2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Deep Learning of Spatio-Temporal Features with Geometric-based Moving Point Detection for Motion Segmentation

Tsung-Han Lin and Chieh-Chih Wang

***Abstract*— This paper introduces an approach to accomplish motion segmentation from a moving stereo camera based on deep learning. Previous work on moving object detection mostly use point features based on 3D geometric constraints. However, point features require good features, and are hard to detect or to be matched correctly in situations where objects have smooth textures. To alleviate this problem, learning high-level spatio-temporal features unsupervisedly from raw image data based on Reconstruction Independent Component Analysis (RICA) autoencoders is proposed. Despite the power of the new spatio-temporal features, these features cannot not learn and be used to interpret 3D geometry of dynamic scenes, which is critical for moving object detection from moving cameras. As detected moving points based on 3D geometric constraints still contain valuable information of 3D scene as well as the camera ego-motion, we propose a framework that incorporates both the detected moving point results and the learned spatio-temporal features as inputs to Recursive Neural Networks (RNN) that performs motion segmentation. Both features effectively complement each other. The proposed approach is demonstrated with real-world stereo video data that contains multiple moving objects, and has achieved 26% better detection rate over the existing 3D geometric-based moving points detector.**

I. INTRODUCTION

Detecting moving objects such as cars and pedestrians is important for safe navigation in dynamic environments. A number of approaches detect hand-engineered point features such as corners and SIFT, then make use of 3D geometric constraints to calculate camera's ego-motion with additional post processing like clustering and tracking in order to detect moving objects [1][2]. These approaches require good corner-like features presented in the scene, and also correct association of feature points in consecutive frames. However, these requirements can be challenging to meet in situations where no such features are present such as smooth texture of cars or pedestrian cloth, or lines on the roads. One may resort to tracking for predicting moving object trajectories, but this would cause delay in detection after the object appearance, and object motion models have to be defined [1]. An alternative approach is to use points that could be tracked as anchor points to reason about nontrackable pixels [3][4].

T.-H. Lin is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan hanklin3@robotics.csie.ntu.edu.tw

C.-C. Wang is with the Department of Computer Science and Information Engineering, and the Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan bobwang@ntu.edu.tw

This work was supported in part by Taiwan National Science Council, National Taiwan University and Intel under grants NSC100-2221-E-002-238-MY2, NSC100-2911-I-002-001 and NTU102R7501.

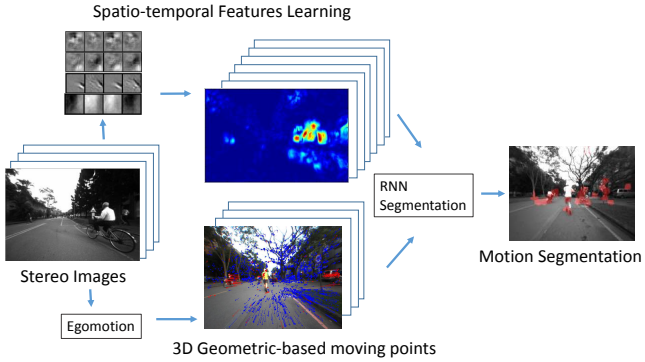


Fig. 1: The proposed deep learning framework for motion segmentation from a moving stereo camera.

In this work, we focus on learning more robust high-level motion features based on deep learning to improve moving object detection at the front-end before any post processing. Our work uses a network based on Reconstruction Independent Component Analysis (RICA) autoencoder [5][6] to unsupervisedly learn high-level spatio-temporal features. This autoencoder has shown to be able to learn receptive fields similar to the visual cortex of our brains [5], and had successes in learning high-level features for object recognition tasks [6]. In comparison to point features, the learned spatio-temporal features can oversee a larger area and represent interesting and meaningful transformations, alleviating the issue of not able to extract features from featureless spots in small areas of images and reducing the chances of mismatches. However, these learned spatio-temporal features still can not learn the information of the 3D geometry of the scene as well as the ego-motion of the stereo camera. Since correctly associated moving points can still provide valuable 3D motion information, we propose a framework that use deep learning to combine both data to learn a hierarchical representation for motion segmentation. The segmentation itself is based on Recursive Neural Network (RNN) [7], which recursively combines small superpixel segments [8] in images into bigger segments. In this way, the RNN learns higher level feature representation to describe motions of bigger object parts instead of only individual small segments. The proposed deep learning framework using both unsupervisedly learned high-level spatio-temporal motion features and 3D geometric cues from moving point detection to accomplish motion segmentation is shown in Fig. 1. The proposed system is trained on real-world data, using only two

consecutive frames from a moving stereo camera in a scene with multiple moving objects. The learned spatio-temporal features can detect moving objects more densely than using only point features alone. In addition, by combining moving points, the ego-motion information of the camera is also considered. The overall motion segmentation results shows that the proposed approach outperforms using either one of the single approach only.

