Proceedings of Seminar and Project

3D Computer Vision and Augmented Reality

Summer Semester 2019

Oliver Wasenmüller and Prof. Didier Stricker
Department Augmented Vision
University of Kaiserslautern and DFKI GmbH
Introduction

The seminar and project 3D Computer Vision and Augmented Reality (NF-73-71-S-7, INF-73-81-L-7) are continuative courses based on and applying the knowledge taught in the lectures 3D Computer Vision (INF-73-51-V-7) and 2D Image Processing (INF-73-53-V-6). The goal of the project is to research, design, implement and evaluate algorithms and methods for tackling computer vision problems. The seminar is more theoretical. Its educational objective is to train the ability to become acquainted with a specific research topic, review scientific articles and give a comprehensive presentation supported by media.

In the summer semester 2019, seven projects, nine seminars and one guided research were completed. The results are documented in these proceedings.

Organisers and supervisors

The courses are organised by the Department Augmented Vision (http://ags.cs.uni-kl.de), more specifically by:

Oliver Wasenmüller
Prof. Dr. Didier Stricker

In the summer semester 2019, the projects were supervised by the following department members:

Yuriy Anisimov
Michael Fürst
Manthan Pancholi
Markus Käechele
René Schuster
Hammad Butt
Ramy Battawry
Mathias Musahl
Stephan Krauß
Jameel Malik
Maria Sanchez
Tewodros Habtegebrial

September 2019
Survey on Depth Post-processing Methods

Ibrahim Abdelaziz and Yuriy Anisimov

\[1\] abdelazi@rhrk.uni-kl.de
\[2\] Yuriy.Anisimov@dfki.de

Abstract. Image-based depth reconstruction methods that estimate the depth information of a scene tend to produce noisy and inaccurate results. To overcome this issue, such algorithms are commonly followed by a post-processing stage to reduce noise and improve the quality of estimated depth maps. This paper provides overview and description of various state-of-the-art depth post-processing methods.

Keywords: Depth recovery, Depth post-processing, Depth upsampling, non-linear filtering, Median filter

1 Introduction

Depth reconstruction is an important problem in the area of computer vision that is used in a wide range of applications such as autonomous driving, 3D reconstruction and pose estimation [8].

A vision system, comprising two cameras placed horizontally, is used to capture a pair of stereo images at the same time. A disparity map, which contains the scene inverse depth information, can then be determined by measuring the amount of horizontal displacement between corresponding pixels in the two frames [8][14].

Stereo correspondence algorithms are classified into three main categories. Local (window-based) methods, which estimate disparity by considering local intensity information inside a finite window centered on a target pixel. Secondly, global methods that assign disparity by minimizing a global energy function consists of data and smoothness terms. Lastly, semi-global methods exploit the low computational complexity of local methods and the high accuracy of global methods by approximating a 2-D global smoothness constraint using local 1-D constraints [9]. However, these algorithms face common problems that cause disparity maps to contain noise and inaccurate values. Some of these issues are illumination variations and sensor noise, matching errors in textureless, repetitive patterns, and occluded regions. A stereo vision system can be extended to use more than two images as in multi-view stereo systems [10].

Another way to acquire depth maps is using depth sensor devices, such as time-of-flight cameras which finds the distance to each point in the scene by measuring round trip time of an infrared (IR) light signal emitted from the device. Structured light sensors is another type of depth devices which projects a known pattern of dots using an IR projector and measures distance by analyzing the change in the projected pattern captured using an IR camera [5]. Nonetheless, depth maps obtained by those devices often contain noise, misalignment with color images, and holes near object boundaries which lead to accuracy degradation of methods that use them as input [16].

Different post-processing techniques are suggested to reduce noise and enhance the quality of depth maps. In this paper, we describe some state-of-the-art methods for depth map post-processing. Those methods span various techniques, such as median filter methods that apply special types of local filters to smooth the disparity map and remove discontinuities, color and depth consistency based methods that remove noise and holes near object boundaries from Kinect depth maps. The other presented methods include depth upsampling which increase resolution of depth sensor devices low-resolution maps, and image segmentation based method that applies image segmentation to disparity map to remove noise inside disparity map segments.
2 Weighted Median Filter Post-Processing

The Weighted Median Filter (WMF) is a local edge-preserving filter that is used to reduce the blurring effect for depth map refinement [6]. The filter works by updating each pixel by the median of a weighted window of size $r$, centered at a target pixel. In this section, we summarize a method for fast WMF computation and selected state-of-the-art methods that use WMFs for depth map post-processing.

2.1 100+ Times Faster Weighted Median Filter (WMF)

In this paper Zhang et al. [8] present three schemes to reduce the computational complexity of finding the weighted median of a WMF from $O(r^2)$ to $O(r)$, where $r$ is the local filter size.

In unweighted median filter values are range-limited and adjacent filters share common values; thus, a counting histogram can be used to find the median value in linear time. However, in weighted median filter the weights of each pixel can vary according to feature distance in use and as the filter is moved all weights change values according to the new center pixel. To overcome this problem, the authors introduce a joint-histogram, a 2D count based histogram that stores the pixel counts according to their feature values. The joint-histogram is divided into $n \times m$ regions, as the local window consists of unique $n$ pixel values and $m$ feature values and each region contains the count of pixels with the corresponding values. After filling the histogram values the median can be found in linear time by summing all the weights in the joint-histogram and then have a second pass to find the point at which the sum is equal half of the total sum. Finally as the local window is shifted the joint-histogram is updated by reducing the count in removed cells and increasing the count for the new cells which is also a linear-time operation.

The next proposed method is a median tracking algorithm that exploits features coherency in adjacent local windows. Experiments on the difference between two adjacent local windows in WMF shows that it has a small average value of 7 to 8 in 8-Bit gray-scale images. This observation leads to the suggested median tracking algorithm, which searches locally around the median of a window to find the median of its adjacent window by using the joint-histogram of features, thus speeding up the weight sum calculation more than 32 times.

The Necklace Table is the last proposed scheme, which is an efficient data structure that enhances traversal of the joint-histogram by avoiding traversing empty cells in the histogram. The data structure is a double-linked list where each node has double pointers to its previous and next non-empty cells. Given that constructed joint-histograms are sparse and comprise up to 98% of empty cells, using this data structure makes element traversal of the joint-histogram 10 to 50 times faster.

2.2 Adaptive Cross-trilateral Median Filter

Local window-based methods for finding correspondences between left and right images of a stereo vision system have a poor performance at borders of objects and inside large textureless regions. The adaptive cross-trilateral median filter is suggested by Mueller et al. [12] to overcome those issues by smoothing textureless areas, and aligning depth discontinuities of the depth map to color discontinuities in the original image.

The method starts by computing two disparity maps, one from left to right and the other from right to left. Those two maps are used to calculate a confidence value based on left-right disparity maps consistency. The total weight for each disparity value inside the filter local window comprises two additional terms, a spatial distance Gaussian kernel and another Gaussian kernel based on intensity difference between pixels in the reference color image. After calculating the total weight, disparity values in both maps are updated to the median of their weighted local window. Finally, this procedure is performed for few iterations until filtered disparity maps converge and stop updating. Fig. 1 shows an example of disparity maps after applying the filter.
2.3 Improved Weighted Median Filter with Superpixel for Disparity Refinement

Fei et al. [6] present an improved version of the WMF based on superpixel segmentation, which provides effective information for local image features by modifying the WMF using a superpixel penalty factor.

The proposed method first uses a weighted least square cost aggregation step to calculate an initial disparity map with reduced noise. Then it generates superpixel labels for the reference colored image pixels using the SLIC superpixel algorithm [2], which clusters pixels based on their color similarity and spatial distance. The final step is to apply the improved WMF weights by multiplying the weights by a penalty factor if the local window pixel has a different superpixel label than the center pixel.

3 Color and Depth Consistency based Post-Processing

Depth maps generated using Microsoft Kinect are found to be noisy, contain misalignments with reference color images, and contain large holes and fluctuating edges near object boundaries [19], as shown in Fig. 2. In this section, we describe two computational methods based on the Markov random field (MRF) optimization framework. Those methods eliminate holes and misalignments by exploiting information from the raw depth map and its coupled colored image to obtain a high quality consistent depth map.
3.1 Depth Map Enhancement based on Color and Depth Consistency

The goal of the depth post-processing method presented by Wang et al. [19] is to remove holes and misalignments from Microsoft Kinect raw depth maps. To achieve this, the method starts by smoothing the raw depth map by exploiting information from its reference color image using a joint bilateral filter (JBF) which depends on pixels color and spatial distances. The JBF is modified by incorporating an additional depth weight that forces smoothing within the same depth layer to overcome depth edge blurring around neighboring pixels with similar color and different depth.

Using the smoothed depth map and the reference color image, multiple cues are extracted to define the different constraints in the MRF energy function. The first term in the energy function is an edge-sensitive smoothness constraint that aligns depth map discontinuities with edges in the reference image. To produce sharp depth edges around objects boundaries, the smoothness constraint defines a weight for each pixel regarding neighboring pixels. The weight is the product of color similarity that forces adjacent pixels with same color to have same depth values, edge penalty weight computed using a Canny edge detector to penalize inconsistent depth edges, and a segmentation penalty that encourages pixels within a single segment to have the same depth value.

The second term in the MRF energy function is a data term that eliminates unreliable depth values inside holes and misaligned edges by computing depth confidence for each pixel based on consistency between the smoothed depth map and the color image. The last term handles large holes in the smoothed depth map by adding constraints to pixels inside holes using estimated depth values from the modified fast marching method (MFMM) image inpainting algorithm [18].

Using this multiple information sources, the MRF energy function is optimized to produce the complete output high-quality depth map as shown in Fig. 3.

![Fig. 3. Pipeline of the depth map enhancement algorithm by Wang et al. [19]](image)

3.2 Edge-guided Depth Map Enhancement

A problem with the previously mentioned method by Wang et al. [19] is that resulting maps contain incorrect depth edges in rich texture scenes. This happens because of using color texture edges for guidance that are not related to depth changes. To solve this issue Song et al. [16] suggest a novel method based on MRF optimization framework that removes redundant texture edges from the color image, thus only make use of depth edge information during the optimization process.

To produce an initial smoothed and filled depth map, the method uses an image pyramid based technique. The pyramid starts from the raw depth map and uses a nearest neighbor based downsampling strategy to produce the depth map at the next level. The propagation is performed till
we reach a downsampled filled depth map at the base of the pyramid. Using the same propagation method in a bottom-up direction, each downsampled map is used to produce a complete one in the level above it till we reach a filled version of the raw depth map in the top of the pyramid. Canny edge detector and a tensor voting strategy [17] are then used to produce a smoothed edges of the filled depth map, as illustrated in Fig. 4.

In order to exploit edges related to depth from the color image, the method removes texture edges from the color edge map by overlapping the smoothed depth edges with the color edges, and then uses small local windows centered at depth edge points to keep edge points around them. Using this edge information from the color image and to align depth edges with it, a local 2D-ICP algorithm [7] is used to find the relative pose between the two edge maps and project the smoothed depth edges to the color edges to get an accurate depth edges aligned with the reference color image.

The reference color image, raw depth map, and the accurate depth edge map are finally combined into an MRF optimization framework to produce high-quality depth map.

4 Depth Maps Upsampling Post-Processing

Many computer vision applications such as 3D reconstruction and driver assistance systems require high-resolution depth maps. However, depth sensor devices such as time-of-flight, LiDAR and Microsoft Kinect cameras are limited to produce low-resolution depth maps [11]. Depth upsampling methods aim to solve this problem by increasing the resolution of depth maps while improving their quality. This section presents two methods that use global optimization framework for depth map upsampling.

4.1 Non-convex Joint Bilateral Guided Depth Upsampling

The joint Bilateral Upsampling (JBU) is a local method used to upsample a low-resolution depth map into the same high-resolution of its coupled color image [11]. JBU is based on local weighted sum strategy to generate the high-resolution depth map, where the local weights depend on spatial distance between pixels in the reference color image and the raw depth map.

The authors start by analyzing JBU and its variants and formulating a global view of them as $L_2$ norm convex optimization problems. Since $L_2$ norm is in use, the JBU based methods produce noisy results at points where there are inconstancies between color and depth edges. Another problem with those methods is that they produce noisy results for large upsampling rates (i.e. 8x, 16x) because
the low-resolution depth maps projected to those high-resolutions, which JBU uses for optimization, lose lots of information and become blurry.

To overcome these two issues of JBU based methods, the non-convex joint bilateral guided depth upsampling (NCJBU) is suggested by Lu et al. [11], which replaces the $\ell_2$ norm with a non-convex error norm function, and solves the optimization problem iteratively till convergence by using the updated depth map at each iteration. Using these modifications, the NCJBU method produces less noisy output at large upsampling rates, compared to JBU, and outputs sharp edges at points with color and depth edges inconsistencies.

4.2 Semantically Guided Depth Upsampling

The goal of the method suggested by Schneider et al. [15] is upsampling of sparse depth maps acquired from depth sensor devices by exploiting object boundary cues guided by structured edge detection and semantic scene labels as shown in Fig. 5. Given the observation that objects surfaces are usually smooth and thus can be approximated by local planes, the method models the coupled high-resolution color image as a set of local planes, where each plane has a unique depth value for all its pixels.

To estimate this planar representation of the scene given a set of sparse depth measurements, the method formulates a pixel-wise global energy function that estimates a set of local planes, their depth values and assigns each pixel in the image to its corresponding plane. To achieve robustness against noisy depth inputs, the energy function additionally estimates a binary outlier indicator for each input depth measurement and discards those outliers in the energy function.

In order to preserve depth discontinuities and fine structures in the upsampled depth map, the method adds a co-planarity indicator and pair-wise weight between each pair of local planes to encourage consistent connected planes to have a smooth depth transition. The pair-wise weight is defined as a geodesic distance [1] between two planes which depends on semantic and structured edge information in the color image acquired by convolutional neural networks. By fusing those multiple cues into the optimized energy function, the problem is solved iteratively to reach a global consistent solution.

![Fig. 5](image)

**Fig. 5.** Output example of the method where cues from the color image are combined with semantic class labels and low-resolution depth measurements to produce a smooth and accurate high-resolution upsampled depth map [15]

5 Image Segmentation based Post-Processing

Image segmentation is the process of dividing the image into homogeneous and non-intersecting regions such that all pixels within a single segment share common characteristics [13]. In this section, we present a novel method suggested by Vieira et al. [4] that uses image segmentation to enhance disparity maps quality by exploiting local information within image regions.

5.1 Disparity Map Adjustment: a Post-Processing Technique

The image segmentation based method starts by calculating a raw disparity map that contains noise parts and wrong estimations in textureless regions. By analyzing one of the textureless regions in the
raw disparity map against the same region in the ground truth, its found that most of the disparity values in the estimated region point to the correct disparity value as shown in Fig. 6.

![Fig. 6. (a)raw and ground truth disparity map regions (b)disparity values histograms][4]

Motivated by this observation, the reference image is segmented into independent regions using the mean shift algorithm [3]. Those segments are then used as disparity map regions. For each region in the disparity map a histogram of the disparity values is calculated to find the most common disparity value $d$ in the region. Then each disparity value in the region is updated to $d$ if it lies within a certain threshold to $d$, or to zero otherwise (unknown disparity). Local supported weighted windows based on color proximity constraints, are finally used to find values for unknown disparities. Fig. 7 shows a qualitative example of the method applied to a raw disparity map.

![Fig. 7. Left: raw disparity map, Middle: Improved map with unknown disparities, Right: after filling unknown disparities][4]

### 6 Conclusion

In this survey, we reviewed several state-of-the-art papers for depth post-processing. We started by explaining the different techniques used to acquire depth maps, their common problems and the motivation for the usage of post-processing algorithms. The reviewed methods covered a wide range of techniques ranging from applying special types of weighted median filters to raw depth maps, through post-processing of Microsoft Kinect depth maps by exploiting color and depth consistency cues, to depth upsampling and image segmentation based algorithms. Despite the large number of proposed methods to solve the problem and given its importance to many computer vision tasks,
this area is still active for research that aims to achieve better results and move towards real-time performance.

References

15. Nick Schneider, Lukas Schneider, Peter Pinggera, Uwe Franke, Marc Pollefeys, and Christoph Stiller. Semantically guided depth upsampling. 08 2016.
Survey on 3D Object Tracking

Kumar Ankit\textsuperscript{1} and Michael Fürst\textsuperscript{2}

\textsuperscript{1} ankit@rhrk.uni-kl.de
\textsuperscript{2} david_michael.fuerst@dfki.de

Abstract. Object tracking deals with estimating the location or trajectory of the object of interest in subsequent frames of a video sequence. This task becomes critical in the context of autonomous vehicles especially when there are pedestrians involved. A well-trained object tracker can overcome the drawbacks of the detector and can be used, especially in case of view/pose variances or occlusions. This paper presents some state-of-the-art and classical approaches with a comparison on KITTI benchmark. We have analyzed different approaches of object trackers considering their temporal aspect with a focus on pedestrians. Finally, we briefly summarise their performance on KITTI tracking benchmark.

Keywords: Object Tracking, Autonomous vehicle, LIDAR, KITTI

1 Introduction

Computer Vision problems such as pedestrian tracking, traffic monitoring, autonomous vehicles, and motion-based recognition includes Object tracking as a prominent step. Object tracking locates the position of the object in subsequent frames of a video by finding its motion path. The advantage of tracking over detection is that it provides more robustness in pose/viewpoint in variance. For instance, if we train a detector to detect frontal faces and if a new image is considered with face side turned, the detector has a high probability of failing than a well-trained tracker. Also, detecting objects in each frame is not efficient. This is like throwing out the information, which is collected in the previous frame, and starting all over for the current frame. Instead, in the tracker, one can use temporal information which gives good results on objects which are occluded for a few frames and then re-appear in the scene. 3D object tracking includes the depth information and plays indispensable role in context to autonomous vehicles, for instance, by detecting pedestrians, and hence, eventually avoiding accidents as such.

Fig. 1. 3D point cloud from the LIDAR sensor in the top and corresponding detection in the below image. Figure taken from [4].

Fig. 2. Vedolyne Lidar sensor mounted on top of a car. Figure taken from [2].

Recently, high quality and inexpensive video cameras and LIDAR sensors are now available, which has powered the interest in tracking algorithms. LIDAR (Light Detection And Ranging) (Figure 2) sensors are one among them, which can measure an object near it. These sensors generate a 3D
map of the environment (Figure 1), which can be used to navigate the robot or vehicle. Fast and Furious (FaF) approach [5] uses these sensors to collect the data to train their network. Through this paper, we survey different latest object tracking algorithms with a focus on pedestrian and car tracking in the context of autonomous vehicles. To compare these approaches, we use KITTI tracking benchmark [2]. KITTI [2] provides a dataset and benchmark with the evaluation metric. The different evaluation metrics used are multi-object tracking accuracy/precision (MOTA/ MOTP) and mostly tracked/mostly lost. MOTP gives average overlap between all correctly matched hypotheses and their respective objects. MOTA combines error made by tracker, misses, mismatches and false positives for all objects over all frames. Tracklets (trajectories) can be classified as mostly tracked (MT) or mostly lost (ML) if the target is either tracked for 80% of its life or 20% respectively. We have categorized the approaches and explained them in section 2 and 3 followed by their comparison in section 4.

2 State-of-the-art approaches

In these approaches, object detection and tracking is done by using deep neural networks. A tracker is used on the results of detection proposals and is either trained jointly with the detector or detector is trained first, and then tracker is trained to reach efficiency.

2.1 Fast and Furious (FaF)

FaF [5] proposes a single convolutional network, which jointly deals with the detection, tracking, and also motion forecasting. They use a 3D sensor (LIDAR) which produces 3D point clouds and operates on a bird-eye-view (BEV) representation (elevated view of an object from above as though observer were a bird) of the 3D world. For input, they use multiple consecutive temporal frames, which creates a 4D tensor which is given to a single-stage detector. The detector performs 3D convolutions over space and time to produce 3D bounding boxes at the current frame and multiple timestamps in the future. Tracklets from these predictions are found by pooling operation that combines evidence from past and current predictions. The end-to-end process completes in about 30 ms. Figure 3 summarises the overview of this approach.

Architecture

FaF approach describes the frame representation by voxel representation. Due to irregularities in point cloud, convolutions are not possible, hence point clouds are converted into 3D voxel grids. However, since the grid is very sparse, the 3D convolution operation would be computationally expensive. Hence, the authors perform 2D convolutions on this 3D voxel representation, using...
height as a channel dimension. The intuition for this is that if the resolution of the grid is high, this approach is similar to applying convolution on every single point. 3D points from past n frames are taken and its coordinates are changed to represent them in the current vehicle coordinate system to undo the ego-motion of the vehicle. After that, the voxel representation for each frame is computed. Hence, each frame is now a 3D tensor, and they append multiples frames along the temporal direction to make a 4D tensor. The advantage of this is that they get more 3D points as a whole and also cues about vehicles heading, and velocity, which is helpful in motion forecasting. 

Now they take this 4D tensor as input and pass it through the single-stage detector (notable examples include YOLO[7]), which directly regresses to object bounding boxes at different time-stamps. To exploit the temporal dimension on the 4D tensor, fusion techniques are used.

**Early and Late Fusion**

Early fusion approach aggregates the temporal information in the first layer of the network but fails to capture complex temporal features. Temporal dimension is reduced, and then spatial convolution and pooling operations are performed. Late fusion approach gradually merges the temporal information using 3D convolutions. Since it is captured in later layers also, this can capture complex motion features. Late fusion improves the accuracy of the model when compared to early fusion because of the fact that it can capture complex features.

**Decode Tracklets**

To decode tracklets, at each timestamp, the model outputs the bounding boxes for \( n \) timestamps, meaning current detection with \( n - 1 \) past predictions. This information from the past is aggregated to produce accurate tracklets without solving any trajectory-based optimization problem. Whenever there is an overlap between detections from current and past prediction, they are considered to be the same object, or else a new object is assumed. The intuition behind this aggregation is that even if the object is occluded at any time frame, it can be tracked.

**2.2 Generic Object Tracking Using Regression Networks (GOTURN)**

GOTURN approach [3] is a single-object tracker where a neural network is trained offline to track generic objects and at test time network is able to track objects without any fine-tuning. Objects are tracked at 100 FPS using this approach. In this method, the network is fed frames of a video, and it learns the generic relationship between appearance and motion, which is used to track objects at test time with no online training.

![Fig. 4. GOTURN](image-url)

*Fig. 4. GOTURN.* The authors input to the network a search region from the current frame and a target from the previous frame. The network learns to compare these crops to find the target object in the current image [3]. Figure taken from [3].
Architecture
In case there are multiple objects in video or image, the network is given information about the target object to be tracked. A two-frame architecture is used for this (Figure 4), where the previous frame is cropped and scaled around the target object, and then the crop is padded to give some contextual information about the object. To find the object in the current frame, the frame is cropped using the search region defined by the previous frame and network then regresses this crop to the location of the target object within the search region. Fully connected layers compare the features from the target object to the features in the current frame to find where the target object has moved [3]. The network works if the object is not occluded and object has smaller motions.

2.3 Recurrent YOLO (ROLO)

ROLO approach [6] extends Deep Neural Network analysis into the spatiotemporal domain. In this approach, Long Short-Term Memory (LSTM) is used in the temporal domain, and visual features from convolutional networks are concatenated with LSTM. This approach is more robust because it considers the location of objects from past frames and also the robust visual features from convolutional network.

Fig. 5. ROLO. It uses CNN and YOLO to train detector and then LSTM for the tracker. Figure taken from [6].

Architecture
This model takes raw video frames as input and returns the coordinates of a bounding box of the tracked object in each frame. The model is trained in 3 phases as also is shown in Figure 5:

Detection Module
In this module, traditional CNN takes raw video frames as input and learns a feature representation of the whole image. After this CNN module is trained, YOLO [7] architecture is used on top of CNN to regress feature representation into region predictions. These predictions are a $S \times S \times (B \times 5 + C)$ tensor as defined by YOLO. This denotes that an image is divided into $S \times S$ grids, each grid cell has $B$ predicted bounding boxes with $x, y, w, h$ as 4 locations and $c$ as the confidence score. $C$ is the number of the classes (these are one-hot encoded).

Tracking Module
To train tracking module, LSTM RNNs are used. The input to the LSTM at time step $t$ is, $X_t$, $B_{ij}$, and $S_{t-1}$ where $X_t$ is the feature vector from convolutional layers, $B_{ij}$ are detections from YOLO and $S_{t-1}$ is output from last time step of LSTM. They use Mean Squared Error for training as defined below in Equation 1 where $n$ is the number of training examples in a batch, $B_{pred}$ is predicted vector, and $B_{target}$ is target ground truth vector.

$$L_{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left\| B_{target} - B_{pred} \right\|_2^2$$  (1)
Alternative HeatMap

In LSTM, coordinates of the detected object is regressed directly, which makes the interpretation of working of LSTM difficult to understand (because the regression is a non-linear operation). The intuition to understand how LSTM performs tracking is to convert ROLO prediction locations into a feature vector of length 1024, which is then translated to $32 \times 32$ heatmap. A Detection box is converted relative to heatmap by transferring corresponding region information. This is again trained using MSE error function as defined in (Equation 2), where $H_{\text{target}}$ is the heatmap vector of target and $H_{\text{pred}}$ is LSTM output. LSTM learns the regression from the YOLO, and temporally learns over the sequences to restrict location prediction into a spatial range.

$$L_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} ||H_{\text{target}} - H_{\text{pred}}||^2_2$$

(2)

3 Classical approaches

In classical approaches, object tracking is usually done by object detection. In other words, tracking is divided into two phases: detection and association. In the detection step, the target object is detected, usually by Deep Neural Network based approaches. And in the association step, tracklets from the previous frame is associated with the current frame to track the target object.

3.1 Near-Online Multi-target Tracking with Aggregated Local Flow Descriptor (NOMT with ALFD)

NOMT approach [1] is a classical approach which combines global tracking algorithm with an online algorithm. In the online method, the association between existing target and detections in the current time frame is found which is good for real-time applications. However, global algorithms are more robust in finding long-term information in the association process with a better association accuracy. NOMT algorithm extends global algorithms to an online application. This algorithm [1] is similar to online as it outputs the association in every time frame, with a difference that decisions in the past can be changed when more observations are available.

NOMT is addressed in 2 phases: 1. ALFD metric which defines affinity measure (the likelihood that two detections belong to the same target) 2. NOMT algorithm [1] which defines the association between target and detections in a temporal window performed at every frame.

Fig. 6. ALFD. In the top figure, detections are shown as colored bounding boxes (red, green, blue). A pair of circles with connecting lines represent tracklets that exist in both the frames. In the below figure, comparisons of 2 ALFDs (red, blue) and (red, green) is shown. It is clear that more tracklets are observed in blue and hence, red and blue are the same target. Figure taken from [1].
Once all the hypothesis is defined for new and existing targets, (Equation 3) is formulated as an optimization is defined as in Equation 3: \[ \Phi(x_1, x_2, \ldots) \]

\[ \hat{x} = \arg\min_{x\in H} E(A^{t-1}, H^t(x), D^{t-\tau}, V^t) \]

\[ E(A^{t-1}, H^t(x)) = \sum_{m} \Psi(A^{t-1}, H^t_{m,x_m}) + \sum_{m,l} \Phi(H^t_{m,x_m}, H^t_{l,x_l}) \]  

\[ \tau \]

\[ \sum_{m} \Psi(A^{t-1}, H^t_{m,x_m}) \]

\[ \sum_{m,l} \Phi(H^t_{m,x_m}, H^t_{l,x_l}) \]

\[ \Psi(x) \]

\[ \Phi(x) \]

\[ \hat{x} \]

\[ \arg\min_{x\in H} E(A^{t-1}, H^t(x), D^{t-\tau}, V^t) \]

\[ \sum_{m} \Psi(A^{t-1}, H^t_{m,x_m}) + \sum_{m,l} \Phi(H^t_{m,x_m}, H^t_{l,x_l}) \]  

\[ \sum_{m} \Psi(A^{t-1}, H^t_{m,x_m}) \]

\[ \sum_{m,l} \Phi(H^t_{m,x_m}, H^t_{l,x_l}) \]

\[ \Psi(x) \]

\[ \Phi(x) \]

\[ \hat{x} \]
inference problem with an undirected graphical model (Figure 7 (c)), where one node is a target and states are the hypothesis indices. To solve this graphical model, independent sub-graphs are found using connected component analysis and inference algorithm on each sub-graph is performed in parallel. If a sub-graph has 1 node, the best hypothesis is selected else it is further solved. Once the states $x$ are found, a new set of targets can be uniquely identified by augmenting $A^{t-1}$ with $H(t)(x): A^{t-1} + H(t)(x) \rightarrow A^t$.

3.2 Beyond Pixels

This approach [8] introduces data association costs based on 3D pose and shape of the object. Object detection phase gives the detections, and then in the association phase, the pairwise affinity between two detections is computed. For these affinity measures, different costs are defined, which are agnostic to the association framework used. Different costs that are defined in this approach are: 3D-2D cost, 3D-3D cost, Appearance cost, and Shape and Pose cost. The overall cost is the weighted linear combination of all these costs.

**Fig. 8. Beyond Pixel.** Two subsequent frames $t$ and $t+1$ are shown in the left. For each detection in the frame $t$, a 3D bounding box is computed and propagated in next frame $t+1$. These boxes are projected to 2D in the frame $t+1$ and then intersection between the detections at $t+1$ frame and these projections constitute 3D-2D cost. The intersection of the 3D bounding boxes in 3D constitute the 3D-3D cost as shown in the right; propagated bounding boxes are colored with their respective 2D box in frame $t$ and 3D bounding boxes of detections in frame $t+1$ are numbered respectively [8]. Figure taken from [8].

**3D-2D cost**

First, the depth estimate $X_t$ of each bounding box detection $d_t$ in the frame $k$ is found in the current camera coordinates. To find the association with detection $d_{t+1}$ in any frame $t+1$, a rough estimate of camera motion is computed. Using this, estimate $X_t$ is transported to camera coordinates of frame $t+1$. The obtained coordinates $X_{t+1}$ are then projected to image frame of $k'$ to obtain 2D search area in which potential matches for $X_t$ are expected to be found as shown in the frame $t+1$ of Figure 8. Intuitively, this cost measures overlap of the 2D region in which the target is expected in the frame $t+1$. This reduces the number of candidate detections to be evaluated.

**3D-3D cost**

Since previous cost still measured overlap in image space, this cost measures overlap in 3D as shown in Figure 8 (right side). Here, candidate $d_{t+1}$ is back-projected via road plane, and then the overlap is measured with respect to the transformed 3D volume from frame $t$.

**Appearance cost**

To calculate dissimilarity between different detections, a weighted combination of activation maps from the outputs of already trained stacked-hourglass CNN is used as a feature descriptor for each detection and a similarity score between detections is computed using $L^2$ Norm.
4 Comparison and Conclusion

In this paper, we reviewed various recent approaches on object tracking. A brief comparison between the approaches has been made in Table 1, based on the KITTI tracking benchmark [2]. However, GOTURN [3] and ROLO [6] are generic object trackers and are included in this survey for the reason that they use different neural network architecture for tracking. These trackers are not evaluated on the KITTI benchmark. For GOTURN approach accuracy for different attributes like occlusion, illumination change, size change, and motion change is measured for each frame of video and averaged for these attributes. Also, comparison of FaF, NOMT, and Beyond Pixel based on KITTI tracking benchmark is made. In conclusion, all the approaches of this survey apply different architectures. Classical approaches (Beyond Pixel and NOMT) are evaluated on KITTI test set and Beyond Pixel performs better than NOMT based on the values of MOTA, MOTP, MT and ML. FaF also has good accuracy but is evaluated on a different dataset.

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Dataset</th>
<th>MOTA</th>
<th>MOTP</th>
<th>MT</th>
<th>ML</th>
<th>Accuracy Ranks</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOTURN [3]</td>
<td>VOT 2014 tracking</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5.54</td>
</tr>
<tr>
<td></td>
<td>challenge Dataset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OTB-30 dataset</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FaF Dataset</td>
<td>80.9</td>
<td>85.4</td>
<td>75</td>
<td>10.6</td>
<td></td>
</tr>
<tr>
<td>ROLO [6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FaF [5]</td>
<td></td>
<td>80.9</td>
<td>85.4</td>
<td>75</td>
<td>10.6</td>
<td></td>
</tr>
<tr>
<td>Beyond Pixel [8]</td>
<td>KITTI test set</td>
<td>84.24</td>
<td>88.73</td>
<td>75.23</td>
<td>2.77</td>
<td></td>
</tr>
<tr>
<td>NOMT [1]</td>
<td>KITTI test set</td>
<td>78.15</td>
<td>79.46</td>
<td>57.23</td>
<td>13.23</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Object tracking accuracy for KITTI tracking benchmark. '-' represents that values are not available and hence no comparison is made. Higher values of MOTA, MOTP, MT and lower values of ML indicates tracker is good.

