Real-time static hand gesture recognition from skeleton data as numeric input for computer systems

Thomas Finlay
Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
tfi18@uclive.ac.nz

Prof. Richard Green
Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
richard.green@canterbury.ac.nz

***Abstract*—This paper proposes a method to enable real-time static hand gestures to control number and command input into computer systems. The method combines static hand gesture recognition and skeletonization using MediaPipe Hands, skeleton classification using a Support Vector Machine (SVM), and an input debouncing method with a novel extension to enable repeating. The classification method has an accuracy of 88%, less than that achieved by earlier work. The debouncer out-performs alternatives, and the method is found to have a viable throughput of 31.76 frames per second.**

Keywords—static hand gesture, SVM, debouncing, MediaPipe

I. Introduction

Hand gestures are recognized as a natural form of human expression and interaction. Vision-based approaches to gesture detection and classification can allow for HCI to become more comfortable for people as computers become increasingly integrated in our lives [1]. Despite this promise, such interfaces remain uncommon in consumer-facing systems today, and most publications focus on implementing individual components of such interfaces without considering the whole.

This paper proposes a method to enable real-time digit-by-digit number input using hand gestures, with the intention of producing a solution suitable for providing an HCI interface for setting and controlling a kitchen timer or microwave, but which can be extended for other applications in future. For applicability in such an environment, the input must operate in real-time on commodity devices while remaining accurate enough to provide a mode of input that is sufficiently efficient and reliable for everyday use.

The solution proposed in this paper builds on prior work to produce a pipeline that consists of the following major steps performed for each frame of input video:

1. Hand detection and pose estimation.
2. Skeleton classification into a number from 0-5 (inclusive) using either Heuristic or Support Vector Machine (SVM) approaches (if a hand is detected).
3. Input debouncing and time-based output repeating.

II. Background

Large parts of this problem have already been solved to a satisfactory degree.

Zhang et al. developed a hand detection and pose estimation (skeletonization) method and published their work for re-use in the Hands module of the MediaPipe framework [2]. Previous work has shown that MediaPipe Hands works with an accuracy of 84.57% for static single hands [3], which should be sufficient for this purpose. In addition, the module provides a simple API and works efficiently on commodity hardware, with their preferred hand landmark model taking only 16.1ms (62fps) to place landmarks on an image using the Pixel 3 mobile phone [2].

Sung et al. devised both a Heuristic and a Neural Network (NN) approach to real-time hand classification of MediaPipe Hands skeleton data into gestures that runs at 30fps on "mainstream mobile devices" [4]. The heuristic classifier was found to be more intuitive for developers, but less accurate and precise than the NN approach, having a recall rate of only 44.4% compared to the NN's 87.9% recall rate, both over an alphabet of 6 gestures.

Osipov and Ostanin devised a similar method that operated at 71fps using an SVM with a one-to-one decision function to classify skeleton data sourced from MediaPipe Hands into an alphabet of 10 gestures [5]. They found that rbf, linear, and poly kernels all provided at least 98% accuracy at 71fps throughput while the sigmoid kernel had low accuracy. They also described experiencing low-accuracy MediaPipe Hands output for some of the training images in their dataset, which they attributed to the low resolution and bounding box cropping of the hand images in their dataset.

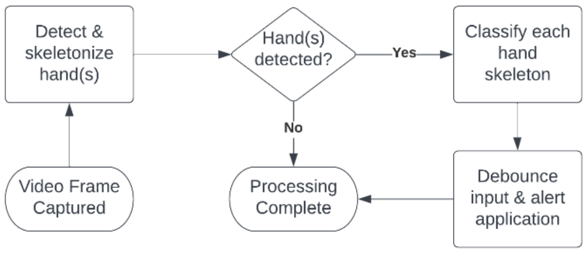
SVMs are a type of supervised machine learning model often used for classification tasks. They are effective for classifying high-dimensional data with a light memory footprint [6].