II. Related Work

Recent work in deep learning have unsupervisedly learned high-level features from raw still images, and have achieved promising results for tasks such as action recognition and object recognition [6][9][10][11][12]. In [11], the same network structure is used for learning motion features for activity recognition. However, the second layer features in their work are more of complex corner shape features, whereas we have learned more interesting second layer transformations. The work of [12] is also based on RICA autoencoder for learning spatio-temporal features. Our work is close to this work in which the features are learned by stacking autoencoders layer-by-layer, and have achieved high-level feature representation that encodes complex local invariances, such as invariance to translation, rotation and local non-affine changes. However, [12] performs manual object tracking and adds a temporal slowness constraint in addition to RICA for learning invariance of features that can be used for object recognition. In this work, we propose to learn motion features for motion segmentation. Therefore without using tracking or temporal slowness constraints, but with only raw stereo images, the proposed approach is able to learn features that are more sensitive to motions and can describe fast moving objects.

A number of works that learn spatio-temporal features are used for object or action recognition on the whole image or video [10][13][11]. Our work has added complexity of segmenting out individual moving objects. Recent developments in segmentation or scene parsing using deep learning have shown promising results for object segmentation in static images using Convolutional Neural Network (CNN) [14][15][16] or Recursive Neural Network (RNN) [7]. The RNN in this work follows the similar paradigm as [7]. However, hand-engineered features were used in [7], whereas we unsupervisedly learned the features. Although this step would increase the load of training, but it has been shown that using learned features would outperform hand-engineered features in scene parsing task [14]. In addition, not just spatial features are considered but spatio-temporal features are considered. The proposed system achieves motion and object segmentation based on both motions and appearances.

III. Unsupervised Spatio-Temporal Feature Learning

This section describes the learning architecture for unsupervised spatio-temporal feature learning. Our inputs are two consecutive stereo images, total 4 frames, which are set as image patches with 4 channels each for autoencoder training.

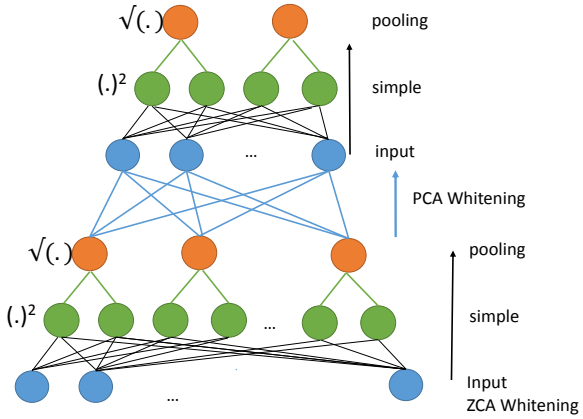


Fig. 2: A simplified architecture of 2 layers stacked network.

The autoencoder based on RICA optimization is used to learn two hierarchical layers. The feed-forward manner of each layer makes it suitable to form deep architecture.

A. Architecture

The network architecture is a variant of Independent Component Analysis (ICA) [17]. Two stacked layers of the network are shown in Fig. 2. Each single layer contains the inputs and two sublayers, where x is the inputs, W is the simple units in the first sublayer, and H is the pooling units in the second sublayer (1). H is the units that pools over a small neighborhood (size of 2) of adjacent simple units W. The element-wise activation function of simple units W is squared $(.)^2$, and for pooling units H is square root $\sqrt{(.)}$.

$$\sum_{i=1}^{N} \sqrt{H(Wx_i)^2} \tag{1}$$

B. Learning Module

The W bases of the network is optimized by the RICA autoencoder and sparsity constraint as shown in (2). The RICA penalty is the first item of (2). In autoencoders, outputs are set as the same as inputs, and it learn bases that can reconstruct the outputs to the inputs. Additional constraint of RICA ensures that the learned W bases are orthogonal and independent of each other, which helps to generalize by learning a diverse number of bases. The encoding step of the autoencoder is Wx, and the decoding step is W^TWx. During the encoding and decoding steps, the W weights are tied (W^TW) to ensure the weights are orthogonal. The benefits of the RICA is that now W can still be optimized even if it is overcomplete, and that optimization still works well with both raw and whitened data (a preprocessing step that decorrelates the input data) [17].

$$\arg\min_{W} \sum_{i=1}^{N} ||x_i - W^TWx_i||_2^2 + \lambda \sum_{i=1}^{N} ||\sqrt{H(Wx_i)^2}||_1 \tag{2}$$

The second item of (2) is the sparsity constraint. The sparsity forces the autoencoder to learn interesting structures

of the data, by enforcing as few output neurons are activated as possible. It is shown that sparsity helps learning bases similar to the receptive fields [9] of our brain, such as edges or corners at the lowest level, and curves, counters, or even objects at higher level. λ is the weight for controlling the importance of sparsity penalty, set to 0.1 in our experiments.