References

Non-parametric Motion Capture Segmentation

Kumar Ankit$^1$ and Manthan Pancholi$^2$

$^1$ ankit@rhrk.uni-kl.de
$^2$ manthan.pancholi@dfki.de

Abstract. Human activity segmentation is the problem of classifying different set of activities performed by humans. The activity data is in the form of sequences and becomes a challenging problem when large number of observations is produced each second. This paper reproduces a Bayesian non-parametric method to solve human activity segmentation. The existing methodology works on multiple time series which are related by activities shared among the sequences. The claim of the existing method is verified on the CMU Mocap dataset and evaluated on the human3.6m dataset to test the method’s robustness.

Keywords: Human activity segmentation, Non-parametric beta process, hidden Markov models, multiple time series

1 Introduction

Human activity segmentation is the problem of classifying the movements of a person, which are captured from sensors. The movements are normal activities such as walking, walking with a dog, sitting or performing jumpjacks. Different sensor modalities can be used to capture this sequence of data (activities) such as optical or IMUs. These sensors measure either the accelerometer or gyroscope data in 3 dimensions (x,y,z). Different activities have different kind of movements and these are captured and shown as a sequence. As shown in Figure 1, different activities have different patterns of sequences in all the 3 dimensions. Intuition is to learn these different patterns and classify them correctly. This task is challenging because each second large amount of observations is collected by sensors and also the classical method use feature engineering to capture hand crafted features and hence analysis of time series data becomes tedious.

Classical time series analysis involves study of single time series. This paper [1] considers analysis of multiple related time series, motivated by increasing abundance of such data in many domains. Motion capture sensors are placed on the joints of people and these people perform certain set of activities, for instance jump jack, side bends. There is a global set of activities defined among which few are uniquely performed and few activities are shared among the subjects.

Fig. 1. Different sequences for each activity. This figure shows that each activity has different patterns of sequences measured in x,y,z directions. Figure taken from [3].
The methodology mentioned in the paper [1] is a non-parametric bayesian approach which uses a beta process prior to infer size of behaviour set from the data. Markov Chain Monte Carlo (MCMC) inference algorithm is used for inferencing. The pipeline of the main approach is explained in section 2. Section 3 explains the experiment section in which first claim of the main approach is verified by reproducing it on CMU Mocap dataset and then is evaluated on Human 3.6m dataset [2] to validate the models robustness. In section 4, comparison of different sampling methods are shown followed by the results section 5 and conclusion in section 6.

2 Existing approach

Fig. 2. Pipeline of the approach. Raw time series is given as input to the BP-AR-HMM model and it outputs the segmented regions of the activities.

Figure 2 explains the basic pipeline of the model. The model takes as input the multiple related raw time series (in this figure shown only 1 time-series). These time series contains the joint angle measurements of the most movable joints captured from the sensor. These angle measurements represents latent set of dynamic activities and the model segments each time series into regions defined by the activities. The binary feature matrix Figure 3 is manually annotated and is used as ground truth. The model used for the segmentation is beta process auto regressive hidden Markov models (BP-AR-HMM). The graphical representation of this model is shown in Figure 4.

Fig. 3. Binary Feature Assignment Matrix. This matrix is produced by manual annotation where each row indicates the activity type present for the particular sequence. Figure taken from [1]
Fig. 4. Graphical representation of BP-AR-HMM model. \( w_k \) (feature inclusion probabilities) and \( \theta_k \) (VAR parameters) of the beta process define the unbounded feature set as explained in Equation 5. Feature vectors \( (f_i) \) is indexed over \( \theta_k \) and also defines transition distribution \( \pi^{(i)} \) (Equation 3) which can also be written in terms of transition weights \( \eta^{(i)} \). The state evolution and VAR dynamics for observations \( y_t^{(i)} \) is defined in Equation 4. Figure taken from [1].

Architecture

To explain the model of BP-AR-HMM, first dynamics of single time series and how it relates with other time series is considered. Single time series dynamics is modelled based on hidden Markov model (HMM). For observations \( y_t \in \mathbb{R}^d \) and hidden states \( z_t \), the HMM assumes Equation 1 for an indexed family of distributions \( F(\cdot) \). Here \( \pi_k \) is the state-specific transition distribution and \( \theta_k \) is the emission parameter for state \( k \) [1].

\[
z_t | z_{t-1} \sim \pi z_{t-1}, y_t | z_t \sim F(\theta z_t)
\]

The paper [1] considers vector auto regressive (VAR) processes (or auto regressive HMMs) for its computational effectiveness. A \( r \) order VAR process is defined as in Equation 2 where \( e_t(z_t) \sim N(0, \Sigma z_t) \) and \( \tilde{y}_t = [y_{t-1}^T ... y_{t-r}^T] \) are the aggregated past observations. \( A_k = [A_{1,k} ... A_{r,k}] \) are set of lag matrices. VAR parameters for \( k \)th state is defined as \( \theta_k = A_k, \Sigma k \). Each VAR process is a dynamic activity and these parameters define linear motion model for the activities like running, jumping walking and so on.

\[
y_t = A_k \tilde{y}_t + e_t(z_t)
\]

To relate the dynamics of multiple time series, a shared set of dynamic activities \( \theta_1, \theta_2, ... \) are defined. Binary feature vector \( f_i = [f_{i1}, f_{i2}, ...] \) as shown in Figure 3, associates these activities by setting the value \( f_{ik} = 1 \), which means that time series \( i \) has that activity for some subset of time sequence. Feature vector \( f_i \) also defines the transition distribution \( \pi_k^{(i)} \) which indicates that for a time series \( i \) activities can switch between those present in feature vector \( f_i \). The transition distribution \( \pi_k^{(i)} \) satisfies the Equation 3.

\[
\Sigma_{j} \pi_{kj}^{(i)} = 1, \pi_{ij} = \begin{cases} 0, & \text{if } f_{ij} = 0, \\ > 0, & \text{if } f_{ij} = 1 \end{cases}
\]

For multiple time series, order \( r \) AR-HMM is defined as in Equation 4.

\[
z_t^{(i)} | z_{t-1}^{(i)}, y_t^{(i)}, z_t^{(i)} \sim N(A_j \overline{y}_t^{(j)}, \Sigma z_t^{(i)})
\]

Feature matrix for each time series \( i \) reduces the model to collection of \( N \) switching VAR processes. Dynamic activities \( \theta_k = A_k, \Sigma_k \) are shared across all time series. This sharing of activities by discovering \( f_{ik} = f_{jk} = 1 \) for some pair of sequences \( i, j \), provides interpretation of how time series relate
to each other. However, to maintain unbounded set of possible activities, Bayesian non-parametric approach based on beta process-Bernoulli process is considered and model is defined for a globally shared set of possible dynamic activities \[1\]. Beta process model defines a prior on feature inclusion probabilities \( w_k \) which helps in inferring the structure of activity set from the data and is defined in Equation 5 where \( \theta_k = A_k, \Sigma_k \).

\[
B = \sum_{k=1}^{\infty} w_k \theta_k 
\]  

(5)

Markov Chain Monte Carlo (MCMC) algorithm computes posterior samples from feature matrix \( F \) and state sequence \( Z \).

3 Experiments

The BP-AR-HMM model was first reproduced on CMU mocap dataset (http://mocap.cs.cmu.edu).

3.1 CMU dataset

The CMU motion capture dataset has available set of 62 positions and joint angles. However, in the paper \[1\], 12 measurements which are most informative are considered. The CMU dataset is captured at 120 frames per second which is downsampled to 10 frames per second by using block average and window size of 12. Also, manual annotations of 12 activities are present. After executing the MCMC inference using Split Merge with Data Driven and Annealing sampling for 50 seconds with initialization of feature matrix as 5 unique features per sequence on the dataset, segmented output as shown in Figure 5 is reproduced. 12 possible set of global activities present are

![Fig. 5. Segmented activities of the 6 sequences (time series) using CMU MoCap dataset. It has true segmentation and estimated segmentation. These sequences are performed by subjects 13 and 14.](image)

A: JumpJack, B: Jog, C: Squat, D: KneeRaise, E: ArmCircle, F: Twist, G: SideReach, H: Box, I: Up-Down, J: ToeTouch1, K: SideBend, L: ToeTouch2. Also, joint log probabilities of data and sampled
variables $p(y, F, z, a, \gamma, \kappa)$ is shown with Hamming distance accuracy in Figure 7 and Figure 6 respectively. From the output, it is verified that the BP-AR-HMM model segments the activities with a good hamming distance accuracy and hence, to test if the model is robust against other Mocap datasets, Human 3.6m dataset is used.

### 3.2 Human 3.6m dataset

This dataset [2] is captured at 50 frames per second. Each subject performs 15 set of global activities. We have focused on 9 activities which have most movements. Also, we have considered 31 joint angles of hips, spine, neck, right upper leg, right leg, right foot, right arm, right forearm, right hand and similar joints for left part. As a preprocessing step, we shifted the mean to zero center. The sudden discontinuities between the angle measurements are removed by smoothing angle measurements. The data is block-averaged using the window size of 5 and then downsampled to 10 frames per second. Different input representations (euler angle and 3D axes angle) are used to check if input representations also affect the segmentation.

### 3.3 Preprocessing - Human 3.6m dataset

As mentioned in the previous section, for this dataset, 31 joint angles are selected first which have maximum movements. This is the raw time series (Figure 8(a)) for a subject who performs certain set of activity from the global set of activities. From the figure (Figure 8), it is evident that there are sharp discontinuities in the angle measurements and to remedy this, smoothing of angle measurements is performed. For instance, if angle measurements shall jump from +178 to +182 and instead it jumps to -178, this jump is not in the boundary and hence causes the discontinuity. After applying the smoothing in angle measurements, the results that are achieved is shown in Figure 8(b).

![Fig. 6. Hamming distance plotted across number of iterations. It decreases and eventually settles down to 0.2](image)

![Fig. 7. Joint log probability of data and sampled variables against number of iterations.](image)

![Fig. 8. (a) Raw time series. Subject S1 performs activity Walking and we have considered flexion/extension knee joints of right leg and left leg. (b) Smoothen joint angles. This time series is achieved after smoothing out the angle measurements and hence, removing discontinuities.](image)
After applying smoothing, joint angles are zero-centred as shown in Figure 9(a). Here the data points are shifted to zero mean. The data is first analysed at 50 frames per second, however the desired results are not achieved. Hence, we decided to downsample the data. Similar preprocessing is used by the paper [1] by applying block-average and window size of 12 followed by downsampling the data from 120 frames per second to 10 frames per second. For this dataset, we used window size of 5 to block-average the data and then downsampled the data from 50 to 10 frames per second as shown in Figure 9(b).

Fig. 9. (a) Zero-centered time series. Data points are shifted to the zero mean. (b) Downsampled data. This time series is achieved after applying block average with a window size of 5 and then downsample to 10 frames per second.

4 Comparison

Different synthetic datasets are created to analyse the behaviour of different sampling methods used in the BP-AR-HMM model (split-merge and data driven). These datasets are created on different frequencies to check if the model can be used real time. Also, different input representations are considered such as euler angle representation and axes angle representation. Apart from this, we also tried Gaussian distribution instead of auto regressive process. The best performance is observed at 10 frames per second with data driven sampling under Var process, as shown in Table 1. The BP-AR-HMM model also shows promising results on unsupervised segmentation, however since manual annotation is not used, it is hard to verify the result.

<table>
<thead>
<tr>
<th>Evaluation metric</th>
<th>Gaussian distribution</th>
<th>Gaussian distribution</th>
<th>VAR process</th>
<th>VAR process</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>zDD</td>
<td>SM + zDD</td>
<td>zDD</td>
<td>SM + ZDD</td>
</tr>
<tr>
<td>logPR</td>
<td>−2.27 × 10^5</td>
<td>−2.10 × 10^5</td>
<td>−2.45 × 10^5</td>
<td>−1.90 × 10^5</td>
</tr>
<tr>
<td></td>
<td>0.39</td>
<td>0.70</td>
<td>0.22</td>
<td>0.85</td>
</tr>
<tr>
<td>Hamming distance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Segmentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classified</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>only 1 activity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Gaussian distribution with Autoregressive-Gaussian is compared. Under both the inferences, data driven(zDD) and Split Merge with data driven (SM+zDD) sampling method is observed. Feature matrix is initialised with 5 unique features shared by each sequence. The model is executed for 300 iterations for both the settings.

5 Results

Figure 12 shows the segmentation of the activities. We used the data from 4 subjects S1, S5, S6, and S8. Different set of activities were taken for these subjects, for instance S1 performs WalkDog followed by Sitting and WalkTogether. Few activity are unique among the subjects (or sequences)
and few are shared. The result shows that the model is almost able to classify all the activities. We tried the BP-AR-HMM model with different feature matrix initialisation such as 3 or 4 or 5 features unique per sequence, with data driven sampling approach (zDD) and it gives the similar segmentations in all the 3 settings. This verifies the claim of the paper that the model is non-parametric approach. However, we could observe that the the model does not perform well, when the feature matrix is initialised with 1 unique feature shared among all the sequences. Figure 10 and Figure 11 is for hamming distance accuracy and joint log probabilities respectively. Hamming distance gradually decreases and has similar curve which is observed in the CMU dataset. Different input representations (euler angle and 3D axes angle) did not create much difference in the segmentation.

![Fig.10. Hamming distance plotted across number of iterations. It decreases and eventually settles down to 0.22](image)

![Fig.11. Joint log probability of data and sampled variables against number of iterations.](image)

6 Conclusion

Human 3.6m dataset [2] is used to test the robustness of the BP-AR-HMM model. It can be concluded that this model can be applied on Mocap data to get the segmentation or classification. Also, the claim that the model is non-parametric is verified. Auto-regressive model perform better than the Gaussian model. Data-driven sampling classifies better than Split-Merge. There is no major difference in performance when using axis angles and euler angle input representations.
References


3D Pose estimation under real world conditions

Siddhartha Dutta¹ and Markus Kächele²

¹ s_dutta1@cs.uni-kl.de
² markus.kaechele@dfki.de

Abstract. Recently deep neural networks have shown impressive results in 3D human pose estimation problems estimating 3D joint positions from the image pixels. However, working with any deep network would need lots of data. In this work we have investigated, evaluated and improved a state-of-the-art 3D pose estimation model by enriching the data in various ways. We augmented the model with random 3D pose rotation, flipping and also added kinematics to create more 3D poses. Furthermore, we also added different levels of Gaussian noise to the data and evaluated its performance.

We find that adding random rotation makes the model better and adding noise converges it faster than the original model. The improved model also showed improvement in 11 out of 15 action scenarios in Human3.6M dataset.

Keywords: 3D pose estimation, pixel to 3D point regression

1 Introduction

Human Pose Estimation which is an active field of research, has found usage in many applications like in animations, computer games, surveillance, VR and AR. One of the active fields of research in this domain is about estimating the human pose from single monocular images. One important reason is because images and videos are the most widely available data currently in public domain. For a number of important applications, understanding the spatial representation is crucial. For example, in autonomous driving, the car/machine needs to understand the environment and it’s one form of the same problem - 3D pose estimation. However, estimating a human pose can be very challenging as there can be varieties of poses that can vary from person to person. Recently deep neural networks have been proved to be good in estimation problems and have performed better compared to the traditional approaches which included hand-crafted features edge direction histograms, SIFT descriptors, silhouettes etc.[2]

In order to estimate the spatial a 3D pose from an image, the algorithm that has been devised should be invariant to number of factors namely - background scenes, clothing and its texture, shape and skin color of persons. In context with deep learning, the models need huge amount of data to produce good result. Unfortunately, most data available in the public domain lack the appropriate 3D ground truth which makes this a difficult problem. We’ve used the deep learning model by Martinez et al. [2] to start as a starting point for our work. She also claims a lower error rate around 30% lower than the state-of-the-art systems. Our contribution is more around investigating and evaluating the existing system and then finally improve the system. We use a number of approaches to on the degraded and incomplete data and try to identify ways to improve the predictions.

1.1 Mathematical Model and Network Design

Even though 2D data carries less information, it is their lower dimensionality that makes it a good choice to work with. For example, the entire dataset of Human3.6M can be kept in the GPU memory for training. The main goal of the problem was to - given a 2D image input, estimate the 3D space.
Fig. 1: Adapted from [2], the figure shows one block of the network architecture. Each 2D input is passed through a linear layer and then batch normalised, dropout with RELU as an activation function. This is again repeated with an added residual connection. This entire structure forms the building block which is again repeated. The final system outputs 3D joint positions.

In mathematical notations, input an array of 2D points $x \in \mathbb{R}^{2n}$, the resulting output should be series of 3D points $\in \mathbb{R}^{3n}$. The deep network aims to learn this function $f^\# : \mathbb{R}^{2n} \rightarrow \mathbb{R}^{3n}$ and minimises the error over the data set.

$$f^\#: \min_f \frac{1}{N} \sum_{n=1}^{N} \varphi(f(x_i) - y_i)$$ (1)

Here, $f^\#$ is a deep network that is scalable and performs well in approximating the function. Figure 1 shows one block of the network architecture which has been repeated twice. The network is a multi-layer deep network using batch normalisation, dropout 0.5, two residual connections and RELUs used as activation function.

Also present are two linear layers applied to the input directly to increase the size to 1024 and one just before of final prediction of size $3n$. So there are six linear layers in all and 2 residual blocks and the model containing 4 to 5 million trainable parameters. RELUs, as with most deep learning approaches, has been a standard choice for adding non-linearity. Residual connections on the other hand has shown general improvement in performance and reduced training time. Martinez claimed to have 10% error reduction with residual connections using a learning rate of $1 \times 10^{-3}$ and exponential decay. Kaiming He initialisation was used to set the starting weights of the linear layers implemented in tensorflow. One forward and backward pass takes around 10ms on NVIDIA GTX 1080 GPU [2].

1.2 Dataset description

Out of many 3D pose datasets available, Human3.6M [1] is currently the most used dataset. It has around 3.6 million images of poses of actors doing 15 daily activities. They have been performed by 7 professional actors doing activities for example eating, walking, in discussion, smoking or making purchases etc. The 2D joint locations and the 3D ground truth positions of the same are available in addition to the camera projection parameters. These have been made available for all the 3.6m frames. In accordance with the usual protocol of Human3.6M, the subjects 1, 5, 6, 7 and 8 were used for training and subjects 9 and 11 were used for testing. The error reported is in millimeters which is the difference between the ground-truth and prediction across all joints [2].

1.3 Data pre-processing

All the 2D joint inputs and 3D joint outputs were normalised by subtracting the mean and dividing by the standard deviation. All the 3D joints were zero centered around the hip joint which is as per
the standard approach of Human3.6M. According to Martinez, it would make more sense to predict the 3D coordinates in the camera frame than to some arbitrary coordinate system. This also makes more appealing as this would mean more training data and also restrict overfitting to any global frame. Any 3D model can be represented in a particular camera space by rotating and translating the data. The 2D predictions can be created using the state-of-the-art stacked hourglass network or using a projection matrix. While the author used either stacked hourglass[3] or camera projection, we have cross-tested the network both on stacked hourglass and using camera projection matrix.

2 Our approach

Within the scope of this project, we have tried to investigate the existing model for unconstrained and incomplete real-world data. As discussed above Martinez et al., used the Human3.6M dataset to train the network, however they have not used any data augmentation methods that can potentially improve the performance of the model. We have rotated, flipped and translated each 3D model per frame. Apart from these we have also added Kinematics and induced Gaussian noise to test the performance of the model.

2.1 Rotating the 3D trajectory

Rotation of the model in random angle would be an ideal candidate for data augmentation. However, if we rotating every 3D model per frame in any arbitrary angle will create inconsistency among the same action in different frames. Hence we take all the frames of a particular action, find the average point and rotate all the frames along the point around some random angle. Fig 2 shows the trajectory of hip joint of each frame of a particular action before and after rotation.

Fig. 2: This figure shows how the rotating of trajectory has been done. (a) shows the action Phoning for subject that has been rotated by -88 degrees. Similarly the same has been done for other actions for Walking and Eating shown by (b) and (c).

2.2 Flipping the model

Flipping of the 3D model is done frame by frame. However, similar to rotation, is done across a local XZ plane. This is because doing it across any arbitrary axis may take it out of the camera bounds. See Fig 3. for reference.
2.3 Augmenting with Kinematics

Adding human kinematics to the model can also create more data. Here we implement dynamics of the human arm into the model. Each upper limb of human is powered with 7 DOF. We have included the kinematics of elbow flexion/extension and shoulder abduction/adduction to our model. For example, the wrist joint can move freely in the range of 2 to 115 degrees in the plane formed by the wrist, elbow and shoulder. This is called elbow flexion/extension. Similarly humans can also perform abduction/adduction of the shoulder in sagittal plane. For each frame, we randomly select an angle from 0 to 15 degrees and then rotate and translate the joints keeping in consideration the restrictions. We have kept the angle small as giving bigger angles might not align with same action annotation. Fig. 4 shows the kinematics in details.

Fig. 3: Flipping of the 3D model around the local axis in XZ plane. The first image is the original data captured and the second one is the flipped one.

Fig. 4: The first figure shows flexion applied to elbows. The second figure shows abduction/adduction applied to the shoulders of both hands. The green joint shows the part of the skeleton that has been modified.
2.4 Adding Gaussian noise to the model

We added Gaussian noise to the joints and wanted to find out how the model behaves with introduction of different levels of noise. We added selective levels of Gaussian noise to different joints and corrupted the 3D model. For example, the maximum standard deviation was applied to the wrists and ankle joints as they are have more probability to be corrupted, while the hip joint was given least considering they have the least probability to noisy. Randomly 0-4 joints were selected and were moved from the original position with the selected standard deviation and zero mean. Thereafter we projected the noisy 3D model to get a noisy 2D projection of the same. The model was trained with the noisy 2D projections and the original 3D model. Fig. 5 shows more details about corrupted frames.

![Images of corrupted frames](image1.png)

Fig. 5: The figures showing how different joints have been corrupted. The shift difference between the green and the red shows how the 3D skeleton has been modified after adding noise.

3 Experiments

With our approach defined in the previous section we trained the model in various scenarios and evaluated how it behaves upon training on the data it hasn’t seen before. The performance was noted down.

3.1 Original model vs Augmented with Random Rotation vs Flip vs Kinematics

Using Kinematics, the data increased by two folds, while rotation and flip doubled the entire data. While flipping the model around the average camera point didn’t show any noticeable performance improvement, the model trained with Rotation and Kinematics showed better results. Each of the models were trained for 100 epochs. See Figure 6 for more details.

3.2 Model trained with different levels of noise

We also trained the model after adding different levels of noises, where we corrupted 0-4 joints per frame randomly. As discussed in section 2.4, different levels of noise levels were introduced, we introduced corruption in 10%, 25%, 50%, 75% and 100% of the frames. Using the noisy 2D projections and the original 3D model, different models were trained. The models with higher noise level showed a slight improvement in performance. Each of the models were trained for 100 epochs.
Fig. 6: (a) Different augmentation of data were performed. Rotation and Kinematics performing better, whereas Flipping is worse. (b) Different level of Gaussian noise added to data with Rotation and compared to the original model. Model with 100% Gaussian noise performing better. (c) Kinematics model with different level of noises and compared to the original model. Model with 50% Gaussian noise performing better.

3.3 Mixed Scenario 1: Rotation with 50% noise vs Rotation with 100% noise

After the individual training we trained the model with mixed varieties of data. Since rotating the data worked well, we wanted to fuse it with the noise. We trained the model with Rotation and 50% noise vs Rotation and 100% noise. We found out that the model with 100% noise and rotation converged quicker than the original model. Refer to Fig. 6 for more details.

Fig. 7: (a) Different levels of noise are added to the data. (a) 10% (b) 25% (c) 75% (d) 100%. With increase in noise level, the model tends to perform better.

3.4 Mixed Scenario 2: Kinematics with 100% noise vs Kinematics with 50% noise

The final mixed scenario was to train the model with Kinematics data with the noise. Similar to the mixed scenario 1, we also trained the Kinematics model with 50% and 100% noise variety. Here however we found out the model with 50% noise performed better than the 100% noise version.

3.5 Testing on Stacked hourglass network

We also wanted to explore how the original model behaved with the stacked hourglass predictions [3]. Stacked hourglass is a state-of-the-art neural network based on the successive steps of pooling and up-sampling. It can estimate key 2D points of human body based on images. We trained the model with stacked hourglass predictions and tested with camera projections of 3D model.
As expected this model had more error compared to the model trained with camera projections. However, during testing with camera projections, the test error was lesser compared to validation error during training. We provide a joint-wise error report for the same.

![Image](image1.png)

![Image](image2.png)

Fig. 8: (a) Joint wise evaluation of the best performing model with the original model. (b) Action-wise comparison of the the best performing model with the original model.

### 3.6 Statistics of Best Performing model

We tested the best performing model (Rotation with 100% noise) in two cases. The first test was made to find out the joint-wise evaluation of our best performing model. It performed better 11 out of 16 cases excepting Spine, Neck/Nose, RShoulder, RElbow and RWrist. The statistics can be found in Fig 8(a).

The second test was to find the error in action-wise based on Human3.6M dataset. The dataset has 15 different scenarios - Directions, Discussions, Eating, Greeting, Phoning, Photo, Posing, Purchases, Sitting, Sitting Down, Smoking, Waiting, Walking Dog, Walking and Walk Together. 11 out of 15 of these performed better with our model, leaving behind Eating, Sitting, Walking and Walking Together which had slightly more errors. Refer to Fig. 8(b) for more details.

### 4 Conclusion

In this project we have investigated the base model by throwing at it different data that the model has not seen before. The network used was a deep neural network trained on Human3.6M dataset. We used this dataset to create more data by augmentations, kinematics and also by adding noise, which has not been seen by the model. We also evaluated each case and also mixed scenarios and noted down the results. While Kinematics also did good, we find that augmenting with Rotation increased the performance of the network and adding noise made it converge faster. Though there hasn’t been a significant increase in the performance, the model trained with Rotation and 100% noise showed better results compared to the original model. We also evaluated the model action-wise and found improvement of performance in 11 out of 15 actions.

As future possible work, it would be interesting to see how the model would also behave by adding more kinematics to other possible joints of the human body. Furthermore, if action accuracy is needed to be improved, a LSTM can also be used.
References


Survey on Scene Flow with Different Sensor Modalities

Oskar F. A. Frohn\textsuperscript{1} and René Schuster\textsuperscript{2}

\textsuperscript{1} ofrohn@rhrk.uni-kl.de
\textsuperscript{2} rene.schuster@dfki.de

Abstract. Aim of this paper is to gain an overview of state-of-art scene flow techniques, whereby we are particularly interested in the various sensor modalities. It should be noted that our survey excludes classical stereo-vision systems, as they go beyond the scope of this survey. We shortly discuss the benefits and applications of camera, RGB-D, LiDAR, RADAR and sonar sensors, while we later summarize recent algorithms for scene flow estimation based on monocular image sequences, depth-maps and point clouds, where we show that the recent advancements in deep neural networks, lead to an significant accuracy and performance improvement.

Keywords: Scene Flow, Sensors, Survey, Monocular-Vision, RGB-D, LiDAR, RADAR, Sonar

1 Introduction

In today’s world, the usage and research of autonomous vehicle is ever increasing. One crucial aspect of operating such vehicles is the question, of how they perceive their surroundings. Over the years there were a multitude of different proposed sensors, starting with simple one dimensional range ultrasound finders, up to complex camera-, LiDAR- and even RADAR systems. Each of these systems handles differently and generates different data. They therefore also need different processes for analyzing the data. One of the processing methods is called scene flow estimation, which tries to calculate the movement of measured points in 3D space. The benefit of scene flow estimation is, that if the measurements are dense and accurate enough, it allows for a precise representation of the surroundings of a vehicle and therefore aids in projects like self-driving cars. In this survey, we summarize the most recent approaches in scene flow estimation algorithms and are going to look at the advantages and disadvantages of different sensors which can be used to generate scene flow compatible data. This being said, we want to clarify that for organizational reasons, we will not focus on classical stereo-vision systems, or research that incorporates depth or motion estimation without scene-flow estimation.

2 Concept of Scene Flow Estimation

According to [26] scene flow is a three-dimensional flow field, describing the motion of three-dimensional points in thee-dimensional space, which can be presented as sparse-, dense- or even parametric data. It is similar to optical flow, which describes the projected two-dimensional motion of three-dimensional space. The core problem definition of scene flow can be formulated as (1), where for each point $P = (X, Y, Z)^T$ we want to know the relative motion $V$.

$$V = \begin{pmatrix} \frac{dX}{dt} \\ \frac{dY}{dt} \\ \frac{dZ}{dt} \end{pmatrix}$$ (1)

This results in three unknown parameters for each point $P$ in the scene, which makes it an inherently ill-posed problem, as the equations do not have an unique solution. To overcome this hurdle, one
can use regularization in order to obtain a solution. Regularization in principle just means, that an equation is added as a constraint \[29\]. Furthermore when speaking of solving scene flow equations, literature often mentions the formulation of an "energy equation". The reason behind this is that energy equations are a powerful representation method for optimization problems in graph theory as elaborated by \[5\]. The actual formulation of these energy equations and their utilized regularization varies heavily depending of which kind of data is used and which types of sensors were used to produce it.

3 The Different Sensor Modalities

Therefore we will now shortly present the different sensor modalities used to generate depth-information containing data, which can therefore also be used to estimate scene flow, furthermore Table 1 provides a rough overview on what properties are to be expected from common used sensors.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>FOV</th>
<th>Points per Measurement</th>
<th>Depth Range</th>
<th>Weather</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotating LiDAR</td>
<td>360° × 30°</td>
<td>10k - 300k</td>
<td>5-20</td>
<td>2 - 120m</td>
<td>No Rain/Snow</td>
</tr>
<tr>
<td>Flash LiDAR</td>
<td>10° × 5°</td>
<td>16k - 1.3m</td>
<td>12 - 100</td>
<td>5 - 1500m</td>
<td>No Rain/Snow, No direct Sunlight</td>
</tr>
<tr>
<td>Depth Camera</td>
<td>70° × 60°</td>
<td>25k - 200k</td>
<td>30 - 60</td>
<td>0.1 - 10m</td>
<td>No Rain/Snow, No direct Sunlight</td>
</tr>
<tr>
<td>Camera</td>
<td>70° × 60°</td>
<td>170k - &gt;6m</td>
<td>10 - 120</td>
<td>0.1 - 40m</td>
<td>No Night</td>
</tr>
</tbody>
</table>

3.1 Camera

At the moment there is a plenitude of different camera modalities for scene flow estimation, ranging from monocular over stereo up to multi-view systems. Each comes with different benefits and downsides. But generally speaking, camera based scene flow estimation works by performing optical flow estimation for each camera and then use the information about the camera’s position in order to estimate the scene flow. A specialty of camera based systems in general is, that they have to choose between applying regularization inside the optical flow estimation or inside the scene flow estimation itself \[26\]. The most efficient and common used sensor modalities, are stereo vision based systems. Unfortunately these are exempt from this survey. That being said, camera based systems mostly are capable of generating more dense depth fields than LiDAR, with the downside of being computational expensive, as well as only useful for short range estimations (< 50-100m).

3.2 RGB-D

Another common used sensor, are RGB-D sensors. These work by typically matching RGB data with depth data. The depth data can then be used to generate depth maps. These depth-maps have the benefit that they convert the scene flow problem into an optical flow problem and therefore reduce the complexity of estimating the scene flow. An example of one of the most common systems used is Microsoft’s Kinect, which acquires the depth data through projecting an infrared mesh and measuring its deformation. The downside of these systems on the other hand is, that the depth-sensor only has a short range (< 10m) and is deceptable to other (infrared)-light sources.
3.3 LiDAR

Light Detection And Ranging (LiDAR), also called LAser Detection And Ranging (LADAR), is one of the more common systems used in 3D-Vision. In its most basic form, it works by emitting a laser beam and measuring the time until its reflection is detected, which allows it to calculate the distance the light pulse traveled. The work of [6] does a good synopsis on the different types of LiDAR systems. The data generated from LiDAR depends on which type was used. Rotating LiDARs often create a point cloud without height, as the laser only scans the same height, while array based LiDAR other hand are capable of creating fully point clouds, that contains height information. Problems of LiDAR systems as mentioned by [4] are their bad performance in occluding weather, their bad performance in close range (<2m) and their inability to extract direct velocity or texture information, as well as, depending on the used system, having moving parts or low-resolution or being problematic in bright daylight, as they might use infra-red light.

