# Near Real-time Face Parsing

Ayush Minocha, Digvijay Singh, Nataraj Jammalamadaka, C. V. Jawahar
CVIT, IIIT Hyderabad
ayush.minocha,digvijay.singh@students.iiit.ac.in, nataraj.j@research.iiit.ac.in, jawahar@iiit.ac.in

***Abstract*—Commercial applications like driver assistance programs in cars, smile detection softwares in cameras typically require reliable facial landmark points like the location of eyes, lips *etc*. and face pose at near real-time. Current methods are often unreliable, very cumbersome or computationally intensive. In this work, we focus on implementing a reliable and real-time method which parses an image and detects faces, estimates their pose and locates landmark points on the face. Our method builds on the existing literature. The method can work both for images and videos.**

***Keywords*—Face detection, Real time, Facial points, Pose, Landmark points, face parsing**

## I. Introduction

Face detection and recognition is one of the oldest problems in computer vision. Recent developments [11] have shown a positive trend towards detecting faces in the wild. As a consequence, many applications demand information beyond location and identity of the person. This information is in the form of landmark points (eyes, nose, ears *etc*. ) and pose (orientation) of the face. Driver assistance programs, for example, process the area around the landmark points on the driver's face and raise an alarm if driver is dozing off. They can also predict when the driver is not paying attention to the road. Similarly, cameras have on-chip programs which can detect smile using the landmark points. Such applications need an accurate and fast detector as a false or a late detection is not viable. In this paper, we aim to parse an image to locate a face, determine landmark points and estimate the pose of the face very accurately and in near real-time.

In the past, each of the face detection, pose estimation and landmark detection problems have been independently addressed until Zhu and Ramanan [11] brought them together using the Pictorial structures framework [3]. Some of the popular face detection methods are [8], [6]. However, Viola Jones detector [10] is undoubtedly most popular among them, and is a real-time solution. For the pose estimation, methods like [5], [7] have state-of-art results. Finally for landmark detection, active appearance model [9] and Constrained Local Models [1] are very popular. The work by Zhu and Ramanan [11] outperforms all the above methods in three tasks of detection, pose estimation and landmark estimation. This performance of course comes at a *very high* computational price. For an image of size $770 \times 500$, the algorithm takes 14.5 seconds giving the output displayed in Figure I. Please note that the output displayed contains the locations, poses (in degrees) and landmark points on the faces detected. For comparison, algorithms like Viola Jones [10] typically take about 0.05 seconds for face detection.

A deeper analysis of Zhu and Ramanan [11] revealed that some of the sub-parts of this algorithm dominate the computation. Further, the prior knowledge about the structure of the human face is not effectively used. For example, all human faces have eyes above the nose-level. If the input to the algorithm is a video, person can be tracked very efficiently at a near-real time speed. In our work, we use the above observations to develop a robust and real-time method which parses an image to detect face, estimate the pose and locate landmark points. An image or a video frame is first processed using a fast face detector. We use the OpenCV implementation of the Viola-Jones [10] algorithm as the detector. The detector quickly gives a set of face detections, some of which can be false positives. The detections are then passed through the proposed method to obtain accurate face detections, pose estimates and land mark points. If the input is a video, results from the previous frame are used to further speed-up the proposed method. The results were observed to be very fast and accurate. A face of size $380 \times 250$ takes 0.3 seconds in our method as compared to 2.4 seconds in Zhu and Ramanan's method [11].