C. Stacked Architecture

Learning bases from the large images can be difficult and computationally expensive. Therefore, deep network are built containing two layers to scale up to large images as shown in Fig 2. The key ideas are as follows. First, around 30,000 small patches (16x16x4) are randomly selected and preprocessed with ZCA whitening [17], which will be inputs to the autoencoder. The autoencoder learns 128 features (pooled from 256 linear bases W, each base is 16x16x4) for the first layer. Next, features are trained for the second layer on larger patches (32x32x4) which are then convoluted with the 128 features from the first layer with a stride of 2. Each patch ends up with 128 response maps (9x9x1 each). PCA is then used to reduce the response maps (9x9x128) of each patch to a small dimension of 300 before learning the second layer. The second layer of autoencoder then learns 150 features (pooled from 300 linear bases, each base is 300x1). ZCA whitening and PCA are used in the process to decorrelate the data and to speed up training.

Regarding extracting features from large stereo images (240x320x4), stereo images are first convoluted with the first layer features to obtain response maps. Note each stereo patch is preprocessed with whitening before convolution. Each response map is then reduced with PCA, and then convoluted with the second layer features. The final response maps are resized to the same size as stereo image (240x320) in order to match the labels for RNN segmentation training.

D. Learned Features Visualization and Analysis

This section discusses different motion transformations learned by the features. The autoencoder of the fist layer learns low-level features, mainly edges as shown in Fig. 3 Each feature is shown with the 4 channels lay out as 2D patch for better visualization. The channels are in the following order: the right image from the previous time step, the right image of the current time step, the left image of the previous time step, and the left image of the current time step. Fig. 3 shows variety of orientations the features are sensitive to. This is the result of the RICA orthogonal constraint that forces the learned features to be as diverse as possible.

The main transformation learned in the first layer is translation. In Fig. 4, the features are in increasing velocities from (a) to (d). One property can be seen is that the edges are increasing in thickness. From our observation, thicker edges detect closer objects that move at faster velocities. This is because the textures of fast moving objects can appear to be more *blurry*, and also that the thicker edges can cover larger area. Motion is also seen as *moving edges* through time, which is best visualized in Fig. 4 (c). The first 2 channels of the feature encode the right image at the previous time and current time, and one can see that the edge is moving toward top-right direction across time.

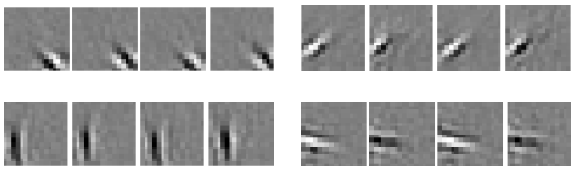


Fig. 3: First layer autoencoder features in different orientations. Four channels are lay out as 2D patch for better visualization.

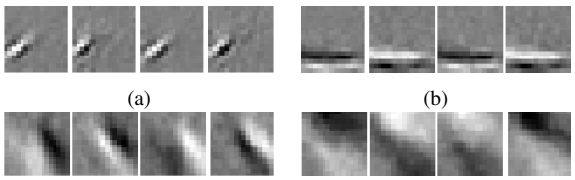


Fig. 4: First layer autoencoder features with increasing velocity from (a) to (d).

The exact positions of the edges in the 4 channels of the learned features encode the spatio-temporal constraints. The relative positions between the edges of the left and right images can provide some sense of *depth*, in which the edges will be farther apart for closer objects, and closer together for far away objects. Likewise, the relative positions between the edges across two time steps provide the the motion property.

At the lowest level, the spatio-temporal features are detecting patterns like moving edges. This means that these features would be overviewing a larger area than corner-like features. This characteristic is critical to detect objects which have smooth textures. Higher level features can oversee even larger areas. For Layer 2 features to learn higher level of transformations, Layer 2 units should see continuous switching between Layer 1 features. More specifically, when an object moves over time, in the output response maps, there should be feature responses from Layer 1 of the same object at time 1 and time 2. The second layer should then associate the features of the same object across time and thus learning the motion patterns. Features of Layer 2 cannot be visualized directly as patches because these features are learned after performing PCA dimension reduction on the response maps from Layer 1. Therefore, Layer 2 features are visualized by seeing how it response to different outputted response maps of Layer 1. First, different possible responses of Layer 1 features are generated by changing each angle value Θ from 0 to 360 degrees and setting $H = cos(\Theta)\sqrt{w_1^2 + w_2^2}$, where H is the pooled unit response, and w the simple units. Response maps from Layer 1 is then convoluted with Layer 2 features, forming a 32x32x4 patch, which can then be visualized in Fig. 5. Interesting transformations are observed, such as rotations or other non-affine transformations.

