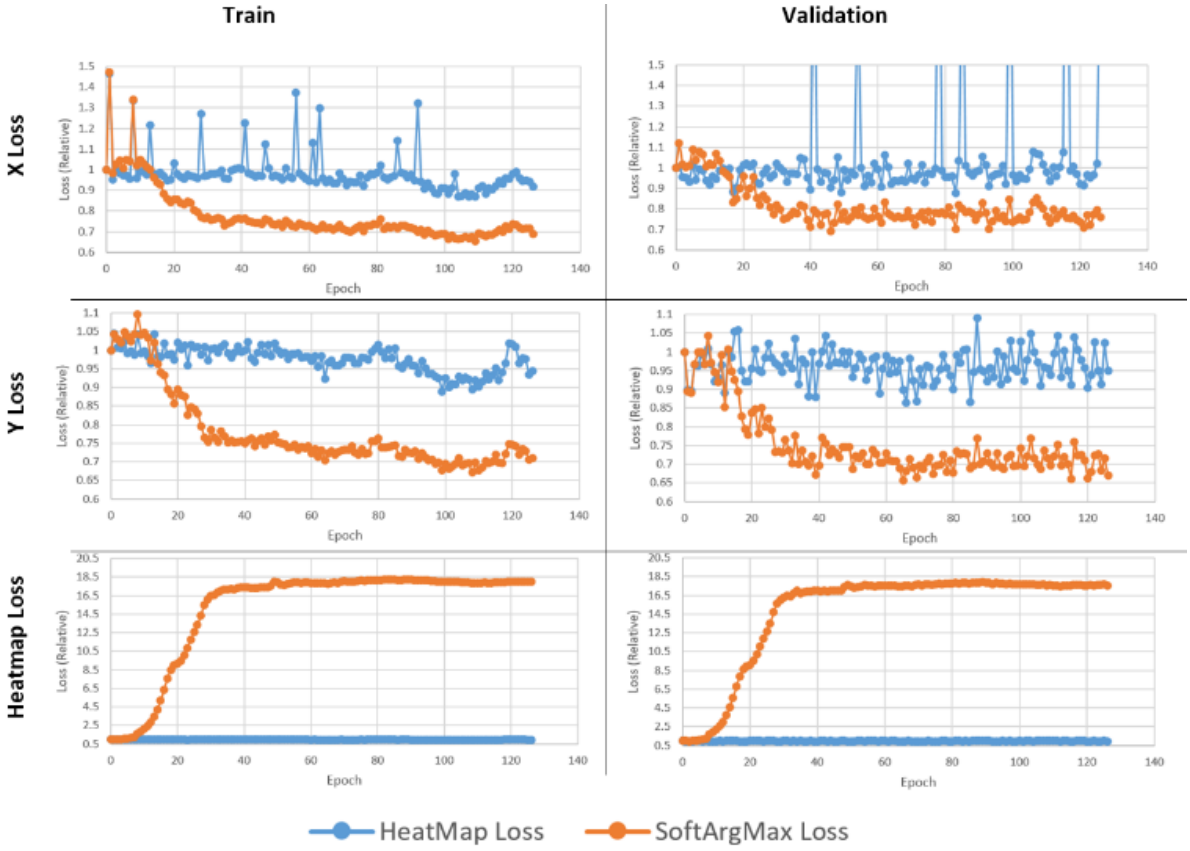
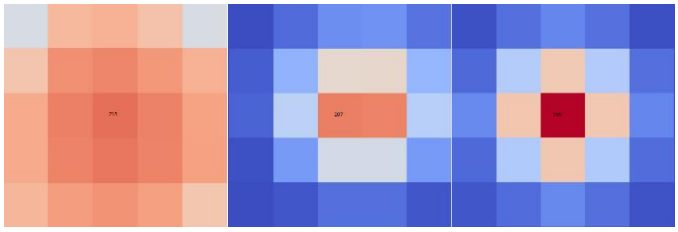


Fig. 13. Train and validation MSE losses (heatmap, Y location and X location) for different loss functions

- it shows that in order to achieve the decreasing in both X,Y loss, the heat map loss increases. This observation makes sense, since the softArgMax, which we use in order to calculate the x and y losses, is an approximation of the real Argmax. Hence, there might be a "preferred" heat map representation which is different compared to the one we chose for the training process.