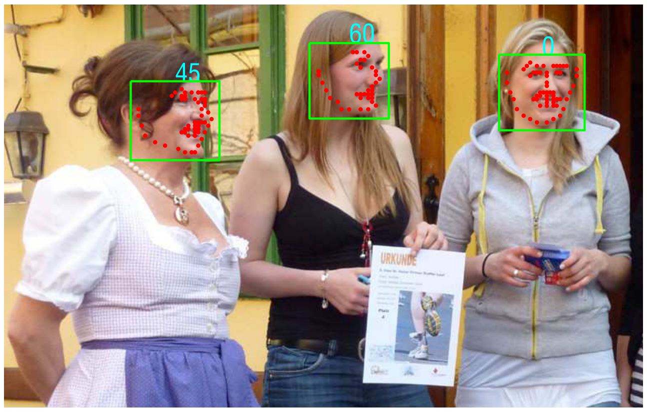

Fig. 1. Zhu and Ramanan algorithm [11] has detected the three faces, the angle of these faces w.r.t to the camera, called as *pose*, and locations of various points on the faces like eyes, ears, nose, chin, jaw-line *e.t.c* ,called as *landmark points*, of the three people in the image. It took about 14.5s to process this image of size $770 \times 500$.

## II. Efficiency and Effectiveness

In this section, we describe and evaluate the two popular face detection algorithms, Viola-Jones [10] and Zhu and Ramanan [11]. Both the algorithms are compared using three measures: precision, recall and computational time. For this purpose, we annotated the first 500 frames of a popular television show "*The big bang theory*" ,which henceforth is called as "BBT dataset", and ran both the algorithms on this data. The dataset has significant variations in size and pose of faces.

**Viola Jones (VJ) [10]** algorithm models the face as a collection of rectangular shaped Haar-like features. Three types of features containing two rectangles, three rectangles and four rectangles are used. These rectangles have same dimensions and placed adjacent to each other. All the pixels within each rectangle are summed up and feature is simply the difference of summations among different rectangles. Using the integral map, these summations and differences can be very efficiently computed. The classifier used is the Adaboost algorithm which combines the series of weak classifiers cleverly to obtain a strong classifier. The weak classifier $h_t$ is given by:

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) < p\theta \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $f$ is the feature, $\theta$ is the threshold and $p$ is the polarity indicating the direction of the inequality. The following equation describes the strong classifier:

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\alpha_t$ are the weights of the classifier. For real-time performance, a series of classifiers with increasing complexity are trained and are arranged as a cascade. The complexity of the classifier is increased by increasing the number of weak classifiers. The first two classifiers in the cascade, for example, use only two features. The threshold of these initial classifiers are adjusted such that extremely high detections rates close to $100\%$ are achieved at the cost of high false positive rates. The idea is to reject all the simple subwindows and pass the difficult ones to the next stage of the classifiers. Each classifier is trained only on those samples which have passed through all the previous classifiers. Thus difficulty of the task increases for the latter parts of the cascade. During testing, the image is passed through this cascade of trained classifiers and the sub-windows which are rejected at any point during the parse are ignored and the ones which goes through all of them are finally selected. We use the OpenCV implementation of the algorithm in our paper.

**Zhu and Ramanan (ZR) [11]** algorithm models the face as a collection of parts and represents them as a collection of tree structured pictorial structures [3]. These parts are the keypoints on the face which include the edges of eyes and lips, tip of the nose and few points on the jaw line. The neighboring points are connected by an edge to capture the structure of the face. The model consists of $M$ mixtures of trees $T_m = (V_m, E_m)$ where $V_m$ are the nodes and $E_m$ are the edges in the mixture $m$. An image is represented by $I$ and each face is represented by configuration $L = (l_1, ..., l_n)$ where $l_i$ is the location of part $i$. A configuration $L$ in a mixture $T_m$ is scored as :

$$S(I, L, m) = App_m(I, L) + Shape_m(L) + \alpha^m \tag{3}$$

$$App_m(I, L) = \sum_{i \epsilon V_m} w_i^m \dot{\phi}(I, l_i) \tag{4}$$

$$Shape_m(L) = \sum_{ij \epsilon E_m} a_{ij}^m dx^2 + b_{ij}^m dx + c_{ij}^m dy^2 + d_{ij}^m dy. \tag{5}$$

The scoring function accumulates the appearance score of each part $l_i$, the relative positions of a part pair $e_{i,j} = (l_i, l_j)$ and bias for the mixture $m$. All the parameters listed above are learned using structured prediction framework [11]. Please note that each mixture component models a particular pose range. In their implementation, a total of 13 components corresponding to $\pm 90°, \pm 75°, \pm 60°, \pm 45°, \pm 30°, \pm 15°$ *and* $0°$ are used.