The machine learning classification space lacks published methods for output debouncing. However, one promising post-processing method (ZYFGAS Debouncing) for debouncing gesture classification output is devised by Young, Stephens-Fripp, Gillett, Zhou, and Alici in [7]. This method minimizes the effect of individual incorrect predictions by culling them as soon as the input returns to the current gesture. This approach's memory requirements scale linearly with the number of gestures recognized, limiting its viability for memory-constrained applications with many classifier outputs. However, this is unlikely to be an issue at current scales as 'large' image categorization datasets contain 11221 [8] or 5247 [9] categories, amounting to only kilobytes of memory required for debouncing. This is no barrier for the proposed solution as it only recognizes ten distinct gestures. The ZYFGAS approach also adds some complexity to the debouncer as it has an 'unknown' state and requires an initial state. Simpler methods such as the time delay approach described in [10] and [11] do not have these deficiencies, though they require some work to be usefully adapted to this problem and are of unknown utility for this application.

Prior work by Chung, Kim, Na, and Lee found that physical touchpads and touch-screen touchpads for numeric input had mean error rates of 5.82% and 5.67% respectively

[12], translating to an accuracy of 94.18% and 94.33% respectively. While their sample size was quite small at only 24 individuals, this result provides an approximate target for the accuracy of the proposed solution if it is to compete with these existing input methods.

III. Method



Detect & skeletonize hand(s)
Hand(s) detected?
Yes
Classify each hand skeleton
No
Video Frame Captured
Processing Complete
Debounce input & alert application

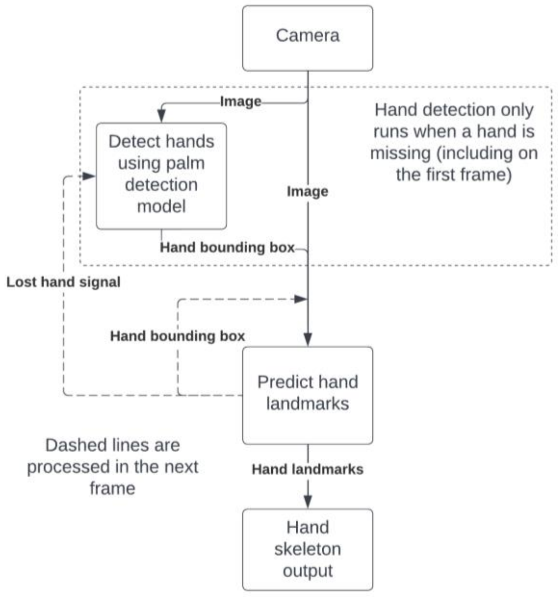
Fig. 1. Flow chart showing an overview of the proposed input handling software pipeline.

The proposed method is composed of three steps performed on each frame of video according to the flow chart shown in Fig. 1. Each of these steps are described in more detail in their own subsections.

A. *Detection and skeletonization of hands*

MediaPipe Hands is a software library developed and maintained by Google that provides an API to detect, track, and provide skeleton data for hands when given an image or frame of video [13]. It is designed to run in real-time "on mobile GPUs with high prediction quality" [4], ensuring that it does not prevent the proposed method from maintaining high throughput on the target devices.

An overview of the MediaPipe Hands hand detection and skeletonization process can be seen in Fig. 2. The hand detection bypass shown in the figure assumes that the model is being passed a video stream, making it suitable for use in the proposed solution. However, when MediaPipe Hands is used on still images to produce training data for the SVM model, this mode is unsuitable as hands will move significantly between 'frames' of this data. As such, this optimization is disabled in this context.



Camera
Image
Hand detection only runs when a hand is missing (including on the first frame)
Detect hands using palm detection model
Image
Hand bounding box
Lost hand signal
Hand bounding box
Predict hand landmarks
Dashed lines are processed in the next frame
Hand landmarks
Hand skeleton output

Fig. 2. High-level overview of the MediaPipe Hands process, including the hand detection bypass optimization. Adapted from [2].

For each hand detected, MediaPipe returns the 3D coordinates of 21 keypoints for the locations shown in Fig. 3. One for the base of the palm, and four for each finger. The annotations always follow the order of the fingers from thumb to pinky rather than from left to right in the image, providing a stable identity for each finger as four keypoints at specific indices. This property means that the positions of individual fingers are captured, rather than the shape of the hand, allowing for gestures to be defined in relation to specific fingers.