3.4 RADAR

Radio Detection And Ranging (RADAR) can also be used to obtain point cloud data, as shown by [4], which presented a system that was designed to be an alternative to LiDAR. The benefits mentioned by [4] in contrary to LiDAR are, that it is not weather dependent and even capable of extracting direct velocity information, also some systems have worse angular-resolution and measurement than its LiDAR counter parts. Interestingly however is that regardless of the benefits of RADAR, it does barely appear in research as an independent system for scene flow estimation, and is more commonly used as an additional range finding device (at least in the field of autonomous vehicle navigation). Also recently, the term of echoic flow was coined by [24], which is based on optical flow but extended to time of flight systems by introducing collision intensity.

3.5 Ultrasound/Sonar

Ultrasound and sonar are both time of flight systems, that can be used to measure distances. Especially sonar keeps to be one of the most important sensors for water-based navigation. While it can be used to calculate the echoic flow [24], we could not find any research on scene flow estimation using this sensors. None the less, we want to mention that it is possible to generate 3D data from 2D sonar images as shown by [20].

4 Scene Flow Estimation Methods

We base our literature research on a prior survey of [28], which does an intensive evaluation of different scene flow procedures. They present the basic energy models, and how said models change based on different input-data-structures, like multi-view, binocular, depth or light-field. Furthermore they compare the more prominent data-sets and evaluation metrics, as well as giving final advice on which method to choose. In the following, we therefore want to build upon their work and present a small selection of more recent research, which nicely encapsulates the improvements made in the field (Table 2).

4.1 General Camera/Depth Scene Flow

A good entry point to begin with, is the approach by [13], they present a general framework, that works with any combination of camera or depth sensors, uses filtering instead of regularization and is capable of handeling occlusion. The algorithm works with a sequence length of two frames,
also preliminary experiments showed that the increase in sequence length dramatically reduces errors. They formulate the scene flow estimation problem as a probabilistic model, which aims to estimate the most likely points of structure and velocity based on an assumed probability distribution and a given set of input observations. They calculate the prior distribution from the posterior of the previous frame and the likelihood is calculated based on the given camera or depth data. They chose the scene particles based on ray resampling [13], while simultaneously ignoring particles that have a high chance of being occluded. Furthermore the algorithm is tested on a multitude of different datasets and different categories, ranging from depth/multi-view comparisons up to 3D hand tracking. Their results often outperform compared algorithms and they are certain that a performance increase from currently 500 seconds per frame up to real-time is possible.

4.2 Monocular Scene Flow

Continuing, as mentioned in Section 3.1, one of the most effective camera based algorithms currently utilize stereo-vision systems. That being said, it also possible to estimate scene flow from monocular images. The current state-of-the-art algorithms of that kind, mainly focus on generating depth maps from image sequences [21] [10]. While, we later discuss algorithms which focus on estimating scene flow from depth maps, we again are limited to only cover publications, which directly involve scene flow estimation. Because of that limitation, the actual selection of papers is quite sparse.

To begin with, we want to shortly summarize the work of [11]. They are the first to present a general approach for the non-rigid scene flow estimation from monocular image sequences. Their framework achieves this without any prior assumptions and its suppose to work even under partial occlusion, but it requires a sufficient diversity of deformations and camera motion. Their approach estimates a scene flow between 2 frames, also these frames have not necessarily to be consecutive as the frame selection is done by their pre-processing step. In their pre-processing step they apply a redundancy removal where two frames are compared and discarded if they are to similar. Furthermore a foreground-background segmentation is applied in order to calculate the translation. The scene flow calculation is then done by computing a prior selected measuring function and factorizing it into non-rigid shapes and motion. They evaluate their algorithms on a multitude of different data-sets. It is pointed out that the algorithm works well on scenarios where a foreground-background segmentation is easy. Unfortunately the implemented algorithms are not usable for real-time computation, as a 40 frames $486 \times 366$ sequence takes 802 seconds to compute. But it is said, that the presented framework should be modifiable enough so, that it can achieve real-time performance.

Another approach we want to present is done by [27], where they also use the variational method for their scene flow estimation. It does require calibration and should deliver good results, even under partial occlusion. Their approach is able to estimate scene flow with a minimum of three frames under classical optical flow assumptions. They first apply an occluded map computation, before continuing on solving their energy minimization problem using coarse-to-fine variational equation on Gaussian pyramid, where the depth is estimated through a inverse depth calculation which needs to be calibrated at frame zero. They test their approach with a multitude of different environments. Most interesting for us, is their evaluation on the KITTI [8] benchmark, where they achieve comparable results to state-of-time stereo vision algorithms. Unfortunately no computational times are mentioned.

4.3 Depth-maps and RGB-D Scene Flow

Following, as mentioned above, it is also possible to estimate scene flow from depth maps. These depth maps can be generated by RGB-D sensors like we presented in Section 3.2 or be the product of an algorithm using different sensors.

We first want to mention an approach by [25], which was already covered in [28], but it is interesting, as it is very simplistic. Rather than calculating the scene flow by assigning each pixel in an image its own depth. Their method segments the image into layers based on the depth and then just keeps track of these layers. This approach therefore handles occlusion way better than other methods, as well as having a faster computation time.

Another approach worth mentioning is presented by [14], where there are not only able to calculate the scene flow based on RGB-D cameras but also the ego-motion of the system within 80ms per estimation. They achieve this using a clustering method. This has the benefit of reducing the amount of necessary regularization and occlusion estimation, but comes at the price of being less accurate. Given two frames, the frame’s 3D points are separated into 24 clusters, whereby these clusters are treated as rigid objects and are tracked using their centres. After a cluster-based foreground-background segmentation, the dynamic clusters are selected and used to solve the energy minimization equation for scene flow and ego-motion estimation. The scene flow estimation was then tested using a moving and a stationary RGB-D camera, which captured indoor images with a resolution of $240 \times 320$. Their calculation time was only beaten by PD-Flow [15] on GPU calculation. Furthermore it was mentioned that their approach could be improved with a faster segmentation method, as well as a more accurate clustering method.

And finally we want to present the approach developed by [22]. They are the first to developed a convolutional network for scene flow estimation based on RGB-D data. Their so called SFNet is capable of real-time end-to-end estimation where it takes a RGB and a depth feed and turns them into an scene flow map. Their networks uses the following structure: In the first layers the RGB and the depth input of two consecutive frames are convoluted separately in order to extract their features. They are then merged in a correlation layer and furthermore downsampled until they reach the lowest resolution. Afterwards their are upsampled again through transpose convolutional layers, with additional input from their corresponding downsampling layers. As for their loss function, they decided to not only include the scene flow error, but also the brightness error, where the brightness error is calculated through image warping. They test their algorithm on a multitude of different datasets where it outperforms most algorithms, among others modified version of FlowNetS and FlowNetC [7], where these optical flow algorithms where adapted to handle RGB-D images.
More specific [22] compared their algorithm with [25] as well as the above mentioned PD-Flow algorithm. While SFNet is faster than both, the SFNet had a significantly higher root mean square error than the algorithm by [25], for the Middlebury dataset [23].

4.4 LiDAR Scene Flow

Finally we are going to look at the most recent (2019) algorithms concerning scene flow estimation from point clouds. The papers we are going to present are all using deep neural networks (DNN), which have only recently been applied to point cloud data sets. While each of the three networks work completely differently, their abstract structure can be summarized as described in Figure 1.

![Diagram of deep neural network structure](image)

**Fig. 1.** General structure of the deep neural networks estimating scene flow from point clouds

We first want to look at [2]. They solve the scene flow estimation using a deep neural network, called PointFlowNet, from end-to-end on unstructured point clouds. Their network not only estimates the scene flow, but also does object detection and local rigid body motion estimation. It has the following structure: First a feature encoding layer, where the input is grouped into voxels, followed by a convolved context encoding layers. Then comes a convolutional layer, which is split into three parallel parts, an ego motion regressor, a scene flow decoder and a 3D object detection decoder. The network was trained and tested with the KITTI object detection dataset [9], as well as an augmented version with more virtually inserted cars. Additionally they compared their estimated dense scene flow with their estimated sparse rigid motion and came to the conclusion that besides a qualitative improvement no significant accuracy improvement occurred.

Another approach at using an end-to-end deep neural network is presented by [17], which works with unstructured point clouds as well. Their FlowNet3D operates in three steps. In the first step they do a hierarchical point cloud feature learning, which clusters nearby points and extracts their local feature. Then in the second step, they use a flow embedding layer, which learns to identify similar features and encode their motion. And finally in the last step, they train their last convolutional layer to do a flow refinement. They train their network on the virtual data of FlyingThings3D [18] and later test it on the KITTI scene flow dataset [19]. When fine-tuned their model with the first 100 frames, they were capable of outperforming the compared methods.

And last but not least we want to present the work of [12], which also created a DNN called HPLFlowNet, which is capable of large-scale (86k points per frame) end-to-end estimation on unstructured point clouds, as well as finding structure within these clouds.
Their method works by using a variation of the Bilateral Convolutional Layer (BCL) [16] and the permutohedral lattice [1]. Instead of clustering or sub-sampling the input data, they map them onto the permutohedral lattice and then downsample them using multiple BCLs. Afterwards multiple BCLs are then used again to do a hierarchical upsampling and remapped to the input cloud. They trained and tested their network on the FlyingThings3D dataset, and also tested it on the KITTI scene flow dataset, where it outperforms all other state-of-the-art scene-flow estimation methods, which shows its great generalization ability.

In more detail, in [12] HPLFlowNet is compared with FlowNet3D on the KITTI scene flow data set and the FlyingThings3D dataset. In all cases the FlowNet3D was significantly outperformed. Furthermore concerning the comparison of PointFlowNet with HPLFlowNet, while a direct comparison is not possible as both papers used different datasets, both do compare themselves against ICP [3], with ICP delivering near identical end-point-error results for the different datasets. Therefore using ICP as a reference, a comparison is possible, where it seems that HPLFlowNet also outperforms PointFlowNet.

5 Conclusion and Future Work

We have seen that it is possible to do scene flow estimation from a monocular image sequence using variational methods and we are quite certain that the utilization of deep neural networks will improve this process in the future. As for right now, numerous approaches already exists that use DNN for optical flow and depth estimation on monocular image sequences. Furthermore we have seen how the process of scene flow estimation from RGB-D data has been highly improved using convolutional networks and we are interested how the usage of Generative Adversary Networks or other complex networks might further improve that process, like they have been for stereo-vision or LiDAR based systems. Another interesting approach would be to use a combination of different sensors systems for scene flow estimation, like there are already used for object detection, or to see more experimentation with nontraditional sensors like RADAR or sonar. And finally it would also interesting to see how improvements in similar fields like optical flow estimation or object tracking could be transferred to improve scene flow estimation.

References

Abstract. This paper represents our approach to estimate dense 3D human pose only from 6 IMU sensor orientation data using Deep Learning. There are many existing popular methods to solve inverse kinematic problem which mostly rely on iterative optimization to seek out an approximate solution. But their approach is infeasible for real time prediction. With deep learning real time prediction becomes possible. Our work is motivated from Deep Inertial Poser and we have used datasets published by them. Our approach can produce relatively convincing results from only orientation data from 6 IMUs. Instead of rotation matrix we used quaternion to represent orientation and pose which reduced the network input size.

Keywords: 3D Human Pose, IMU, Deep Learning

1 Introduction

3D Human pose estimation has been an active field of research among computer vision community for decades. Although with the help of deep learning recent camera based methods can generate satisfactory result, it becomes infeasible and fails in the wild or outdoor setting due to occlusion and other environmental factors. In contrast wearable sensors like IMUs are small, lightweight and low-cost and give measurement of orientation and acceleration. So they can facilitate capturing real time outdoor activities in a more natural way. However, to estimate full human pose, at least 17 sensors need to be attached at different body segments which makes this approach non-viable for complex motions. So our goal is to predict full human pose from sparse IMUs (i.e. IMUs only placed at body extremities) using deep learning. More specifically subject needs to wear only 6 IMUs at left & right lower legs, left & right arms, head and pelvis. This is an under-determined problem and so we impose anatomical body constraints (SMPL [3] body model) to discard impractical human poses. Besides, there exist a temporal dependency in human motion sequences like walking, jumping, eating or any particular limb movements. So Recurrent Neural Network can be an effective approach towards this. Our work is mainly inspired from Deep Inertial Poser(DIP)[2] and for this task we have used their published datasets. We have performed some experiments with different approaches and compared their qualitative and quantitative results.

2 Relative Work

Our work starts with the work represented by Deep Inertial poser[2] which tried to address this problem and came up with a novel deep neural network capable of reconstructing full human body pose from 6 IMU sensors at real time.

The end to end pipeline of DIP[2] can be described as follows: 6 IMUs are placed on the subject body at head, pelvis, left and right arms, left and right lower legs. They give raw measurements of orientation and acceleration. These raw data are first calibrated to SMPL[3] reference frame. Then all sensors are normalized with respect to the root sensor which results into 5 sensor readings (root sensor reading becomes identity). This calibration and normalization is done for all the frames in the sequence. The orientation is represented by 3 X 3 rotation matrices and acceleration comprises
of three components \((x,y,z)\). Orientation \((5\times9=45)\) and acceleration \((5\times3=15)\) from 5 sensors are concatenated resulting a single long input feature vector of size 60. This is fed into the network (fig 1) which can predict pose parameters for 15 joints. As reported in the DIP paper, during training their network had access to full sequence but we found that in the published DIP.IMU.nn training set, all activities were split into chunks of 300 time-steps which arises confusion if their network was trained over full sequence or in chunks. However, for evaluation they preferred window based approach with 20 past and 5 future frames. So to predict pose at time-step \(t\), the input window contains total 26 frames starting from \(t-20\) up to \(t+5\) frames. DIP has provided their best performing two pre-trained models. One was trained on synthetic data and another one was further fine-tuned on real DIP.IMU data. We performed test on these two models with the activities from their DIP.IMU.nn test set and the quantitative result in terms of mean joint angle error is shown in fig 2. The result ascertains the effectiveness of fine-tuning on DIP.IMU data which mitigates the gap between synthetic and DIP.IMU real data distributions and difference in motions.

![Network Architecture](image)

Fig. 1: Network architecture of Deep Inertial poser [2]: Bi directional stacked two layered LSTM with two fully connected layers at the two ends.

![Comparison of Model Performance](image)

Fig. 2: Error of DIP models on DIP.IMU.nn test activities
2.1 SMPL body model

Skinned Multi-Person Linear model (SMPL[3]) is a skinned vertex-based model that accurately represents a wide variety of body shapes in natural human poses. It is parameterized by 72 pose, and 10 shape, parameters $\theta$ and $\beta$ respectively, and returns a mesh with $N = 6890$ vertices. Here, in our case, we are interested in pose parameters which are represented as axis angle form (3 components) for total 24 joints ($24 \times 3 = 72$). As aforementioned DIP model predicts 15 joints, to map into SMPL the remaining joints are assumed as equivalent rotation vector of identity rotation matrix. In fig 3 marked joints are predicted by the model.

![Fig. 3: SMPL needs 24 joints. The marked joints are predicted ones.](image)

2.2 Orientation representation

There are several methods to represent orientation in three dimensions.

**Euler angle:** The first attempt to represent an orientation was owed to Leonhard Euler. Euler angles provide a way to represent the 3D orientation of an object using a combination of three rotations about different axes. In aerospace they are known as yaw-pitch-roll rotation ($z - y' - x''$). So this is parameterized by three variables. Euler angles are limited by a phenomenon called “gimbal lock,” which prevents them from measuring orientation when the pitch angle approaches +/- 90 degrees.

**Axis-angle representation:** This representation describes a rotation or orientation using a unit vector aligned with the rotation axis, and an angle indicating the magnitude of rotation about the axis. This form is equivalent to more concise form known as rotation vector which is obtained by multiplying the angle with the unit rotation axis vector. It also has 3 parameters.

**Rotation matrices:** The product of two rotation matrices is the composition of rotations. Therefore, the orientation can be given as the rotation from the initial frame to achieve the frame that we want to describe. The rotation matrices are 3 X 3 orthogonal matrices and it has 9 parameters.

**Quaternion:** Another way to describe orientation in 3D coordinate system is unit quaternions which is a four element vector and is composed of one real element and three complex elements. Compared to Euler angles they are simpler to compose and avoid the problem of gimbal lock. Compared to rotation matrices they are more compact, more numerically stable, and more efficient.
3 Our Methodology

3.1 Data preparation

For our work we have used the datasets published by DIP.IMU. The dataset can be downloaded from http://dip.is.tuebingen.mpg.de/pre_download. At the time of writing this paper, three main folders were available. DIP.IMU: This was their recorded uncalibrated raw IMU data from 17 Xsens sensors worn by 10 subjects. DIP.IMU.nn: This was calibrated and normalized version of the activities from DIP.IMU. This was further split into training set consisting of 40 activities, test set consisting of 18 activities and validation set consisting of 3 activities. Synthetic60FPS: This was uncalibrated IMU data synthesized at 60 frames/sec mainly from CMU, H36 and AMASS dataset.

**Calibration** Before feeding into the model raw IMU data must be transformed to our reference target SMPL frame $F^T$. For the detailed calibration procedure, we contacted with authors of DIP and tried to implement it following their direction which is summarized below.

All rotation matrices given below should be understood as $F^{AB}$: rotation from frame A to frame B. We will talk about four frames denoted as follows T: SMPL frame, B: bone coordinate, I: inertial frame, S: sensor local coordinate. Fig 4 gives the visual understanding of the coordinate frames involved in the entire calibration process.

1. IMU provides absolute orientation of each sensor relative to a global inertial frame $F^I$ in terms of $R^{IS}: F^S \to F^I$. So our goal is to find the mapping $R^{TI}: F^I \to F^T$, relating the inertial frame to the SMPL body frame.
2. All IMU readings can then be expressed in the SMPL body frame by the following equation $R^{ TS} = R^{TI} R^{IS}$.
3. Here $R^{TI}$ is constant over all frames and calculated as $R^{TI} = inv(R^{IS} R^{SB} R^{BT})$ where $R^{IS}$ is considered as the head sensor readings at 1st frame assuming head sensor is aligned with SMPL axes and value of $R^{SB}$ and $R^{BT}$ needs to be known. Here $inv(\cdot)$ denotes matrix inverse.
4. In the first frame of each sequence, each subject stands in a known straight pose with known bone orientation $R_0^{BT}$.
5. We compute the per-sensor bone offset as $R^{BS} = R_0^{BT} R^{TS}$.
6. The last step is to calculate $R^{TB} = R^{TS} inv(R^{BS})$ for every frame. The output $R^{TB}$ can be interpreted as the bone orientations measured by the IMU in SMPL frame. Here $inv(\cdot)$ denotes matrix inverse.

**Normalization** The calibrated IMU data are further normalized with respect to root sensor to make the motions invariant to any directions. Below equation calculates the normalized orientation $R_s^{TB}(t) = (R_{root}(t))^{-1} R_s^{TB}(t)$, while $R_{root}(t)$ denoting the orientation of the root at time step $t$. 

![Fig. 4: Calibration steps from sensor coordinate to SMPL coordinate](image)
and $R^B_s(t)$, the orientation of the bone corresponding to sensor $s$ at time step $t$. Thus the root sensor readings becomes identity matrix and so discarded and remaining 5 sensor data are used for training.

### 3.2 Our approaches

**1st approach** As mentioned earlier, due to more numerical stability, we have chosen quartenion over rotation matrices. A rotation matrix is 3 X 3 square matrix and has 9 parameters while a quaternion has 4 parameters ($w,x,y,z$). So now orientation of 5 sensors is given as $(5 \times 4 = 20)$ 20 dimensional vector. So after concatenating orientation and acceleration our new input size reduces from 60 to $(4+3) \times 5 = 35$. Similarly prediction of 15 joints was represented by $(15 \times 4 = 60)$ 60 dimensional vector. Now we trained a similar Bidirectional 2 layered LSTM as in DIP with the required modification in input and output layer on synthetic H36 dataset. Some results on the unseen test data are shown in fig 5.

![Target and predicted frames from 1st approach](image)

(a) Correct  
(b) Erroneous

(c) Correct  
(d) Erroneous

Fig. 5: Target and predicted frames from 1st approach. Fig 5a & fig 5b represent correct & erroneous frame respectively of same activity. Same applies for fig 5c & 5d.

**2nd approach** When we visualized the prediction from 1st approach, we found that the predicted poses are near to ground truth pose but there was a lot of jittering. Next, we further eliminated acceleration from our input and predicted pose only from orientation of 5 IMUs. Now with 20 dimensional input we trained a model and found a very smooth satisfactory result. Acceleration data from IMUs are generally very noisy and so, if it is added in input, it needs to be handled in the loss function.
So rather than making it complex, we first tried a simpler approach of not considering acceleration as determining factor. With this we achieved far better results and also it eased the task of the network by reducing the number of parameters to learn. Some example results from this model are presented in the fig 7.

3rd approach Next, we have performed an experiment with similar input and output on a very simple 3 layered perceptron. The network architecture is given in fig 6. We wanted to compare the result from MLP with LSTM. A MLP is able to learn a non-linear mapping from input variables to output variable while LSTM network is mainly used to learn the temporal dependency for sequence data. The result from MLP was worth to consider and some glimpses of that can be visualized in fig 8.

Fig. 7: Target and predicted frames from 2nd approach. Compared to fig 5 error is much reduced and prediction became smoother without jittering.

Fig. 8: Target and predicted frames from 3rd approach. There were no jittering like fig 7 but predicted poses are less accurate.

Comparison of approaches Fig 9 shows the quantitative comparison of our aforementioned three approaches on left out test data (H36-S11) containing total 30 activities. It is clear that our 2nd approach outperforms others for all the activities. So BiLSTM network with only orientation input in quaternion form gave the best results so far to predict 15 joints in quaternion. As desired, MLP performs poor than BiLSTM network for this problem because human motion has temporal pattern between the subsequent frames which can be better learned by long short term memory network.
The performance of models are compared in terms of mean joint error across all frames for all activities.

![Comparison of our three approaches](image)

Fig. 9: Comparison of our approaches on left out test set H36-S11 containing 30 activities in terms of mean joint error in euler angle(degree).

### 3.3 SMPL Model Visualization

For the qualitative understanding of the prediction of our model the visualization of activities is very important. For that we adapted SMPL python packages ([http://smpl.is.tue.mpg.de/downloads](http://smpl.is.tue.mpg.de/downloads)) to our problem. To generate human poses at each frame we only change the pose parameters as per the prediction from our model and other parameters remain constant for any activity. We have used openCV[1] for visualization of frames and further merge into video.

### 4 Miscellaneous Experiments

#### 4.1 Single vs all dataset

As seen in the previous section our 2nd approach produced the best results, so we proceeded with it and trained a model across all synthetic datasets. With larger dataset, our model generalizes overall but the performance drops for any specific dataset. To illustrate, in fig 10, we compared model a: trained on H36 dataset with model b: trained across all datasets. Model a outperformed model b when tested on H36 test set but it performs worse when tested on different dataset, e.g AMASS Transition. This gives us an intuition of multimodal distribution across our synthetic datasets.
4.2 LSTM training strategy

Throughout the implementation of our experiments, we followed different training strategies for better learning of the model. Our synthetic datasets contain activities of varied sequence length in the range from 200 to 5500 almost. From some failed attempts, we found that the longer sequences are very much prone to destroy LSTM hidden and cell state learning. Empirically it was found that 200 sequence length was optimal to learn in our problem. So, we divided all the longer activities into chunks of smaller sequence length (200 in our case). We created batch of 10 activities of 200 time steps each randomly with replacement. We also tried stateful batch learning where for each new activity hidden and cell state are initialized by 0 and the intermediate state is maintained throughout the whole activity by initializing cell and hidden state from last timestep of previous batch. We observed that maintaining state did not improve our result due to comparatively long sequences for LSTM.

5 Challenges

This section explains the main challenges encountered in our task. While we tested the DIP_IMU dataset on the model trained on synthetic dataset, it failed. We could not get any interpretable output for DIP_IMU data from the same model that worked before for synthetic data.
5.1 Dataset distribution

The root cause lies in the different distribution of initial poses for all activities across all datasets. For correct calibration measurement, subject should start any activity either with T-pose or I-pose with known bone orientation in that pose. Fig 12a shows the output pose distribution at the first frame for all activities of different datasets. It is clearly visible that the first pose is not same for different activities in a single dataset and further the whole distribution of one dataset is far different than another. This effects our calibration procedure.

5.2 Concept shift

Concept shift is one type of dataset shift problem in machine learning. Dataset shift refers to the change in the distribution of data contained in train and test set that will lead to unreliable predictions. In our case, train set was synthetic data and test set was DIP_IMU real recorded data. Now in fig 12, we have highlighted three datasets, DIP_IMU, AMASS_ACCAD, AMASS_SSM.

Fig 12a shows they have similar output pose distribution but fig 12b shows they have different input orientation distribution. So the relationship between input and target variable differs which is known as concept shift problem. In ideal case after accurate calibration, dataset having similar pose should have similar input orientation. But in our case the problem was not solved even after calibration as visible in fig 12c.

6 Analysis of our approach with DIP_IMU data

6.1 Training

Further we verified our network and approaches on provided calibrated DIP_IMU nn data which is already split into train set(40 activities), test set(18 activities) and validation set(3 activities). For that we trained two models - short BiLSTM (Fig 11) and MLP(fig 6) on this data. The results ascertain that the problem discussed above being the reason of not getting any satisfactory results from our previous models for DIP_IMU data.

Fig. 12: Insights of dataset distribution
6.2 Comparison with benchmark

These two models were tested on the DIP.IMU.nn test set containing 18 activities. We have compared the performance of following four models, i.e. provided best two models of DIP, one trained on synthetic data, another finetuned on DIP.IMU.nn data, our trained short BiLSTM and MLP models trained only on DIP.IMU.nn data. The test was performed on 18 activities of DIP.IMU.nn test set. Some qualitative results are presented in fig 13. Out of these 4 models, state of the art model trained on synthetic data performs worst on this test set. We also analyzed the quantitative results and presented in fig 14. Clearly, their model trained on synthetic data gave highest error for almost all activities, and their finetuned model performs best. Our trained MLP comes in 3rd rank. Our short BiLSTM model can produce good results considering the limitation of dataset. Also it performed better in some activities than their finetuned model.

7 Conclusion

Towards the conclusion of this project, we have found that orientation and pose parameters presented in quaternion can reduce input parameters and cost computation. Further it simplifies the training
of model and stabilizes the weight parameters fast. We trained our model for maximum 30 epochs and it started converging. After that, eliminating acceleration improved our result from jittering. Due to temporal dependency in human motions, LSTM performs better than MLP. Considering synthetic data for training and DIP IMU data for test we observed that dataset suffers from concept shift problem even after calibration. The limitations of calibration procedure were not mitigated properly due to varied starting pose, unknown bone offset and different alignment of calibration sensor between synthetic and DIP IMU dataset. Our proposed model still performs better than DIP:synthetic model mostly for all activities and in some cases than DIP:finetuned model, when using their calibrated data of DIP IMUs.

References

Survey on Synthetic Data for Smart Traffic Analysis

Praveen Kumar Gummadi\textsuperscript{1} and Markus Kächele\textsuperscript{2}

\textsuperscript{1} p.gummadi@rhrk.uni-kl.de
\textsuperscript{2} markus.kaechele@dfki.de

Abstract. Deep neural networks are extremely hungry for data. For models based on images, hundreds of thousands of labeled images are required to train the network. Unfortunately, manual data collection and annotation with ground truth labels is both expensive and time-consuming. Synthetic data is information that’s artificially manufactured rather than generated by real-world events. So, instead of real-world data, synthetic data can be used to train the network. This paper presents some techniques used to generate synthetic data for respective datasets along with a comparison with standard benchmarks.

Keywords: Synthetic data, autonomous vehicle, traffic analysis, object detection.

1 Introduction

Training a neural network requires millions of images. Collecting and annotating images from real world events involves manual work. Task which include manual work are both time-consuming and costly. Synthetic data is artificially created rather than being generated from recordings of real-world events. Synthetic data can be used for training networks instead of real world data. For example, boundary & object detection, semantic labeling, depth estimation, optical flow, pedestrian, and traffic sign detection are several tasks in traffic analysis and autonomous driving systems where synthetic data can be used. Synthetic data for several tasks is shown in Fig. 1. There are several benefits of using synthetic data. Generating data is fast and very cheap. An unlimited amount of data can be generated as required and lastly ground truth labeled data can be generated for specific requirements based on context, even if there is no real data available for the same.

Normally, image related training data is generated from the real world. Cameras with high frame rate are used to capture real-world events. Frames are extracted from the video recordings and then images are generated. Once the data is collected, annotations were manually done to generate ground truth labeled data. Traditionally, most datasets were labeled by human annotators, utilizing the human perception system which is expensive and time-consuming. Some tasks with
capabilities of boundary detection [3], segmentation [3], semantic labeling [4,6] etc. Nowadays, with the increase in graphics and GPU processors, virtual environments can be used to generate synthetic data in real-time and use the data for training networks. The remainder of this work is structured as follows. In Section 2, different approaches to generate synthetic data are discussed. In Section 3, several experiments and results on using synthetic data with the comparison with real-world data are discussed. In Section 4, conclusions are derived from the experiments and results.

2 Synthetic Data Generation Approaches:

In this section, four different approaches for generating synthetic data are discussed. These approaches can be characterized into two different categories. The first category, data is generated from virtual environments such as video games and virtually created city (see Section 2.1 & 2.2). The second category, data is generated using the composition or blundering of 3d object models with background scenes (see Section 2.3 & 2.4).

2.1 Data generation from a virtual world

In this approach [8], a virtual world is created with cities, cars which is similar to a city in the real world. Once the virtual world is generated, cameras are placed in the scene at different locations and images and the ground truth is captured.

![Sample (left) with semantic labels (center) and general view of city (right) from Synthia dataset. Image adapted from [8].](image)

Synthia [8] (SYNTHetic collection of Imagery and Annotations) The Main purpose of the dataset is aiding semantic segmentation in the context of Autonomous Driving (AD). The data has been generated by the rendering of a virtual city created with the Unity development platform. The important elements of the real-time driving environment are included in the virtual city. Realistic models of cars, pedestrians, cyclists and more are used to populate the city (see Fig. 2.). Additionally, properties like color, size, shape, etc can be changed to produce new looks for the data. Two complementary sets of images are generated referred to as Synthia-Rand and Synthia-Seqs from the virtual city rendered in previous step. All images share standard properties as frame resolution of $960 \times 720$ and horizontal field of view of 100 degrees. Synthia-Rand set contains images captured by an array of virtual cameras moving randomly around the city. Several frames are captured by changing the texture in the scene like the texture of sidewalk, road, etc. Synthia-Seqs simulates four video sequences of approximately 50,000 frames each one up to a total of 200,000 frames, acquired from a virtual car across different seasons.

Semantic annotations for objects are generated with ease whenever an element or random object is placed in scene. New cities can be created by using different combinations of elements and blocks. Multiple viewpoints of a same scene can be captured using multiple cameras, which can be used in scene reconstruction tasks as well. Initial creation of random objects catalogue and virtual city is manual work. Other limitation is data set contains only labels for 13 classes.
2.2 Data generation from a virtual world

Recent video games use rendering techniques on dedicated hardware to increase visual quality and effects. Note the richness in detail and grade of realism. All objects in the game such as 3D meshes, textures and their rendering parameters are stored in GPU. During a game, each object is rendered with several rendering calls. In each rendering call, the object undergoes several transformations (see Fig. 3.). Different techniques to generate high-quality scenes in a game are vertex shader, geometry shader (tessellator and hull shader), rasterization, fragment shader (textures).

![Fig. 3. Overview of the DirectX rendering pipeline. Image adapted from [5].](image)

To make use of the hardware, the game load’s library into the application memory, a wrapper is injected around this graphics library. This injection is known as detouring. Wrapper modifies and monitors all subsequent communication between game and graphics. The injection is done around the transformations in rendering pipeline to generate synthetic data.