Let us analyse this last claim even further; on *Fig. 14* we visually compare the 3 different heat map representations. On the right side of the figure we can see the "ground truth" heat map which we create for the training process. On the middle, the network's output for the heat map loss is shown. It can be spotted that, as expected, the network has learnt to output an image with relatively high MSE fidelity compared to the ground truth image. However, is this the right measurement for revealing the coordinate locations by using softArgMax approximation? The left part of the figure suggests that the answer is 'no'. It shows that the output when using the softArgMax loss, has higher maximum value (215 vs 207) with more gradual spatial reduction. As a result, we conclude that this change is responsible for the x and y loss reduction as shown on the upper parts of *Fig. 13*.



(1) (2) (3)

Fig. 14. The "on" area of heatmap for softArgMax loss (1), MSE loss (2), compared to train heatmap (3)

F. Outlier Detection Evaluation

To estimate our outlier detection method, we shall now focus in the results of experimental setup 'D', which supplied the worst result over the tested scenarios. Like any "classical" tracking method, we would like to see if we could automatically detect the outlier results in order to filter them out. Like previous session, we compare the heat map loss against the softArgMax loss as in our AWM training process.

As first step, we compare the standard deviation (STD) of the points' scores for each frame. Given four output points, we would like to sort them according to quality scores, which require some deviation between the results. In other words, STD=0 would not allow us to distinguish between "good points" and "bad points". The results of this test are shown in *Fig. 15*. It can be seen that without AWM, the STD is zero almost in all samples, which makes the sorting process highly unreliable. On the contrary, when using AWM (with softArgMax loss) the STD increases significantly. This property of the AWM is a crucial advantage for any "real life" application which requires results differentiation for automatic outlier detection.

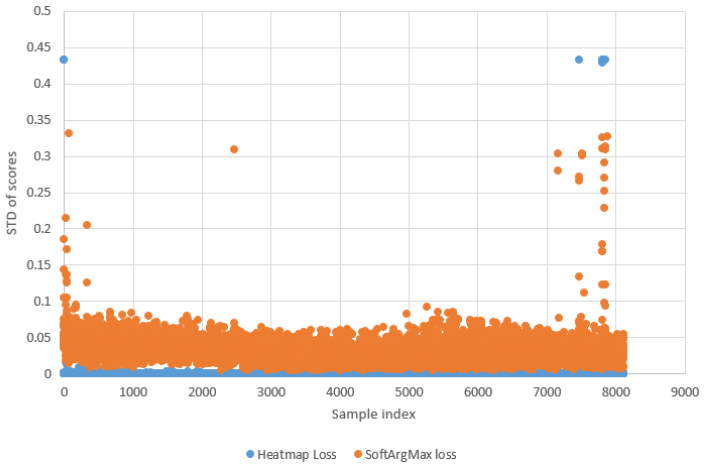


Fig. 15. STD of points – scores as a function of sample, for experimental setup D

However, there is still a question whether we could separate "bad" points from "good points" based on their calculated scores. To answer this, we detected the point which yielded the lowest prediction score in each frame. Then, we sorted all the frames according to the maximum prediction error of the worst point, based on the ground truth.

According to the previous steps we create *Fig. 16*, which shows the accumulative success rate of detecting the worst point, as a function of the relative

maximum error in a given frame (0 is the highest error, 1 is the lowest error).

The results show the superiority of our AWM training method. It can be seen that for high error rate (x values closer to 0 in the graph), softargmax loss dramatically increases the probability of correctly detecting the outliers, compared to the case when only heat map loss is used. In addition, throughout all the relative error range, AWM provides higher probability to select the "right" outlier among the tracking points. Note that as the relative error becomes lower (values closer to 1 in the graph), the effect of choosing the wrong outlier decreases, since the error value becomes mush lower.

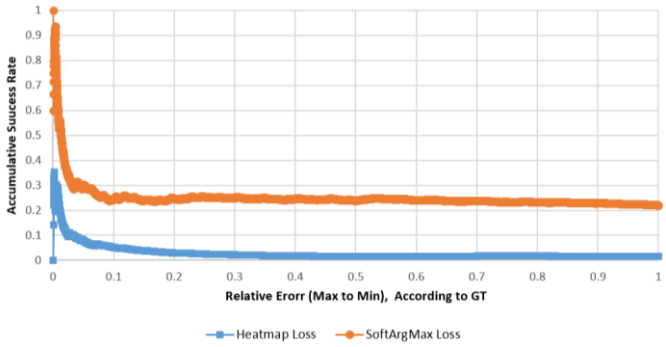


Fig. 16. Outlier detection accumulative probability as a function of maximum error

To end up this session,

Fig. 17 visually compares the outlier detection capability between softArgMax(AWM) and heatmap-only based training. It can be seen that when using only heatmap loss, the same score ('1') is achieved for all points, while in softArgMax configuration the scores can be sorted to detect the minimum. We assume the reason for this phenomenon is the different properties of heat map "on" area as we demonstrated in *Fig. 14*. It seems like "smoother" heatmap, as in AWM, encourages results disparity which allows us to effectively use our method for automatic scoring and outlier detection.