By pooling, the features have some invariances to these

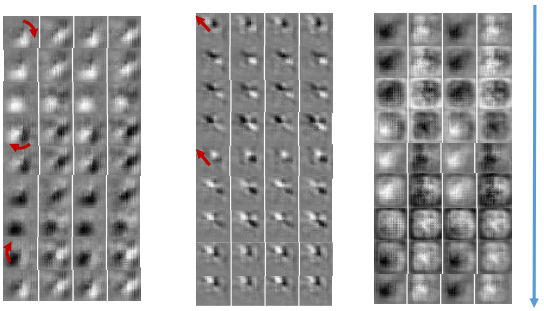


Fig. 5: Visualization of layer 2 feature responses after convolution on layer 1. Each row shows the 4 channels of the receptor fields, and downward means increasing in time. Non-trivial transformations can be seen, like rotation (left), and non-affine transformation (right).

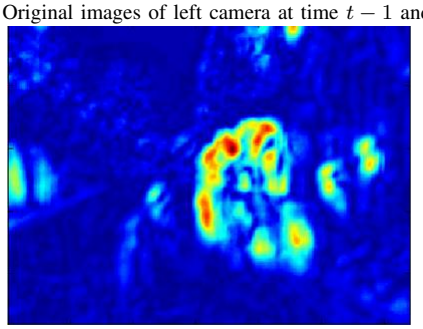
transformations, that is there are some *local* invariances to these transformations. However, the features of the pooling units are still sensitive to motions as also observed in [11]. The pooling simply combines similar transformations described by simple units W. For example, at Layer 1, moving edges in similar orientations and velocities are combined. This helps the pooling units to have some local translation invariance (not too sensitive to the exact pixel location of the motion), but sensitive to that particular motion. Likewise for Layer 2, transformations like rotations in the same direction are combined into one.

Currently huge dataset for the evaluation is not used due to the bottleneck of RNN supervised training. However, because the spatio-temporal features can be trained unsupervisedly, larger data are trained and results are found to be similar, i.e. the features cover similar orientations, velocities and transformations for both first and second layers. This is due to the fact that RICA orthogonal constraint helps to generalize to unseen data by not repeating the learned bases. Our traffic scene dataset is sufficient now for RNN training because the motions needed to be detected are simple over a short period. These include object movements such as left and right, forward and backward, up and down at different orientations and velocities, or small rotations when objects are turning. Scaling up RNN is for future work and is beyond the scope of this paper.

E. Discussion

The spatio-temporal features itself do have some limitations. Although the features can detect motions in 2D image space (such as moving edges), one of the issues is that it could not learn differences of motions in 3D geometry. This would be problematic for a moving camera scenario. In situation of a forward moving camera, close-by static objects such as street signs will appear to have big motion difference from the background faraway, and would be misclassified as moving objects by the RNN using the learned spatio-temporal features.

(a) Original images of left camera at time $t-1$ and t



(b) Feature response map

Fig. 6: Spatio-temporal feature response map of layer two.

Another issue is that the feature response locations in the response maps are imprecise as the feature responses do not map to the exact pixel locations of the objects in the original images. The responses are somewhere between the objects at two different time steps. The main reason is that when responses of the same object from a lower layer are combined by higher layer features, the resulting responses are activated somewhere between the objects at two time steps as shown in Fig. 6. One can see that the motion responses of the bicycles are somewhere between the bicycles at two different time steps.

Note that the discussed issues of the learned features do not affect much on object segmentation [14][7] and activity recognition [11]. In object segmentation, features can map close to the object location in images because objects do not move. For activity recognition, the exact feature locations are not a concern because the classification is on the whole videos. Next section will describe how to design the segment features to deal with these issues for RNN segmentation.

IV. Segment Features

Instead of using point features in geometric-based motion segmentation approaches, we propose segment features as the inputs to the RNN motion segmentation module. In this work, two kinds of segment features are used: the learned spatio-temporal feature segments and the 3D geometric-based moving point segments.

A. Spatio-temporal Segment Features

The segment features of spatio-temporal features are designed to deal with the imprecision of feature responses. Since the final response maps are smaller than input images due to convolution and pooling, the response maps are upsampled to match the label image size. Labels are not downsampled in order to not miss small moving objects.