Data generated using this approach contains photo-realistic images as recent video games use very high graphics. Ground truth for several dense predictions problems like optical flow, depth estimation can be generated using this approach. Limitation of this approach is source code of video game is not accessible for improving the performance of data generation model.

2.3 Data generation using domain randomization technique

Object detection (such as pedestrians, cyclists, sign, etc) is one part of Autonomous Driving Systems. To train such a system, ground truth labels with bounding boxes are required [1]. This approach uses domain randomization (DR) to generate synthetic data. The first step is collecting 3D models of interested objects, for traffic analysis interest is on objects such as cars, pedestrians, cyclists, etc. Random numbers of these objects are placed in the 3D scene at random positions and orientations. For better training of the network, distractions are included in the scene and then different textures are applied to the objects, a random number of lights of different types are inserted at different locations in the scene and then scene is rendered from different camera viewpoints. Once the scene is generated, it is composed over a random background image. The idea is illustrated in the image Fig. 4. More specifically, images were generated by randomly varying the different aspects of the scene. Such as different type of objects, number of objects in the scene, Types, color, shape, number of distractions, etc.

Huge variety of images can be captured using this approach. Inclusion of flying distractions within the image, forces the network to ignore nearby patterns and deal partial occlusion of the
object of interest. Data that are hard to capture in real world can be easily generated using this approach. Additional care has to be taken with lightning and contrast parameters during the composition of objects onto background images.

![Fig. 4. Synthetic objects (in this case cars, top-center) are rendered on top of a random background (left). Image adapted from [1].](image)

### 2.4 Data generation using blender modeling technique

The main idea of the approach is to develop a rendering system [7]. The system renders an image of an object of interest with an increased field of view (see Fig. 5). Both the object and viewpoint are sampled from defined sets of objects and distributions respectively. For the image, the depth map is also rendered which is used for determining pixel-precise segmentation of the object.

![Fig. 5. Synthetic image of a vehicle from a defined set (left), render its depth map (center) and output image (right). Image adapted from [7].](image)

The rendering is parameterized by further scene and rendering parameters. An environment map is randomly selected with a predefined distribution. One light source simulating the sun is defined with different position and lighting parameters. Then motion blur is included to obtain realistic images. Finally, the complete scene is rendered using Blender Internal Renderer.

Using of blender tool is highly customizable: view point, sun intensity and direction, motion blur, vehicle appearance, background environment map, and many other factors can be changed easily and selected randomly from a defined distribution.
3 Results

For the approaches discussed, Data generation and performance evaluation is done on different tasks. Hence, comparison among the approaches is not possible and feasible. Instead, performance of networks trained on data generated using different approaches are compared with networks trained on real world datasets respectively. In this section, experiments performed on the synthetic data generated with three approaches (approaches mention in Section 2.1, 2.2 and 2.3) and their performance on real-world data are discussed.

3.1 Experiments on object detection:

Several experiments for object detection were performed using Synthia dataset and the data generated through domain randomization. Separately for each approach, performance of the network is compared with the networks trained on different standard datasets and different state-of-art algorithms.

**Synthia dataset** Experiments mentioned below are performed by training fully connected network (FCN) and T-Net network on low-resolution images from Synthia dataset. These experiments are confined to detection of 11 specific object models, results of per-class and global accuracy are mentioned in Table. 1. All images are resized to a common resolution of $180 \times 120$. This is done to speed up training process and save memory [8]. Experiments performed are compared with networks trained on standard benchmark datasets KITTI, CamVid, CBCL, U-LabelMe generated from real world through recordings.

First experiment, FCN and T-Net networks trained on Synthia-Rand dataset and validation is done on different standard datasets. Performance of network in terms of percentage of accuracy is shown in Table 1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Validation</th>
<th>Per-class global</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-Net</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CamVid (V)</td>
<td>48.9 79.7</td>
<td></td>
</tr>
<tr>
<td>KITTI (V)</td>
<td>39.0 61.9</td>
<td></td>
</tr>
<tr>
<td>U-LabelMe (V)</td>
<td>38.3 53.4</td>
<td></td>
</tr>
<tr>
<td>CBCL (V)</td>
<td>41.8 66.0</td>
<td></td>
</tr>
<tr>
<td>FCN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CamVid (V)</td>
<td>62.5 74.9</td>
<td></td>
</tr>
<tr>
<td>KITTI (V)</td>
<td>47.1 62.7</td>
<td></td>
</tr>
<tr>
<td>U-LabelMe (V)</td>
<td>50.0 59.1</td>
<td></td>
</tr>
<tr>
<td>CBCL (V)</td>
<td>56.9 68.2</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Results of networks which are trained on Synthia-Rand data set and validated on datasets generated from real world recordings. Information gathered from [8].

Second experiment, networks are trained on Synthia-Rand data along with real data from standard datasets. For this purpose, for each batch of 10 images, 6 images are taken from real dataset and 4 images are taken from synthetic dataset. Results are shown below (see Table 2), compared against network trained on just datasets KITTI, CamVid, CBCL, U-LabelMe generated from real world video recordings.

Notice that for both, T-Net and FCN there are improvements of more than 10 percent (up to 18.3 percent) in per-class accuracy (marked in green color in Table 2). According to [8], the decrements of global accuracy (marked in red color in Table 2) for FCN may be related to the combination of early and late layers during the up-sampling process.
<table>
<thead>
<tr>
<th>Method</th>
<th>DataSet</th>
<th>Per-class</th>
<th>global</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-Net</td>
<td>CamVid</td>
<td>46.3</td>
<td>81.9</td>
</tr>
<tr>
<td></td>
<td>CamVid+SR</td>
<td>56.5 (10.2)</td>
<td>90.7 (8.8)</td>
</tr>
<tr>
<td></td>
<td>KITTI</td>
<td>44.2</td>
<td>80.5</td>
</tr>
<tr>
<td></td>
<td>KITTI+SR</td>
<td>51.6 (7.4)</td>
<td>80.8 (0.3)</td>
</tr>
<tr>
<td></td>
<td>U-LabelMe</td>
<td>36.4</td>
<td>62.4</td>
</tr>
<tr>
<td></td>
<td>U-LabelMe+SR</td>
<td>46.7 (10.3)</td>
<td>72.1 (9.7)</td>
</tr>
<tr>
<td></td>
<td>CBCL</td>
<td>37.9</td>
<td>73.9</td>
</tr>
<tr>
<td></td>
<td>CBCL+SR</td>
<td>48.4 (10.5)</td>
<td>75.2 (1.3)</td>
</tr>
<tr>
<td>FCN</td>
<td>CamVid</td>
<td>52.8</td>
<td>78.4</td>
</tr>
<tr>
<td></td>
<td>CamVid+SR</td>
<td>72.1 (18.3)</td>
<td>83.6 (5.2)</td>
</tr>
<tr>
<td></td>
<td>KITTI</td>
<td>51.5</td>
<td>82.3</td>
</tr>
<tr>
<td></td>
<td>KITTI+SR</td>
<td>59.4 (7.9)</td>
<td>80.8 (-1.5)</td>
</tr>
<tr>
<td></td>
<td>U-LabelMe</td>
<td>60.1</td>
<td>79.4</td>
</tr>
<tr>
<td></td>
<td>U-LabelMe+SR</td>
<td>64.4 (4.3)</td>
<td>76.2 (-3.2)</td>
</tr>
<tr>
<td></td>
<td>CBCL</td>
<td>53.4</td>
<td>79.7</td>
</tr>
<tr>
<td></td>
<td>CBCL+SR</td>
<td>53.5 (0.2)</td>
<td>75.2 (-4.5)</td>
</tr>
</tbody>
</table>

Table 2. Comparison of network trained on mix of different dataset and Synthia-Rand to the networks trained only on real world dataset in terms of percentage of accuracy. Information gathered from [8].

**Domain Randomization Dataset:** Object detection experiment is performed on three State-of-art neural networks [1]. Neural networks used are Faster R-CNN, R-FCN, SSD. VKITTI dataset is set of synthetic images generated from a game engine with an idea of recreating real world KITTI dataset as closely as possible. So, different architectures trained on DR based approach is compared with VKITTI dataset (see Table 3). The testing is done on real world KITTI dataset. R-CNN and SSD network trained on dataset generated through domain randomization performed well when compared to VKITTI dataset.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>VKITTI</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster R-CNN</td>
<td>79.7</td>
<td>78.1</td>
</tr>
<tr>
<td>R-FCN</td>
<td>64.6</td>
<td>71.5</td>
</tr>
<tr>
<td>SSD</td>
<td>36.1</td>
<td>46.3</td>
</tr>
</tbody>
</table>

Table 3. Comparison of three different state-of-the-art object detector networks trained on two different datasets and tested on real world KITTI datasets. Information gathered from [1].

Precision-Recall curves of different networks trained on both DR based and VKITTI datasets are shown in Fig. 6. From the plot one can observe the precision for DR based data is higher than VKITTI for most of the recall values across the architectures.

### 3.2 Experiment on sign detection and distance to sign:

Experiment is performed to detect a stop sign and recognize the distance to the stop sign [2]. The network is trained on synthetic data generated from the GTA V video game. To model the network, a huge number of images (over 1.4 million images) with and without stop signs are collected from the GTA V game. Fig. 7 shows a sample image of stop sign generated from video game and image of a stop sign from the real world. The images collected from the game are split into long range and short range sets. Images with stop sign within a distance of 70 m are considered in long range set and distance within 40 m are considered in short range set.
At first, deep convolutional neural network (DCNN) is trained for 300,000 iterations on long range set along with images of no stop sign and this network is referred to as long range CNN. The final co-coefficients of long range CNN network are further fine-tuned by training for another 100,000 interactions on short range set. This network is referred to as Short range CNN.

<table>
<thead>
<tr>
<th>Range</th>
<th>Accuracy</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 m - 10 m</td>
<td>0.903</td>
<td>0.097</td>
</tr>
<tr>
<td>10 m - 20 m</td>
<td>0.934</td>
<td>0.066</td>
</tr>
<tr>
<td>20 m - 30 m</td>
<td>0.801</td>
<td>0.199</td>
</tr>
<tr>
<td>30 m - 40 m</td>
<td>0.512</td>
<td>0.488</td>
</tr>
<tr>
<td>40 m - 50 m</td>
<td>0.499</td>
<td>0.511</td>
</tr>
<tr>
<td>50 m - 60 m</td>
<td>0.396</td>
<td>0.604</td>
</tr>
<tr>
<td>60 m - 70 m</td>
<td>0.501</td>
<td>0.499</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Range</th>
<th>Accuracy</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 m - 10 m</td>
<td>0.961</td>
<td>0.039</td>
</tr>
<tr>
<td>10 m - 20 m</td>
<td>0.949</td>
<td>0.051</td>
</tr>
<tr>
<td>20 m - 30 m</td>
<td>0.798</td>
<td>0.202</td>
</tr>
<tr>
<td>30 m - 40 m</td>
<td>0.440</td>
<td>0.380</td>
</tr>
</tbody>
</table>


From 1.4 million images collected from the video game, over 1 million images are used for training purpose and remaining are used for testing purpose. Performance of networks on the respective test set in terms of accuracy and false positive rate is shown in Table 4. Short range CNN has outperformed long range CNN with accuracy of 96% to 90% within range of 10 m.
A set of real images were collected from a single road lane for evaluation of short range CNN on real-world data. Performance of short range CNN on real-world data in terms of accuracy and the false positive rate is shown in Table 5. The network has 100% accuracy and 0% false positive rate within a range of 10 m.

<table>
<thead>
<tr>
<th>Range</th>
<th>Accuracy</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 m - 10 m</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>10 m - 20 m</td>
<td>0.750</td>
<td>0.250</td>
</tr>
<tr>
<td>20 m - 30 m</td>
<td>0.968</td>
<td>0.032</td>
</tr>
<tr>
<td>30 m - 40 m</td>
<td>0.687</td>
<td>0.313</td>
</tr>
</tbody>
</table>

Table 5. Performance of short range CNN when tested on data collected from real world. Information gathered from [2].

4 Conclusion

After evaluating the performance of networks with real data, networks trained on synthetic data performed equivalently good in comparison with networks trained on real data in terms of accuracy rates. In some state-of-art architectures and synthia dataset, the accuracy of a network trained on the combination of synthetic and real data is higher when compared with networks trained on only real-world datasets. After seeing the results, for semantic labeling tasks, data generated using virtual city is most appropriate. For training of model related to autonomous driving systems, data generated using video games like GTA 5 is appropriate because most of real time driving scenarios are included in the game. So, for tasks where real data is not available or collection of real-world data is time consuming and costly, synthetic data can be used for training deep neural networks. The datasets generated with different approaches are confined to a limited amount of classes. Further work on applying techniques for including different objects (e.g. road signs), scene structures (e.g. parking lots), scene characteristics (e.g. shadows in the scene) can be done.

References

Synthetic Data for Smart Traffic Analysis

Praveen Kumar Gummadi¹ and Markus Kächele²

¹ p.gummadi@rhrk.uni-kl.de
² markus.kaechele@dfki.de

Abstract. Training a neural network to learn robust features requires millions of labeled images. Collecting the images from real world and annotation of images with labels involves manual effort. Unfortunately, tasks involving manual work are expensive, time-consuming and error prone. Instead, synthetic data can be used to train the networks. In this project work, we present an approach to generate labeled data from video game GTA 5. Ground truth labels are generated in real time while playing the game. Using this code labels for instance segmentation, semantic labeling, optical flow, depth estimation, instance tracking can be generated.

Keywords: Render call, rendering pipeline, DirectX, GTA 5, Synthetic data.

1 Introduction

Manual data collection for deep learning tasks in computer vision, especially annotation of images with ground truth labels is time-consuming, expensive and prone to error. For depth estimation, optical flow, image decomposition related tasks annotations by humans is very hard, resulting in very few amounts of data available for training the networks. Using synthetic data instead of real data for training the neural networks for various tasks have shown good results. One of the approaches for generating synthetic data is from video games. Current graphics quality of video games and computing performance of graphics processing unit (GPU) allows to generate the high quality data while playing the video game. A wrapper is injected in between the game and graphics library which would interpret the communication between the hardware and game engine and generate the additional data along with images captured during the game. Using this approach, ground truth for several tasks can be generated in one go while playing the game. The remainder of the project work is as follows. In Section 2, composition of a frame from the hardware during a game, DirectX rendering pipeline and operations performed by the injected tasks is explained. In Section 3, post processing using semantic labels and instance labels data generated from video game to generate bounding box information for training a YOLO network is discussed. In Section 4, various mods integrated with the game to enhance the quality and variety of images are presented. In Section 5, results of training a network using the data generated is showed. Finally, in Section 6, observations and conclusions based on the results are discussed.

2 Data generation process

The video game communicates with hardware via API such as DirectX, a library which is dynamically loaded by the system. Basic idea of the process is a wrapper around the DirectX library is injected into the game through dll injection (shown in Fig. 1). This method is called detouring. For the implementation of a wrapper, clear understanding of frame composition during the game and rendering pipeline is required. First, how each frame is rendered during a game and pipeline of each rendering call is discussed. Later different tasks performed by the wrapper and injection of wrapper into the game to generate the ground truth is explained.
2.1 Composition of a frame:

Each 3d geometry, texture, mesh in a scene/frame are stored separately in GPU memory locations called G-buffers. All the intermediate output during the rendering and composition of a frame are stored in these G-buffers (shown in Fig. 1 Left). Later, game communicates with the graphics library with information related to composing a frame based on the information stored in G-buffers. This communication is done through rendering parameters. Now the graphics library composes the frame by rendering the 3d geometry, texture and mesh stored in the hardware using the rendering parameters (Shown in Fig. 1 Right).

Fig. 1. Left: Each 3d geometry and texture stored in different locations in G-buffers. Right: Composition of a frame based on rendering parameters and G-buffers data.

2.2 Rendering of each object:

During the composition an object is rendered by several render calls. Each render call renders a part of object/frame which has the same geometry and texture. In case of rendering of car, wheels, mirrors, body is rendered by three different renders calls as their geometry and texture are different (illustrated in Fig. 2).

Fig. 2. Each color represents a different render call. For this object 4 render calls are required.

2.3 Rendering call pipeline:

Each Rendering call undergoes at least 3 transformations. Vertex shader, rasterization and pixel shader are the transformation involved in rendering an object. The overview of the pipeline is shown in Fig. 3.

- **Vertex Shader**: 3d Geometry is transformed into screen space using a vertex shader.
- **Rasterization**: Transformed mesh is rasterized into 2d image.
- **Pixel Shader**: Post processes each and every pixel based on the texture information.
2.4 Wrapper operations:

- **Monitor** and store the communication like rendering parameters between the game and graphics hardware. Copy the information in G-buffers (GPU memory) to C-buffers (CPU memory). This information is sufficient to recreate the frame.

- During the vertex shader transformation, group the render calls for each object. But as discussed in Section 2.2, each object is rendered by several renders, so how to group the calls? Each object is rendered in a separate reference frame (shown in Fig. 4). So, during vertex shader transformation render calls are grouped based on the target reference frame and passed onto modified pixel shader.

- **Modified pixel shader** assigns a unique id for each group and encodes the information to the corresponding pixels. Each id consists of
  - 1st bit: indicates the class label.
  - Rest of bits: indicates the instances of objects.

3 Prepossessing of data for training

Semantic labels and instance labels are generated in the previous step. To train an object detection network for example a YOLO network [1], bounding boxes for objects of interest has to be generated. In the preprocessing step, bounding boxes are generated for vehicles and pedestrians using the semantic label and instance id generated by the wrapper during the game.
3.1 Bounding box for Vehicles:

- Using class labels for generating bounding boxes. Issue is single combined bounding box is overlapping vehicles (shown in Fig. 5 Left).
- Using instance id for generating bounding box. Issue is instance id are generated based on texture and 3d geometry in the frame. In a frame there are too many textures and geometry changes resulting in too many unnecessary bounding boxes (shown in Fig. 5 center).

- Solution: Combine both class labels and instance id to generate bounding boxes for vehicles. Within each class label region, use instance id and create bounding boxes (shown in Fig. 5 Right).

![Fig. 5. Bounding Box using class label (Left), using instance id (Center), using both class label and instance id (Right).]

3.2 Bounding box for pedestrians:

- Using the same generation logic used for vehicles. Issue with the logic is bounding boxes are also created for the drivers (shown in Fig. 6 Left).
- Solution: Ignore the pedestrian bounding boxes which are within the vehicle bounding boxes (shown in Fig. 6 center). Vehicle drivers apart from cyclists and bike riders are excluded as pedestrians. Open problem still is cyclists and bikers are included as pedestrians and bounding boxes are created.

![Fig. 6. Bounding Box using vehicle logic (Left), using changed logic (Center), blacked out drivers in original image (Right).]

Finally, for the bounding boxes for pedestrians which are ignored, the corresponding region in the image is blacked out, so that network doesn’t learn them as negative examples.
4 Mods integrated with the game:

To generate photo realistic images with a huge range of variety, several mods were integrated with the game. In this section, three main mods integrated with game are discussed.

4.1 Natural Vision Remastered:

This mode enhances the image quality and ensures to generate photo realistic images. In Fig. 7, image without mod (Left), Image with mod (Right) in GTA 5.

![Fig. 7. Left: Image from GTA 5 without mod. Right: Image from GTA 5 with Mod.](image)

4.2 Aikido Free Cam:

This mod is for generating images from different angles instead of first and third person view of player. In Fig. 8, image captured with player view (Left), image captured with the mod (Center)

![Fig. 8. Two different objects rendered on to separate reference frames.](image)

4.3 Time Scalar:

This mod is used to change the time in game and change game time rate with respect to real time world.

Additionally, to change the weather in the game, use the cheat code provided by the game itself ‘makeitrain’. Image in Fig 9 shows the images captured in different seasons.
5 Results:

For checking the quality of synthetic data, YOLO network has been trained for object detection of vehicles and pedestrians.

- Network is completely trained on synthetic data and no pretrained weights were used.
- Network with 3 YOLO layers was used.
- Trained on 15,000 images for 30,000 iterations with a split 10,000 iterations.
- Final mAP: 93% and loss: 0.7.

Fig. 9. Images captured during different seasons: sunny, rain, snow, overcast.

Fig. 10. mAP and loss graph for first 10,000 iterations.
Some results on images are shown in the Fig. 11.

![Network prediction on unseen images generated from game.](image)

**Fig. 11.** Network prediction on unseen images generated from game.

### 6 Conclusion:

Network performed good enough for unseen images from game and real time videos. To detect smaller objects as well, network with 5 YOLO layer can be used. To enhance the performance of network, training can be started from a pretrained weights real data along with combination of synthetic data can be used. Network trained with pretrained weights was able to detect real time night videos as well. Further work can be focused on data generation process. Below are couple of points where further work can be done:

- During the generation process, a person has to play the game or move around the camera in the game. This task can be automated by scripts.
- For this project, all the vehicles are treated as same label, labels for subdivision of vehicles like cars, trucks, vans etc can be generated separately.

### References

Survey on 3D Pedestrian Detection

Mahdi Javadi1 and Michael Fürst2

1 JavadiMahdi1991@gmail.com
2 David_Michael.Fuerst@dfki.de

Abstract. Autonomous ground vehicles is a hot topic. Developing the system, we encounter many challenges, among which collision with pedestrian is of great importance. The major challenge is the development of reliable pedestrian detection system. Due to the varying appearance of pedestrian and the unstructured environment, it is very difficult to cope with the demanded robustness of such systems. The aim of this paper is to survey different approaches on pedestrian detection in 3D. 6 state-of-the-art-approaches are compared in detail from various aspects. All papers are evaluated on KITTI dataset, which makes the comparison results realistic. Finally, a conclusion has been drawn with respect to a comprehensive and precise data analysis.

Keywords: Pedestrian Detection, Autonomous Vehicles, Object detection in 3D

1 Introduction

Developing autonomous systems capable of assisting human in daily life is one of the grand challenges in the modern computer science. One example is autonomous driving systems which can help decrease fatalities. Deploying autonomous vehicles in urban environments poses a difficult technological challenge. Among other tasks, 3D understanding is strongly in demand and autonomous vehicles need to detect and track moving objects in realtime. 3D object detection task classifies the object category and estimate oriented 3D bounding boxes of physical objects, such as vehicles, pedestrians and cyclists, from input data. In this paper, we focus on a particular 3D object detection, 3D pedestrian detection. This problem faces many challenges such as ambiguity of 3D pose from images and complexity of human appearance. Self-driving cars are equipped with multiple sensors, such as LiDAR and cameras. LiDAR hand outs the accurate path information while camera represents much more detailed semantic information. The aim of this paper is to provide an overview of state-of-the-art approaches for the above mentioned task and presents detailed analysis and discussion on the structure and accuracy. Thus, analysing numerical data is of great significance, while we need to realize the performance of each approach in the task of pedestrian detection in 3D. Therefore, having a reliable benchmark is in demand. KITTI dataset [2] is one of the benchmarks that pushes forward the performance of the visual recognition systems. This dataset took advantage of an autonomous driving platform to develop a challenging benchmark for pedestrian detection. This benchmark provides us the opportunity to compare the approaches from different aspects, such as Bird’s Eye View (BEV) and 3D detection, in different modes: easy, moderate and hard. For the fair comparison, all the numerical data is taken from KITTI benchmark, not from papers. The paper is organized as follows. Section 2 represents the most important aspects of the architecture of each individual approach. Discussion of the experimental data and performance comparison is presented in Section 3. Finally, Section 4 puts forward the conclusion.
2 Approaches

For the purpose of dealing with the task of 3D detection, self-driving cars are equipped with different sensors, among which LiDAR and camera are the most common that output point clouds and images, respectively. There are two major differences between point cloud representation and image data: 1) the point cloud is 3D, while the image is 2D and 2) a point cloud is sparse, while an image is dense [5]. Making use of diverse combinations of these two widely used sensors, results in different outcomes for the task of 3D detection. Thereby, the primary feature addressing the comparison among approaches is the architecture. These approaches are categorized into three types: RGB-Based Architecture, LiDAR-Based Architecture, RGB-LiDAR-Based Architecture.

2.1 RGB-Based Architecture

Monocular 3D Object Detection Leveraging Accurate Proposals and Shape Reconstruction [4].

This paper introduces a proposal based monocular 3D object detection method that leverages the related task of shape reconstruction [4]. Taking advantage of a 2D object detector, the 3D search space is reduced by designing a 3D bounding box proposal for each object detected in the scene. Location of the proposal is determined by two factors: 1) re-projection of the box center and 2) the relation between height and depth of an object in perspective transformation of a pinhole camera model.

![Fig. 1. Monocular 3D object detection. Figure is taken from [4].](image)

**Framework.** The approach used a two stage proposal regression design, which makes the learning process easier by regressing distributed anchor boxes, to obtain the final amodal, oriented 3D bounding box. For each instance in local object coordinate system, a point cloud is predicted with the aid of Instance Reconstruction Module. Object detection and shape reconstruction tasks are connected by transforming the object point cloud into the camera coordinate frame using the instance centroid regression output. Finally, local scale and shape of each instance are jointly optimized with their localization in the scene through multi-task learning and a projection loss. An overview of the architecture is provided in figure 1.

The basic idea of the method is to reduce search space by using a single proposal per object and to leverage shape reconstruction for accurate localization. This paper took advantage of the 2D detectors to generate classified 2D bounding boxes. Using these classified 2D bounding boxes,
image crops are then passed through an encoder to generate a feature map. This feature map is then shared by three modules shown in figure 1. The objective of Proposal Generation module is to generate high quality 3D proposals. These proposals are regressed by Proposal Refinement module, which outputs amodal, oriented 3D bounding boxes. An estimation of a point cloud per instance is estimated by the aid of Instance Reconstruction Module. Using centroid regression from Proposal Refinement Module, the point cloud is transformed to camera coordinate frame.

2.2 LiDAR-Based Architecture

**Pointpillars: Fast Encoders for Object Detection from Point Clouds** [5]. This paper addressed the problem of 3D detection by the approach of encoding a point cloud into a format which is appropriate for a downstream detection pipeline. Pointpillars is an encoder that utilizes PointNets [7] to learn a representation of point clouds organized in vertical columns (Pillars).

**Framework.** PointPillars takes point clouds as input and estimates an oriented 3D box for an object in 3D, such as cars, pedestrians and cyclists. This process consists of three main stages which is illustrated in figure 2.

**Pointcloud to Pseudo-Image.** To apply a 2D convolutional architecture, point cloud is converted to a pseudo-image as the first step. \( l \) is denoted as a point in a point cloud with coordinates \( x, y, z \) and reflectance \( r \). The point cloud is discretized into an evenly spaced grid in the \( x - y \) plane and creates a set of pillars \( \mathcal{P} \). The points in each pillar are then augmented with \( x_c, y_c, z_c, x_p \) and \( y_p \) where the \( c \) subscript denotes distance to the average of all points in the pillar and the \( p \) subscript denotes the offset from the pillar \( x, y \) center. Then, a PointNet is applied for each pillar.

![Fig. 2. PointPillars for 3D detection. Feature encoder network is responsible for converting a point cloud to a sparse pseudo-image. 2D convolutional backbone processes the pseudo-image into high-level representation. Detection head is in charge of detection and regression of 3D boxes. Figure is taken from [5].](image)

**Backbone.** The backbone module has two sub-networks: one top-down network that produces features at small spatial resolution and a second network that performs upsampling and concatenation of the top-down features. The final features from each block are combined through upsampling and concatenation.

**Detection Head.** A Single Shot Detector (SSD) is used for 3D object detection task. The anchor boxes are matched to the ground truth by using 2D Intersection over Union (IoU). In this approach, height and elevation of the bounding boxes were not used for matching, but instead, height and elevation become additional regression targets.
2.3 RGB-LiDAR-Based Architecture

Multi-view 3D Object Detection Network for Autonomous Driving [1].

A Multi-View 3D network (MV3D) is introduced in this paper that takes both LiDAR point clouds and RGB images as input and predicts oriented 3D bounding boxes. The complete structure of the approach is illustrated in figure 3. MV3D was one of the first networks that took advantage of both image and LiDAR for 3D object detection. The network is evaluated just for cars in KITTI.

**Framework.** MV3D is composed of two stages: one for 3D object proposal generation and the other for multi-view feature fusion [1]. Making use of bird’s eye view data, the 3D proposal network generates 3D candidate boxes, which can be projected to any views in 3D space. The multi-view feature fusion network extracts region-based features by projecting 3D proposals to the feature maps from multiple views, bird’s eye view (BV), front view (FV) and the image plane (RGB). This sub-network is also benefited from a deep fusion approach to satisfy the need of interactions of intermediate layers from multiple views. Finally, with the given multi-view feature representation as input, the network performs oriented 3D box regression, capable of predicting location, size and orientation of objects in 3D space.

![Fig. 3. MV3D network for object detection. The network takes three views as input. First, it generates 3D object proposals from BV and project it to other three views. A deep fusion network is then uses these proposals and combine them. Finally, these fused features are used for prediction of object class and oriented 3D box regression. Figure is taken from [1].](image)

**3D Proposal Network.** Given a bird’s eye view map as the input, network generates 3D box proposals from a set of 3D anchor boxes. Each 3D box is parameterized by \((x, y, z, l, w, h)\), which are the center and size (in meters) of the 3D box in LiDAR coordinate system. Taking advantage of a multi-task loss, classification and 3D box regression is performed simultaneously.

**Region-based Fusion Network.** Since features from different views usually have different resolutions, ROI pooling is employed for each view to obtain feature vectors of the same length. Then, a region-based fusion network is employed to combine features from multiple views and jointly classify object proposals and do oriented 3D box regression. Lastly, a multi-task loss is used to jointly predict object categories and oriented 3D boxes.
Frustum PointNets for 3D Object Detection from RGB-D Data [6].
In this work, the search space is reduced by applying the following steps:
Making use of a Convolutional Neural Network, 2D object region proposals in RGB images are
generated. Benefited from depth data of point cloud, every single 2D region is then extruded to a
3D viewing frustum. Eventually, taking advantage of two variants of PointNets, 3D bounding box
for the object from the points in frustum is predicted.

Framework. Depicted in figure 4, the Frustum PointNet Network for 3D Object Detection
consists of three modules: frustum proposal, 3D instance segmentation and 3D amodal bounding
box estimation. This 3D box is parameterized by its size \( h, w, l \), center \( c_x, c_y, c_z \) and orientation
\( \theta, \phi, \psi \) relative to a predefined canonical pose for each category [6].

Frustum Proposal. With a known camera projection matrix, a 2D bounding box can be lifted
to a frustum that defines a 3D search space for the object. All points in the frustum are then
collected in the frustum to shape a frustum point cloud. Frustum proposal generation is the process
for extracting frustum point clouds from RGB-D data.

3D Instance Segmentation. 3D instance segmentation is achieved by using a PointNet-based
network on point clouds in frustums. This approach is similar to Mask-RCNN, which achieves
instance segmentation by binary classification of pixels in image regions [6]. Based on 3D instance
segmentation, the 3D bounding box center in a local coordinate system is predicted. This Module
takes a point cloud in frustum as input and predicts a probability score for each point [6]. This
score indicates how likely the point belongs to the object of interest [6]. At last, points that
are classified as the object of interest are extracted.

Amodal 3D Box Estimation. Benefited from a box regression PointNet, This module estimates
the object’s amodal oriented 3D bounding box for a given segmented object points as input.
Employing a combination of 3D box estimation network, output of T-Net and the masked points’
centroid, an absolute center of the box is obtained.

Joint 3D Proposal Generation and Object Detection from View Aggregation [3].
This Paper presents AVOD, an Aggregate View Object Detection network for autonomous driv-
ing [3]. Taking advantage of BEV map and RGB image, the method uses feature extractors to
generate feature maps. Both feature maps are then shared by two sub-networks: region proposal
network (RPN) and second stage detector network. Finally, oriented 3D bounding box is regressed
to predict the extents, orientation and classification of objects in 3D space.
Fig. 5. Architectural diagram of AVOD. The feature extractors are displayed in blue. The Region Proposal Network is shown in pink. The second stage detector is demonstrated in green. Figure is taken from [3].

**Framework.** The network structure is illustrated in figure 5. Generated features are shared by two sub-networks: region proposal network (RPN) and second stage detector network. RPN is capable of performing multi modal feature fusion on feature maps to generate 3D object proposals for multiple classes of objects. The second stage detection network employ these proposals to carry out oriented 3D bounding box regression and category classification to predict orientation, extent and classification of objects in 3D space [3].