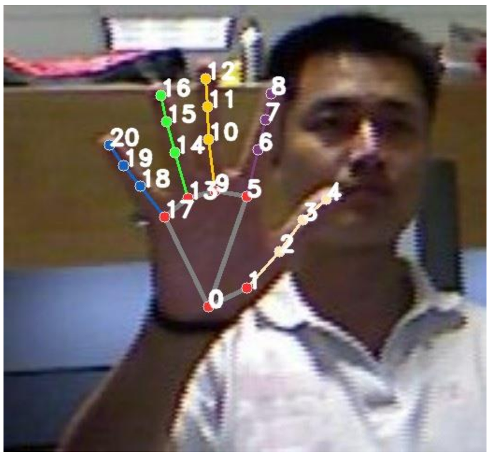


Fig. 3. A cropped image from the SVM training data [14] annotated with MediaPipe Hands keypoints and their indices.

Once generated, each hand's skeleton keypoints are passed into the skeleton classification step.

B. *Skeleton classification*

During skeleton classification, the keypoint data defining the skeleton of a single hand is classified into one of six classes corresponding to the integers in the range [0, 5] with each corresponding to a specific static hand gesture being performed.

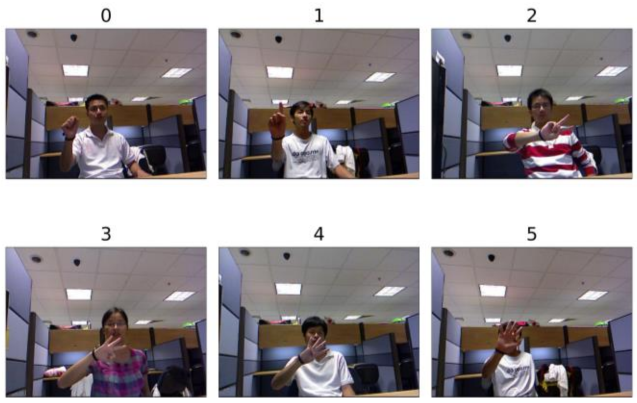


Fig. 4. An image from the Ren, Yuan, Meng, and Zhang dataset [14] for each label.

The classification methods were evaluated for accuracy against the RGB images in the dataset developed by Ren, Yuan, Meng, and Zhang [14] (sample images shown in Fig. 4), with horizontally mirrored images generated to increase the size and diversity of the dataset. The dataset was then split into training and testing sets, with 80% of the images in each label (mirrored images were considered their own label for this purpose) randomly picked into the training set and the remaining 20% moved into the testing set. The data was then pre-processed through MediaPipe Hands, producing a CSV file of the coordinates of the landmarks detected by MediaPipe Hands for each label directory, increasing the speed at which the classification methods could be trained and evaluated.

This dataset was chosen as it contains high-resolution images of the required hand gestures, uncropped in a setting with a realistic background. This meant that MediaPipe Hands detected and skeletonized a hand in 96.8% of the images used for training and testing, with at least 94.0% (188) of the 200 images per label being skeletonized. Another dataset containing 3600 100x100px images cropped to just the hand for each label [15] was evaluated and found to be unusable as MediaPipe detected only 15.4% of hands in total with 0.39% of images with the label '4' skeletonized. This follows Osipov and Ostanin's experience of poor MediaPipe Hands performance for cropped and low-resolution images.

The following four classification methods were evaluated for accuracy:

1. An adaptation of feature angle thresholding (FAT) as proposed by Sung et al. in [4].
2. Linear kernel for an SVM model operating with MediaPipe Hands skeleton data as proposed by Osipov and Ostanin in [5].
3. RBF kernel for an SVM model operating with MediaPipe Hands skeleton data as proposed by Osipov and Ostanin in [5].
4. RBF kernel for SVM operating with feature angles (SVM-FA) derived from FAT (A linear kernel was also evaluated but is omitted due to very similar accuracy results).

1) *FAT*

The FAT method was implemented in pure Python 3.10, using only the standard library. Unlike in [4], this implementation operated in two dimensions.

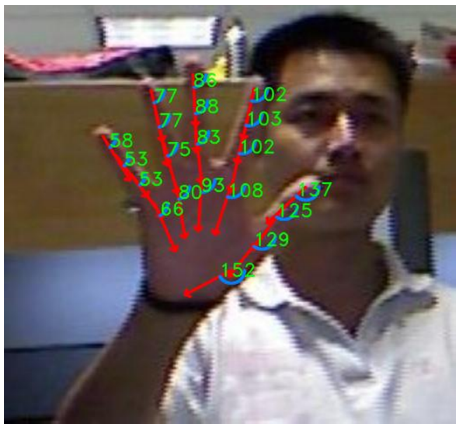


Fig. 5. Finger segments shown running from S_{i+1} to S_i and each segment's angle from the horizontal.

