Gesture Controlled Virtual Mouse Using Colour Caps

Prithvi J, S Shree Lakshmi , Suraj Nair and Sohan R Kumar
Department of Computer Science And Engineering
B.M.S. College of Engineering
Bengaluru, Karnataka, India
{prithvijayadev27, nurashree24, surajnair9912, sohanrk.2001}@gmail.com

Ms. Sunayana S
Department of Computer Science And Engineering
Visveswaraya Technological University, Belgaum
Bengaluru, Karnataka, India
sunayanas.cse@bmsce.ac.in

***Abstract*— This research paper presents a mouse simulation system that enables users to control the computer's cursor and perform various mouse functions using hand movements and gestures. The system utilizes a camera to capture the user's video, and through image processing techniques, it extracts the positions of three colored finger tips. The relative positions of these finger tips are calculated using the method of moments, and based on the results, the system determines the appropriate mouse action to be performed, such as moving the cursor, left-clicking, right-clicking, dragging, selecting, or scrolling. The implementation is based on Python programming language, leveraging the cross-platform image processing module OpenCV and the Python-specific library PyAutoGUI. To run the system, a computer needs to have a webcam (commonly found in most laptops) and the required packages, including Python 2.7 interpreter, OpenCV, Numpy, and PyAutoGUI. By eliminating the need to physically reach for the mouse, this system offers convenient control of the computer's mouse interface from any location. The experimental results demonstrate the effectiveness and feasibility of the proposed mouse simulation system.**

***Index Terms*— Mouse simulation system, Hand gestures, Cursor control, Image processing, Method of moments, Python 2.7 interpreter, OpenCV, Computer vision, Gesture recognition, Numpy, PyAutoGUI, User interface, Computer-human interaction, Convenience Control systems, Webcam processing, Finger tracking, Gesture-based interaction, Human-computer interface.**

I. INTRODUCTION

In the realm of human-computer interaction, the mouse has long been a ubiquitous input device, enabling users to interact with computers through cursor control and various mouse functions. However, the conventional mouse requires physical proximity to the computer, often necessitating users to interrupt their activities to access its functionalities. To address this limitation, we propose a novel solution: a mouse simulation system that allows users to control the computer's mouse through hand movements and gestures, thereby eliminating the need for direct physical contact with the mouse. The objective of this project is to provide users with an intuitive and convenient method of interacting with the computer by leveraging hand gestures. Through the integration of a camera, the system captures the user's video feed, enabling real-time interpretation of their hand gestures. By moving their hand and fingers, users can perform a wide range of mouse functions, including cursor movement, left and right clicks, dragging, selection, and scrolling. These hand gestures are predefined and are associated with specific actions, enhancing user control and improving the overall user experience.

The core of this project is a program developed in Python, which incorporates image processing techniques to extract relevant data from the video feed and subsequently maps it to the mouse interface of the computer. The program utilizes the OpenCV cross-platform image processing module to process the video captured by the camera. Additionally, the Python-specific library PyAutoGUI is employed to implement the desired mouse actions based on the recognized hand gestures. To run the system, a computer needs to be pre-equipped with the necessary software packages, including the Python 2.7 interpreter, OpenCV, Numpy, and PyAutoGUI. In most cases, modern laptops are already equipped with a built-in webcam, which serves as the video input device for capturing the user's hand gestures. By leveraging the power of image processing algorithms, the system focuses on extracting the centers of the three colored finger tips, employing the method of moments for accurate position calculations. Based on the relative positions of these finger tips, the system determines the appropriate mouse action to be performed.

The proposed mouse simulation system offers numerous practical applications and benefits across various domains. For instance, in a social setting, such as watching a movie with friends on a laptop, the system eliminates the need for someone to physically pause the movie when they receive a call. Similarly, during presentations using projectors, switching between applications can be achieved seamlessly without the presenter having to move

across the stage to access the mouse. By enabling users to control the mouse from any location, the system greatly enhances convenience, productivity, and flexibility in diverse scenarios. Furthermore, the system's versatility extends beyond professional settings. Users can leverage its capabilities for recreational purposes, such as gaming or navigating media players, by using hand gestures to control the mouse. This not only enhances the immersive experience but also offers a novel and engaging approach to interaction with computers. This research project presents a mouse simulation system that revolutionizes human-computer interaction by enabling users to control the computer's mouse through hand gestures. By harnessing image processing techniques and leveraging the power of Python programming, the system provides an intuitive and convenient alternative to traditional mouse control. The system's applications span professional presentations and recreational activities, offering users enhanced control and flexibility in their computer interactions.