**Generating Feature Maps from Point Clouds and Images.** Making use of voxel grid representation of the point cloud, a six-channel BEV map is generated. The first five channels are encoded with the maximum height of points in each grid cell. The sixth channel contains point density information computed per cell [3].

**The Feature Extractor.** AVOD is benefited from an identical feature extractor for each view. The feature extractor made up of two segments: an encoder and a decoder. The encoder takes an image or BEV map as input and outputs a feature map \( F \), which is divided by eight in length and width. The decoder takes the output of the encoder as input and produces a new feature map.

**Region Proposal Network.** The difference between a set of anchor boxes and ground truth is regressed by the aid of this module. By clustering the training samples for each class, dimensions of anchors are then determined [3]. Crop and resize operation is used for feature crop extraction. Projecting anchors onto BEV and image feature maps, two regions of interest are achieved for a given anchor in 3D. These regions are then used to extract feature map crops from each view. The results of the crop and resize operation are equal-sized feature crops from both views. Using fused feature crops, two fully connected layers are then employed to regress axis aligned object proposal boxes and produce an object/background "objectness" score.

**Second Stage Detection Network.** Each bounding box is encoded with four corners and two height values, which represents the top and bottom corner offsets from the ground plane. To determine orientation from a 3D bounding box, the four possible orientations of the bounding box, four corners, is extracted. Finally, the closest one to the regressed orientation vector is chosen.

**IPOD: Intensive Point-based Object Detector for Point Cloud** [9].

The paper presents a 3D object detection framework based on raw point cloud [9]. The raw point cloud is taken as input. The novelty in this paper is on the proposal generation module. This module provides proposals relying on each point and effective selection of object proposals with corresponding ground-truth labels, which ease network training. Context and local information are extracted for each proposal. This information is given to a PointNet to infer final results.
**Framework.** The network in this paper takes point cloud as input and produces the feature representation for each proposal. Figure 6 provides an overview of the framework. A general strategy is used to seed object proposals based on each point independently and raw point cloud is processed directly [9]. The entire procedure is called Point-Based Proposal Generation.

**Selecting Positive Points.** The first step is to filter out background points. A 2D semantic segmentation network, subsampling network, is used to predict the foreground pixels and then project them into point cloud as a mask to gather positive points.

![Fig. 6. Architectural diagram of IPOD. Figure is taken from [9].](image)

**Proposal Feature Generation** The feature of each proposal has two parts. The first one is cropped from the extracted feature map. Second part is the canonized coordinates of the $M$ selected points. These two are then fed to a T-Net which results in a real object center.

**Bounding-Box Prediction Network.** In this module, a PointNet++ [8] for each proposal is used to predict its class, size ratio, center residual as well as orientation.

### 3 Experimental Results and Comparison

All approaches are compared on 3D pedestrian detection in KITTI dataset using the Bird’s Eye View (BEV), 3D AP metrics and Average Orientation Similarity (AOS) which is shown in Table 1. BEV provides a top view or 360 degrees images which visualize the direct car surroundings. Measuring the accuracy of object detectors, average precision, AP, computes the average precision value for recall value over 0 to 1 and AOS metrics provides the results for joint object detection and orientation estimation. For a fair comparison, all information is taken from KITTI benchmark and not the one in each paper. The reason lies in the fact that, papers used a part of the train set as their test set, which leads to inconsistent results with the KITTI. Each method is evaluated on different difficulty level and the best result in each column is shown in bold type letters. The MV3D is evaluated just for cars in KITTI and will not be a part of comparison.

In this comparison, the best results in all modes owned by RGB-LiDAR-Based approaches. A quick glance at the table, one can conclude that IPOD outperforms all other methods, which is obtained best results in most of the columns.

The reason of the superiority of RGB-based approach stemmed from the fact, that for evaluation of orientation using AOS metrics, the 3D box needs to be projected into the image. This operation is pose dependent and results in loose boxes. Considering the results of either RGB or Laser based approaches, RGB-based approaches obtained the worst results, while Pointpillars method accomplished remarkable results, even close to the best one in each column. With this in mind, it can be stated that LiDAR has an outstanding effect on the performance of pedestrian detection in 3D. An important criteria can be expressed by observing the results of AVOD and Pointpillars. Although
AVOD employed both LiDAR and camera, it achieved relatively similar results in comparison to Pointpillars. To this end, a conclusion can be drawn that network architecture has also significant impact on the accuracy of pedestrian detection in 3D.

Table 1. Evaluation results of all approaches in percentage as reported by KITTI. 'R' and 'L' are stands for 'RGB' and 'LiDAR', respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>3D Object Detection</th>
<th>BEV</th>
<th>AOS</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Easy</td>
<td>Moderate</td>
<td>Hard</td>
<td>Easy</td>
</tr>
<tr>
<td>Monocular [4]</td>
<td>R</td>
<td>12.65</td>
<td>10.66</td>
<td>10.08</td>
<td>14.27</td>
</tr>
<tr>
<td>Pointpillars [5]</td>
<td>L</td>
<td>52.08</td>
<td>43.53</td>
<td>41.49</td>
<td>58.66</td>
</tr>
<tr>
<td>F-PointNet [6]</td>
<td>R+L</td>
<td>51.21</td>
<td><strong>44.89</strong></td>
<td>40.23</td>
<td>58.09</td>
</tr>
<tr>
<td>AVOD-FPN [3]</td>
<td>R+L</td>
<td><strong>50.80</strong></td>
<td>42.81</td>
<td>40.88</td>
<td>58.75</td>
</tr>
<tr>
<td>IPOD [9]</td>
<td>R+L</td>
<td><strong>56.92</strong></td>
<td>44.68</td>
<td><strong>42.39</strong></td>
<td><strong>60.83</strong></td>
</tr>
</tbody>
</table>

4 Conclusion

This paper presents a survey on state-of-the-art approaches on the task of pedestrian detection in 3D. All methods were discussed individually in details. For the reasonable comparison, numerical data is taken from KITTI benchmark. Experiments on the benchmark show the superiority of the RGB-LiDAR-Based approaches, among which IPOD outperforms all the state-of-the-art approaches in the task.

References

Survey: A 2D/3D Combined Descriptor for Matching Enhancement

Avik Mukherjee\textsuperscript{1} and Ramy Battrawy\textsuperscript{2}

\textsuperscript{1} avimukh08@gmail.com
\textsuperscript{2} ramy.battrawy@dfki.de

Abstract. This survey aims at exploring techniques of improvement over descriptors that use 2D spatial information for image matching by incorporating geometrical constraints. Image-based descriptors often perform poorly subjected to extreme photometric or geometric variations giving wrong correspondences in case of image matching. Many approaches use 3D geometrical constraints in order to filter out mismatches during pre-training process or during training itself.

Keywords: geometric constraint, homographies, geometric context, visual context, feature detection

1 Introduction

Features (e.g. edges, corners etc.) are positions in images that encode a lot of information. Patches around features are encoded using feature descriptors. Features and their corresponding descriptors can either be extracted using conventional algorithms like Scale Invariant Feature Transform (SIFT) \cite{4} or by using neural networks for learning feature detectors and their associated descriptors. Descriptors and keypoints extractors can be classified either as handcrafted or learned approaches \cite{11}\cite{12}.

Robust feature detectors and descriptors are made invariant to photometric and geometric changes in images making them an indispensable part of many applications like image matching, wide-baseline matching, structure from motion and many other applications. Conventional algorithms \cite{4}\cite{5}\cite{8} often underperform in cases of extreme variation in illumination and viewpoint. In order to counteract the above limitations many approaches use 3D information in the form of geometrical constraints. Geometrical constraints are restrictions applied on the optimization procedure that make use of various 3D Scene information and characteristics like the homographies between various viewpoint images, point cloud data of a scene described from various images and many more. Geometrical constraints are mainly used for data generation and refinement, loss formulation, feature augmentation and many other purposes. Approaches using geometrical constraints \cite{2}\cite{3}\cite{13} provide better results for image matching data sets as opposed to simple descriptors handcrafted \cite{4}\cite{5}\cite{8} or learned using only 2D local neighborhood information \cite{11}\cite{12}.

This survey provides a comprehensive summary of various techniques using geometrical constraints for descriptor learning, advantages of such descriptors over state-of-the-art 2D descriptors and a comparative discussion of the individual performance of various descriptors on some state-of-the-art image matching benchmark \cite{10}.

2 Approaches

Recent studies \cite{3}\cite{13}\cite{14} show that 2D handcrafted feature descriptors like SIFT give below par performance on various challenging benchmarks. Many approaches also aim at using various deep
learning solutions in order to learn the detectors and their descriptors either using manually labelled data or in a semi-supervised fashion [3]. New state-of-the-art approaches focus more on incorporating 3D information acquired from training data along with earlier 2D spatial information in order to make the descriptors more robust to extreme changes which increases their accuracy in a host of tasks. The following section lists various approaches of using geometrical constraints in order to develop better descriptors for matching enhancement.

2.1 GeoDesc

GeoDesc [14] uses 3D point cloud on image pairs for generating clean data and optimizing loss functions, consequently making the learning task more difficult. The learning is leveraged by a host of loss formulations which again improve on the accuracy of the descriptor. GeoDesc outperforms some state-of-the-art descriptors in image matching over various standard benchmark data sets [14].

GeoDesc uses the network architecture of L2-Net [12]. In contrast to pooling layers used by L2-Net, GeoDesc uses strided convolution for in-network downsampling [14]. It uses batch normalization in all except last convolutional layers and gives a 128-dimensional feature vector as an output. This approach employs geometrical constraints in order to leverage training data generation and refine outliers in the training data. This approach uses a semi-supervised way for training by automatically generating ground truth correspondences for training the data. The used training data already contains matches having strong correspondences. Training on a refined data set helps the network develop more robust feature descriptors. Ground truth correspondences for training are generated using standard Structure-from-Motion pipeline. The point cloud is then used in order to generate correspondences in the image data sets. GeoDesc uses Delauney triangulation for outlier filtering of the generated 3D point cloud as shown in the Fig. 1. This helps in eradicating correspondence errors. The data used for training thus contains almost perfect high confidence matches which help in generating more robust correspondences.

Fig. 1: First four columns from the left show faulty Correspondences generated by using 3D point cloud information and their matching score given on the left side of each of them. The right-most column denotes the point cloud results obtained after SfM and outliers in red filtered using Delauney triangulation Method [14].

The patches around detected keypoints are generated using a sampling algorithm given by:

$$
\begin{bmatrix}
  x'_{i1} \\
  x'_{i2} \\
  y'_{i1} \\
  y'_{i2}
\end{bmatrix} = \begin{bmatrix}
  k\sigma\cos(\theta)/2 & k\sigma\sin(\theta)/2 & x_i \\
  -k\sigma\sin(\theta)/2 & k\sigma\cos(\theta)/2 & y_i
\end{bmatrix} \begin{bmatrix}
  x_i \\
  y_i
\end{bmatrix}
$$  

(1)

Where:
\((x_i^t, y_i^t), (x_i^f, y_i^f)\) = input and output regular sampling grids.

\((x, y, \sigma, \theta)\) = keypoint parameters

\(k = 12\) similar to LIFT [7].

This approach uses two different similarity criteria: patch similarity and image similarity. Patch similarity denotes the similarity between two patches around the features in the image pairs and the image similarity denotes the sum total of patch similarities over the entire image. The similarities thus generated are used to formulate the training loss.

The training data is divided into batches where each batch is sampled using a favourable batch sampling strategy [14]. Each image is first broken down into a set of N1 patch pairs X as per the patch similarity and SfM filtering criteria [14] and a batch of N2 such sets is considered for training. GeoDesc uses two main approaches for loss formulation: Structured Loss and Geometric Loss.

Structured Loss is given by \(E_1\):

\[
E_1 = \frac{1}{N1(1 - N1)} \sum_{i,j} \left( \max(0, l_{i,j} - l_{i,i}) + \max(0, l_{i,j} - l_{j,j}) \right)
\]  

where:

\(l_{i,j}\) = element in L which is given by

\[L = S - \alpha \text{diag}(S)\]

where:

\[\alpha \in (0, 1)\] is the distance ratio mimicking the behavior of ratio test

and \(S\) is given by:

\[\text{Similarity Score } S = F^1(T)F^2\]

where:

\(F^1, F^2\) are normalised features generated over \(X \in \mathbb{R}^{N1 \times 128}\)

Geometric Loss is given by \(E_2\):

\[
E_2 = \sum_i \max(0, \beta - s_{i,i}), \beta = \begin{cases} 
0.7 & s_{\text{patch}} \geq 0.5 \\
0.5 & 0.2 \leq s_{\text{patch}} \leq 0.2 \\
0.2 & \text{otherwise}
\end{cases}
\]

2.2 ContextDesc

ContextDesc [13] aims at refining the repeatability of the feature detectors and descriptors by trying to eradicate the wrong matches that occur between nearest neighbors. In order to do that it uses the geometrical information of the keypoints and the visual information or the high level information of their patches in order to make an augmented feature descriptor [13]. It also uses special softmax loss in order to train the descriptor learning network.
Fig. 2: Network Architecture of ContextDesc [13].

Fig. 2 shows the network architecture used in ContextDesc. The learning in the case of ContextDesc is broken down into the preparation and the augmentation stages. In the data preparation stage, feature detectors are used in order to find out the keypoints in the images and a local feature extractor is used in order to encode the feature information in the images. The extractor network takes $32 \times 32$ patch samples and forms $K \times 128$ feature vector out of it which is then passed to the context encoder layer for geometric and visual context encoding. The ResNet-50 architecture is used as regional feature extractor which outputs $2048$ activation maps of dimension $H/32 \times W/32$. The ResNet-50 regional feature extractor encodes the regional information in $32 \times 32$ patches across the images into a vector of length $2048$. Geometrical context encoding is achieved by using a variant of a PointNet [1] which uses context normalization over putative matches. ContextDesc uses a technique called the matchability predictor [6] that provides matchability scores over all the feature points detected in image pairs. This method uses an unsupervised learning approach where each image pair is broken down to quadruples containing two feature points of each image and the quadruple is then sent to a function called the matchability function which sends a single matchability score for the pixel. The final objective is then optimised using a hinge loss.

The visual features are encoded using interpolation where the output of the regional feature extractor block is sampled and then interpolated in the locations of the feature coordinates. The raw features are thus passed into a multi layer perceptron in order to get out a more robust feature vector. The geometric and the visual feature vectors thus formed are aggregated and passed through a SoftMax classifier over which the logistic loss is calculated.

### 2.3 SuperPoint

SuperPoint [3] provides a combined architecture for detector and descriptor learning [3]. It uses a pre-trained network for learning. The encoder consists of convolutional layers, spatial downsampling via pooling and non-linear activation functions. It uses three max-pooling layers for spatial downsampling. For feature detection, each pixel of the output corresponds to a probability of pointness for that pixel in the input. The feature detector head computes $X \in \mathbb{R}^{H_e \times W_e \times 65}$ where, $H_e$ and $W_e$ are $H/8$ and $W/8$ respectively, and outputs a tensor sized $R^{H \times W}$. The 65 channels correspond to local, non-overlapping $8 \times 8$ grid regions of pixels plus an extra no feature dustbin. The descriptor head computes $D \in \mathbb{R}^{H_e \times W_e \times D}$ where, $H_e$ and $W_e$ are $H/8$ and $W/8$ respectively, and outputs a tensor sized $R^{H \times D}$. Learning descriptors semidensely rather than densely reduces training memory and keeps the run-time tractable. The decoder then performs a bicubic interpolation of the descriptor and then normalizes the activations to be unit length. The final loss is the sum of two intermediate losses: one for the feature detector, $L_p$, and one for the descriptor, $L_d$. SuperPoint uses homography adaptation in the form of geometrical constraint in order to facilitate ground truth creation and increase the robustness of the feature detections [3]. It uses a synthetically generated data set and
applies various homography parameters and estimations over the data set. It pre-trains the encoder network for detecting correspondences on the adapted data set and then trains the detector and the descriptor networks on the original data set by using suitable loss formulation.

2.4 D2-Net

D2-Net [9] aims at providing a joint detection and description pipeline. It targets on providing sparse and more robust description of images by facilitating descriptor learning before detector learning. It uses the activation maps coming at the output of the CNN in the form of a 3D tensor $R^{W \times H \times n}$ where $n$ is the number of activation maps coming at the output of the CNN layer [9]. This 3D tensor is considered to be a dense set of $n$-dimensional feature descriptors. The descriptors $d_{i,j}$ are normalized in order to make them more robust to matching during training.

Feature detection is being broken down into two stages hard and soft Feature Detection. Hard feature detection step considers a point to be detected as a feature only if it comes out to be the local max of the feature points in the activation map. During training the hard feature detection step is made more soft by first applying a soft local-max score given by:

$$\alpha_{i,j}^k = \frac{\exp D_{i,j}^k}{\sum_{i',j' \epsilon \nu(i,j)} \exp D_{i',j'}^K}$$

(6)

where $\nu_{i,j}$ denotes the 9-neighborhood of the pixel i,j. Then for each spatial location a soft channel selection score is selected given by:

$$\beta_{i,j}^k = \frac{\exp D_{i,j}^k}{\max_i D_{i,j}^i}$$

(7)
Then in order to take both the scores into account a accumulated score is defined given by:

$$
\gamma_{ij} = \max_{k} \alpha_{ij}^{k} \beta_{ij}^{k}
$$

Finally a normalized soft score is selected for each of the possible feature points in the image. Image pyramids are used as in case of traditional handcrafted feature detectors in order to make them more robust to multi-scale detections.

D2-Net uses a triplet margin ranking loss which aims at reducing the distance between the correct correspondences while maximizing the distances between the incorrect correspondences that lie outside a given local neighborhood [9]. The proposed loss produces a weighted average of the margin terms over all the correspondences. Thus maximizing the scores of the most distinctive correspondences and making descriptors corresponding to them more similar then the rest.

### 2.5 Universal Correspondence Network (UCN)

UCN [2] unlike other feature descriptors aims at learning the feature that preserve matches among image pairs under generic correspondences [2]. The mapping is invariant to geometric changes and viewpoint variation. In order to achieve the above the UCN uses a fully convolutional network along with various specialized loss function.

![Cross-Sectional Diagram of the UCN detailing the functioning of each proposed part of the network [2].](image)

Using a fully convolutional layer for correspondence learning has advantages threefold, the network can reuse some of the activations computed for the overlapping regions, a large number of correspondences can be trained per image, and the network can be used for dense descriptor learning as opposed to patch based approaches [2]. The loss formulation known as Corresponding Contrastive Loss encourages two corresponding points if belonging to the same 3D point to be close to each other and atleast a marginal m distance otherwise. The loss formulation is given by:

$$
L = \frac{1}{2N} \sum_{i}^{N} s_{i} ||F_{I}(x_{i}) - F'_{I}(x'_{i})||^{2} + (1 - s_{i}) \max(0, m - ||F_{I}(x_{i}) - F'_{I}(x'_{i})||)^{2}
$$

(9)
This approach uses the technique of hard negative mining where it considers mostly negative correspondences and pushes them more towards the metric boundary of a minimum distance of $m$ as is evident in Eq. 9. This is followed by the concept of the spatial transformer where each feature is transformed separately in image pairs in order to achieve more invariance to transformations [2].

3 Discussion

All the approaches discussed above use geometrical constraints over conventional learned 2D feature descriptors in order to obtain increased matching accuracy. Geometrical constraints are either used in the process of loss computation during training or are used separately as a constituent for descriptor aggregation. These approaches have been seen to have outperformed their conventional counterparts in cases of matching performance and accuracy in their respective benchmarks. The main objective of these methods have not only been to increase matching accuracy but also to target the accuracy of a host of other computer vision applications. Most of these methods have targeted the repeatability criteria of the feature detectors trying to reduce wrong correspondences [2][3][9]. One of the most successful methods, GeoDesc which has greatly outperformed other conventional hand-crafted feature descriptors like SIFT in cases of image matching, retrieval, verification and many other tasks, uses geometrical constraints not only for training loss formulation, but also more accurate ground truth correspondence generation [14]. It tries to make the feature descriptor more robust by training it on harder and more accurate ground truth correspondences. This approach along with its successor ContextDesc [13] though providing better feature descriptors concentrate only on descriptor learning as opposed to other methods like SuperPoint, D2-Net and Universal Correspondence Network which focus mostly on joint detector and descriptor learning. The latter three approaches try to create more robust detectors and descriptors by reducing the distance between positive correspondences and increasing the distance between negative correspondences. SuperPoint trains it joint detector and descriptor network on a Homography adapted data set in order to get more robustness to variations [3]. D2-Net and UCN on the other hand use 3D point cloud information in order to determine positive and negative correspondence to leverage training loss [2][9].

<table>
<thead>
<tr>
<th>Method</th>
<th>No. Features</th>
<th>No. Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hes. det. + RootSIFT</td>
<td>6.7K</td>
<td>2.8K</td>
</tr>
<tr>
<td>HAN + HN++</td>
<td>3.9K</td>
<td>2.0K</td>
</tr>
<tr>
<td>LF-Net</td>
<td>0.5K</td>
<td>0.2K</td>
</tr>
<tr>
<td>SuperPoint</td>
<td>1.7K</td>
<td>0.9K</td>
</tr>
<tr>
<td>DELF</td>
<td>4.6K</td>
<td>1.9K</td>
</tr>
<tr>
<td>D2 MS</td>
<td>4.9K</td>
<td>1.7K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GeoDesc</th>
<th>Verification</th>
<th>Matching</th>
<th>Retrieval</th>
</tr>
</thead>
<tbody>
<tr>
<td>91.1</td>
<td>59.1</td>
<td>74.9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ContextDesc</th>
<th>Verification</th>
<th>Matching</th>
<th>Retrieval</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.2</td>
<td>59.2</td>
<td>76.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparative performance of the D2-Net and the SuperPoint on the top and the performance of the GeoDesc on the HPatches benchmark data set on the bottom [9][10][13][14].

The tables above list the comparative performance of the various methods on the HPatches benchmark. The table on the top compares using the total number of features detected and the total number of matches between corresponding images in the HPatches benchmark. The table in the bottom lists the comparative mean Average Precision between ContexDesc [13] and its predecessor GeoDesc [14] on HPatches benchmark [10].
4 Conclusion

Tables in Table 1 portray the comparative performance of most of the above discussed approaches on the single benchmark matching data set HPatches [10]. The results of the UCN approach cannot be used in the above comparative study as the data set used for evaluation in literature is different as compared to the other approaches. As can be seen ContextDesc and GeoDesc have a very large margin of training accuracy as compared to other handcrafted feature descriptors. GeoDesc and ContextDesc focus more on training descriptors rather than both detectors and descriptors. The approaches like SuperPoint and D2-Net on the other hand focus on joint detector and descriptor learning and also provide better performance as compared to other handcrafted feature detectors and descriptors as listed in Table 1(a). Consequently, approaches like D2-Net and SuperPoint can be considered a better alternative when it comes to image matching scenario as compared to GeoDesc or ContextDesc.

References

3D Visualization of Realtime Body Tracking Data

Avik Mukherjee\textsuperscript{1} and Mathias Musahl\textsuperscript{2}

\textsuperscript{1} avimukh08@gmail.com
\textsuperscript{2} musahl@dfki.uni-kl.de

Abstract. Body sensor networks produce data that can map the motion of the human body with high accuracy. Proper techniques of visualizing such data should be attempted in order to add to the visual accuracy and aesthetics of the software. 3D human models have thus been employed in this project in order to visualize the data coming from the body sensor networks. The calibration of the sensor network should be in accordance to the initial pose of the 3D model in order to reproduce accurate animations.

Keywords: 3D Visualization, Skinned Mesh, Skeleton, Bones, Three .Js, BSN

1 Introduction

Body Sensor Networks are made up of devices called Inertial Measurement Unit or Magnetic Inertial Measurement Unit. These devices are mainly comprised of Accelerometer, Gyroscopes and Magnetometers in case of MIMUs. IMUS/MIMUS help in measuring the position, and orientation of the body. These two quantities combined together form the pose of the body. A network of IMUs/MIMUs worn over different joints of the body can accurately help predicting the pose of each joint of the body and thus estimating the motion of the body.

Fig. 1: Architecture of a Body Sensor Network including its application areas [1].
IMUs find their use in a vast range of application areas involving physical motion. Human Motion Analysis is one such area where IMUs are used in order to analyse the behaviour of Human Motion. In order to achieve such a task body worn suits are fitted with IMUs. These suits consist of an intricate network of such devices that help in modelling the motion across each joint and thus help in analysing any abnormalities if present in the motion. The data acquired from these suits is then processed in order to find out the position and orientation of each node as shown in Fig. 2 and then is leveraged by proper visualization. This project emphasizes the work on visualizing the processed data coming from the IMU focusing on the pipeline used in the development and then summarizing the methods used for achieving each such step in the pipeline. The theory section of the report is adapted from online articles and blogs and no formal literature is used due to the readily available material.

2 Theory

3D modelling is the process of creating digital representations of 3 dimensional objects that can be viewed from several angles. 3D modelling deals mainly with modelling the surface of a 3D object into a 2D plane such that it can cover the whole 360 degrees view of the object. There are two basic functions that define the world of 3D modelling. They are NURBS and NURMS. These functions can be thought of as curves representing geometrical surfaces especially as polygons. A 3- Dimensional model is mainly constituted of meshes which act as the basic building blocks of the geometry. Depending on the application and the model employed the meshes can be triangular or polygonal in shape. The process of 3D modelling has been simplified to a great extent with the help of advanced 3D modelling tools available freely in the internet. These software take care of the complex mathematical calculations undergoing in the background and give the end users a seamless and easy 3D design experience.

Fig. 2: 3D modelling tool Blender for designing 3D Models and creating animations.
The focus of this project is limited to using 3D rendering of human figures that can act as the perfect virtual model of the real human wearing the body sensor suit. In order to visualize the data being sent by the body sensor network in a 3D Human Avatar form, 3D rendering engines should be used in order to model or map the data to an existing human body model. This section enlists a few important components or building blocks for the design of the human body model.

2.1 Skinned Mesh

3D design software mainly use the word Mesh to define a large collection of points or vertices that when combined with the right amount of intensity information help in providing an accurate 3D rendering of a physical 3D Dimensional object. Depending on the number of vertices these meshes can either be defined as low-poly, mid-poly or high-poly meshes. Many 3D objects can be considered as a combination of more than one meshes. These meshes act as rigid bodies that cannot undergo animation. In order to animate these meshes one has to change the position of the vertices, which in turn would bring the deformation in the polygons. It is not possible to manually control and modify each of the vertices due to their shear size and their highly intricate design. Skinned meshes basically represent a family of meshes that can be deformed dynamically after creation. The vertices of these meshes are either attached to an underlying rigid structure that acts as the bone or are assigned automatic weights that help in determining their deformations. The animations provided by the former arrangement is called Skeletal Animation as the deformations are being managed by an underlying skeleton. In order to undergo animations, each vertex of the mesh is attached to a bone of the underlying skeleton, and is assigned a weightage which determines the level of movement the vertex will undergo in the rendered space as the bone is moved by a fundamental unit. The bones are connected to each other and provide motion relative to each other which in turn helps further in modelling real life motion. Skinned meshes also come with material information that provide texture information and are deform proportionately to the mesh.

Any 3D rendering software provide features for skinning meshes. 3D rendering apis like Three .Js provide inbuilt classes for creating skinned meshes and applying skeletal animation to them.

2.2 Bones

Bones are the basic building blocks of the skeletal animations. Any complex animation is constituted of animations applied to individual bones of the skeleton. Three .Js provide extensive features for working with bones of an underlying skeleton. Each bone is considered as a 3D object with a set of attributes and methods for manipulating the bones in a 3D space. The bones basically comprise of two parts the head and the tail. The head is attached to the parent bone or the bone above in the hierarchy and the tail part is attached to the child bone or the bone below in the hierarchy.
Fig. 4: A single armature as displayed in 3D rendering software Blender and A single armature displayed as an 3D Object in Three .Js

2.3 Skeleton

Skeleton is the hierarchy of the bone that act as the animation machine of every deformable meshes. A skeleton comprise either of one bone or a large number of bones. The richer the bone information provided to the skeleton, the more detailed the animation of the model. The bone that stays in the top of the hierarchy is known as the root bone. The bones that come successively afterwards are known as child bones of the root bone. Each child bone by default inherits the property of the root bone and thus inherits the transformations of the root bone. This helps in providing motions better resembling human motion. Three Js provides inbuilt classes for describing skeleton. Bone information of the skeleton can also be accessed using the skeleton object.

This project uses Skinned Meshes, Bones and Skeletons in extensive details. Already available skinned meshes have been analysed under the scope of this project in order to extract skeleton information and real time sensor data has been applied to them in order to model the human movement.

3 My Approach

Creating 3D models resembling humans is a tedious task and takes a lot of work hours. Additionally, it requires certain level of expertise in various 3D Modelling Softwares. Thus, this project has been developed on existing 3D Models that have skeleton information already attached with their meshes, in order to create the visualization. This project uses Three .Js Api under React Framework in order to model the sensor data coming from the BSN and visualize them in an already created human skeleton.

Fig. 5 shows the development pipeline employed under the scope of this project. The body sensor data is first collected and arranged in a suitable MoCap format for being loaded into a Node .Js application. The data is then calibrated with respect to a basic skeleton. The calibrated data is then applied to the bones of the reference figure and the 3D body model. This section explains the visualization steps in details one by one.

3.1 3D Model Creation/ Selection

3D rendering has been made easily available with the help of a lot of open source and proprietary softwares like Blender, 3Ds Max, Maya etc. This project mainly dealt with creation/selection of a
rigged human model i.e. a skinned mesh connected with a fully developed skeleton in order to model the movements of the mesh properly. The process of creation of a human model is a time taking process and is beyond the limited time frame of the project. As a result of which already developed and rigged models are being used for the task of visualization. A total of four human models have been examined under the scope of this project.

Fig. 6 states the models that are selected for development of the visualization software. Each of the models have a highly elaborate mesh that have been rigged using a detailed skeleton. The models were selected on the basis of their skeleton information. Each of the models had skeletons matching the reference figure used in the software. The models were also available in formats that could be loaded in Three.js boilerplate. Out of these 4 models 2 have been incorporated into the visualization software. The models were first loaded into a simple Three.js boilerplate. Three.js provides an exhaustive number of classes for loading 3D models. Out of these, the two in the first row are of .FBX format and the two in the bottom row belong to COLLADA format. The models were loaded using Object Loader classes provided by Three.js.

3.2 Model Rigging/Createion of Skeleton for Animation

As discussed in the theory section the skeleton acts as the center for animation in the 3D model. Each of the models incorporated in this project have a well designed skeleton to help in skeletal animation. The bone structure of model in the right bottom matched with that of the reference skeleton whereas the one in the top had a different root bone configuration as compared to the reference figure.

3.3 Applying Sensor Data to the model

Body Sensor Network used in the software is calibrated to stick figure as shown in the Fig. 9. The network data was emitted in the form of a dictionary storing the transformations in the form of a special matrix data structure used by Three.js called Matrix4. Consequently, the data for the joint angles had to be mapped separately to the skeleton. The separate models used in the software had different skeleton structures, and thus separate maps were created for mapping the separate models.
4 Experiments

A total of 4 3D models were selected based for implementation in the software. All the models were independently tested by loading them in separate Three.Js boilerplate. The male .FBX model as shown in top left hand corner of Fig. 6 had more than 4 vertices attached to every skeletal bone, which is above the permissible limit for providing animations in Three.Js. Consequently, that model could not be used for providing animation on real time sensor data. The female model shown in the top right hand corner was loaded with the reference figure for doing the real time visualization, but it had a different root bone as compared to the reference figure used in the software. Therefore, the model and the reference figure did not share the same orientation after the initial alignment with the sensors. Adding on to that, the bone coordinate system of the model were differently aligned with respect to the coordinate system of the reference figure, leading to completely incoherent movements between the reference figure and the 3D Model.