The upsampled response maps are then segmented with the superpixel approach [8], around 15x15 pixels per segment. The segment size are designed to be big enough by containing the *in between* feature responses in order to map it back to its corresponding object, and from experiments segment size cannot be too small. Each segment is cropped out and performed with mean pooling of 4x5. With mean pooling, data dimension is reduced while maintaining the spatial positions of feature responses in images. Larger pooling size can represent segment data better, however large pooling size will increase RNN training load. Therefore, size 5 is chosen for the columns to better capture frequent occurrences of left and right object movements, and size for up and down movements are smaller due to infrequent occurrences.

B. 3D Geometric-based Moving Point Segments

In order to compensate the issues of the spatio-temporal features, 3D geometric-based moving points are also used in the system as additional features. The point feature matcher (optical flow) and camera ego-motion estimation are based on libviso2 [18], which achieves accurate camera 6DOF ego-motion estimation with 1.3% position error of the traveled distance. Moving points are determined by checking the association of measured 2D points in stereo images over two time steps, whether the association is consistent based on the camera ego-motion. Given the associated 2D points $(u^1, v^1, disparity^1), (u^2, v^2, disparity^2)$, project the points in the first time step into 3D XYZ^1 (3).

$$\begin{aligned} X^t &= \frac{(u_{n,left} - c_{u,left}) * baseline}{d_n^t} \\ Y^t &= \frac{(v_{n,left} - c_{v,left}) * baseline}{d_n^t} \\ Z^t &= \frac{baseline * focal}{d_n^t} \end{aligned} \tag{3}$$

where u, v are the 2D points, c_u, c_v is the camera center, $baseline$ is the stereo baseline, $focal$ is the camera focal length, and d is the disparity.

Ego-motion is used to obtain XYZ^2 of the next time step, then it is projected back to 2D $(u^{proj}, v^{proj}, disparity^{proj})$ and its difference with the measured 2D point $(u^2, v^2, disparity^2)$ is checked. If the difference is above a certain threshold then it is determined as a moving point. Tighter threshold is used for points near the center of images (where motions are small) to help detect objects with small motions, and looser threshold near the outside to reduce noise. Finally all moving points are clustered into groups with 3D Euclidean clustering, which helps to reduce noise by removing groups that contain less than 4 points. An example of sparse moving points detection before clustering is shown in Fig. 8 (a).

The 3D geometric-based moving points need to be converted to segment features as input to the RNN motion segmentation module. Besides the 4 image channels of the stereo images, 5 additional image channels are added to represent the moving point features. These 5 individual channels are the moving points, static points, differences of 2D u, v, and $disparity$. Superpixels, cropping, and pooling are then performed. Since the detected moving points are sparse, each channel contains many missing data, which makes mean pooling unsuitable. With mean pooling, the missing data, which are set to 0, will reduce the mean response score of each segment, and will be an inaccurate feature representation. Therefore, max pooling is proposed and used, which only looks at maximum response values of a segment. In this way feature responses are accurately represented and are not affected by missing data.

V. Recursive Neural Network Motion Segmentation

This section discusses the Recursive Neural Network (RNN) algorithm used for motion segmentation. We follow the same recursive framework in [7]. During training, the RNN first is initialized by building one layer of representation, by combining only the good pairs (segments of same class) and bad pairs (segments of different classes). Next a *greedy structure predicting* algorithm is used to find one possible correct tree, which is defined as segments of the same class are combined before one from different class. Many possible correct trees exist because the order of of segment merging does not matter as long as they are in the same class. The greedy algorithm only seeks for the highest scoring tree. To ensure that the highest scoring tree is in the set of correct trees, the max-margin estimation framework [7] is used . Furthermore, the highest scoring (correct) tree y_i is set to be larger than a proposed (incorrect) tree $\hat{y}$ up to a margin defined by the loss $\triangle$.

$$s(RNN(\theta, x_i, y_i)) \geq s(RNN(\theta, x_i, \hat{y})) + \triangle(x_i, l_i, \hat{y}) \tag{4}$$

Thus the cost function J that needs to be minimized is:

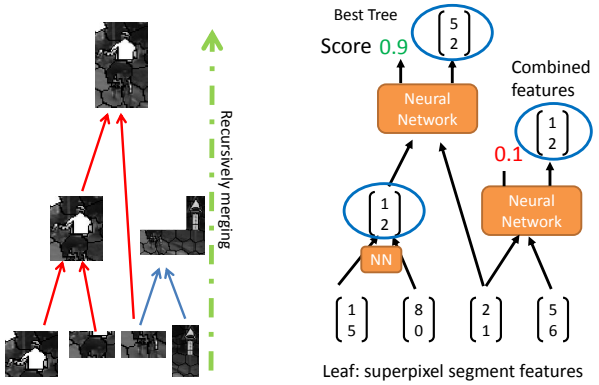
$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} r_i(\theta) + \frac{\lambda}{2} ||\theta||^2, \tag{5}$$

where

$$\begin{aligned} r_i(\theta) = &\max_{\hat{y} \in T(x)} (s(RNN(\theta, x_i, \hat{y})) + \triangle(x_i, l_i, \hat{y})) \\ &- \max_{y_i \in Y(x_i, l_i)} (s(RNN(\theta, x_i, y_i))) \end{aligned}$$

and x_i is an instance of training data from $i = 1, ..., n$, l_i is the label of training instance x_i, y_i is the correct tree, and $\hat{y}$ is a proposed incorrect tree. The right side of the cost function is a regularization term that decreases the magnitude of the weights to prevent overfitting. The loss $\triangle$ is a penalty that increases if segments of different classes are merged before segments of the same class.