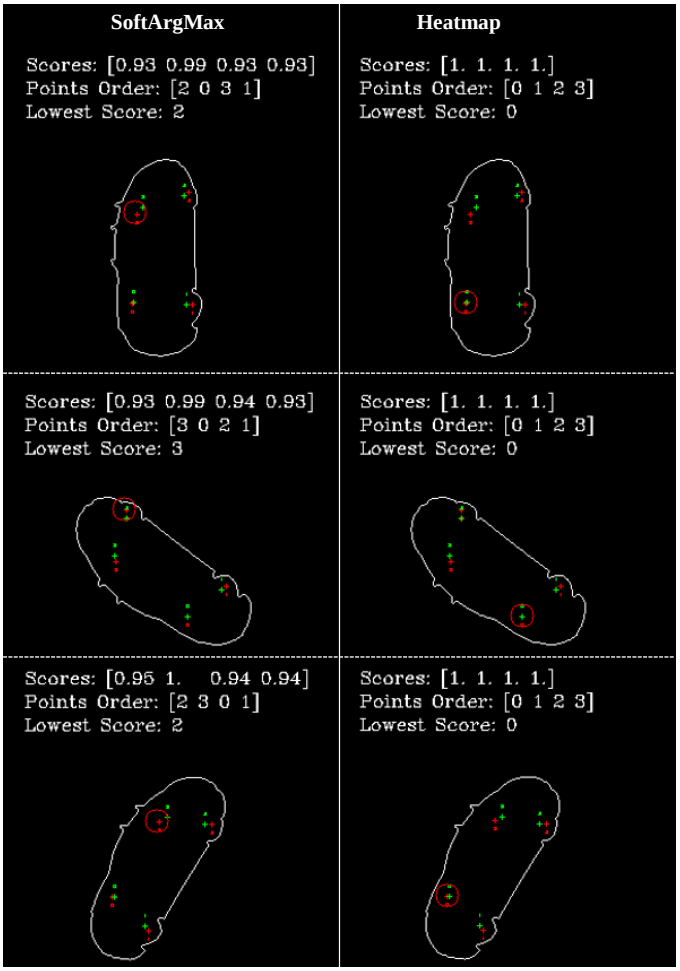


Fig. 17. Outliers detection capability when using softargmax (left)as part of our AWM method Vs. using only heatmap

V. CONCLUSION

We have addressed the problem of tracking several 3D locations on event- based cameras, by using time surface data representation. We reviewed multiple articles which deal with tracking and localization problem on event based cameras, showing that despite its high relevance, the 3D tracking problem has not been widely addressed.

In this work, we proposed a deep learning framework, which is specially designed for time surface input. We presented some improvements to existing architectures, which improve the learning capability and the perception of the network. In addition, we proposed a novel training method, AWM, which combines heat map loss with a specially designed softArgMax coordinate loss.

Finally, we proposed a novel method for detecting and filtering tracking outliers, which leverages the advantages of AWM.

The solution has been quantitatively evaluated on dataset which we created in order to simulate time surface data of event based cameras. We achieved remarkable perception (<1 pix) on the test set, and showed that our network is capable of dealing with unseen car model, at the price of decreasing the accuracy (~5 pix).

In addition, we investigated the properties of our AWM method, showing it improves the coordinate loss by encouraging the creation of slightly different heat map "on" area, compared to the case when only a heat map loss is used. Finally, we demonstrated the outlier detection capability of our solution, showing how it benefits from our AWM training technique.

Further work should test the solution on real data sets to fully evaluate its performance, when they become available.