The male model in the bottom left hand corner of Fig. 6 was not used in the software because of the computational time being taken as a result of the detailed texture information present in the model. The model in the bottom right hand corner came out to be the perfect choice for visualizing the data. This model shared the same skeletal structure with respect to the reference figure and also the same root bone configuration. The initial poses of the models were not aligned as the 3D model was in T-Pose and the reference figure was in the I-Pose as is evident from Fig. 7 and Fig. 8. In order to align the poses the upper-arm bones of the 3D model was rotated by 90 degrees in the Z axis. This alignment led to a rotation in the coordinate axes of the child bones in the X and Y direction. The misalignment correction was attempted by using a - 90 degrees offset correction on the incoming orientation data. The Y axis of the child bones especially the lower-arm(both left and right arms) bone was found to be originally misaligned with respect to the reference figure and thus could not be corrected. The offsets were applied after each and every sensor rotation in order
to match the model and sensor orientations. While the misalignment in the X axis was removed and the movements of the 3D model completely resembled that of the reference figure for any movement of the arms in the Z and the X axis. The misalignment in Y axis hindered the movement of the 3D model in a few composite rotations, but it resembled the stick figure for most of the rotations.

Fig. 7: Offset Corrected in X axis  
Fig. 8: Movement in Z axis

Fig. 9: Horizontal arm movements  
Fig. 10: Vertical arm movements

5 Conclusion

This project aimed at visualizing real-time data coming from a body sensor network using a suitable 3D Model. It was found out that it is important for the model to share the same skeletal structure as compared to the reference figure used for calibrating the body sensor network. Misalignment in bone orientations can lead to in-coherence between the reference figure and the 3D model. It was also found important for the initial pose of the sensor and the rest pose of the 3D model to be aligned. Applying external rotation during visualization of the real time data can lead again to miscalibration and thus incoherence in movements.

References

A Correction System For Calibration Patterns

Marcel Rogge\textsuperscript{1} and Yuriy Anisimov\textsuperscript{2}

\textsuperscript{1}m Rogge14@cs.uni-kl.de
\textsuperscript{2}Yuriy.Anisimov@dfki.de

Abstract. Camera calibration requires images of calibration patterns. These calibration patterns need to be detected before the intrinsic parameters of the camera can be computed. However, it is possible that the pattern cannot be accurately detected. In those cases, the calibration of the camera can be faulty. Ensuring that the calibration patterns are correctly detected is therefore necessary. However, doing so manually is time-consuming. The automatic correction system proposed here will discard incorrectly detected calibration patterns automatically. This will reduce the amount of manual labour required to calibrate cameras.

Keywords: pattern-based calibration, camera calibration, RANSAC, M-estimator

1 Introduction

Calibration patterns are required to compute the intrinsic parameters of a camera. For this, an image of e.g. a chessboard is taken. From this image, the corners of the squares on the chessboard can be found. With the position of these corners, it is possible to compute the intrinsic parameters of the camera. However, the detection of these corners is not always accurate. Many factors can affect this. Two scenarios where this is likely to happen is if the chessboard is too far away or on the boundary of the image. Every image where the calibration pattern was incorrectly detected will add errors into the computation of the intrinsic parameters. Therefore, it is necessary to confirm that the calibration patterns have been correctly detected. Naturally, doing this manually is tedious and time-consuming. This was our motivation to proposes an automatic approach: A correction system for calibration patterns that will automatically discard patterns that cannot be accurately detected.

This can be achieved by fitting lines to the detected corners. The resulting line should roughly match the pattern, even if a few corners were incorrectly detected. Given a line that matches the pattern, correctly detected corners should be on the line. If there is a significant distance between any of the corners and the line, then the pattern cannot be correctly detected. A visualization of this can be seen in Fig. 1. The distance between red points and the blue line can be considered to discard a pattern.

Obviously, the detected line will not always match the pattern perfectly. In those cases, there is also an angle consideration to improve the approach. For more details on this, see Section 2.

2 My approach

The approach can be separated into three steps: 1. Detect corners of the calibration pattern. 2. Compute lines according to the corners. 3. Classify pattern as correct or incorrect. These three steps will be explained in the following.

(1) Corner detection is achieved using the OpenCV function findChessboardCorners [2]. It detects the internal corners of the chessboard where black squares are connected. The results are restricted to pixel coordinates, which may not be perfectly accurate. This can be improved using the cornerSubPix function from OpenCV [1]. It enables subpixel accurate results using gradients.
(2) Line computation was tested with two different methods: RANSAC [4] and an M-estimator [5]. First, all detected corners are separated into different sets for each horizontal and vertical line inside the pattern. The line computation is then performed for each of these sets individually. Our RANSAC implementation considers all combinations of two corners to compute lines, rather than iteratively selecting random corners. This is due to the small number of corners that make up each line inside the pattern. We can consider all combinations inside a set without increasing computational time. Besides this, the method works the same as a standard RANSAC approach: Given a combination of two corners, compute a line and sum the distance between the line and all corners. The best line is the one with the smallest summed distance out of all combinations tested. Distance calculation will be explained in the next step.

The M-estimator method is the fitLine function available in OpenCV [3]. It works by minimizing the function shown in Eq. 1, where $p(r)$ is a distance function. There are different distance functions available, the one used in this work can be seen in Eq. 2, where $r$ is the distance between a point and the line.

$$\sum_i p(r_i) \quad (1)$$
$$p(r) = r^2/2 \quad (2)$$

(3) Classification is achieved by using thresholds. The idea of the approach is to consider the distance between all corners and the computed line. If the largest distance value exceeds a threshold, the pattern is classified as incorrect.

Fig. 2 helps to visualize the distance calculation, which will be explained now: Given the green point, we want to know its distance to the black line. We need to find the norm vector (red line) of the black line and any red point that lies on the black line. The norm vector can be found by simply rotating the direction vector of the black line by 90°. This means, given the direction vector is $[x_1y_1]$, the norm vector can be $[-y_1x_1]$. Next, a vector pointing from the red point to the green point is needed. This is the blue line, which is given by $[x_{green} - x_{red}, y_{green} - y_{red}]$. When performing scalar projection of this blue line onto the norm vector (follow the orange line), the result is equal to the length of the brown line. Therefore, the distance is given by Eq. 3, where $P_g$ and $P_r$ are the green and red points respectively and $L_b$ is the black line. The $\cos \theta$ can be substituted through the dot product definition from Eq. 4, where $n$ is the norm vector. That leads to Eq. 5, which is the formula by which distances between points and lines are calculated in this work.
In Section 1 it was mentioned that the fitted lines will not always match the pattern perfectly. This can have a bad impact on the performance of the classification. Consider Fig. 3: In the left image, there is only one corner that has a significant distance to the line. This distance is relatively large, which makes it easy to discard by using a threshold. The right image has more points with a noticeable distance to the line. However, the maximal distance is smaller than for the left image. When having to adjust for general inaccuracies of detected corners and fitted lines, the threshold cannot be set close to zero. Otherwise, many correct patterns could be discarded. Therefore, distance thresholds are not reliable for fitted lines that do not match the pattern. For this reason, an additional criterium is used to classify patterns: A comparison of the relative position between different fitted lines. This is motivated by the observation that the lines in the pattern are parallel to each other. Depending on the orientation of the pattern, this will not be exactly true in the image. However, neighbouring lines will still be close to parallel, which means that they intersect at a very far away point. Another observation is that fitted lines that do not match the pattern can only be the result of incorrectly detected corners. This means that we can check if neighbouring fitted lines are almost parallel to each other (within a threshold). If this parallel threshold is exceeded for any two neighbouring lines, then it must mean that the pattern was incorrectly detected.

To check whether two lines are parallel, we first determine a line that is orthogonal to the first of both lines. Should both lines be parallel, then the orthogonal line will also be orthogonal to the second line. Determining the orthogonal line is done the same way as for the norm vector in the distance calculation. To check if it is orthogonal to the second line, we can simply compute the dot product. As can be seen from Eq. 4, if the angle $\theta$ between both lines is $90^\circ$ then the dot product will be zero. Again, due to general inaccuracies and since the orientation can make them not exactly parallel, we have to consider a sensible threshold. This will make it possible to improve classification in cases where the fitted lines do not match the pattern.

In conclusion, a pattern is classified as incorrect if the distance of any detected corner to the fitted line exceeds a threshold or if any fitted line is not parallel (given a threshold) to other fitted lines.
3 Experiments

There are two special cases of calibration patterns that are likely to be incorrectly detected. The first is when a pattern is captured on the boundary of the image. The second are patterns that are captured from a far distance to the camera. Experiments have been conducted on 272 images that include both of these cases. Of these, 85 images could not have a pattern detected at all. This results in 187 detectable patterns. These patterns were evaluated manually and classified as correct or incorrect: 142 patterns are correct, while the remaining 45 are incorrect.

Two methods were used to fit lines to the pattern: RANSAC and M-estimator as described in Section 2. In the following, the experiments will be discussed for both separately before comparing the two.

RANSAC generally computes accurate lines that match the pattern of the board well. An example of this can be seen in Fig. 4 on the left. However, there are rare cases where it also produces bad lines that do not fit the pattern at all. This can be seen in Fig. 4 on the right. In those cases, distance thresholds become unreliable, as was explained in Section 2. For this reason, it becomes necessary to check whether the fitted lines are parallel to each other. This is shown in Fig. 5: The left image shows a case where the distance threshold is too large to discard the pattern due to the inaccurately fitted line. This could also be addressed by lowering the distance threshold. However, the right image of the same figure shows a case where this would cause problems. It shows a correctly detected pattern where, due to general inaccuracies, the distance between the fitted line and detected corners is rather large. If the distance threshold is lowered, the algorithm would discard correct patterns like this even though they are correct. Adding the second criterion of checking whether the fitted lines are parallel makes lowering the distance threshold unnecessary. The pattern on the left side of Fig. 5 can be discarded using the second criterion and the image on the right will not be wrongly discarded.

The M-estimator tends to produce lines that do not fit the pattern perfectly. An example of this can be seen in Fig. 6 on the left side. At the same time, there are also cases where the fitted line significantly differs from the pattern. This can be seen in the right image of Fig. 6. Checking whether the fitted lines are parallel to each other seems like a good choice for all patterns since the lines tend to not fit perfectly. However, similarly as with RANSAC and the distance threshold, if we want to discard all patterns using this one criterion, we would have to lower the threshold for parallel lines to a point where many correct patterns would be discarded. The thresholds for parallel lines are only reliable for significantly skewed lines. This is why we still need to check the distance
**Fig. 4.** RANSAC line accuracy for different patterns. On the left is an example of RANSAC producing an accurate line. The right shows an example for badly fitted lines.

**Fig. 5.** RANSAC problem with distance thresholds. The left image is an example where a smaller distance threshold is required to discard the pattern due to inaccurate lines. The right image shows an example where due to general inaccuracies a small distance threshold would discard a correctly detected pattern.
between lines and corners as well. So the M-estimator method requires similarly as the RANSAC method two criteria to ensure the best possible classification.

![M-estimator line accuracy for different patterns.](image)

**Fig. 6.** M-estimator line accuracy for different patterns. On the left is an example of the M-estimator producing a not perfectly accurate line. The right shows an example of a badly fitted line.

In conclusion, both RANSAC and M-estimator benefit from using both criteria for classifying patterns. This is why the evaluation will be conducted while using both classification criteria for both methods. All results are summarized in Table 1 and Table 2. We can see that both methods produce nearly the same results. The first difference is that RANSAC has one more case where it classifies an incorrectly detected pattern as correct. Specifically for a pattern that was captured on the boundary. The other difference is that the M-estimator also classifies one more incorrectly detected pattern as correct. However, this happens for a pattern that was far away from the camera. Besides this, using both line fitting algorithms, the system arrives at the same classifications for each image. There are also two cases where correctly detected patterns are classified as incorrectly detected. These are due to general inaccuracies that cause the distance between detected corners and fitted lines to pass the distance threshold. However, increasing the threshold further would result in too many incorrectly detected patterns passing the threshold as well. The currently used parallel threshold does not have any cases of misclassifying correctly detected patterns.

Overall, it can be said that both methods have a total accuracy of 95.2% with 178 of the total 187 patterns correctly classified. When separating by type of image, then RANSAC has an accuracy of 95.3% for patterns on the boundary and an accuracy of 95% for patterns that are far away. The M-estimator has an accuracy of 98.1% for patterns on the boundary and 93.8% for far away patterns. We can also consider the percentage of correctly as correct or incorrect classified patterns. For correct patterns, both RANSAC and the M-estimator have accuracies of 98.6% while for incorrect patterns they are 84.4%.

**Table 1.** RANSAC results: Classification of patterns that were correctly and incorrectly detected. Results are separated by patterns that are on the boundary of the image and far away.

<table>
<thead>
<tr>
<th>Correct (boundary)</th>
<th>Incorrect (boundary)</th>
<th>Correct (far)</th>
<th>Incorrect (far)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Right</strong></td>
<td>75</td>
<td>27</td>
<td>65</td>
</tr>
<tr>
<td><strong>Wrong</strong></td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 2. M-estimator results: Classification of patterns that were correctly and incorrectly detected. Results are separated by patterns that are on the boundary of the image and far away.

<table>
<thead>
<tr>
<th></th>
<th>Correct(boundary)</th>
<th>Incorrect(boundary)</th>
<th>Correct(far)</th>
<th>Incorrect(far)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right</td>
<td>75</td>
<td>28</td>
<td>65</td>
<td>10</td>
</tr>
<tr>
<td>Wrong</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

4 Conclusion

A correction system for calibration patterns is proposed that can automatically discard incorrectly detected patterns. This is achieved by first fitting lines to the detected points, which can be computed using one of two methods. Using these fitted lines, two different classification methods are used: One considers distances between detected corners and previously fitted lines. The other compares the fitted lines between each other. These two methods complement each other, increasing accuracy. Overall, the system achieves an accuracy of 95.2% for correctly classifying patterns. The accuracy for specifically incorrect patterns can be increased by adjusting the thresholds. This would, however, result in an increased amount of correctly detected patterns being discarded. It has to be decided on an individual basis whether this can be afforded.

References

Survey on Long-Tail Learning Methods

Marco Stricker\textsuperscript{1} and Stephan Krauss\textsuperscript{2}

\textsuperscript{1} m.stricker13@cs.uni-kl.de
\textsuperscript{2} Stephan.Krauss@dfki.de

Abstract. The long-tail problem describes the issue, that the dataset our machine learning method is supposed to learn from, is imbalanced and heavily skewed towards a few head classes with many examples, while there are many underrepresented tail classes with few samples to learn from. Classic machine learning algorithms have the problem, that they overfit to the head classes and disregard the tail classes. This survey gives an overview of several approaches of how to handle this problem. The presented approaches are categorized to several main ideas, like knowledge transfer, modifying the loss function, feature transfer and data generation.

Keywords: Long Tail, Imbalanced Dataset, Machine Learning

1 Introduction

Modern machine learning algorithms require a balanced dataset in order to learn a classification which is equally good over all classes. However, real world datasets are often not balanced and show a skewed data distribution [12,14]. This means, that there are many samples for a few classes, but few samples for many classes. This problem is known as an imbalanced dataset or learning with long-tail data. A simple example for this would be a face identification task. While there will be thousands of samples of famous people like politicians, there will be only a few for private persons. The classes with many samples are also denoted as head classes, while the ones with few samples are known as tail classes in this context. As a consequence of such an imbalanced dataset, normal machine learning algorithms will overfit on the head classes and will mostly disregard the tail classes [12]. However, in order to develop a robust method which can also deal with underrepresented classes novel approaches are necessary. Naive methods try to solve this problem by including a naively weighted loss function, over-sampling the tail classes or under-sampling the head classes. These approaches overfit on the few samples in the tail classes and have problems with unknown samples of a tail class. On the other hand under-sampling of head classes will discard important information on which we want to train [12]. This survey paper aims to give an overview of more advanced methods, which deal with the long-tail problem and overcome the shortcomings of the naive ones. We have grouped these methods into different categories. First, we explain knowledge transfer methods in Section 2. Then we explain one of the most popular approaches which is modifying the loss function. In Section 4 we explain how certain methods try to transfer features from one class to another. While the most recent literature tries to avoid data generation approaches for overfitting reasons, one of these methods is introduced in Section 5. Finally, we give a summary of these methods, where we evaluate the proposed methods on how well they would work on different problems other than the ones which are presented in the paper. Furthermore we state how easy they are to apply to other problems. This easiness is rated based on how much work an adaption of the method would need to be incorporated into another problem. We evaluate these methods in such a way, because a qualitative evaluation is impossible, because many methods are dealing with different problems using different datasets and evaluation metrics. A short overview of all presented methods can be found in Table 1.
Table 1. Summary of all introduced long-tail training methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
<th>Main Idea</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning to Model The Tail</td>
<td>Knowledge Transfer</td>
<td>Change of network parameters</td>
<td>[12]</td>
</tr>
<tr>
<td>Graph Embeddings and Graph CNNs</td>
<td>Knowledge Transfer</td>
<td>Build relations between classes</td>
<td>[15]</td>
</tr>
<tr>
<td>Two Stage Training</td>
<td>Knowledge Transfer</td>
<td>Train on full dataset and undersampling</td>
<td>[2]</td>
</tr>
<tr>
<td>Finetuning Deep Model for Object Detection</td>
<td>Knowledge Transfer</td>
<td>Cluster similar classes</td>
<td>[9]</td>
</tr>
<tr>
<td>Tail Class Promotion Loss</td>
<td>Loss Function</td>
<td>Align norm of weight vector</td>
<td>[4]</td>
</tr>
<tr>
<td>Range Loss</td>
<td>Loss Function</td>
<td>Decrease intra-class and increase inter-class variation</td>
<td>[10]</td>
</tr>
<tr>
<td>Class Balanced Loss</td>
<td>Loss Function</td>
<td>Include weighting factor</td>
<td>[1]</td>
</tr>
<tr>
<td>Uncertainty Factor</td>
<td>Loss Function</td>
<td>Bigger region in feature space for tail classes</td>
<td>[7]</td>
</tr>
<tr>
<td>Max-margin Class Imbalance</td>
<td>Loss Function</td>
<td>Similar to range loss</td>
<td>[5]</td>
</tr>
<tr>
<td>Reweighting Examples</td>
<td>Loss Function</td>
<td>Weight-factor for every sample</td>
<td>[10]</td>
</tr>
<tr>
<td>Quintuplet Sampling</td>
<td>Feature Transfer</td>
<td>Use quintuplets for feature representation</td>
<td>[6]</td>
</tr>
<tr>
<td>Feature Transfer Learning</td>
<td>Feature Transfer</td>
<td>Modify features of tail class to resemble head class</td>
<td>[14]</td>
</tr>
<tr>
<td>Large-Scale Long-Tailed Recognition</td>
<td>Feature Transfer</td>
<td>Special mapping into feature space</td>
<td>[8]</td>
</tr>
<tr>
<td>Conditional Generative Adversarial Networks</td>
<td>Data Generation</td>
<td>Use cGANs to create data</td>
<td>[3]</td>
</tr>
</tbody>
</table>

2 Knowledge Transfer

The goal of the following methods is to gather knowledge from the head classes and transfer it to the tail classes. Which information is being extracted from the head classes depends on the method. Because the tail classes are getting different information, depending on the method, the way how they incorporate it differs from method to method, too.

2.1 Learning to Model The Tail

The approach of Wang et al. is to train a meta-model [12]. The idea is that a model learns how the network parameters behave during the training of the head classes and then applies this behaviour to the tail classes. Therefore, first a model $H$ is trained solely on the head classes. After that another model $T$ is trained on a subset of samples of the head classes. This is a so-called few-shot model. Following this, the meta-network $M$ can be trained by learning a mapping from the parameters defined by $T$ to the parameters defined by $H$. Choosing the size of the samples is crucial. Therefore, in order to train a network as general as possible, multiple meta-networks are trained. Each for a different split into head and tail classes (i.e. the number of samples which needs to be classified as a head or tail class) and for each split multiple sample sizes. Finally, the results are combined in a single network via residual connections. Furthermore, the authors propose an efficient training method. Consider one model for a possible split in head and tail classes. Then, one should start by training the model with the fewest number of classes in the head. Using the resulting parameters the authors then successively add one more class at a time. Each block itself is trained to be tuned for a few-shot model of the size $2^i$ where $i$ denotes the index of the model. Note, that they start at index $N$ (number of models) and end at 0.

2.2 Long-Tail Relation Extraction via Knowledge Graph Embeddings and Graph Convolution Networks

The goal of Zhang et al. is to extract relationships between entities [15]. Their entire approach is designed to deal with long-tailed data. First, they need to learn relational knowledge, which aims to help with long-tail data. According to my understanding it should be possible to adapt this step to work with other applications, too. In latent space similar classes are close to each other. In the case of relations between words, this space is achieved with an embedding and an encoding layer. Now implicit and explicit relational knowledge between classes can be extracted with knowledge graphs and graph convolutional networks. The idea behind this is to build a graph which represents how
the classes are connected or related. For example “place of burial” (a long-tail class) will end up as a subcategory of “place of death” (a class with many samples) in the graph. Using this relation we know from which classes we can transfer knowledge to which tail classes in order to increase performance. It would decrease performance if we transfer knowledge from a class to another completely unrelated class. The second step is to incorporate this knowledge into a specific problem. In this case, the authors used an knowledge-aware attention mechanism in order to transfer knowledge from head to tail classes. However, this is a specific way of achieving it for this kind of problem and would likely not work for different tasks.

In our opinion this approach is not as general as the first one and would need a lot of changing in order to be fit to a custom task. We also think that the idea of building implicit and explicit knowledge relations synergizes well with the author’s task of relation extraction. Therefore, it would be interesting to see how well it deals with different kinds of problems.

2.3 Two Stage Training

Cui et al. tried to solve the problem of long-tailed distributed data by employing a two stage training procedure [2]. The first stage trains the network on the complete dataset. In this stage the imbalance is still there, however there are also many training samples on which the network can learn a first feature representation. In the next stage some images are removed from the training set in order to achieve a more balanced dataset. The network is then trained again on this smaller set and small learning rate. This approach tries to transfer knowledge from the first phase of training to the second one. With this approach an increased performance could be achieved, especially in the tail classes.

2.4 Factors in Fine-tuning Deep Model for Object Detection with Long-Tail Distribution

Ouyang et al. follow a similar idea as the one described in Section 2.2 by defining some form of hierarchy where similar classes are grouped together and the features are then transferred from the head classes to the similar tail classes [9]. The authors mention, that to group classes together any clustering method can be used. However, the authors applied the visual similarity, since it performed best among the tested ones. This clustering method creates a tree of clusters. To transfer the learned features, the authors specify the following algorithm: Each node in the clustering tree corresponds to a model. A model is fine-tuned by using the model parameters of the parent node. Also for training a node, only a subset of classes is used. Namely, the classes of the node as positive samples and the classes of negative samples are only extended by the classes of the parent node.

3 Loss Function

Another very popular approach in order to handle long-tailed data, is to adapt the loss function in such a way, that it can account for imbalanced data. Modifying the loss is popular, because it is very easy to adapt it to other tasks.

3.1 Tail Class Promotion Loss

Guo and Zhang introduce an “Underrepresented Classes Promotion Loss” [4]. This loss aims at aligning the norm of the weight vector of the tail classes to the weight vector of the head classes. This loss specifically iterates over the set of tail classes. Therefore, this loss is jointly trained with another loss function, which is usually used for such a task. However, in Guo and Zhang’s case of face identification, they also used another newly defined loss function, which is detailed in their paper.
3.2 Range Loss for Face Recognition

The idea of the loss function defined by Zhang et al. is to decrease intra-class variations and increase inter-class variations [16]. A similar idea has already been proposed in the contrastive loss [11]. However, it needs enough samples in order to work effectively, which is not always the case in long-tailed distributed data. Therefore, the range loss is defined over all samples within a minibatch, unlike the contrastive loss which is defined on pairs. Furthermore, the concept of hard negative mining is incorporated in the loss function. This concept focuses on samples at the classification boundaries. Therefore, the range loss is defined as a combination of intra- and inter-class loss:

\[ \mathcal{L}_R = \alpha \mathcal{L}_{\text{intra}} + \beta \mathcal{L}_{\text{inter}} \]

with \( \alpha \) and \( \beta \) being some weights. The intra-class loss models the concept of hard negative mining, by penalizing the maximum harmonic range. Since this loss is responsible for the intra-class variations, the harmonic range is calculated for each class as:

\[ \mathcal{L}_{\text{intra}} = \frac{1}{\sum_{j=1}^{k} \frac{1}{D_j}} \]

Where \( D_j \) is the j-th largest distance. The loss to increase the inter-class distance is defined as:

\[ \mathcal{L}_{\text{inter}} = \max(M - D_{\text{center}}, 0) \]

\( M \) is defined as the max optimization margin of \( D_{\text{center}} \), which represents the shortest distance between two classes. Finally, the authors combined the range loss with the softmax loss as their final loss function. Lastly, the authors mention that their approach seems to work really well with residual networks.

3.3 Class-Balanced Loss Based on Effective Number of Samples

Cui et al. define a weighting factor which can be added to the loss function [1]. The advantage of this approach is, that it can be applied to any loss function and is therefore independent of the problem. The step to calculate the weighting factor is done in a way, that is independent of the data distribution. Instead of just defining a factor inverse to the number of occurrences, random samples are selected from each class. Cui et al. define a space around the samples and they then try to find samples, such that they cover the whole space assigned to the class. A data point contained in a space covered by a sample does then not need to be included. With that similar datapoints (for example those created through data augmentation) are then grouped together and therefore the number of effective samples is smaller than the number of actual datapoints. Using this the loss can be defined as:

\[ CB(p, y) = \frac{1}{E_{n_y}} = \frac{1}{1 - \beta n_y} \]

Where \( E_{n_y} \) represents the number of effective samples. The weight \( \beta \) is being calculated solely based on the number of samples per class \( N \), namely: \( \beta = (N - 1)/N \). The authors also give further examples, of how to include this factor for the softmax cross entropy loss, the sigmoid cross entropy loss and the focal loss.

3.4 Striking the Right Balance with Uncertainty

Khan et al. define a loss function which incorporates uncertainty [7]. The effect of this uncertainty should give classes with low confidence, i.e. rare classes or difficult classes, a bigger region in feature space. Therefore, the model’s ability to generalize is supposed to improve. Calculating this Bayesian uncertainty is achieved with deep CNNs. Then, the softmax loss is modified in order to incorporate this uncertainty. In this loss function \( w_y^T f \) is being calculated for a feature \( f \). This can be rewritten as \( w_y^T f = \|w_z\| \|f\| \cos(\alpha_y) \) where \( \cos(\alpha_y) \) can be exchanged with a function \( \psi(\alpha_y) \), where \( \psi(\alpha_y) \) is responsible for incorporating the uncertainty. This function penalizes samples in the margins harder for certain classes.
3.5 Max-Margin Class-Imbalanced Learning with Gaussian Affinity

Similar to the Range Loss of Section 3.2, Hayat et al. try to increase inter-class distance and decrease it for intra-class samples [5]. This “affinity-loss” as the authors call it, maps the input to a discriminative Euclidean space. Furthermore, the distance in this space can be used as a similarity measure. This results in clusters where class points are grouped together, which gives their formulation of the loss function: \( L = L_{mm} + R(w) \), where \( L_{mm} \) includes the similarity measure in Euclidean space and \( R(w) \) enforces a uniform distribution of the class centers.

3.6 Learning to Reweight Examples for Robust Deep Learning

In this approach Ren et al. try to learn weights which are then applied in the loss function for each sample separately [10]. The descent direction of a set of samples is observed during training. The reweighting is then done according to the similarity to the descent direction on samples in the validation set. The author first introduce a naive version of realizing that and then present a faster solution to this problem of setting \( w \) for sample \( i \) at time step \( t \): \( w_{i,t} = max(u_{i,t}, 0) \) where \( u_{i,t} = -\eta \frac{\partial}{\partial \epsilon} f_j^\theta(\theta_{t+1}(\epsilon)) \mid_{\epsilon_i = 0} \). This formula rectifies the output and takes the validation samples into account depending on the value of \( \epsilon \). The rectification is done, in order to get non-negative weights. Lastly, this formula represents one step of the gradient descent. Note that the authors further constrain the weights to sum up to 1.

4 Feature Transfer

Feature transfer methods are taking the features of an imbalanced dataset and map them into a feature space in such a way, that they are balanced in that space.

4.1 Quintuplet Sampling Based Hinge Loss

Huang et al. combines feature transfer methods and loss methods [6]. However, the most crucial idea is a specific feature transfer. Namely, the construction of quintuplets whose aim is to embed an image into a feature space, in which the features are discriminative and balanced with respect to the classes. As a prerequisite the data needs to be clustered. This can be achieved with k-means, for example. Having a clustered dataset, one can find quintuplets by finding a point for each class and then combine this point with the most distant point within the same cluster, the nearest and farthest point from another cluster, but within the same class and the closest point of another class. These quintuplets can be considered the most discriminative representation of their class, while the other samples representing this class are ignored and therefore effectively removing the imbalance of the data. Then, the authors formulate a “Triple Header Hinge Loss” which optimizes those four distances.

4.2 Feature Transfer Learning for Deep Face Recognition with Long-Tail Data

Yin et al. took an idea from low-shot learning [14]. Namely, they use center-based feature transfer to change the distribution of long-tail class data points to resemble the distribution of the head-classes in the feature space.

In detail, their framework consists of a concatenation of multiple networks. First, we have an encoder-decoder network whose aim it is to train an internal representation \( g \). This representation is then forwarded to a module \( G \) which applies the feature transfer. To perform the feature transfer, the authors calculate the mean of each class as the average over all features. Specifically, for long-tail classes, images with extreme pose variations were removed. Then, the intra-class variance from head
classes is transferred to the tail classes. As an optimization Principal Component Analysis, with taking the top 150 eigenvectors, is applied. Following $G$ a filtering network is used to prepare the features as an input to the fully connected layer, which performs the classification. During training they alternate between feature transfer and feature learning.

4.3 Large-Scale Long-Tailed Recognition in an Open World

Liu et al. try to combine long-tail learning with few-shot learning in an open world setting [8]. This is done by mapping an image into the feature space such that the following property is satisfied: Visual concepts should relate to each other in order to represent a closed world classification. In order to achieve this, two steps are necessary. In the beginning an embedding calculates the features, which are then augmented by meta-learning methods. To calculate the embedding, the centroids of each class need to be estimated in such a way, that the distances between each other are high, while the distances inside the classes are low. This concept of increasing inter-class and decreasing intra-class distances is an often occurring idea for many methods. When the network encounters a tail class, it should use these calculated features in order to enhance the features of the tail class. This is done by calculating a special factor using another neural network and multiplying it with the set of all centroids. Combining this result with the feature vector of the tail class results in a final feature vector used for classification. Finally, the authors note, that they could further increase the performance by adding a “modulated attention”, which modifies the feature map to give a special focus on certain areas. This is supposed to help the network to differentiate between head and tail classes.

5 Data Generation

One simple approach to overcome the problems of a long-tailed data distribution is by just generating more data and oversampling the tail-classes. However, the most recent literature does not see this as the solution for this problem. Because with oversampling the tail classes, one will overfit on the few samples in these classes. However, for the sake of giving a complete overview of the different approaches, the data generation approach is also covered by choosing one of the more recent methods as a representative.

5.1 Effective Data Generation for Imbalanced Learning using Conditional Generative Adversarial Networks

Douzas and Bacao focus on data generation and oversampling [3]. The authors are using cGANs (conditional Generative Adversarial Networks) to generate more of the underrepresented values. A cGAN takes additional information as an input compared to classic GANs. The authors use the labels of the tail classes as the additional input. Therefore, the GAN will learn to create more data for the tail classes. But a disadvantage is, that the authors only applied and tested their approach in a context of binary classification, so far.

6 Additional Techniques

Lastly, we present three additional and novel methods introduced by Wertheimer and Hariharan [13] which do not fit into any of the categories above. Each single technique improves long-tail learning by itself.

The authors use prototypical networks as a basis combined with a new training method, a preprocessing step and finally they add bilinear pooling to the network.
During training the authors apply batch folding. This procedure performs leave-one-out cross validation within each batch. Prototypical networks work with a split into reference and query images for a specific batch. Leave-one-out cross validation aims to remove this split by interpreting the whole batch as the reference part. Additionally, the contribution of a single element gets subtracted, or as the authors call it, “folded”, if it is part of a query in a prototype. This results in a more stable training.

In the classification model some preprocessing is included before the final prediction is done. Namely, a localization method to find regions of interest is being used. The authors experimented with two different approaches: unsupervised and few-shot learning. Unsupervised tries to learn this localization automatically as a sub-module in the learner module of the prototypical network. Few-shot learning on the other hand has images containing ground truth bounding boxes in the evaluation set. The advantage of incorporating localization in such a way is, that the network can still be trained in an end-to-end fashion. In our opinion this approach can be transferred to any domain by using some form of attention mechanism.