Minimizing the cost J maximizes the correct tree's score and minimizes (up to a margin) the score of the highest scoring but incorrect tree. RNN computes a score for the best segmentation tree. Segment features are first mapped to *semantic* features that RNN operates on. Let x_i be the segment features, then for each segment, map x_i to semantic features a_i that RNN can recursively reads as inputs:



(a) RNN merging of image segments.

(b) RNN mechanism of feature merging.

Fig. 7: Recursive Neural Network learns the best tree that merges segments of the same class before merging other classes.

$$a_i = f(W^{sem} x_i + b^{sem}) \tag{6}$$

where W^{sem} is the weights, b^{sem} is the bias, and f is the sigmoid function $f(x) = 1/(1+e^{-x})$. Now the input to RNN will be the semantic features or activation a_i, and an adjacency matrix A. RNN then recursively merges two adjacent segments i and j by concatenating the two segments:

$$a_{(i,j)} = f(W[a_i; a_j] + b) \tag{7}$$

where the new features or activation of the merged parent segment is $a_{(i,j)}$, and W and b is the weights and bias of RNN to be learned. A local score s can be computed for the merged segments with a simple inner product with W^{score}:

$$s_{(i,j)} = f(W^{score} a_{(i,j)}) \tag{8}$$

Fig. 7 shows a result of the RNN moving object segmentation module.

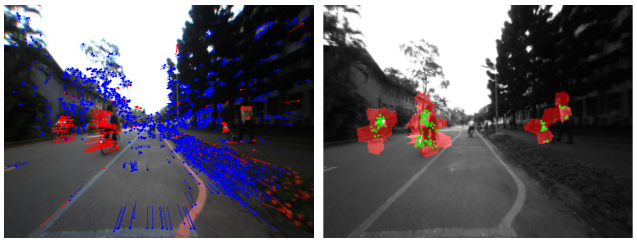
VI. Experiments

A. Dataset

Our dataset contains 400 stereo images recorded in front of the campus cafeteria. The camera is constantly moving forward, and containing multiple moving objects such as cars, pedestrians and bicycles, which makes motion segmentation challenging. All moving objects are labeled by drawing contours encompassing the objects. Bounding boxes are not used because we try to achieve fine-cut and dense segmentation. For RNN traning, we took 200 images for training, and 200 for evaluation. All moving objects were labeled with class 1, and static objects with class 0.

B. Evaluation

The performance is evaluated by following [19], in which the motion segmentation has different object motions separated into different clusters. Only two clusters were consider



(a) Sparse moving points detection (red)

(b) Dense moving points detection (red).

Fig. 8: (a) Sparse moving points detection and (b) Dense moving points (red) after clustering. Green points are the original sparse moving points

in our system, moving or static, therefore the detected moving objects are assumed to be in separate clusters when evaluating *average accuracy*. Results of three different methods are compared: 3D geometric-based moving points detector, the spatio-temporal motion features plus RNN, and then the final combination of the two mentioned approach.

Due to the fact that 3D geometric-based moving points detection is sparse and the RNN is a dense pixel-wise detection, these methods can not be compared directly. Therefore for each sparse moving point that is detected, we count all pixels of the superpixel segment the point lie on as moving, as shown in Fig. 8. This will be categorized as the 'Moving Pts (dense)' as depicted in Table I. Then we can directly compare the performance of dense 3D geometric-based moving point results to the spatio-temporal motion features + RNN.

Density shows the percentage of the number of pixels that are detected as either moving or static over all image pixels, for all images. Note that higher density is much more likely to have detection errors, resulting in higher false positive. Next is accuracy. Note that our groundtruth labels use fine boundaries, and we are classifying pixels of 'rough' superpixel segments. Therefore even if the all segment are classified correctly, the accuracy would not be 100% because the segments are bigger in order to cover the objects, and the *spilled over* pixels would be treated as errors. The **overall accuracy** is the number of correctly detected moving pixels out of all moving pixels, for all images. Correctness is defined as detected moving pixels overlapping moving pixels of the groudtruth labels. The **average accuracy** is the number of correctly detected moving pixels out of all moving pixels, averaging over each object. For the overall accuracy, a large object that is correctly detected can contribute more to the overall accuracy. Therefore, the average accuracy focuses on smaller objects by looking at the correctness for each individual object. Average accuracy would drop substantiatlly if smaller objects are missed. The **false positive** is the incorrectly detected moving pixels out of all moving pixels. This test the precision of detection. Finally, the **extracted objects** shows how many objects are detected in the whole video. A moving object is counted as detected if it contains at least one pixel that is classified as moving. One object in each image gives one count, in other words the same object

TABLE I: Performance of our system.