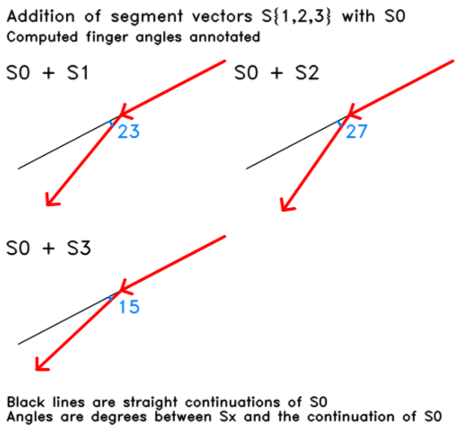

Addition of segment vectors S{1,2,3} with S0
Computed finger angles annotated
S0 + S1
23
S0 + S2
27
S0 + S3
15
Black lines are straight continuations of S0
Angles are degrees between Sx and the continuation of S0

Fig. 6. Vector addition diagrams showing S_1, S_2, and S_3 added to S_0 in the context of the thumb from Fig. 5. The blue angles shown are the finger angle candidates. In this case, 27° is the greatest angle and so is chosen as the finger angle F_0.

The method models each finger as four 'segments' (unit vectors) with an angle from the horizontal as shown in Fig. 5. For each finger the acute angle formed by the addition of the first segment to every other segment in the finger is computed and the difference between 180° and that angle is taken as a candidate finger angle, shown diagrammatically in Fig. 6. The finger angle is the candidate with the greatest magnitude.

Formally and using Radians, the computation follows like so:

For each pair $(P_i,\ P_{i+1})$ (where $i \in [0, 3]$ is an integer) of consecutive keypoints in a finger, the angle of the segment S_i is defined as:

$$S_i = atan2(P_{i+1_y} - P_{i_y}, \qquad P_{i+1_x} - P_{i_x})$$

The angle of the finger F_i is then taken to be:

$$F_i = \max\left\{ \left| \left| (\pi - S_0) + S_j \right| - \pi \right| \,\middle|\, j = 1..3 \right\}$$

Finally, each finger is assigned a hand-tuned finger angle threshold. The finger is counted as 'straight' if the finger angle is at least the threshold. The number produced by the FAT method is the number of fingers that are 'straight'. The thresholds were tuned manually by experimentation, then later tested using the dataset and compared for accuracy against other classification methods.

2) *SVM models*

The first two SVM models differ only in the kernel chosen and so are discussed simultaneously in this section.

The models are trained using identical datasets pre-processed as described above. Each hand skeleton produced by MediaPipe Hands is vectorized by flattening the 21 triplets of (x, y, z) coordinates describing each keypoint in the skeleton into a single 63-dimensional vector. These vectors are then used as input for the SVM models.

For training, all vectors from hand skeletons, and their labels in the training dataset are combined to form a data matrix of size (1162, 63) and a label matrix of size (1162), both are then passed into a scaler which centers and transforms each vector in the data so that it approximates a normal distribution. It is then passed into an SVM classifier using one-vs-rest decision function with the given kernel type.

For classification, a single hand-vector is passed through the scaler and into the train SVM classifier, and an integer corresponding to the predicted class is returned.

3) *Fusion of FAT and SVM models*

For this approach, the SVM model is given a 5-dimensional vector containing the angles of each finger as computed by the FAT method. 1162 such vectors are used for training, and individual vectors are used for classification. As with the SVM models described above a scaler is used to pre-process the data, and the SVM model returns an integer corresponding to the predicted class.

Both the linear and RBF kernels are evaluated for this approach.

C. Debouncing and repeating

The ZYFGAS debouncing method is implemented in Python 3.10 as described in [7]. A 'debouncing tick' occurs for each frame in which at least one hand is detected and classified. The debouncer is given an alphabet of 12 gestures (integers 0..11). Eleven cannot be generated from the classification algorithm but is included for use as a dummy initial value, allowing the user to enter any valid gesture when the application starts.