References

[1] *Gallego, Guillermo, et al. "Event-based vision: A survey."* arXiv preprint arXiv:1904.08405 *(2019).*

[2] *Zhu, Alex Zihao, Nikolay Atanasov, and Kostas Daniilidis. "Event-based feature tracking with probabilistic data association."* 2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.*

[3] *Tedaldi, David, et al. "Feature detection and tracking with the dynamic and active-pixel vision sensor (DAVIS)."* 2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. IEEE, 2016.*

[4] *Rebecq, Henri, Timo Horstschaefer, and Davide Scaramuzza. "Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization." (2017): 1-8.*

[5] *Kueng, Beat, et al. "Low-latency visual odometry using event-based feature tracks."* 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016.*

[6] *Manderscheid, Jacques, et al. "Speed invariant time surface for learning to detect corner points with event-based cameras."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.*

[7] *Mueggler, Elias, Chiara Bartolozzi, and Davide Scaramuzza. "Fast event-based corner detection." (2017): 1-8.*

[8] *Mahendran, Siddharth, Haider Ali, and René Vidal. "3d pose regression using convolutional neural networks."* Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017.*

[9] *Mitrokhin, Anton, et al. "EV-IMO: Motion segmentation dataset and learning pipeline for event cameras."* arXiv preprint arXiv:1903.07520 *(2019).*

[10] *Newell, Alejandro, Kaiyu Yang, and Jia Deng. "Stacked hourglass networks for human pose estimation."* European conference on computer vision*. Springer, Cham, 2016.*

[11] *Berger, M-O. "Resolving occlusion in augmented reality: a contour based approach without 3D reconstruction."* Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 1997.*

[12] *Bulat, Adrian, and Georgios Tzimiropoulos. "How far are we from solving the 2d & 3d face alignment problem?(and a dataset of 230,000 3d facial landmarks)."* Proceedings of the IEEE International Conference on Computer Vision*. 2017.*

[13] *Barranco, Francisco, Cornelia Fermüller, and Yiannis Aloimonos. "Contour motion estimation for asynchronous event-driven cameras."* Proceedings of the IEEE *102.10 (2014): 1537-1556.*

[14] *Mustikovela, Siva Karthik, et al. "Self-Supervised Viewpoint Learning From Image Collections."* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.*

[15] *Honari, Sina, et al. "Improving landmark localization with semi-supervised learning."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.*

[16] *Maqueda, Ana I., et al. "Event-based vision meets deep learning on steering prediction for self-driving cars."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.*

[17] *Remelli, Edoardo, et al. "Lightweight Multi-View 3D Pose Estimation through Camera-Disentangled Representation."* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.*

[18] *Dardelet, Laurent, Sio-Hoi Ieng, and Ryad Benosman. "Event-based features selection and tracking from intertwined estimation of velocity and generative contours."* arXiv preprint arXiv:1811.07839 *(2018).*

[19] *Zhu, Alex Zihao, et al. "EventGAN: Leveraging Large Scale Image Datasets for Event Cameras."* arXiv preprint arXiv:1912.01584 *(2019).*

[20] *de Tournemire, Pierre, et al. "A large scale event-based detection dataset for automotive."* arXiv *(2020): arXiv-2001.*

[21] *Kogler, Jürgen, Christoph Sulzbachner, and Wilfried Kubinger. "Bio-inspired stereo vision system with silicon retina imagers."* International Conference on Computer Vision Systems*. Springer, Berlin, Heidelberg, 2009.*

[22] *Rebecq, Henri, et al. "High speed and high dynamic range video with an event*

camera." IEEE Transactions on Pattern Analysis and Machine Intelligence *(2019).*

[23] *Amir, Arnon, et al. "A low power, fully event-based gesture recognition system."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.*

[24] *Lichtsteiner, Patrick, Christoph Posch, and Tobi Delbruck. "A 128x128 120 dB 15 uS Latency Asynchronous Temporal Contrast Vision Sensor."* IEEE journal of solid-state circuits *43.2 (2008): 566-576.*

[25] *Delbruck, Tobi. "Frame-free dynamic digital vision."* Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*. 2008.*

[26] *Lagorce, Xavier, et al. "Hots: a hierarchy of event-based time-surfaces for pattern recognition."* IEEE transactions on pattern analysis and machine intelligence *39.7 (2016): 1346-1359.*

[27] *Gallego, Guillermo, and Davide Scaramuzza. "Accurate angular velocity estimation with an event camera."* IEEE Robotics and Automation Letters *2.2 (2017): 632-639.*

[28] *Gallego, Guillermo, Henri Rebecq, and Davide Scaramuzza. "A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.*

[29] *Piątkowska, Ewa, et al. "Spatiotemporal multiple persons tracking using dynamic vision sensor."* 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2012.*

[30] *Conradt, Jörg, et al. "A pencil balancing robot using a pair of AER dynamic vision sensors."* 2009 IEEE International Symposium on Circuits and Systems*. IEEE, 2009.*