	Density	Overall Accuracy	Average Accuracy	Extr. Objs	False Positive
Moving Pts (dense)	100 %	35.0 %	30.7 %	476	59.2 %
motion features +RNN	100 %	58.1 %	41.5 %	680	61.8 %
motion features +RNN +Moving Pts	100 %	**61.5**%	**45.8** %	**708**	**55.3** %

appearing across consecutive images are counted separately. In our experiments there are 946 objects in total.

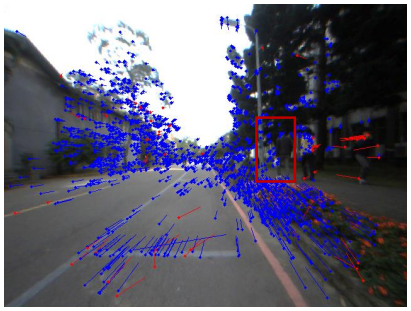
Our results are shown in Table I. All results are the detection results over 2 consecutive time steps (4 frames). This would makes it harder to detect far away objects with small motions, and because the groundtruth labels contain objects far away, the accuracy is not that high for all methods. One result can be seen is that the spatio-temporal motion features were able to extract more objects than Moving Pts (dense). One reason is that points are hard to detect and associate correctly for objects far away with small motions, because the textures appear to be dark and smooth. Our motion features were able to extract features like *moving edges* that help to detect smaller objects. An example of a missed pedestrian is shown in Fig. 9. Another reason is that when new objects just enter the scene or was occluded, the point features need to wait for objects to appear and then match it at the next frame. This causes a small delay in detection. Our motion features was able to detect incoming objects at first appearance. See a result sample in Fig. 10.

The average accuracy shows the detection for each individual object. Comparing Moving Pts (dense) detection to the motion features + RNN, the motion features + RNN achieves higher detection rate for each object. This shows that the motion features were better at picking up more challenging moving objects.

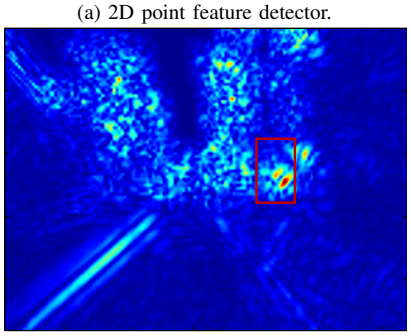
The sparse moving point detector cannot be compared directly. However, if we define accuracy as the number of detected sparse moving points that are correctly classified (and ignored all unclassified pixels), the accuracy is 85.4%, with low density of 13.1%, 376 extracted objects and 14.5% false positive. This results indicate sparse moving point detector has high precision but low detection rate. Therefore, by adding the sparse moving point features along with the spatio-temporal features to the RNN segmentation, the overall results achieve higher accuracy and detection rate with a lower false positive rate. The combined final results can be seen in Fig. 11.

C. Training and Prediction Time

The system is implemented in Matlab running on a 4th generation Intel i7 CPU with Nvidia GTX 670 (for convolution computation only). The prediction time for one 240x320x4 stereo image is shown in Table II. For improvements, the parallel nature of convolution and pooling can be implemented on chips demonstrated in [20]. Their 3-layers

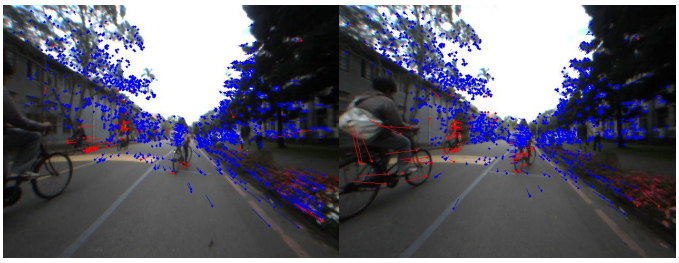


(a) 2D point feature detector.



(b) Spatio-temporal feature.

Fig. 9: (b) Spatio-temporal features detect moving edges where it was missed by (a) 2D point detector. The red box shows the missed pedestrian whose cloth has a smooth texture.



(a) 2D point feature detector.

(b) Spatio-temporal feature.