TABLE I. **Performance Analysis.** *The performance of Viola Jones [10], Zhu and Ramanan [11] and proposed method are compared in terms of precision, recall and computational time.*

| Algorithm | Viola Jones [10] | Zhu and Ramanan [11] | Ours |
|---|---|---|---|
| Precision | 15.6 | 10.7 | 72.0 |
| Recall | 59.1 | 64.1 | 57.6 |
| Time | 0.08 | 162.86 | 5.05 |

For this work, the analysis of inference is very relevant. Here the inference algorithm is briefly described. Given any image, first a feature pyramid is constructed. The feature used is histogram of oriented gradients [2]. Then the part filters across all the mixture components are convolved to obtain responses. These responses serve as the appearance/unary term given in equation 4. Since the CRF described above is a tree-structured graph, exact inference is performed using belief propagation. The following equation describes the inference procedure for one pyramid level,

$$S^*(I) = \max_m [\max_L S(I, L, m)]. \tag{6}$$

Let $L$ be the number of levels, $M$ be the number of mixtures, $K$ be the number of parts and $N$ be the number of possible part locations. The number of edges would be $K - 1$. The standard belief propagation [3] has a complexity of $O(KLMN^2)$. Using the distance transform [4], the complexity of the algorithm can be reduced to $O(KLMN)$. Distance transform solves the following problem very efficiently:

$$dt(x_2) = \min_{x_1} f(x_1) + (x_1 - x_2)^2. \tag{7}$$

For each value of $x_2$, the minimum value of the function on the right has to be found. A simple algorithm takes $O(N^2)$ where $N$ is the number of possible values that both $x_1$ and $x_2$ can take. Distance transform has an order complexity of $O(N)$. The rhs of the above equation can viewed as a shifted parabola in terms of $x_2$. Thus for each $x_1$, a parabola can be described. Distance transform finds the lower envelope of these set of parabolas in $O(N)$ time. After the belief propagation algorithm is run, only those configurations are considered whose max-marginals at the root node are greater than $-1$.

On the whole the algorithm can be split into four parts: (i) constructing feature pyramid, (ii) convolution with part filters ($App_m$ in equation 4), (iii) belief propagation and distance transform (equation 7) and (iv) non-maximal suppression and miscellaneous tasks. In this work, we reduce the complexity of the algorithm to $O(KMN)$ in case of detection and $O(KN)$ in case of tracking. Further, we search over a small of fraction $K$, $M$, $N$ reducing the time taken.

**Comparison:** Table I gives the results of Viola-Jones' [10] and Zhu and Ramanan's [11] approach on the BBT dataset. It can be observed that Zhu and Ramanan's [11] approach has better recall values, but is computationally very expensive taking more than 2.5 minutes per frame. Viola-Jones [10] on the other hand has relatively low precision and recall but is very fast at 0.08 seconds. Our method, as the table I suggests, strikes a nice balance between quantitative performance and computational cost.

## III. Face Parsing

Given an image or a frame of a video, it is first processed using a *face detector*. In our implementation, we use VJ algorithm and get face detections. To ensure that there are no misses, the threshold of the algorithm is lowered at the cost of having more false positives. The detected bounding boxes are then expanded by a scaling factor $F$ to allow the subsequent components of the method to refine the detections. We then pass on the expanded bounding boxes to the two subcomponents of our method, *Face-Track* and *Face-Detect*. *Face-Track* tracks the faces from the previous frame by initializing the ZR algorithm to detection from the previous frame and then efficiently finding the face in the current frame. If the method is run on a stand-alone image, *Face-Track* algorithm returns an empty set. The outputs of both *face detector* and *Face-Track* are sent to *Face-Detect* algorithm. *Face-Detect* is an optimized version of ZR which takes advantage of the face structure and the location of the face pointed by face detector. It runs only on those *face detector* bounding boxes which are not associated by *Face-Track* algorithm.