[31] *Ni, Zhenjiang, et al. "Asynchronous event-based high speed vision for microparticle tracking."* Journal of microscopy *245.3 (2012): 236-244.*

[32] *Ni, Zhenjiang, et al. "Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics."* IEEE Transactions on Robotics *28.5 (2012): 1081-1089.*

[33] *Valeiras, David Reverter, et al. "An asynchronous neuromorphic event-driven visual part-based shape tracking."* IEEE transactions on neural networks and learning systems *26.12 (2015): 3045-3059.*

[34] *Fischler, Martin A., and Robert A. Elschlager. "The representation and matching of pictorial structures."* IEEE Transactions on computers *100.1 (1973): 67-92.*

[35] *Mueggler, Elias, Basil Huber, and Davide Scaramuzza. "Event-based, 6-DOF pose tracking for high-speed maneuvers."* 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014.*

[36] *Mueggler, Elias, Guillermo Gallego, and Davide Scaramuzza.* Continuous-time trajectory estimation for event-based vision sensors*. No. CONF. 2015.*

[37] *Gallego, Guillermo, et al. "Event-based, 6-DOF camera tracking from photometric depth maps."* IEEE transactions on pattern analysis and machine intelligence *40.10 (2017): 2402-2412.*

[38] *Bryner, Samuel, et al. "Event-based, direct camera tracking from a photometric 3D map using nonlinear optimization."* 2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.*

[39] *Zhu, Alex Zihao, et al. "EV-FlowNet: Self-supervised optical flow estimation for event-based cameras."* arXiv preprint arXiv:1802.06898 *(2018).*

[40] *Alonso, Inigo, and Ana C. Murillo. "Ev-SegNet: semantic segmentation for event-based cameras."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019.*

[41] *Ye, Chengxi, et al. "Unsupervised learning of dense optical flow, depth and egomotion from sparse event data."* arXiv preprint arXiv:1809.08625 *(2018).*

[42] *Schneiderman, Henry, and Takeo Kanade. "A statistical method for 3D object detection applied to faces and cars."* Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*. Vol. 1. IEEE, 2000.*

[43] *Pepik, Bojan, et al. "Teaching 3d geometry to deformable part models."* 2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012.*

[44] *Hejrati, Mohsen, and Deva Ramanan. "Analysis by synthesis: 3d object recognition by object reconstruction."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014.*

[45] *Tulsiani, Shubham, and Jitendra Malik. "Viewpoints and keypoints."* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.*

[46] *Su, Hao, et al. "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views."* Proceedings of the IEEE International Conference on Computer Vision*. 2015.*

[47] *Tompson, Jonathan J., et al. "Joint training of a convolutional network and a graphical model for human pose estimation."* Advances in neural information processing systems *27 (2014): 1799-1807.*

[48] *Pagano, Christopher C., and M. T. Turvey. "Eigenvectors of the inertia tensor and perceiving the orientation of a hand-held object by dynamic touch."* Perception & psychophysics *52.6 (1992): 617-624.*

[49] *Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation."* International Conference on Medical image computing and computer-assisted intervention*. Springer, Cham, 2015.*

[50] *Antholzer, Stephan, Markus Haltmeier, and Johannes Schwab. "Deep learning for photoacoustic tomography from sparse data."* Inverse problems in science and engineering *27.7 (2019): 987-1005.*

[51] *Schwab, Johannes, et al. "Real-time photoacoustic projection imaging using deep learning."* arXiv preprint arXiv:1801.06693 *(2018).*

[52] *Han, Yo Seob, Jaejun Yoo, and Jong Chul Ye. "Deep residual learning for compressed sensing CT reconstruction via persistent homology analysis."* arXiv preprint arXiv:1611.06391 *(2016).*

[53] *Hu, Jie, Li Shen, and Gang Sun. "Squeeze-and-excitation networks."* Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.*

[54] *Rundo, Leonardo, et al. "USE-Net: Incorporating Squeeze-and-Excitation blocks into U-Net for prostate zonal segmentation of multi-institutional MRI datasets."* Neurocomputing *365 (2019): 31-43.*

[55] *He, Kaiming, et al. "Deep residual learning for image recognition."* Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.*

[56] *Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization."* arXiv preprint arXiv:1412.6980 *(2014).*

[57] *Trajković, Miroslav, and Mark Hedley. "Fast corner detection."* Image and vision computing *16.2 (1998): 75-87.*

[58] *Derpanis, Konstantinos G. "The harris corner detector."* York University *2 (2004).*