II. PROBLEM STATEMENT

Conventional mouse control restricts user mobility and convenience. This project introduces a mouse simulation system that interprets hand gestures to perform mouse actions. By leveraging image processing and computer vision techniques, users can seamlessly navigate computer interfaces and perform tasks without physical proximity to the mouse. This research aims to develop an alternative method for controlling the mouse pointer and its functions, such as left click, right click, scrolling, selection, and cursor movement. The objective is to enable users to interact with computers from a distance without physically touching the mouse, reducing hardware requirements. The proposed solution utilizes a webcam for real-time video capture. The proposed system benefits professional presentations, allowing users to switch applications and control multimedia content from a distance. In recreational activities like gaming, hand gestures provide an immersive interaction. This project revolutionizes human-computer interaction, enhancing productivity and user experience across various domains.

III. LITERATURE SURVEY

A. Background

Gesture-based mouse control using computer vision has been a topic of interest for researchers for a long time. Various methods have been proposed for gesture recognition, but in this paper, the authors have proposed a method based on colour cap. This section presents the foundational background for the literature survey, focusing on an algorithmic design that encompasses finger detection, gesture identification, and mouse control operations. The research encompasses three distinct subsections: color identification, gesture identification, and virtual mouse control, complemented by an overview of the overall system.For algorithm implementation, Python has been selected as the programming language of choice. The OpenCV library is leveraged for robust image processing, while the pyautogui library facilitates mouse control. The algorithm employs two distinct methodologies for mouse control implementation: color caps and bare hand gesture recognition.

The overall system architecture is divided into two primary methods: "fingertip detection" using colored caps and "gesture recognition." The system encompasses video interfacing, image processing utilizing background subtraction techniques to eliminate stationary objects and focus solely on the foreground. Fingertip detection involves color identification, circle identification, and finger estimation. Gesture recognition encompasses skin color identification, contour detection, convex hull formation, and subsequent gesture estimation. The respective mouse operations are performed based on the recognized gestures.

By harnessing these advanced techniques, the proposed system presents a novel solution for accurate finger detection, precise gesture identification, and intuitive mouse control. These advancements significantly enhance the user experience and pave the way for further advancements in human-computer interaction.

B. Literature Survey

In the previously proposed system by Vantukala VishnuTeja Reddy and team's "Virtual Mouse Control Using Colored Finger Tips and Hand Gesture Recognition" research paper published in IEEE-HYDCON (2020), color detection is done by virtual mouse control using finger tip identification and hand gesture recognition is proposed. This section presents an extensive review of the existing literature pertaining to real-time finger gesture detection using coloured caps. The research in this field encompasses four essential stages: video extraction, background subtraction, color identification, and circle identification.

A. Video Extraction: The initial stage involves capturing video using a webcam and extracting a frame for subsequent processing. The OpenCV library's Video Capture function facilitates this process, yielding a 512x512 pixel-sized frame. The frame, represented in the BGR (Blue, Green, Red) format, is then converted to the HSV (Hue, Saturation, Value) color space using the BGR to HSV conversion function in OpenCV.

B. Background Subtraction: To isolate the hand with color caps from the background, a robust background subtraction algorithm is employed. This algorithm effectively distinguishes between moving foreground objects, particularly color caps, and stationary background objects, resulting in a precise foreground mask. By comparing the current frame image with a reference frame captured 21 frames prior, the algorithm identifies and neglects the background objects that exhibit no motion for a significant duration. Moreover, shadows of the hand in the frame image are efficiently eliminated using the "detect shadows" command in the background subtraction algorithm. The resulting masked grayscale image represents

the foreground objects, and a bitwise 'and' operation with the original frame image generates the background-subtracted image.

C. Color Identification: The background-subtracted image is subsequently analyzed to identify color caps of interest. By defining the lower and upper HSV color ranges for each specific color (e.g., blue, red, green) and employing the "in range" function in OpenCV, a grayscale image containing only the objects of the desired color is obtained. The bitwise 'and' operation between this grayscale image and the background-subtracted frame produces an image with exclusively the objects of interest, facilitating precise color cap identification.