As we have noted earlier, ZR is computationally expensive. It handles the variations in translation, scale and pose by sliding the part templates over the entire image, searching over feature pyramid and searching among the components of the mixture model respectively. All the three operations are computationally expensive considering that they have to be repeated over all the parts of the face. To speed-up the algorithm, we constrain the number of feature pyramid levels and locations that sliding window visits in *Face-Detect* algorithm. In *Face-Track* algorithm, we further restrict the number of components in the mixture model. Each of these steps significantly speeds-up the algorithm and cumulatively have a factor times speed-up. Below, we describe all the three optimizations.

### A. Reducing Scale Space levels

Zhu and Ramanan [11] algorithm is built using faces of certain standard size. Any face whose size is less than the standard size cannot be detected. But faces whose size is larger than the standard size are detected using the feature pyramid trick. A pyramid consists of a collection of images with base image being the original image itself and the subsequent levels having images of gradually lesser sizes. Thus images stacked on top of each other in a progression reminds of a pyramid. A feature pyramid is simply a pyramid containing feature responses to a particular filter. In our case, we use HoG [2] as the feature. After building the pyramid, Zhu and Ramanan [11] algorithm is run over all the scales.

We have observed that a maximum response is obtained at that level where the detector's dimensions match that of the face. Unfortunately, obtaining the detector's dimension is not straight-forward, since the Zhu and Ramanan [11] has a collection of part templates rather than a single large face template. Therefore cross validation has been used to find the detector's [11] dimensions. Given the face dimensions and

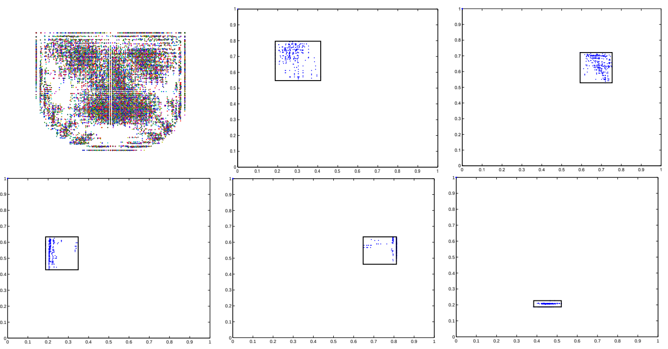

Fig. 2. **Mixture components and Parts:** In the clock-wise direction, mixture component corresponding to $0°$ and locations of left eye, right eye, chin, left ear and right ear are plotted in a normalized space.

the detector dimensions, the level at which the best detection occurs can be approximately estimated using,

$$l = \lceil \log_2 H - \log_2((h-d)S))I + I + K \rceil \tag{8}$$

where $l$ is the desired level, $H$ is the the height of the face, $h$ is the height of the detector, $d$ is a constant, dependent on the size of padding, $S$ is the HoG [2] cell size, $I$ is the octave interval in the feature pyramid and $K$ is a constant. Using the above formula, a level $l$ is estimated given the face dimensions by the *face detector* [10]. The feature computation is then restricted to levels $\{l-1, l, l+1\}$ to accommodate for approximations in the above formula.

Reducing the scale space to 3 levels significantly reduces the computation time. In the section II, we noted that the order complexity of the algorithm is $O(KLMN)$. By reducing the number of levels from $L$ to 3, the order complexity is reduced to $O(KMN)$. Reduction in the scale space levels saves time in the following steps of the algorithm: (i) feature pyramid construction, (ii) convolution with part filters ($App_m$ in equation 2) and (iii) belief propagation. Both *Face-Detect* and *Face-track* algorithms use this optimization.

### B. Reducing Spatial Configurations

To optimize the search in possible locations parts, we use the fact that every keypoint on the face can be localized to some approximate region. For example, the right ear cannot be found on the left side of a frontal face. Thus it is not necessary to do a sliding window search over the whole search space. These regions are obtained by collecting statistics from Zhu and Ramanan's [11] outputs on a number of images from our dataset. First VJ algorithm is run on each image and correct detections are retained. These detections are then expanded by a factor of F and passed to ZR algorithm. The value of F is determined to be 1.6 using cross-validation. After running the Zhu and Ramanan [11] algorithm, bounding boxes of face detections are computed using the minimum and maximum indices of the face keypoints. Next the keypoints are normalized with respect to the previously obtained bounding box from VJ algorithm. The key points from all the images in the validation data are collected to obtain an approximate region for each keypoint. These approximate regions are obtained for each part-mixture component combination.