The debouncer is extended to emit notifications to listeners each time the current gesture changes to a known state (transitions to the 'unknown' state do not trigger notification).

The accuracy of the debouncer is then compared to two alternatives:

1. Raw input – listeners receive an event every time the output from the classifier changes.
2. Time-delay – listeners receive an event a pre-set amount of time (counted in frames) after the output from the classifier changes if and only if the classifier continues emitting that value for the duration of the time. This implementation is based on the debouncer described in [11].

As a standard debouncer evaluation method does not appear to have been described in the literature, a novel method is devised to quantitatively compare the three debouncers:

A short video is labelled manually by removing short errors from the raw output, and the error count and input delay of each debouncer was computed. The error count is taken to be the Levenshtein Distance between the string of events emitted by the debouncer and the events defined by the label. The input delay is computed by subtracting the index of the frame during which the debouncer emitted a notification of change and the index of the frame when the label changed. To address cases when the debouncer emits too many or too few notifications, the edits proposed by the Levenshtein algorithm to make the debouncer notification string match the labels, are applied to the notification frame index array before comparison to the label notification frame index array. This ensures that cases where the debouncer emits an extraneous notification or fails to emit a notification do not affect the input delay score.

Relevant properties are varied for the ZYFGAS and time-delay debouncers and their resulting error counts and input delays are recorded for comparison. The time-delay debouncer's time delay varies from 1..20 frames, and the ZYFGAS debouncer is evaluated for each (activation threshold, max value) parameter pair in the set $\{(t, m) \mid t \in 1..20,\ m \in t..30\}$.

Finally, another novel extension to the debouncer provides a repeater which records the datetime of the last notification that was sent to listeners (the combination of the debouncer and this extension will be referred to as ZYFGAS-R). For each debouncing tick, if the current gesture is not 'unknown' and its counter remains above the activation threshold, and at least a pre-defined amount of time has passed since the last notification, the listeners are re-notified of the current gesture.

D. *Throughput*

The suitability of the proposed method for real-time applications requires that it can process frames at a rate fast enough to keep up with the video camera. To evaluate this, the frames-per-second throughput of the proposed method is computed by:

1. Recording a 62-second video of hand gestures and loading its frames entirely into memory.
2. Repeatedly capturing the time in seconds required to execute a pipeline of MediaPipe Hands, SVM-FA classification, and ZYFGAS-R debouncing over the buffer of frames (the time taken to initialize these components was excluded as it would be expected to occur only during application startup in real applications).

After repeating the second step five times, the average of the times is taken and the number of frames in the video is divided by the average time taken, producing the maximum throughput of the solution, measured in frames-per-second.

IV. Results and Discussion

The above method was implemented on a PC running Window 10 Pro, with an AMD Ryzen 9 5900X 12-Core Processor (3.7 GHz), 16GB of DDR4 RAM, and using a Microsoft LifeCam HD-3000 webcam producing 1290x720px video at 30fps as input. JetBrains PyCharm Professional 2022.1 IDE was used, and the following key Python 3.10.4 packages were installed in a miniconda environment:

- Numpy 1.22.3 [16]
- MediaPipe 0.8.9.1
- Scikit learn 1.0.2 [17]
- OpenCV 4.5.5.64 [18]