Finally, bilinear pooling is applied as the last step. The authors call the way of how they include it “Covariance Pooling”. The idea behind this method is to artificially increase the feature space and therefore increases the capabilities of the model, so that it can express a more complex problem. Covariance Pooling calculates the pixel-wise outer product of two feature maps. The two feature maps are the foreground (areas which the localization method pays attention to) and background maps from the localization phase. After that the signed square-root normalization is computed which is not followed by a projection into the unit sphere. Lastly, average pooling is applied.

7 Evaluation

Table 2 provides a brief evaluation of all presented methods. Each method is evaluated with respect to how easy it is to incorporate it into other solutions. The evaluations range from easy to medium to hard. We define easy as: For incorporation only a small addition needs to be done, e.g. adding a factor in the loss function. While in medium difficulties another major step, e.g. training a second network, is necessary. Hard further requires to adapt the given solution by changing some processing steps, so that they fit to the problem. Furthermore, we mention whether the given method is applicable to other problem domains, if it was mentioned in the source paper. This means, whether you can use a solution tested on face recognition for other tasks like e.g. translation with neural networks.

<table>
<thead>
<tr>
<th>Method</th>
<th>Incorporation</th>
<th>Different Problem domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning to Model The Tail</td>
<td>Medium</td>
<td>Yes</td>
</tr>
<tr>
<td>Graph Embeddings and Graph Convolution Networks</td>
<td>Hard</td>
<td>Maybe</td>
</tr>
<tr>
<td>Two Stage Training</td>
<td>Easy</td>
<td>Yes</td>
</tr>
<tr>
<td>Finetuning Deep Model for Object Detection</td>
<td>Medium</td>
<td>Some</td>
</tr>
<tr>
<td>Tail Class Promotion Loss</td>
<td>Easy</td>
<td>Yes</td>
</tr>
<tr>
<td>Range Loss</td>
<td>Easy</td>
<td>Some</td>
</tr>
<tr>
<td>Class Balanced Loss</td>
<td>Easy</td>
<td>Yes</td>
</tr>
<tr>
<td>Uncertainty Factor</td>
<td>Easy</td>
<td>Some</td>
</tr>
<tr>
<td>Max-margin Class Imbalance</td>
<td>Easy</td>
<td>Some</td>
</tr>
<tr>
<td>Reweighting Examples</td>
<td>Easy</td>
<td>Yes</td>
</tr>
<tr>
<td>Quintuplet Sampling</td>
<td>Medium</td>
<td>Some</td>
</tr>
<tr>
<td>Feature Transfer Learning for Face Recognition</td>
<td>Medium</td>
<td>Some</td>
</tr>
<tr>
<td>Large-Scale Long-Tailed Recognition</td>
<td>Medium</td>
<td>Some</td>
</tr>
<tr>
<td>Conditional Generative Adversarial Networks</td>
<td>Medium</td>
<td>Some</td>
</tr>
</tbody>
</table>
8 Conclusion

In this survey we presented several methods which aim at improving machine learning algorithms for long-tailed datasets. We defined several categories which we assigned the different methods to. Furthermore, we tried to evaluate these methods by making an educated guess which methods are better in the following categories: how easy they are to adapt to your own problem from the presented use case and how tailored they are for a specific case e.g. will the method used for face detection be able to achieve similar results on different tasks.

References

Survey on Data Augmentation and other Tricks for Object Detection

Resha Thacker and Michael Först

1 thacker@rhrk.uni-kl.de
2 David_Michael.Fuerst@dfki.de

Abstract. Recent advancements of deep learning have been prominent in the field of computer vision. Researchers achieved great success with the deep learning algorithms but are still facing some challenges. Insufficient amount of training data is one of the challenges for deep learning algorithms. Researchers show that the performance of deep learning algorithms is based on the amount of training data. There are a few methods to resolve this problem. For instance, data augmentation, drop out, transfer learning. Data augmentation is the technique which increases data diversity for neural network models and reduces the overfitting. This paper gives analysis of six different approaches of data augmentation. All these approaches are mainly divided into classical based approaches and deep learning-based approaches. This paper presents the result of all approaches and compare them with the baseline.

Keywords: Deep learning, Machine learning, Data Augmentation, Neural Network

1 Introduction

To train a neural network enough training data should be present. Not having enough training data leads to overfitting, where a model will be too good for the training data such that the model fits almost perfectly for the training data but gives a poor performance on the unseen data. To deal with the overfitting problem, there are some regularization techniques. In general, two regularization techniques are used, which are drop out and batch normalization [5], which avoids overfitting and leads to generalization. Another method to deal with this problem is data augmentation. Analysis of different approaches of data augmentation are briefly described in this paper, which are based on classical methods and deep learning-based methods. The classical data augmentation approaches are Texture and Geometry [7], Cut and Paste [3], Data generation for Urban Driving [1] and deep learning-based approaches are Synthetic Data Augmentation using GAN’s for liver lesion classification [4], Smart Augmentation [6], Auto-Augmentation [2]. Texture and Geometry approach shows that changing the texture and geometry of the images gives the more diverse and large dataset. Another approach is the ‘cut’ and ‘paste’ method, which generates the synthetic data by placing the cut objects on a random background. Data generation for urban driving shows that the combination of real and synthetic images is also one approach to get diverse data. Deep learning-based approaches mostly using GAN’s for generating more diverse data. Synthetic Data Augmentation using GAN’s for liver lesion classification (SDAG), use classical data augmentation for training purpose and use GAN’s to enlarge the data size and data diversity further. In the paper of smart augmentation, they feed the combination of two or more images to the target network to reduce the loss of the target network. We have many data augmentation methods but which will work best based on the problem statement and dataset is still manual work. In the paper of Auto Augmentation, with the help of reinforcement learning, they show that a neural network can pick the best technique of data augmentation automatically. Section 2 and 3 described classical and deep learning based approaches respectively. In the section 4, results and conclusion shows the effect of diverse Data Augmentation methods in terms of accuracy.
2 Classical Data Augmentation Approaches

This section describes classical data augmentation approaches in which the methods apply rotation, translation, and place cut objects on a random background. Some approaches perform data augmentation by combining real and synthetic data.

2.1 Texture and Geometry

Romera et al. [7] uses texture and geometry augmentation to improve robustness in the semantic segmentation. Deep neural networks rely on the amount of training data. More diverse data leads to more robustness. This approach has two parts, one is generating more data by changing the texture of the training data (for instance, brightness, contrast, and color) and second is producing data by changing the geometry of the training data (for instance, rotation, horizontal flip, and translation).

Texture Augmentation

1. Brightness: Changing of brightness leads to change in the absolute values of the colors. This changes the tone (lightness and darkness) of the colors. Improves ⇒ Illumination invariance.
2. Contrast: Variance between the darker and brighter area of the images. To change the shadows with respect to original images. Improves ⇒ Illumination invariance.
3. Saturation: Saturation describes the intensity of the color. Less saturation gives more grey images. Hence, it is a variance between colors. Improves ⇒ Illumination invariance.
4. Color-Jitter: To obtain invariance against camera distortion, some random noise is added in each pixel.
5. Salt- Pepper: It saturates black and white pixels with some random probability.

Geometry Augmentation

1. Horizontal flip: It horizontally mirrors the image, which helps to train the network with different orientations. For instance, a car can be parked on any side of the road. So horizontal flip gives diversity with respect to the orientation. Vertical flipping is not used for the training purpose, because it changes the meaning of the image which will not help the neural network to learn anything. For instance, a road can never take the place of the sky in real scenarios [7].
2. Translation: It involves moving the image along both the axes. Translation allows the object to move around the image, which forces the network to search for an object everywhere in the image.
3. Scaling and Cropping: Scaling the image means resizing the image. It can be done in two ways inward and outward, which helps the neural network to learn the images with different resolutions. In the cropping, sample a section from the real image and then resize it to the original image.
4. Aspect Ratio: Changing the scale with respect to any specific dimension, adds invariance, and helps neural network against different aspect ratios.
5. Rotation: It rotates the image by some angle. Here the noticeable thing is that the image size will be preserved after rotation or not, depends on the image size itself. For instance, if the image is rectangular, then it preserves the size if the rotation is done by 180°.

Romera et al. [7] the experiment was conducted on Cityscapes and CamVid dataset.
2.2 Cut and Paste Approach

Dwibedi et al. [3] uses cut and paste approach to generate the synthetic data, for instance, detection. In this method, they cut the object instances and paste them on a random background (Figure 1). At the time of placing those objects on the random background, training detectors find pixel artifacts in the image. When those images are fed to the network, it reduces the detection performance. With the help of the data augmentation and changing the blending parameter settings, they improved the training of the neural network.

Fig. 1: They collected a set of images of the instances and background scenes. Then they extract the object mask and segment the object. They use different blending techniques so that local artifacts can be ignored by the detection model. Then paste the objects on the scenes [3]. Figure taken from [3].

Main steps of this method are:

1. Collecting a diverse and large amount of object instance images.
2. Collecting scene image, which will work as the background images. This approach will work with new scenes also.
3. Predict a foreground mask with the help of Convolutional Network, which separates the foreground pixels and background pixels from the image. Put that foreground mask on the scene images [3].
4. Before paste, they use two methods. First one is Blending, which deals with the problem of boundary artifacts at the time of placing those objects. It helps to smooth out artifacts (Figure 2). Second is Data Augmentation, which gives a diverse viewpoint and scale coverage.

Data Augmentation methods:

1. 2D rotation: Objects rotate by some angle. The angle of rotation is between 30° to -30°.
2. 3D rotation: Data is collected by humans, which fails to detect instances from some viewpoints. With the help of 3D rotation, it is able to synthesize data with diverse viewpoints.
3. Occlusion and Truncation: Partially visible objects occur naturally in the images. To do truncation, some part of the object (usually 0.25) is kept at the boundary of the image. For occlusion, objects are taken with an overlap of IOU 0.75 with each other [3].
4. Disaster objects: In the real world, images are filled with disaster objects (extra objects in the frame). To add disaster objects, they use BIRD dataset.

In this experiment, UV Scenes dataset served as the background images. They used 33 objects instances from the BigBird dataset overlapping with the 11 instances from GMU Kitchen Dataset and the 33 instances from Active Vision Dataset. They train on either their synthetic data or the GMU Dataset and evaluate on the Active Vision Dataset [3].
2.3 Data Generation for Urban Driving Scenes (DGUDS)

Alhaija et al. [1] show that the semantic instance segmentation and object detection models can be learned by the training data, which combines real and synthetic data. They provide real background with the dataset of augmented urban driving scenes, which are captured by 360° (Figure 3). This also helps to preserve realistic lights and reflections on objects. In parallel, it is also able to generate diverse foreground object configuration.

There are two main components of the method. First is detailed high-quality 3D models of car. They obtained 28 high-quality 3D models of a car of 7 different categories from online model repositories. For diversity, car color is chosen randomly. Second is to find the best location. It is very important to find the best location to paste those 3D models of the car, which will help to achieve more realistic images (Figure 3). They explored four different location sampling strategies. First is manual car location annotations. With the help of birds eye view, which they get it by transforming the perspective image, they find and mark the places where the car can be placed. Also, they use a random rotation around the vertical axis of the object. After then to segment the image into road and non-road, they used one algorithm which is proposed by Teichmann et al. [8] and use a random rotation around the vertical axis of the car. The disadvantage of this method is overlapping with real objects because of random rotation. Next step is to estimate the ground plane in the scene will be easy because they know the parameters of the camera and its direction, which will help to overcome the problem of sampling from 6D to 3D. At the end, to deal with the problem of automatic road segmentation, they found manual car location annotation [1].

To create the environment maps, they create KITTI-360 dataset, in that each frame comes with two 180° images taken by two fish-eye cameras on top of recording platform. It gives a full 360°
Omni-directional images. VKITTI dataset is the proxy of KITTI 2015 dataset, used for the comparison with their augmented images. By using the KITTI 2015 dataset, they created KITTI-15 test dataset, which is used to demonstrate the advantage of data augmentation for training robust models. For generalization purpose, they test their model on Cityscapes validation dataset [1].

3 Deep Learning-based Data Augmentation Approaches

In this section, we describe Deep Learning-based Data Augmentation Approaches. In these approaches, they use Generative Adversarial Networks, which helps to generate new data with the same statistics as the training set. Also, they use reinforcement learning to find the best augmentation technique.

3.1 Synthetic Data Augmentation using GAN (SDAG)

This method [4] is used for liver lesion classification. In the medical domain, they have to cope with a limited amount of data, but for the better classification, large datasets are needed. In this method, they use classical data augmentation approach to generate more data. Those data serves as training data to the Generative adversarial networks, which creates more diverse data for synthetic data augmentation. Main two parts of the approach are: 1) Classical Data Augmentation 2) GAN Networks for Lesion Synthesis.

Classical Data Augmentation Method

In general, on the grayscale images, mostly applicable augmentation tricks are the translation, rotation, scaling, flipping, and shearing. Here they have to preserve the liver lesion characteristics they didn’t apply to shear. They kept the ROI in the center of the lesion. Each lesion ROI is first rotated Nrot times, and each rotated ROI is flipped Nflip times, then translated Ntrans time, and at the end, ROI is scaled Nscale times[4]. So the total number of augmentations are $N = N_{rot}(1 + N_{flip} + N_{trans} + N_{scale})$.

Fig. 4: Lesion ROI examples of Cysts (top row), Metastases (middle row) and Hemangiomas (bottom row). Left side shows Real lesions and Right side shows Synthetic lesions [4]. Figure taken from [4].

GAN Networks for Lesion Synthesis

There are three different class lesion: Cysts, Metastases, and Hemangiomas. They use the Deep Convolutional GAN for synthesizing labeled lesions as per class (Figure 4). They use the Deep Convolutional GAN Architecture in which there are two Deep CNNs. A generator will try to generate images which looks like real images and discriminator tries to distinguish between real and
synthesized images [4]. They used a liver lesion classification CNN of the following architecture: three pairs of convolutional layers where each convolutional layer is followed by a max-pooling layer, and two dense fully-connected layers ending with a soft-max layer to determine the network predictions into the three lesion classes. [4].

### 3.2 Smart Augmentation

Lemley et al. [6] describes the method to generate augmented data during training process by creating a network (Figure 6). The network learns to combine two or more images of the same class and feed those images to a target network in a way that reduces that networks loss (Figure 5).

![Fig. 5: Image on the left is a learned combination of the two images on the right as produced by network A [6]. Figure taken from [6].](image1)

There are two networks. Network A is used for the data generation. By combination of two or more images Network A generates more samples or more training data. Network B is used for the specific target task. For instance classification or regression. With the help of this approach network B will get enough training data for the target task. Main constrain on network A is that input and output should be of same shape and type [6]. They parameterized it by the inclusion of $\alpha$ and $\beta$ as loss function $f(L_A, L_B; \alpha, \beta)$. The results show that these can impact final accuracy [6].

![Fig. 6: The image on the left is a learned combination of the two images on the right as produced by network A [6]. Figure taken from [6].](image2)

Network A can be implemented as a single network or as multiple networks (Figure 6). The benefit of the multiple networks is that it can learn the class-specific augmentation [6]. The experiment was conducted on five different datasets: Highly constrained faces dataset, Augmented Highly constrained faces dataset, FERET, ADIENCE, MIT places dataset.
3.3 Auto Augmentation

Cubuk et al. [2], with the help of reinforcement learning, the network automatically searches for improved data augmentation policies.

![Fig. 7: Main policy is consists of 5 sub policy. Each sub-policy consists of 2 operations, each operation is associated with two numerical values: the probability of calling the operation, and the magnitude of the operation. There is a probability of calling an operation, so the operation may not be applied in that mini-batch. However, if applied, it is applied with a fixed magnitude [2]. Figure taken from [2].](image)

In the implementation of [2], they design a search space, which consists of one main policy. That main policy further consists of other sub policies. Each sub policy can do two image operations, in which each operation has their two parameters values. One is the probability of that operation and another one is the magnitude. In this paper [2], there are 16 operations: ShearX/Y, TranslateX/Y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout, Sample Pairing. From (Figure 7), we can understand that every image transforms differently for different batches. Another main component of this approach is the search algorithm. The search algorithm is a combination of two components one is a controller, Which is a recurrent neural network. It decides how good a policy is, by giving reward signals. To decide that, first they train child model with sub policies. Then one of those policies is selected randomly to augment the image and reward is a given to that sub policy if it has better validation accuracy [2]. Second component is Proximal Policy Optimization algorithm. They use this algorithm at the time of training [2]. The experiment was conducted on CIFAR10, Reduced CIFAR-10, CIFAR100, SVHN, Reduced SVHN, and ImageNet dataset.

4 Conclusion and Results

In this paper, we analyzed different data augmentation methods. We start with an introduction about the background and the concept of data augmentation. Then, a brief overview of those approaches. The effect of data augmentation methods has been made in Table 1. The results are not directly comparable, since all approaches were used on different datasets. All the approaches have different evaluation metric. So we show the improvement in that particular evaluation metric. Hence, we can conclude that with the help of data augmentation we can increase the accuracy of the network.
Table 1: Comparison of all approaches with their benchmark (IOU: Intersection over Union and mAP: mean Average Precision)

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Dataset</th>
<th>Evaluation Metric</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture[7]</td>
<td>Cityscapes</td>
<td>IOU</td>
<td>0.3</td>
</tr>
<tr>
<td>Geometry[7]</td>
<td>CamVid</td>
<td>IOU</td>
<td>17.5</td>
</tr>
<tr>
<td>Texture and Geometry[7]</td>
<td>Cityscapes</td>
<td>IOU</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>CamVid</td>
<td>IOU</td>
<td>10.7</td>
</tr>
<tr>
<td></td>
<td>Cityscapes</td>
<td>IOU</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>CamVid</td>
<td>IOU</td>
<td>29.6</td>
</tr>
<tr>
<td>'Cut and 'Paste'(Bending) [3]</td>
<td>GMU Kitchen dataset</td>
<td>mAP</td>
<td>7.8</td>
</tr>
<tr>
<td>'Cut and 'Paste'(Data Augmentation) [3]</td>
<td>GMU Kitchen dataset</td>
<td>mAP</td>
<td>6.5</td>
</tr>
<tr>
<td>DGUDS(KITTI15)[1]</td>
<td>VKITTI</td>
<td>mAP</td>
<td>-7.3</td>
</tr>
<tr>
<td>DGUDS(KITTI15)[1]</td>
<td>Augmented</td>
<td>mAP</td>
<td>1.7</td>
</tr>
<tr>
<td>DGUDS(KITTI15)[1]</td>
<td>VKITTI+Real</td>
<td>mAP</td>
<td>5.1</td>
</tr>
<tr>
<td>DGUDS(KITTI15)[1]</td>
<td>VKITTI+Augmented</td>
<td>mAP</td>
<td>5.7</td>
</tr>
<tr>
<td>DGUDS(Cityscapes)[1]</td>
<td>VKITTI</td>
<td>mAP</td>
<td>6.8</td>
</tr>
<tr>
<td>DGUDS(Cityscapes)[1]</td>
<td>Augmented</td>
<td>mAP</td>
<td>3.0</td>
</tr>
<tr>
<td>DGUDS(Cityscapes)[1]</td>
<td>VKITTI+Real</td>
<td>mAP</td>
<td>4.9</td>
</tr>
<tr>
<td>DGUDS(Cityscapes)[1]</td>
<td>VKITTI+Augmented</td>
<td>mAP</td>
<td>7.1</td>
</tr>
<tr>
<td>SDAG(Cyst) [4]</td>
<td>Tomography(CT) images of 182 liver lesions</td>
<td>Accuracy</td>
<td>-0.7</td>
</tr>
<tr>
<td>SDAG(Met) [4]</td>
<td>Tomography(CT) images of 182 liver lesions</td>
<td>Accuracy</td>
<td>5.1</td>
</tr>
<tr>
<td>SDAG(Hem) [4]</td>
<td>Tomography(CT) images of 182 liver lesions</td>
<td>Accuracy</td>
<td>6.8</td>
</tr>
<tr>
<td>Smart Augmentation[7]</td>
<td>AR Faces</td>
<td>Accuracy</td>
<td>6.85</td>
</tr>
<tr>
<td>Smart Augmentation[7]</td>
<td>Adience</td>
<td>Accuracy</td>
<td>6.0</td>
</tr>
<tr>
<td>Smart Augmentation[7]</td>
<td>FERET</td>
<td>Accuracy</td>
<td>4.9</td>
</tr>
<tr>
<td>Auto Augmentation[2]</td>
<td>CIFAR-10</td>
<td>Accuracy</td>
<td>1.2</td>
</tr>
<tr>
<td>Auto Augmentation[2]</td>
<td>Reduced CIFAR-10</td>
<td>Accuracy</td>
<td>7.1</td>
</tr>
<tr>
<td>Auto Augmentation[2]</td>
<td>CIFAR-100</td>
<td>Accuracy</td>
<td>3.3</td>
</tr>
<tr>
<td>Auto Augmentation[2]</td>
<td>SVHN</td>
<td>Accuracy</td>
<td>0.4</td>
</tr>
<tr>
<td>Auto Augmentation[2]</td>
<td>Reduced SVHN</td>
<td>Accuracy</td>
<td>6.4</td>
</tr>
</tbody>
</table>

References

A Robust and Accurate In-Air Signature Acquisition Method based on 3D hand Pose Estimation

Resha Thacker¹ and Jameel Malik²

¹ thacker@rhrk.uni-kl.de
² Jameel.Malik@dfki.de

Abstract. Recent advancements in the field of signature-based user authentication lead towards the In-Air signatures, which is important for non-contact and distant modes. Existing methods for in-air signature acquisition are based on heuristic approaches for fingertip tracking or hand pose estimation method based on single 2D depth frame. In this project, we propose a new in-air signature acquisition method which is based on Hand PointNet method for hand pose estimation. To this end, a single depth frame is first converted into point cloud representation and preprocessed for standardization. Afterwards the normalized point cloud is given to a CNN-based regression network which estimates accurate 3D hand joint locations. The 3D signature is recorded by tracking the position of the index fingertip. One forward pass through the regression network takes 0.6s. In order to make our method robust to viewpoint variations, we use 3D data augmentation in the training. The proposed approach works in real time and it is not restricted to any specific hand pose for capturing in-air signatures. The experiments are performed on MSRA-2015 hand pose dataset which contains several hand poses including small variations in viewpoints.

Keywords: Hand pose estimation, In-Air Signature, Data Augmentation, Regression Network

1 Introduction

Traditionally people have been using pen and paper for the signature. Advancement in the technology gives birth to digital signature. Digital signatures are based on public key cryptography, which is based on some algorithms such as RSA (Rivest-Shamir-Adleman). Biometric-based authentication is based on fingerprint, voice, eye and many other characteristics.

Fig. 1: (a) shows the 3D In-Air signature trajectory, (b) shows the 2D spatial view, and (c) shows the depth pattern [2].
New bio-metric signature acquisition technique ‘In-Air signature’ is a non-contact mode of signature which gives more freedom for the different kind of hand gestures and movements. To capture the in-air signature, the complete 3D hand pose is estimated. 3D hand pose gives the information about the depth of the hand which is an important feature. Figure 3 shows the 3D In-Air-signature trajectory, 2D spatial and depth pattern of that signature respectively. Depth pattern of each signature is unique and hence is considered to be of great significance. However, occlusion, self-similarity of fingers and large variation in the movement of hand are still one of the major challenges and could be addressed to increase the accuracy and robustness. Many methods are available based on 2D CNN, which do not use 3D spatial information of the hand. Some methods are based on 3D CNN, which takes 3D volumes as an input but because of the limited resolution, loose useful details. To tackle all these problems Ge et al. [1] proposes a point cloud based hand joints regression method for the 3D hand pose estimation in single depth image. Depth image is first converted into point cloud representation and preprocessed for standardization. The normalized point cloud is given to a CNN-based regression network which estimates accurate 3D hand joint locations. In this project, we take the HandPointNet method of Ge et al. [1] as baseline approach. The HandPointNet method provides more efficient and effective results because of the 3D representation of the input depth image. In order to make the method robust to viewpoint variations, 3D data augmentation in the training.

The pipeline of the main approach is described in detail in section 2. Section 3 describes the experiment in which first claim of the main approach is verified by reproducing it on MSRA dataset and then the existing approach with 3D data augmentation is applied in real time on unseen images. Section 4 shows the results of the experiments and conclusion is described in section 5.

2 Methodology

Methodology is divided in three sections. First section describes the whole pipeline of the approach. Second section provides detailed description of ‘Hand segmentation’ step. Section 3 describes the ‘Pose CNN’ step. Pose CNN is the main baseline approach [1].

2.1 Pipeline Overview

The main pipeline of our method is shown in Figure 2. Depth images are taken from the depth camera, which is shown in Figure 2 as \(D_i\). We apply depth value based thresholding and separate out the segmented hand, which is shown in Figure 2 as \(D_s\). The segmented hand image is given to the Pose CNN. Accuracy and robustness can be improved in this part. Hand segmentation is described in section 2.2 and Pose CNN is described in section 2.3.

![Fig. 2: Shows the whole pipeline of the approach. Depth images are captured from the depth camera and by applying depth value based thresholding, hand segmentation was done. Pose CNN takes the segmented hand as an input and gives prediction of hand joints as output [2].](image-url)
2.2 Hand Segmentation

Input to the baseline approach is the depth image of hand. When a depth image is captured from the camera, background of the image is also captured. To avoid the background, hand regions is segmented by applying some threshold on depth value of the image.

![Hand Segmentation Image](image)

Fig. 3: Shows the segmented hand from the whole depth image based on hand thresholding value.

2.3 Baseline Approach (Pose CNN)

The segmented hand is given to the Pose CNN [1] as depth image, which is shown in Figure 4. The next step is preprocessing. In this step, the hand depth image is converted to a set of 3D points. To deal with the large variation in global orientation of the hand, the authors [1] proposed OBB-based Point Cloud Normalization (Figure 4). OBB is a tightly fitting bounding box of the input point cloud in which hand point cloud is transformed into a canonical coordinate system in which the global orientations of the transformed hand point clouds are as consistent as possible [1].

![Pose CNN Architecture](image)

Fig. 4: Shows the architecture of the HandPointNet approach. Depth image is first converted to a set of 3D points. In the next step OBB-based Point Cloud Normalization is done. OBB based normalized hand point cloud is given to the Hand Pose regression network. Hand Pose regression network estimates the 3D hand joint locations [1].

After the preprocessing step, OBB based normalized hand point cloud is given to the Hierarchical PointNet Network for the 3D Hand pose regression, which has three abstraction level. Figure 5 shows the two abstraction level, each abstraction level consist of the 3 layers, sampling layer, grouping layer and pointnet respectively. Sampling Layer takes the subset by FPS (Farthest Point Sampling) and centroid is calculated using kNN (K nearest neighbour) algorithm. Grouping Layer gives K number of points in the neighbourhood of centroid points. At the end, in the PointNet
each local region in the output is abstracted by its centroid and local feature, that encodes the centroid’s neighbourhood. Fully connected layers gives the prediction of the 3D hand joint locations. [3]. Figure 6 shows the detailed architecture with convolution layers.

Fig. 5: Shows the detailed abstraction levels with sampling, grouping and pointnet layer. Fully connected layers gives the prediction of the 3d hand joint locations. [3].

Fig. 6: Detailed architecture of the Hierarchical Pointnet with convolution layers. Each layer shows the detailed description of 2D convolutions and max pooling. Each 3-layer 2D convolution is an abstraction level, followed by max pooling.

3 Experiments

The hand pose estimation with hand pointnet is first reproduced on MSRA dataset and then we applied 3D data augmentation on the MSRA dataset. To test the robustness of the approach in real time, an experiment is conducted using SR300 camera.

3.1 MSRA dataset

MSRA hand dataset consists of 9 subjects (P0 to P8) who are all right handed. Each subject has 17 gestures. Each gesture has 500 frames in terms of bin file. Each bin file starts with 6 unsigned int: img_width img_height left top right bottom. [left, right) and [top, bottom) is the bounding box coordinate in this depth image. The bin file then stores all the depth pixel values in the bounding box in row scanning order, which are (right - left) * (bottom - top) floats. The unit is millimeters. Ground truth joint.txt file stores 500 frames x 21 hand joints per frame. Each line has 3 * 21 = 63 floats for 21 3D points in (x, y, z) coordinates. The 21 hand joints are: wrist, index_mcp, index_pip, index_ip, index_tip, middle_mcp, middle_pip, middle_dip, middle_tip, ring_mcp, ring_pip, ring_dip, ring_tip,
little_mcp, little_pip, little_dip, little_tip, thumb_mcp, thumb_pip, thumb_dip, thumb_tip. The images are captured using Intel’s Creative Interactive Gesture Camera. The camera intrinsic parameters are principle point (image center), at 160, 120 and the focal length which is 241.42.

### 3.2 3D Data Augmentation

We applied rotation on MSRA dataset around XYZ. First we applied rotation at 45° around XYZ. After then we applied rotation in the range of -40° to 40°, -50° to 50° and -90° to 90° respectively. We applied rotation matrix as is described in the Figure 7.

\[
R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

Fig. 7: Rotation matrix consisting of rotations around X, Y and Z axes respectively. \(R_x(\theta)\) shows the rotation around X axis, \(R_y(\theta)\) shows the rotation around Y axis and \(R_z(\theta)\) shows the rotation around Z axis. These are the standard forms defined for rotation in the euler angle representation.

### 3.3 Experiment with SR300 camera

Creative SR300 camera is used as an interface, which has the focal length of 475.06. The depth image is captured from the depth camera and mass center is computed based on the value of the mass of the depth image. Bounding box of the depth image is computed from the mass center to set up the depth image according to the format of MSRA bin file. The bounding box gives the 6 unsigned int: img_width, img_height, left, top, right and bottom as per the MSRA bin file. Preprocessing is done on the cropped image (after applying bounding box on the original depth image) and is evaluated on pretrained model.

### 4 Results

This section is also divided into three parts as per the experiments. It also consists the graph of the comparison which shows the augmentation results.

#### 4.1 MSRA Dataset without data augmentation

Figure 8 shows the results of the whole pipeline on different gestures. In the preprocessing step, depth image is converted into 3D point cloud and then normalization is performed on the 3D point cloud. Normalized point cloud is fed to the regression network and 3D hand joints are estimated as the output. To plot the 3D hand joints on to the original image de-normalization (inverse of the preprocessing step) is performed and then converted into to the pixel coordinates.
Fig. 8: Each row shows the different gestures and corresponding results. Depth image is first converted into 3D point cloud which is normalized in the preprocessing step. Normalized point clouds are fed to the regression network which predicts the normalized joint locations. These normalized joint locations are plotted on the normalized point cloud. Following this, normalized joint locations are denormalized into pixel coordinates and plotted on the original image.

4.2 MSRA dataset with 3D Augmentation

Figure 9 shows the results of the 3D augmentation. We applied rotation around XYZ by some random degree. It shows the rotation in X, Y, Z and XYZ axes respectively.

Fig. 9: Shows the augmented images of the MSRA dataset. Rotation was applied on X, Y, Z and XYZ axes respectively.

4.3 Results from the unseen images

Figure 10 shows the results of the depth images which are taken from the SR300 camera. It shows the different poses with different viewpoint variation and the prediction of the 3D hand joint locations. Data augmentation helps in successfully detecting different viewpoint variations. We
have shown the results of 2 gestures, however, we verified that it works on many other gestures also.

![Diagram showing the results from the SR300 camera with preprocessing and prediction of 3D hand joint locations.](image)

**Fig. 10**: Show the results from the SR300 camera with preprocessing and prediction of 3D hand joint locations. Second row shows the results with view point variation for another gesture. First and third row are different gestures, and second and last rows are the viewpoint variations of the respective first and third rows.

### 4.4 Comparison

Figure 11 and Figure 12 shows the result of HandPointnet with fingertip refinement network, HanPointNet_project_new without fingertip refinement network. The figure 11 shows the mean error with respect to 21 joints of the hand. HandPointNet shows the mean error of the approach with Fingertip Refinement Network, which works on straight fingers with the help of the inverse kinematics. We didn’t include this as our baseline approach. HandPointNet_project_new shows the mean error without Fingertip Refinement Network. HandPointNet_project_45_new shows the mean error after applying rotation at 45°. HandPointNet_project_range shows the mean error after applying rotation at random degree.
5 Conclusion

In this project we proposed a new in-air signature acquisition method, based on Hand PointNet method. Our approach gives more efficient and effective results because of the 3D representation of the input depth image. One forward pass through the regression takes 0.6s. Our approach is more robust to viewpoint variations by using 3D data augmentation. The approach works well in real time and is not restricted to any specific hand pose to capture in-air signatures. This approach can be used for the in-air signature trajectory because of its real time performance.