Using the localized regions for each part, we limit the search for a part to be in a definite region. This saves time while computing the appearance term $App_m$ as the convolution operation is on a much smaller region. Similarly the belief propagation is limited to the above localized regions. Figure III-B illustrates that the parts are well localized regions occupying a small region in the search space. While this step does not change the order complexity, it reduces the time taken by the algorithm as the number of spatial locations searched by the algorithm is now limited to regions determined above. While this optimization can be used both for *Face-Detect* as well as *Face-Track* algorithms, *Face-Track* algorithm can be further optimized. *Face-Track* algorithm uses the detection from previous frame to refine the current detection. Thus the algorithm can be limited to the previous frame's part positions and a small neighborhood around them. The scaling factor for converting the part positions from original scale to a particular level in the pyramid can be directly computed using,

$$SF = 2^{-\frac{(l-I-K)}{I}} \tag{9}$$

where *SF* is the scaling factor, $l$ is the level of the pyramid under consideration, $I$ is the octave interval in the feature pyramid and $K$ is a constant. The constant $K$ in case of Zhu and Ramanan [11] is 1.

### C. Reducing number of components

In the *Face-Detect* algorithm, since it is not known apriori what is the pose of the face detection given by the VJ algorithm, the ZR algorithm has to search over all mixture components. *Face-Track* algorithm on the other hand has the prior information from the previous frame. It can be safely assumed that between two consecutive frames the pose of the face will not significantly change. Hence the *Face-Track* algorithm simply searches for the best match around the pose obtained from the previous frame. To be more precise, if the mixture component of the previous detection is $m$, the search space is restricted to $\{m-1, m, m+1\}$. This step needs a careful consideration. When the search transitions from a mixture component corresponding to near frontal angle to the one corresponding to the a side profile, the part correspondences have to be carefully mapped. This will prove very useful in initializing the locations of parts and thus optimizing the search over spatial configurations.

## IV. Experiments

In this section we evaluate our algorithm on both performance and efficiency. To evaluate the algorithm we have created two datasets and used another standard dataset. We evaluate only the face detection accuracies and the speed of the algorithm. For the performance of pose estimation and landmark localization please refer [11] as our algorithm does not effect performance of these two tasks. The next few sections describe the datasets used for evaluation, experimental set-up and the results obtained.

**Datasets:** The proposed method is evaluated on three datasets *viz.*, driver dataset, AFW dataset [11] and BBT dataset. The datasets are in the increasing order of complexity. The driver dataset consists of four YouTube videos containing people driving a vehicle and facing camera at different poses. This dataset has been chosen to mainly showcase the algorithm for driver assistance application. The dataset is relatively simple with typically one or two persons in a vehicle and a relatively simpler backgrounds. The AFW dataset has been introduced in Zhu and Ramanan [11]. It consists of images with multiple people at varying poses located in cluttered background. Finally the BBT dataset, introduced in section II, is the most challenging of the three. It has significant variations in pose and size of the faces and imaging conditions.

**Experimental set-up:** Each image is processed by the proposed method to obtain face detections. The outputs and the computation times of (i) feature pyramid creation, (ii) Part filter convolution, (iii) Belief propagation, (iv) Non maximal suppression and other miscellaneous codes and (v) total time are noted. Precision, recall and computational times are used as the measures.

**Results:** The first step of the algorithm is the VJ face detection algorithm. Both the frontal face and side profile detectors of VJ algorithm are run. As mentioned before, we lower the threshold of the algorithm to increase the detection rate. Table II shows the performance of the face detection before and after lowering the threshold and the total time taken on all the datasets. As expected, the precision of the algorithm drops and the recall increases as the threshold is lowered. The time taken by the algorithm is unaffected by the threshold used. Please note that the value of the threshold has implications on the speed of the whole algorithm. For lower thresholds, more detections appear and both *Face-Detect*, *Face-Track* have to do more work.