A. *Classification accuracy*

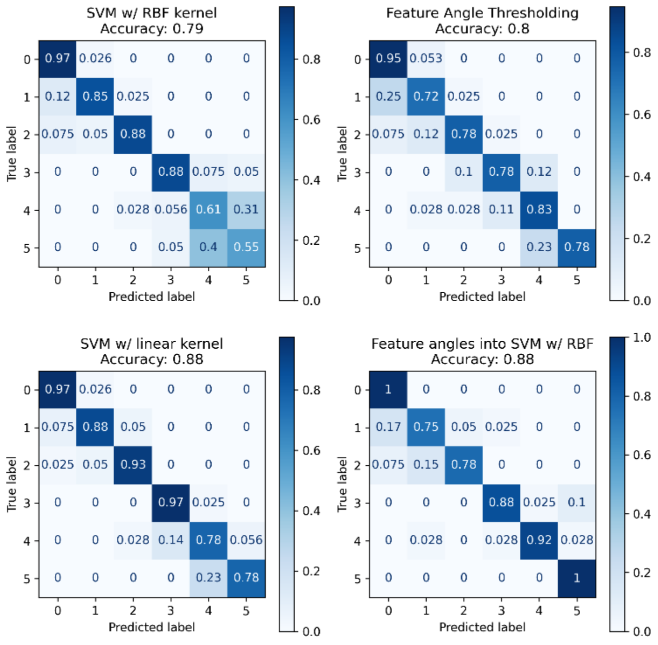


Fig. 7. Confusion matrices for each of the investigated classification approaches in order of increasing accuracy. *Feature angles into SVM w/ linear* is omitted as it performed similarly to *Feature angles into SVM w/ RBF* except with 86% accuracy.

After training, the linear and RBF SVM models and the FAT method were tested against the same dataset of 36 - 40 images per label (MediaPipe Hands did not detect any hands in 6 of the images in the dataset, so these images were excluded). As shown in Fig. 7, the SVM with a linear kernel and SVM-FA with an RBF kernel were the most accurate overall at 88%. This is 10 percentage points lower in accuracy than the SVM models with the same kernels proposed in [5], likely due to a difference in the quality of the training and testing data as approximately the same number of samples for each label was used in this study. It is also significantly lower than the 94.33% accuracy reported for touch-screen touchpads in [12], though the comparison is not entirely fair as these accuracy results measure classification accuracy on individual frames rather than the accuracy of people using the solution.

All models except SVM-FA had significant difficulty distinguishing between the gestures for '4' and '5', and all models had difficulty distinguishing between '0', '1', and '2'.

This may be resolved by improvements in size, diversity, and quality of training data or may reveal a deficiency in MediaPipe Hands' skeletonization when most fingers are curled into a fist.

B. *Debouncing and repeating*

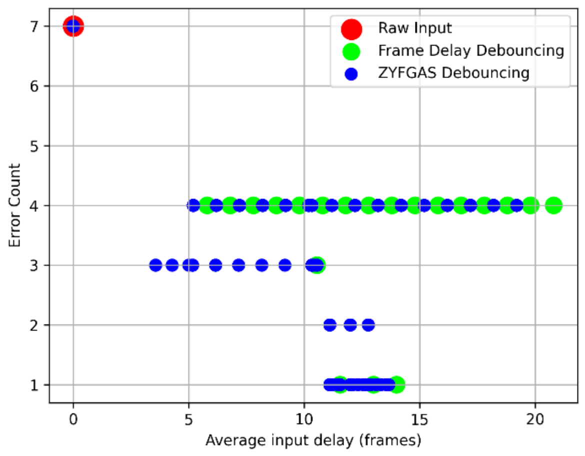


Fig. 8. Scatter plot comparing the error count and average input delays of each configuration of each debouncer type. Point size has no significance.

Parameter combinations of the ZYFGAS debouncer were compared against parameters of the time-delay debouncer and to raw (or un-debounced) data as described above. An overview of the results, displayed in Fig. 8, shows a general trend of error count decreasing with increasing average input delay (although there is a grouping of poorly performing Frame Delay and ZYFGAS debouncers across a large range of input delays at error count = 4). The raw input method has the lowest average input delay of 0 frames (by definition), but the greatest error count at 7 errors. ZYFGAS can also be seen on the far left of each error-count grouping, meaning that it can provide the best error prevention capability for the least input delay.

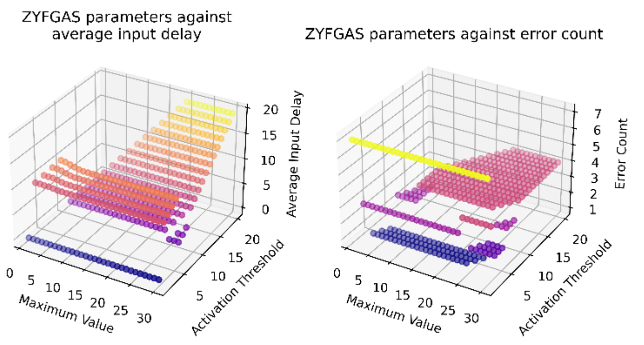


Fig. 9. Scatter plots visualizing the effects of ZYFGAS parameters on average input delay and error count. Point colors show dependent variable value increasing from purple to yellow following the Matplotlib 'plasma' color map[1].