References

Seminar Survey: Temporal alignment in distributed systems at receiver end

Janik Westers\textsuperscript{1} and Maria Alejandra Sanchez\textsuperscript{2}

\textsuperscript{1} j.westers13@cs.uni-kl.de
\textsuperscript{2} maria.sanchez@dfki.de

Abstract. In Body Sensor Networks (BSN) drift caused by inaccurate clocks in Inertial Measurement Units (IMU) and data loss are common problems. Therefore, this survey paper presents three approaches to handle time inaccuracies at receivers end of a BSN. Additionally, three approaches are presented and evaluated to reconstruct lost data. All presented approaches are evaluated regarding their usability in a wireless BSN.

Keywords: temporal alignment, data reconstruction, body sensor network

1 Introduction

This paper provides an overview of methods for temporal data alignment and data reconstruction in Body Sensor Networks (BSN). A BSN is a network consisting of multiple sensors or wearable computing devices connected with a central computation unit. Possible application areas for BSNs are patient recovery monitoring or human motion detection, tracking and evaluation. In the scope of this survey paper, a BSN consisting of multiple Inertial Measurement Units (IMUs) is taken as reference. The IMUs are placed at different parts of the body and used to record the movements of a participant. Currently, the sensors are connected through wires and it is planned to use a wireless network in the future.

Very accurate clocks are expensive and therefore most IMUs have clock drift. This is an increasing desynchronization of the clock in comparison with an (ideal) reference clock. A first step to reduce drift is to use clock synchronization. But even after this procedure, a fixed or slowly increasing influence may remain. Therefore, section 2 presents and reviews solutions for the remaining inaccuracy. In section 3 approaches are presented to retrieve lost data in BSNs.

2 Methods for temporal alignment

In this section three approaches are presented to temporal align data and retrieve accurate time stamps. The first approach uses FIFOs to reconstruct accurate time stamps. Then an approach using auto- and cross-correlation is presented. The last approach aligns data streams based on temporal dependencies.

2.1 Accurate Sample Time Reconstruction of Inertial FIFO Data

In this paper, the authors propose a data alignment approach based on a First In First Out queue (FIFO) interface [1]. A FIFO is a ring buffer data structure that allows stored data to be read in a chronological order. To apply this approach the sensor should have a free-running sensor time value and the timestamp of the sensor must be synchronized with its sampling times.
Concept: The algorithm calculates the reconstructed time stamp for each sensor individually. The FIFO has an amount \( n \in \mathbb{N}, 0 \leq n \leq \text{#samples} \) of data samples in its queue. To reconstruct the timestamps it is needed to know the timestamp of the first sample \( (t_0) \) and the sampling period of the sensor \( (\text{period}_{\text{sensor}}) \). Then the accurate timestamp of the last sample in the queue is

\[ t_n = t_{n-1} + n \times \text{period}_{\text{sensor}}. \]

The FIFO can be configured to send an interrupt message if the queue exceeds a certain number of samples and the host starts reading the FIFO data. Then the interrupt message’s time \( t_{\mathrm{int}} \) is a precise timestamp of the last data sample in the queue. This is used to calculate \( t_0 \).

\[ t_0 = t_{\mathrm{int-lvl}} + \text{period}_{\text{sensor}} \]

where \( t_{\mathrm{int-lvl}} \) is the sampling time of the last sample in the previous queue. Both equations above result in the following equation:

\[ t_n = t_{n-1} + D \times \text{period}_{\text{sensor}}, \]

where \( D \) is the relative drift.

When the host reads the sensor timing information it stores the time stamps as \( ST_a \) and the local time stamps as \( HT_a \). Then the drift \( D \) can be calculated as in algorithm 1b, where it is assumed for the sensor to have a 24-bit wide timer with a granularity of 39.0625\( \mu \)s.

It remains to do the time offset correction. Therefore the age \( t_{\mathrm{age}} \) of the last generated sensor sample is calculated by

\[ t_{\mathrm{age}} = (ST_a \text{mod} 2^n) \times D \times 39.0625 \mu s. \]

The age describes the time difference between the sampling time of the last sample in the FIFO queue and the read access of the host system. The remaining delay is determined with communication time \( t_{\mathrm{OR}} \) of the number of over-read bytes \#\text{\textoverread}. With the knowledge of the data rate of the communication medium \( (t_{\text{comm/byte}}) \) it can be calculated by

\[ t_{\mathrm{OR}} = \#\text{\textoverread} \times t_{\text{comm/byte}}. \]

Finally, the timing corrections can be put together to calculate the accurate time stamps by

\[ t_n = (HT_a - t_{\mathrm{age}} - t_{\mathrm{OR}}) + (n + 1) \times D \times \text{period}_{\text{sensor}}. \]

The time and space complexity of the whole algorithm is in \( O(n) \).

Experiments: To test the system two inertial sensor devices were connected to a raspberry pi 2B with a python implementation of the approach. A sampling rate of 200Hz was used and the FIFO stream was fetched every 25 to 250 ms. Independent of the FIFO fill level the approach was able to compensate the relative drift. Also, with a sample time of 39\( \mu \)s the standard deviation of the reconstructed sampling rate was bound below 40\( \mu \)s.

Conclusion: The approach is overall promising for the application in body sensor networks. However, the communication medium assumed in the paper is a bus, not a wireless network. Realtime processing without delays caused by FIFO queues could be archived by modifying the approach. This could be done by implementing the FIFO at the host system and not at the sensor. Then the drift can be calculated by storing past data samples in the host’s FIFO queue and run the calculation on them.

2.2 A method for sub-sample computation of time displacements between discrete signals based only on discrete correlation sequences

The approach uses auto- and cross-correlation sequences to calculate the time displacements between two sampled signals [2]. Figure 1a presents a graphical overview over the proposed approach.

Concept: The method assumes as input two morphological similar discrete time signals \( s_1[n] \) and \( s_2[n] \) of length \( N \). The sampling frequency is \( f_s - 1/T_s \) and the delay between the signals is \( \delta \). The
first step is to calculate the normalized cross-correlation sequence $\hat{R}_{s_1,s_2}[m]$ as in equation 1 and the integer delay estimate $ID = \arg\max_m \hat{R}_{s_1,s_2}[m]$. This step is shown on the left side of Figure 1a.

\[
\hat{R}_{s_1,s_2}[m] = \begin{cases} 
\frac{1}{N} \sum_{k=0}^{N-m-1} s_1[k+m]s_2[k], & \text{if } m \geq 0 \\
R_{s_2,s_1}^*[m], & \text{if } m < 0 
\end{cases}
\]

With the ID the two signals could be synchronized. However, it remains a sub-sample displacement. To calculate this displacement one signal has to be taken as reference signal (e.g. $s_1[k]$) and its normalized auto-correlation sequence $\hat{R}_{s_1,s_1}[m]$ must be calculated similar to equation 1. Furthermore the normalized cross-correlation sequence $\hat{R}_{s_1,s_2}[m]$ between two synchronized signals is calculated. It is again similar to equation 1 but using the signal $s_2$ shifted by ID samples written as $s_2'$. The difference between those two correlation is $\hat{R}_d[m]$. The delay between the two discrete signals can then be calculated by multiplying ID with the sampling period and add it to the maximum $\hat{R}_d[m]$ divided with a scaling factor: $\delta = ID \ast Ts + \max(\hat{R}_d[m]) \ast factor$. The scaling factor is calculated beforehand by artificially delaying a reference signal and computing its auto- and cross-correlation. Then the sampling period is divided by the maximum difference between the auto- and cross-correlation: $factor = \frac{T_s}{\max(R_{s_1,s_1}[m] - R_{s_1,s_1}')}$. All this together results in the process as shown in Figure 1a.

Experiments: The algorithm was implemented in Matlab on a standard computer and compared with a cosine method, a method using parabola fitting and a phase modulation method. As test data artificially crated ultra-sonic echoes and real-world ultrasound signals were used. For a pair of signals, one was delayed by a fraction of the period. Furthermore, the signals were disturbed by Gaussian white noise. Under moderate and noisy conditions, the proposed approach managed to outperform the other methods except the phase modulation method. However, on real world data, the proposed algorithm outperformed all other algorithms.

Conclusion: The approach presented above can be used to synchronize the signals of IMUs, which are attached on the same body parts. Also, it can be used during the calibration process to calculate the delays and drifts. Therefore, a known movement is performed with several IMUs or a magnetic field is induced.

2.3 Temporal alignment model for data streams in wireless sensor networks based on causal dependencies

The authors propose an approach to synchronize data-streams in Wireless Sensor Networks (WSN) by identifying temporal relationships [3]. No clock synchronization or global reference is needed, since the approach purely relies on ordering the data streams sent by the wirelessly connected sensors.

Concept: Each sensor or sink is a process and each process can only do a single execution at a time, called an event. It can be either sending or receiving a message. Events can be ordered with the “happened before relation”. This temporal relationship expresses that two events of a single process have an order ($a \rightarrow b$) and a sending of a message always occurs before its receiving. Furthermore, it is transitive (if $a \rightarrow b$ and if $b \rightarrow c$ then $a \rightarrow c$). Events of different sensors may be concurrent. A data-streams of a single sensor is called local-stream and consist of events executed by this process. An event-stream is a collection of subsets of different local-streams ordered by their dependencies. Such a subset is identified by $Q^R_q$, where $R_q$ is the set processes that generated events in the subset. Initially two local-streams $X_c$ and $Y_d$ are aligned to get an event stream $ES_\Theta$. Therefore, three subsets are created. The first subset contains all processes of one local-stream that precede the other local-stream as shown in Figure 2a. The second subset contains all simultaneous occurring
(a) Proposed method. (A) and (B) original signals, (C) cross-correlation function used to determine the discrete time-delay, (D) and (E) synchronized original signals, (F) auto-correlation of the synchronized signal, (G) cross-correlation between the synchronized signals, and (H) difference between the auto-correlation of $s_1$ and cross-correlation between $s_1$ and $s_2$.

\[
\text{if } ST_a \geq ST_{a-b} \text{ then } \\
\quad ST_{\text{delta}} = ST_a - ST_{a-b} \\
\text{else} \\
\quad ST_{\text{delta}} = ST_a + 2^{24} - ST_{a-b} \\
\text{end}
\]

\[
HT_{\text{delta}} = HT_a - HT_{a-b} \\
D = HT_{\text{delta}} / (ST_{\text{delta}} \cdot 39.0625 \mu s)
\]

(b) Drift calculation. The influence of communication jitter is reduced by increasing the time window, using $b \in \mathbb{N}, b > 1$.

Fig. 1: 1a is taken from [2] and 1b is taken from [1].

processes as in Figure 2b. The last subset contains all processes that occur after one of the streams have finished.

Next the event-stream is aligned with another local-stream. Therefore, the created event-stream $X_{\beta}$ is aligned with another local-stream $Y_k$ to form a new event-stream. Subsets of the event-stream preceding the local-stream are the first subsets of the new event-stream. This is shown in Figure 2c. The first concurrent subset with the local-stream may be only partly concurrent with the local-stream and must then be divided into two new subsets. The first containing the events not concurrent with the local-stream. The second subset containing the concurrent events of the remaining part of the subset and the local-stream. This and the following part are shown in Figure 2d. Next, the subsets of the event-stream and the concurrent local-stream are aligned. Therefore, the subsets of the new event-stream are the subsets of the old event-stream additionally containing the concurrent local-stream events. If the local-stream ends before the event-stream, the last concurrent subset may again only be partially concurrent with the local-stream. Then again two new subsets are created one containing the concurrent events and one containing the reaming events of the old subset. All other succeeding subsets of the event stream remain unchanged. This case is shown in Figure 2e. If the event-stream finishes first, the remaining events of the local-stream are grouped to a new single subset. The complete procedure is repeated until all concurrent local-streams are merged.

**Experiments:** The authors proved mathematically that their approach creates a virtual timeline in which each subset represents a timeslot. Furthermore, they used a simulator to test their approach in a simulated wireless sensor network. Therefore, in a 200 × 200 meter field 50 nodes were arranged in a multi-hop network and sampling rates between 25 and 1000 milliseconds were used. In this
(a) Aligning the first subset of the first event-streaming.
(b) Aligning the second subset of the first event-streaming.
(c) Aligning the first subsets of events of an event-streaming.
(d) Aligning the subsets of events with concurrences.
(e) Aligning the last subsets of events without concurrences when \( Y_k \) finishes first.

Fig. 2: Figures taken from [3].

Simulation, the synchronization error was bounded below the transmission delay for sampling rates between 75 to 830 ms.

**Conclusion:** The approach was created for wireless multi-hop sensor networks in a wide physical area. Therefore, a BSN with higher sampling rates is expected to have a lower transmission delay boundary. But this approach was never implemented in an actual physical sensor network and its computational complexity is unknown. Therefore, other approaches may be preferable.

### 3 Data prediction

Even after applying the above presented methods for synchronizing packages, data loss may occur. Reasons may be transmission problems, inference or too big delays. This data loss can be a single sample from one sensor up to multiple samples from multiple sensors. Therefore, three methods are presented to reconstruct lost data. The approach in section 3.1 reconstructs lost data with the knowledge of previous data of the sensor. In section 3.2 an approach is presented, which uses periodicity in movements. Finally, the approach in section 3.3 models dependencies between different sensors as Bayesian network and thereby reconstructs the most likely sensor value.

#### 3.1 Deep learning-based real-time query processing for wireless sensor network

The authors present in their paper a deep learning approach for query processing based on data from wireless sensor networks [4]. Since the neural network always needs input data, they proposed a method to predict data, if it is too much delayed or lost. Therefore, they introduced an exponential weighted moving average. The algorithm computes for the past samples an average which gives a weight to each sample. It is calculated with: 
\[
s_t = \alpha \times x_t + (1 - \alpha) \times s_{t-1}
\]
where \( \alpha (0 < \alpha < 1) \) is the smoothing factor and \( t \) represents the time. \( s_t \) represents the new predicted value of \( x \). For a larger value of \( \alpha \) more weight is given to more recent data of \( x \).
**Conclusion:** The approach is easy to apply, computationally inexpensive and can easily be applied for real time data processing. However, the approach performs best on linear data and reducing its accuracy on nonlinear data. Also, it is not suitable for large amounts of missing data.

### 3.2 Missing Sample Recovery for Wireless Inertial Sensor-Based Human Movement Acquisition

The authors propose an approach for data reconstruction of wirelessly sent IMU data during high mobility tasks [5]. The idea is to make use of periodic movements as the limb movement during sidesteps.

**Concept:** The Approach consists of two parts: 1) decomposition of the corrupted signal and 2) reconstruction of the missing samples. The first step divides the signal into several subcomponents, more precisely into sets of intrinsic mode functions (IMFs). This IMFs are later used to describe periodicity. Theoretically, the received IMFs can be interpolated and summed up to reconstruct the original signal. However, often the samples on the border of the missing data appear as extrema and lead to errors. Therefore, the local extrema near those boundaries are used for further processing. The goal of the second step is to fill the gaps in the IMFs obtained from the decomposition step. Figure 3a shows how such data with gaps can look like.

The approach makes use of two correlations. First the short-term correlation of a data sample with its close samples. Secondly a long-term correlation of a data sample with its corresponding sample a period T away. This long-term correlation uses the periodicity of movements. The combination of those two correlations results in the following predictor:

\[
x_0[n] = \sum_{k=1}^{p} a_k x_c[n-k] + \sum_{k=-q}^{q} b_k x_c[n-T-k] + e[n]
\]

where \(p\) and \(2q+1\) are the model orders. The vector \(c = [a_1, \ldots, a_p, b_{-q}, \ldots, b_p]\) and the step period \(T\) specify the model. \(T\) is the time difference between the peaks of the auto-correlation functions.

For a gap with \(G\) missing samples between two short term segments of length \(N_1\) and two long term segments of length \(N_2\) the following steps are done.

1) Estimation of the vector \(c\) with the Burg’s method for the preceding \(N_1\) sample and the one period away \(N_2\) sample.
2) With inverse filtering calculate the excitation \(e[n]\).
3) Extension of the excitation. This is done by combining the Excitation and the first \(G\) samples of its flipped (reversed in time) version.
4) Filter the result of the previous step by the above presented model. The result is then the reconstructed signal in forward direction.
5) Repeat steps 1 to 4 for the segment, which reconstructs the period in the reversed (backward) direction.
6) Combining the reconstructed forward and backward directed signals. This is done with a cross-fading window.
7) Repeat this for all gaped IMFs. The reconstructed signal is obtained by summing up all reconstructed IMFs.

**Experiments:** To test their approach the authors recorded participants doing the modified Edgren sidestep test. Therefore, the participant is doing three times sidesteps up to 4 meters to the left and right. The data was recorded by custom-designed IMUs with a Bluetooth transceiver, which were placed on the thighs and shanks. The calculation was done in Matlab on the recorded data. For testing gaps with up to 25 samples were artificially created at critical events as heel strike or change of direction. The proposed method was compared with spline interpolation, auto-regression model-based interpolation, expanded auto-regression model-based interpolation, and EMD (Empirical mode decomposition)-based interpolation. It was able to outperform the other approaches, mainly because it is the only approach using the long-term correlation of periodic movements.
Conclusion: The approach is useful for data reconstruction in periodic movement data. It cannot be applied in real-time, as it is more computationally expensive and needs the samples of data a period of the movement away. For non-periodic movement, a reduced approach could be applied by only using the short-term correlation of missing data with its close past samples. Only using the short-term correlation, the approach may also be applicable in real-time with only a short delay.

3.3 A Data Reconstruction Model Addressing Loss and Faults in Medical Body Sensor Networks

The proposed method uses a Bayesian network to reconstruct lost data [6]. Therefore, conditional probabilities are used to recover data by dependencies modeled in the network.

Concept: For n sensors exist the ground truths $X_1, X_2, ..., X_n$ for the sensor readings $Y_1, Y_2, ..., Y_n$. Furthermore $X_{n+1}, X_{n+2}, Y_{n+1}, Y_{n+2}$ describe the previous and next sensor reading of the analysed sensor. Then figure 3b shows how the sensors are related for value $X_1$ in the Bayesian network.

In the following $b^k_{x_ky_k}$ is the probability to observe the kth sensor value. $c_{x_1...x_n}$ is the probability to observe $X_1$ under the known values of all other sensors. Furthermore $a^1_x_x$ is the transition probability of $X_{n+2}$ being equal to $x$ for a given value of $X_1$. Also, $a^{-1}_x_x$ is the conditional probability of $X_{n+1}$ being equal to $x$ for a given value of $X_1$. The initial probability of a sensor k is denoted as $\pi^k_{x_k}$.

To use the Bayesian Network for reconstruction first the above parameters must be estimated by history training data. This is done by the maximum likelihood estimation method on training data. With those parameters estimated it remains to find the ground truth value $E$ for $X_1$, where the sample $Y_1$ is missing. This is achieved by calculating for all possible ground truth data of $X_1$ the conditional probabilities under the known sensor readings $Y_2, ..., Y_{n+2}$. The value with the maximum conditional probability is then selected. For a sequence $y_1...y_{n+2}$ of sensor readings ($y_1$ is set to -1) and a possible ground truth value $x_1$ this probability can be calculated as in equation...
2. Here $\mu$ is the reconstructed sequence of data samples of $\bar{y}_1$ to $\bar{y}_n$ and $U$ represents all possible reconstructed sequences. $\chi_i$ is the range over the discrete domain of sensor $X_i$. Furthermore, $\alpha_{x\mu} = b_{x\bar{y}}(\sum_{x \in \chi_1} a^{-1}_{x1} b_{1 \bar{y}_{n+1}})(\sum_{x_n \in \chi_n} \pi_{x_n} b_{x_n \bar{y}_n})c_{x_1 \ldots x_n}$.

$$P_x|y_1\ldots y_{n+2} = P(H = x_1|Y_1 = y_1, \ldots Y_n + 2 = y_n + 2) = \frac{\sum_{\mu \in U} \alpha_{x_1\mu} \sum_{\hat{x}_1 \in \chi_1} \sum_{\mu \in U} \alpha_{\hat{x}_1\mu}}{\sum_{x_1 \in \chi_1} \sum_{\mu \in U} \alpha_{x_1\mu}} \quad (2)$$

**Experiments:** For the experiment, a sequence of 9000 time slots with values of vital signs was used as ground truth. Each time slot has seven attributes as mean blood pressure and heart rate. The first 8000 data groups were used for training to estimate the parameters. The approach was compared to environmental space time improved compressive sensing (ESTI-CS) and K nearest neighbor (KNN). For data loss, up to 50% the approaches were evaluated and the presented approach outperformed the others in determining the ground truth.

**Conclusion:** The presented approach can retrieve data of large data loss caused by a single sensor. However, the approach only works with discrete data, while the IMU data is continuous. Also, a large amount of data is needed to construct the dependencies between the sensors and the computational complexity is unknown. Therefore, other approaches are preferable.

4 Conclusion

In this survey paper, three approaches for temporal alignment and three approaches for data reconstruction were evaluated. Some approaches were not designed specifically for BSNs but are applicable. The subsample approach presented in section 2.2 may be used to determine and eliminate drifts in the calibration process. Furthermore, the FIFO approach in section 2.1 can be implemented for temporal alignment on run-time. For data prediction of single values of a single sensor the exponential weighted average presented in section 3.1 is a good approach. For a larger amount of lost data and periodic movements the approach presented in 3.2 should be used.

References

Neural Network-based Light Field Depth Estimation

Yunxia Zhu\textsuperscript{1}, Yuriy Anisimov\textsuperscript{2}, and Tewodros Habtegebrial\textsuperscript{3}

\textsuperscript{1} emiliazyx@gmail.com
\textsuperscript{2} yuriy.anisimov@dfki.de
\textsuperscript{3} tewodros.amberbir.habtegebrial@dfki.de

Abstract. In this paper, we present a method to estimate depth based on light field data. We utilize an approach proposed by Changha Shin. The dataset used for this task is HCI Light field dataset. And we have applied a fully convolutional neural network to our model. The input is 4D light field benchmarks consisting of 16 scenes for all 81 views. To overcome overfitting problem methods of data augmentation are used. Our method was measured on test datasets and received a considerable result.

Keywords: depth map, Convolutional Neural Network, HCI Light field dataset, data augmentation

1 Introduction

Human beings have brilliant ability to assess the distance between themselves and other objects, which makes a 2D image created by eyes into a 3D image. However, it is a challenging task in computer vision to get the third dimension information. Depth information is crucial to the development of many industries, such as robotics, 3D reconstruction, self-driving cars and so on. With the rapid progress of light field camera and neural network, it is possible to get a more accurate depth map.

1.1 Epipolar geometry

Epipolar geometry offers a way to have each pixel’s distance away from the camera lens in 2D images, which is vital for 3D reconstruction [1]. Two cameras are used to take a photo of one same object and left and right images are gained. Comparison of these two images is to fetch depth information. As shown in Fig. 1 there are many potential positions for object $X$. Point $X_L$, $X_R$ and camera center point $O_L$, $O_R$ are coplanarity, so $X$ is the intersection of line $X_LO_L$ and $X_RO_R$ [10].

Fig. 1. Epipolar geometry is taken from [10].
1.2 Light Field

The light field is a model to record light distribution. Light field camera takes images of one scene from different views without moving the camera at one time. 4D representation of light field information \( L(u, v, s, t) \) [9] combines the information of spatial and angular by light field camera. As shown in Fig. 2, \((u, v)\) is first plane, and \((s, t)\) is the second plane. \( L(u, v, s, t) \) means that there is a line with direction connecting two points from first plane to second plane. Light field that was put forward helps us go further in the computer version area.

![Light field image representation as two planes method is taken from [9].](image)

1.3 Convolutional Neural Network

Over the past few years, the neural network has developed very rapidly and has been applied to a wide range. With the fast development of neural networks, we have the power to do deeper research in computer vision. And convolutional neural network is widely applied in lots of tasks. As shown in Fig. 3, it is a typical structure of 2D CNN. With the extra depth channel, 2D image is extended to 3D image and CNN on 2D images is extended to 3D convolutional neural network [4]. The structure of 3D CNN is as same as it of CNN, which means that 3D CNN also has convolutional layer, pooling layer and fully connected layer [8].

Convolution is the vital operation in CNN. Eq.1 [7] denotes the convolutional function. \( b_{ij} \) is the bias of kernel which connects \( j \)th feature map in \( i \)th layer. \( w_{ij}^{pqr} \) is the weight at the position \((p, q, r)\) of the kernel which connects \( j \)th feature map in \( i \)th layer, \( P_i, Q_i \) and \( R_i \) are height, width and length of the kernel and \( v_{xyz}^{ij} \) is the value of a unit at position \((x, y, z)\) in the \( j \)th feature map in \( i \)th layer. The filter is slided over the whole area in each feature maps in \((i-1)th\) layer to get one feature map in \( i \)th layer. The output of convolutional layer is feature map which is also called an activation map [8]. The first activation map describes the locations where certain low-level features like corner, edge appear. Second activation map represents higher-level features like semicircles, squares and so on. In pooling layer, pooling operation is used to reduce the spatial size of the matrix in order to decrease the number of parameters and computation. Neurons in a fully connected layer connect to all activations in the previous layer to get the class score.

\[
v_{xyz}^{ij} = b_{ij} + \sum_{k=1}^{m} \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} \sum_{r=0}^{R_i-1} w_{ij}^{pqr} v_{(x+p)(y+q)(z+r)}^{(i-1)k}
\]
2 Related Work

Light field depth estimation becomes a vital field of research in the domain of computer vision especially its connection with neural networks. Many people propose their ideas in this area. Generative Adversarial Networks for depth prediction [3], EPINET [12], neural EPI-volume Network [5] and depth estimation by CNN [11] are some popular and typical algorithms.

Pilippo et.al [3] introduces GAN paradigm to deal with unsupervised depth estimation of a single image. The generator generates a warped target image by reckoning depth from the given image and the discriminator distinguishes between fake images and real one.

Changha et.al [12] uses a fully convolutional neural network to train HCI 4D light field benchmark in order to estimate the depth. Because of the limitation of light field images some data augmentation methods are proposed to improve its performance.

Stefan et.al [5] presents u-shaped network architecture and tests it on Epipolar Plane Image volumes. The input of 3D convolution is two spatial dimensions and one-directional dimension of the LF.

Jiayong et.al [11] proposes an unsupervised CNN-based method for depth estimation from light field, which learns a mapping from light field to the matching disparity map with unlabeled light field image.

3 Dataset

The dataset we manipulate is from [6]. In training dataset it has 16 scenes like boardgames, kitchen, tower and so on, with RGB images and depth from all 81 views. The size of light field image and depth is $512 \times 512$. We deploy same dataset as training for validation, but in training we use small patch $25 \times 25$ which is different as $512 \times 512$ used in validation. Instead of using whole image to train it definitely reduces time cost. In testing dataset there are four scenes with light field images from all 81 views and depth in center view.

4 Depth Estimation Method

We implement EPINET which is proposed by paper [12], using a fully-CNN model and scale, rotation, center view shifting and gamma correction as a way to solve overfitting problem. $L(u, v, s, t)$
shows the spatial resolution \((u, v)\) and angular resolution \((s, t)\) as 4D light field image. Eq. 2 displays how center and other viewpoints are connected, where \(d(u, v)\) is the disparity of the pixel \((u, v)\) and \(\theta\) is an angular direction.

\[
L(u, v, 0, 0) = L(u + d(u, v) \ast s, v + d(u, v) \ast s \tan \theta, s, s \tan \theta)
\]

\[
\tan \theta = t/s
\]

4.1 Neural Network Model

The model we use is from [12]. As shown in Fig. 4, a fully-convolutional neural network is applied to depth estimation. 81 light field images in each scene are arranged to 9 * 9 array by sequence. In case of the occurrence of uninteger viewpoint index, four streams, horizontal view, vertical view, right diagonal view and left diagonal view, are chosen with same baseline. In horizontal stream we choose them from left to right and in vertical stream we select from down to up. Right diagonal view is from left down to right up and left diagonal view is from left up to right down. As we pick out angular resolution as 7, we stack 7 light field images together. The input of the model is \([\text{batchsize}, 7, \text{height}, \text{width}]\). In first layer of neural network "Conv, ReLU, Conv, Batch, ReLU" is utilized and repeated for three times. The kernel is very small, which is only 2 * 2 to get \(\pm 4\) parallaxes. The stride is 1 without any padding because the influence of border can be ignored. For each time 70 kernels are used. After concatenation these four streams, the same structure as it in first layer is utilized in previous 7 blocks and for each block 280 kernels are used. For the last block "Conv, ReLU, Conv" is trained to get disparity map.

![Fig. 4. Architecture of EPINET is taken from [12].](image)

4.2 Data Augmentation

When there is no enough data to train, it leads to negative effect to test dataset even it performs well in training step. Thus, it is very necessary to propose some ways to extend the number of light field images.

Rotation the image and corresponding depth groundtruth by randomly choosing of 90 degrees, 180 degrees or 270 degrees can increase the number of dataset into 3 times [12]. We not only keep attention on light field images but also focus on the angular sequence. As shown in Fig. 5 black arrow stands for the original angular sequence of four streams, horizontal stream, vertical stream, right diagonal stream and left diagonal stream. After rotation by 90 degrees each one comes to the position which is represented by red arrow. New horizontal view is equal to the rotated original
vertical view with the same angular sequence, which is same as new right diagonal view. However, for new vertical view we turn the original horizontal view by 90 degrees and reverse the original angular sequence in original horizontal view in order to keep concordant, which also happens for new left diagonal view.

Besides rotation, some methods are used in [12]. Sampling by skipping one pixel is to get small patch 25*25 in train. Compared to continues sampling the advantage of it is to stay stable of images with noise. There are many options for angular resolution, such as 3*3, 5*5, 7*7 or 9*9. For example, if 5*5 angular resolution is choosen, we are able to shift the center view as shown in Fig. 6, which totally extends 25 times of the original dataset from red matrix to orange matrix or from red matrix to green matrix or from green matrix to blue matrix or from orange to blue matrix. In addition, gamma correction[0.8, 1.2] is applied for changing its illumination.

5 Result

The model is trained for 5 to 6 days with GPU GTX1080Ti. Main parameters are listed in Tab. 1. The result of 9*9 angular resolution with all data augmentation is shown in Fig. 7. Mean squared
error and bad pixel percentage are computed for 4 light field images. The mean squared error is 2.50 and bad pixel percentage (> 0.07 px) is 17.51, which means the percentage of pixels whose absolute loss between prediction and ground truth is greater than 0.07 is 17.51. The result of 7 * 7 angular resolution with all data augmentation including center view shifting is shown in Fig. 8. The mean squared error is 2.41 and bad pixel percentage (> 0.07 px) is 27.65. Mean squared error between them are very close, but for 7 * 7 angular resolution mean squared error is better, which shows center view shifting helps improve the performance. We can assume that if 3 * 3 or 5 * 5 is used for training, the performance of this model will be better because the number of dataset is greatly increased.

Table 1. Main parameters in EPINET

<table>
<thead>
<tr>
<th>Library</th>
<th>Python 3.6(pytorch 1.2.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
<td>16</td>
</tr>
<tr>
<td>Epoch</td>
<td>10000000</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.1**4</td>
</tr>
<tr>
<td>Optimizer</td>
<td>RMSprop</td>
</tr>
<tr>
<td>Loss Function</td>
<td>L1Loss</td>
</tr>
</tbody>
</table>

Fig. 7. Result of 9 * 9 angular resolution. The first row is prediction and second row is ground truth.
6 Conclusion

Epipolar geometry in light field not only focuses on left and right two views, but also pays attention to multiple other views, which offers more useful information to estimate depth. Besides, we make use of a fully convolutional neural network for processing huge amount of data. Thus, EPINET [12] has an outstanding performance of depth estimation in 4D Light field Benchmark [6]. But it needs to be trained around 5 to 6 days in GPU GTX1080Ti, which greatly consumes time. And it is also sensible for the images with reflection or noise even with data augmentation. There is still lots to be explored to see if there is more efficient and intelligent way to get depth map.

References