Next these face detections are passed through the *Face-Detect* and *Face-Track* parts of the algorithm. Table III shows the performance of our algorithm on the Driver dataset, AFW dataset [11] and BBT dataset respectively and compares it with [11]. As expected, the algorithm has performed excellently on the simple Driver dataset with perfect precision and recall. Due to the relatively simpler settings of the dataset, Viola-Jones algorithm [10] was able to detect most of the faces with high precision and recall. On the AFW dataset, it has very good precision and recall. On the BBT dataset, the method has surprisingly good performance. We believe the improved performance is because of the tracking step (*Face-Track* algorithm) which the other two algorithms [10], [11] (please refer to Table I) do not leverage by design. On the whole, the performance drop in terms of recall (table III) is minimal on the driver dataset and acceptable on the BBT dataset. On both these datasets the algorithm performs *Face-Track* step. On AFW dataset the drop in recall is tolerable. Figure IV shows some of the detections on the three datasets.

Tables III,IV give the statistics on average computational time of our method. As summarized before, the algorithm can be broadly split into (a) feature pyramid computation, (b) convolution and (c) belief propagation. Table IV gives the computational time of the above mentioned sub-parts of our method , the total time taken on Driver dataset and compares it to that of Zhu and Ramanan [11]. Clearly the overall method has achieved a speed-up by a factor of 35 times over Zhu and Ramanan [11]. The table also shows the computational times of *Face-Detect* and *Face-Track* steps. As expected *Face-Track* is faster than *Face-Detect* step and hence it is desirable to

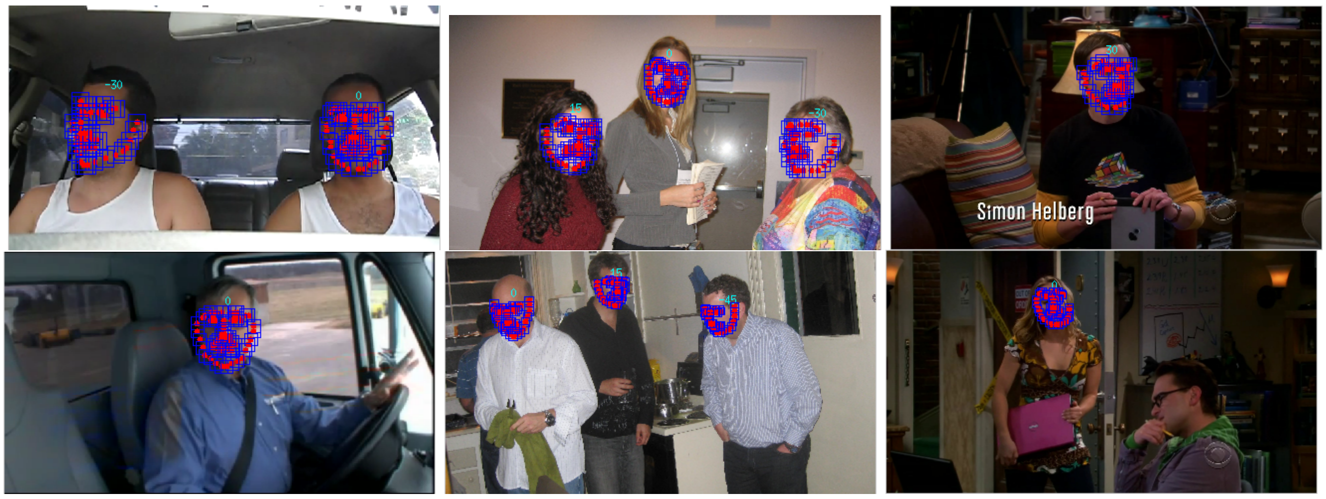

Fig. 3. **Qualitative Results.** *The example detections of the proposed method on Driver dataset (column 1), AFW dataset [11] (column 2) and BBT dataset (column 3).*