As shown in Fig. 9, ZYFGAS' average input delay is linearly correlated with the activation threshold, while there is no clear relationship between ZYFGAS properties and the error count. The presence of lines along the maximum value axis implies that the activation threshold is a stronger error count indicator than the maximum value. Overall, error count appears to be minimized when the activation threshold and maximum value properties are in specific ranges, these are likely to vary by application, dependent on the level of noise in the input and the application's tolerance to error output and input delay. For this application, ZYFGAS appears to perform best against both metrics with an activation threshold near 10, and a maximum value near 25.

Due to a dearth of publications in this space, it is impossible to quantitatively compare debouncer results to debouncer implementations in other publications.

C. *Throughput*

Across the five tests, the solution was found to have a mean throughput of 31.7fps. As a result, the proposed method can feasibly be used in real-time applications. While the number is substantially smaller than the throughput of over 71fps reported by Osipov and Ostanin in [5], they recorded only the throughput of their classifier, while the 31.7 fps number reported here captures the entire process, including hand detection and skeletonization, classification, and debouncing.

V. Conclusion

These result show that the proposed method can be used to produce accurate number input into computer systems from real-time static hand gestures. The mean throughput of 31.76fps, and classification accuracy of 88% (likely higher with the addition of a ZYFGAS debouncer) means that the method can provide a useful basis on which real-time computer systems requiring accurate and reliable gesture number input can be constructed, though no such system is evaluated in this paper.

The best gesture classification methods attempted had an accuracy of 88%. This is less than the accuracy of 98.74% reported in [5], but likely sufficient to provide reliable static gesture classifications when appropriately debounced by the ZYFGAS method. When properly configured, the ZYFGAS debouncer was found to be more effective than the time-delay debouncer and no debouncer alternatives, emitting the fewest erroneous notifications with the least average input delay.

A. *Future research*

1) *Gesture classification*

All classification methods had significantly lower accuracy than some comparable approaches in literature [5] [19]. Future work could likely improve these results by increasing the size, diversity, and quality of the training dataset used, and by altering values of the SVMs (for example the C parameter can be adjusted to increase accuracy depending on the noise level of the input [6], although such accuracy improvements can become negligible after a point [20]). Training data including negative cases could also be used to evaluate and improve the classifiers' handling of unknown inputs.

The band of errors directly neighboring correct predictions in the confusion matrix for FAT shows that the thresholds used for each finger could be fine-tuned in future studies to produce better accuracy.

Future studies could also evaluate whether FAT and SVM-FA become more accurate if the feature angles are computed in three dimensions.

Implicit in the design of the FAT method is that it recognizes a superset of the gestures supported by the SVM models since it simply counts the number of fingers that are

[1] See: http://matplotlib.org/3.5.0/tutorials/colors/colormaps.html#sequential

deemed to be straight while the SVM models are sensitive only to specific fingers being straight or bent. Future studies could improve on the specificity of FAT's classification without machine learning by mapping specific straight finger combinations to gesture outputs.

2) *Debouncing & repeating*

Future work can improve the repeating model to make it more 'natural' for human input. One area for improvement would be for user interaction to provide separation to denote repetitions of numbers, rather than relying on a purely time-based repeating approach. This would allow people that are beginning with the solution to gesture more slowly without causing accidental repetitions while still enabling advanced users to repeat quickly.

Another issue is that the ZYFGAS debouncer (and derivatives) is designed for continuous streams of data. However, with the proposed solution it only receives data when hands are in frame. This leads to biases from previous interactions affecting the next interaction, even if they are spaced apart in time. With the ZYFGAS-R method, erroneous repetitions often occur in the first frame of the next interaction, though this could be remedied by altering the repeater to count frames instead of comparing time. Future work could eliminate or mitigate this problem across the rest of ZYFGAS by adding a 'controller' that manages the ZYFGAS[-R] Debouncer, resetting it to initial values after a pre-set amount of time without input.

In future studies, it would also be useful to further refine debouncer evaluation methods and to evaluate debouncer implementations against different labelled video of greater length and gesture complexity and speed. This could allow, for example, quantification and comparison of the speed at which different debouncers can recover from long streams of noise, and how well they perform against streams of different noise levels.