Fig. 10: Incoming bicycle from the left of image. (a) 2D point features need two frames to match in order to detect moving points, so it need to wait for objects to appear first. (b) Spatio-temporal features detect the motion at the first appearance.

(a) 2D point feature detector.

Fig. 11: Deep learning motion features + geometric-based moving points + RNN final results. Detection of various object motions in clutter scenes.

TABLE II: Prediction Time for one stereo image.

	Time (secs)
Layer 1 Convolution + Pooling	1.9
Layer 2 Convolution + Pooling	9.8
Segment Pooling	1.8
Moving Points Detection	0.8
Moving Points Detection Segment Pooling	0.09
Total	14.4

ConvNet with 500x500 input image, 920 bases of size 16x16 gives about the same number of connections as our network. The proposed chip implementation of the ConvNet reports real-time performance of 10 FPS on FPGA and 20 FPS on ASIC. This would suggest the convolution bottleneck of the proposed system can be improved for feasible real-time performance. For training, the unsupervised feature learning module takes around 1.5 hours for learning 150 features in layer 1 and 300 features in layer 2 over 30,000 patches. The RNN takes about 1.5 days to train over 200 stereo images.

VII. Conclusions and Future Work

In this paper, high-level spatio-temporal features are learned to extract motion features from challenging moving objects. Our experiments show that the proposed features are able to extract more moving objects than point features. However, moving points with 3D geometric constraints still provide valuable 3D informaiton and is more accurate given that data association is correct. Therefore, in this work a new framework is proposed that combines both features for an overall improvement. Although two consecutive stereo frames were used to demonstrate the potential of the spatio-temporal features, the overall accuracy could be further improved since only 2 time steps were considered in this paper. RNN currently is the training bottoleneck. For future work, we will investigate the improvement of learning over longer time steps.

References

[1] P. Lenz, J. Ziegler, A. Geiger, and M. Roser, "Sparse scene flow segmentation for moving object detection in urban environments," in *Intelligent Vehicles Symposium (IV)*, 2011, pp. 926–932.

[2] A. Kundu, K. Krishna, and J. Sivaswamy, "Moving object detection by multi-view geometric techniques from a single camera mounted robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009, pp. 4306–4312.

[3] P. Ochs and T. Brox, "Object segmentation in video: A hierarchical variational approach for turning point trajectories into dense regions," in *IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 1583–1590.

[4] K. Fragkiadaki and J. Shi, "Detection free tracking: Exploiting motion and topology for segmenting and tracking under entanglement," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 2073–2080.

[5] J. H. van Hateren and D. L. Ruderman, "Independent component analysis of natural image sequences yields spatiotemporal filters similar to simple cells in primary visual cortex," in *Royal Society: Biological Sciences*, 1998.

[6] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng, "Building high-level features using large scale unsupervised learning," in *International Conference in Machine Learning (ICML)*, 2012.

[7] R. Socher, C. C. Lin, A. Y. Ng, and C. D. Manning, "Parsing natural scenes and natural language with recursive neural networks," in *26th International Conference on Machine Learning (ICML)*, 2011.

[8] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels," *École Polytechnique Fédéral de Lausssanne (EPFL), Tech. Rep*, vol. 149300, 2010.

[9] A. Coates and A. Ng, "Selecting receptive fields in deep networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2011, pp. 2528–2536.

[10] R. Memisevic and G. Hinton, "Unsupervised learning of image transformations," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007, pp. 1–8.

[11] Q. Le, W. Zou, S. Yeung, and A. Ng, "Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[12] W. Y. Zou, S. Zhu, A. Y. Ng, and K. Yu, "Deep learning of invariant features via simulated fixations in video," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[13] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler, "Convolutional learning of spatio-temporal features," in *European Conference on Computer Vision (ECCV)*, 2010, pp. 140–153.

[14] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Scene parsing with multiscale feature learning, purity trees, and optimal covers," in *International Conference on Machine Learning (ICML)*, 2012.

[15] D. Grangier, L. Bottou, and R. Collobert, "Deep convolutional networks for scene parsing," in *ICML Deep Learning Workshop*, vol. 3, 2009.

[16] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano, "Toward automatic phenotyping of developing embryos from videos," *IEEE Transactions on Image Processing*, vol. 14, no. 9, pp. 1360–1371, 2005.

[17] Q. V. Le, A. Karpenko, J. Ngiam, and A. Ng, "Ica with reconstruction cost for efficient overcomplete feature learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2011, pp. 1017–1025.

[18] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *Intelligent Vehicles Symposium (IV)*, 2011.

[19] T. Brox and J. Malik, "Object segmentation by long term analysis of point trajectories," in *European Conference on Computer Vision (ECCV)*, 2010, pp. 282–295.

[20] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 257–260.