TABLE II. **Quantitative Results.** *The precision (P), recall (R) and computational time (T) of Viola jones before and after lowering threshold.*

| | P-before | P-after | R-before | R-after | Time |
|---|---|---|---|---|---|
| Driver | 100 | 100 | 97.5 | 100 | 0.35 |
| AFW | 68.4 | 9.65 | 76.9 | 61.88 | 9.56 |
| BBT | 72.0 | 14.03 | 56.6 | 89.52 | 5.05 |

TABLE III. **Quantitative Results.** *The precision (P), recall (R) and average computational time (T) on various datasets.*

| | P-ours | P [11] | R-ours | R [11] | T-ours | T [11] |
|---|---|---|---|---|---|---|
| Driver | 100 | 100 | 97.5 | 100 | 0.35 | 12.97 |
| AFW | 68.4 | 10.2 | 76.9 | 93.8 | 9.56 | 242.9 |
| BBT | 72.0 | 10.7 | 56.6 | 64.1 | 5.05 | 162.9 |

have more *Face-Track* runs. Table III gives the computational time of our algorithm on various datasets and compares it with Zhu and Ramanan [11]. Our algorithm has a speed of 2.9 FPS on the driver dataset. The reason for this speed is because the algorithm ran primarily the *Face-Track* step. On the AFW dataset the speed of the algorithm has dropped. There are two reasons for decrease in speed: (i) Viola-Jones [10] outputted many false positives and as a consequence *Face-Detect* step had to be run many more detections and (ii) since the dataset has stand-alone images, *Face-Track* could not be used. On the BBT dataset, the method has a good speed. Overall the algorithm has a near-real time speed with good performance.

## V. Conclusion

In this paper, we proposed a near real-time method which detects face, estimates pose and locates landmark points. The algorithm of Zhu and Ramanan [11] has been throughly optimized to achieve this speed. We introduced two new datasets (a) BBT dataset and (b) driver dataset. We demonstrated that on videos and particularly those with simpler backgrounds, the algorithm runs with good performance and high speed. This algorithm is particularly suitable for applications like driver-assistance programs and smile detection softwares on cameras.

TABLE IV. **Average computational time.** *The timings in seconds computed on Driver dataset of various components of the algorithm are displayed. 'Feat.Pyr' stands for feature pyramid computation and 'BP' stands for belief propagation.*

| Algo component | Feat.Pyr | Convolution | BP | Total |
|---|---|---|---|---|
| Face Detect | 0.013 | 0.40 | 0.19 | 1.18 |
| Face Track | 0.013 | 0.03 | 0.19 | 0.31 |
| Overall | 0.013 | 0.07 | 0.19 | 0.35 |
| [11] | 0.620 | 6.590 | 5.587 | 12.974 |

## References

[1] P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar, "Localizing parts of faces using a consensus of exemplars," in *CVPR*, 2011, pp. 545–552.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR (1)*, 2005, pp. 886–893.

[3] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 55–79, 2005.

[4] ——, "Distance transforms of sampled functions," *Theory of Computing*, vol. 8, no. 1, pp. 415–428, 2012.

[5] L. Gu and T. Kanade, "3d alignment of face in a single image," in *CVPR (1)*, 2006, pp. 1305–1312.

[6] B. Heisele, T. Serre, and T. Poggio, "A component-based framework for face detection and identification," *International Journal of Computer Vision*, vol. 74, no. 2, pp. 167–181, 2007.

[7] M. Jones and P. Viola, "Fast multi-view face detection," MERL, TR2003-96, Tech. Rep., 2003.

[8] Z. Kalal, J. Matas, and K. Mikolajczyk, "Weighted sampling for large-scale boosting," in *BMVC*, 2008, pp. 1–10.

[9] I. Matthews and S. Baker, "Active appearance models revisited," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 135 – 164, November 2004.

[10] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, May 2004.

[11] X. Zhu and D. Ramanan, "Face detection, pose estimation, and landmark localization in the wild," in *CVPR*, 2012, pp. 2879–2886.