It would also be informative for the accuracy of people performing number entry with the solution to be measured, so a fair comparison can be made against existing HCI methods such as the touchpads studied in [12].

3) *Throughput*

Future studies could likely optimize the implementation of the solution to improve throughput. For example, it may be that the throughput of the hand detection and skeletonization, and the SVM classification steps can be increased significantly with GPU acceleration.

It would also be informative for future studies to evaluate the throughput of the proposed method on different devices, such as mobile phones or Raspberry Pi's, to better understand the feasibility of the method for low-powered devices.

REFERENCES

[1] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review,* vol. 43, no. 1, pp. 1-54, 2015/01/01 2015, doi: 10.1007/s10462-012-9356-9.

[2] F. Zhang *et al.*, "MediaPipe Hands: On-device Real-time Hand Tracking," p. arXiv:2006.10214. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2020arXiv200610214Z

[3] J. Sanalohit and T. Katanyukul, "TFS Recognition: Investigating MPH]{Thai Finger Spelling Recognition: Investigating MediaPipe Hands Potentials," p. arXiv:2201.03170. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2022arXiv220103170S

[4] G. Sung *et al.*, "On-device Real-time Hand Gesture Recognition," p. arXiv:2111.00038. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2021arXiv211100038S

[5] A. Osipov and M. Ostanin, "Real-time static custom gestures recognition based on skeleton hand," in *2021 International Conference "Nonlinearity, Information and Robotics" (NIR)*, 26-29 Aug. 2021 2021, pp. 1-4, doi: 10.1109/NIR52917.2021.9665809.

[6] s.-l. developers. "1.4. Support Vector Machines." https://scikit-learn.org/stable/modules/svm.html (accessed 30/04/2022, 2022).

[7] S. Young, B. Stephens-Fripp, A. Gillett, H. Zhou, and G. Alici, "Pattern Recognition for Prosthetic Hand User's Intentions using EMG Data and Machine Learning Techniques," in *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 8-12 July 2019 2019, pp. 544-550, doi: 10.1109/AIM.2019.8868766.

[8] B. Wu *et al.*, "Tencent ML-Images: A Large-Scale Multi-Label Image Database for Visual Representation Learning," *IEEE Access,* vol. 7, pp. 172683-172693, 2019, doi: 10.1109/ACCESS.2019.2956775.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, *ImageNet: a Large-Scale Hierarchical Image Database*. 2009, pp. 248-255.

[10] Malk and Vsync. "debouncing - Can someone explain the "debounce" function in Javascript." https://stackoverflow.com/a/24004942 (accessed 30/04/2022, 2022).

[11] L. Contributors. "Lodash Documentation." https://lodash.com/docs/4.17.15#debounce (accessed 03/05/2022, 2022).

[12] M. K. Chung, D. Kim, S. Na, and D. Lee, "Usability evaluation of numeric entry tasks on keypad type and age," *International Journal of Industrial Ergonomics,* vol. 40, no. 1, pp. 97-105, 2010/01/01/ 2010, doi: https://doi.org/10.1016/j.ergon.2009.08.001.

[13] Google. "Hands - mediapipe." GitHub. https://google.github.io/mediapipe/solutions/hands (accessed 27/04/2022.

[14] Z. Ren, J. Yuan, J. Meng, and Z. Zhang, "Robust Part-Based Hand Gesture Recognition Using Kinect Sensor," *IEEE Transactions on Multimedia,* vol. 15, no. 5, pp. 1110-1120, 2013, doi: 10.1109/TMM.2013.2246148.

[15] P. Koryakin. *Fingers*. [Online]. Available: https://www.kaggle.com/datasets/koryakinp/fingers

[16] C. R. Harris *et al.*, "Array programming with NumPy," *Nature,* vol. 585, no. 7825, pp. 357-362, 2020/09/01 2020, doi: 10.1038/s41586-020-2649-2.

[17] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825-2830, 2011

2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html.

[18] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools,* 2000.

[19] M. H. Ismail, S. A. Dawwd, and F. H. Ali, "Static hand gesture recognition of Arabic sign language by using deep CNNs," *Indonesian Journal of Electrical Engineering and Computer Science*, 2021.

[20] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A Library for Large Linear Classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, 2008.