D. Circle Identification: Given that the color caps employed in this project possess a circular shape, the circle identification stage aims to detect circular objects corresponding to the color caps. The OpenCV library's Hough Circles function is employed to identify circles based on their radii and center coordinates. By specifying an appropriate range of radii (e.g., 50 to 200 pixels, corresponding to a distance of 0.5 m to 1 m from the camera), the function accurately identifies the circular color caps. The resulting information allows for the visualization of the identified circles by plotting them using their respective coordinates.

Through an extensive review of the existing literature, the aforementioned stages have been widely explored and adopted in various finger gesture detection systems. The subsequent sections of this paper elaborate on the implementation details and evaluation of these stages, culminating in the development of an efficient finger gesture recognition system.

IV. METHODOLOGY

This section presents a comprehensive methodology for reducing the impact of illumination and achieving robust hand gesture recognition. The proposed approach encompasses several key steps, including chrominance color space conversion, color detection, background subtraction, noise removal, Gaussian filtering, edge detection, hand contour extraction, gesture recognition, and cursor control.

A. Chrominance Color Space Conversion: To mitigate the adverse effects of illumination variations, the input image is transformed into the chrominance color space, which exhibits reduced sensitivity to illumination changes. Among various color spaces, HSV (Hue, Saturation, Value) is chosen as the most suitable color space for accurate skin detection.

B. Color Detection: The subsequent stage involves differentiating selected color pixels from non-color pixels in the image. Through a robust color detection method, the desired color pixels are identified, enabling the isolation of the hand and distinguishing it from the background. This step plays a pivotal role in subsequent hand gesture recognition.

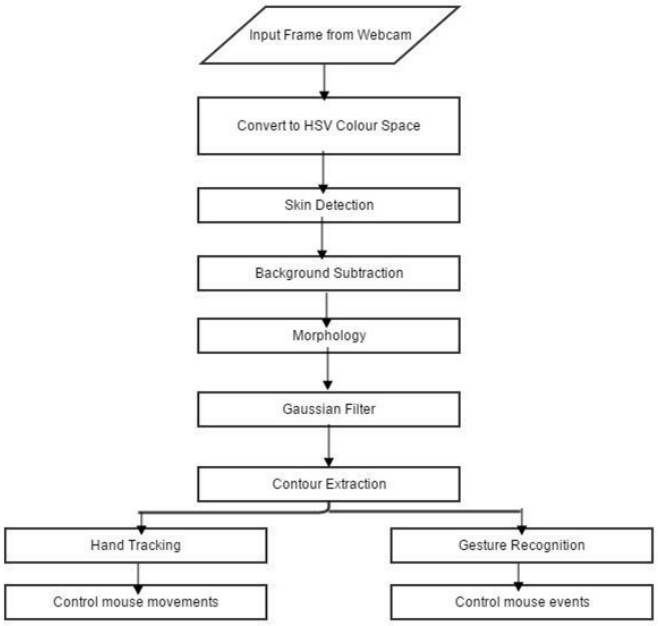


Fig. 1. Architecture

C. Background Subtraction: To further refine the hand region, background subtraction is applied, effectively eliminating facial features and other skin-colored objects present in the background. This process enhances the accuracy and reliability of subsequent hand tracking and gesture recognition.

D. Noise Removal: To mitigate the adverse effects of noise, a morphological opening operation, comprising erosion followed by dilation, is employed. This operation efficiently removes unwanted artifacts and enhances the clarity of the hand region, contributing to precise hand tracking and gesture recognition.

E. Gaussian Filtering: A Gaussian filter is then applied to the preprocessed image, facilitating effective smoothing and enhancing edge detection. This step aids in obtaining well-defined hand contours, which serve as crucial features for subsequent hand tracking and gesture recognition.

F. Edge Detection: The image undergoes edge detection, enabling the extraction of the hand contour within the frame. Through this process, the boundaries of the hand region are accurately delineated, forming the basis for robust hand tracking and gesture recognition.

G. Hand Tracking: Utilizing the obtained hand contour, the tip of the index finger is located, enabling precise hand tracking and control of mouse movements. This tracking mechanism facilitates the translation of hand gestures into corresponding cursor movements, enhancing user interaction.

H. Gesture Recognition: In addition to hand tracking, the hand contour is leveraged for gesture recognition. By analyzing the shape and spatial configuration of the hand contour, distinct gestures can be identified and associated with specific commands or actions. This enables intuitive

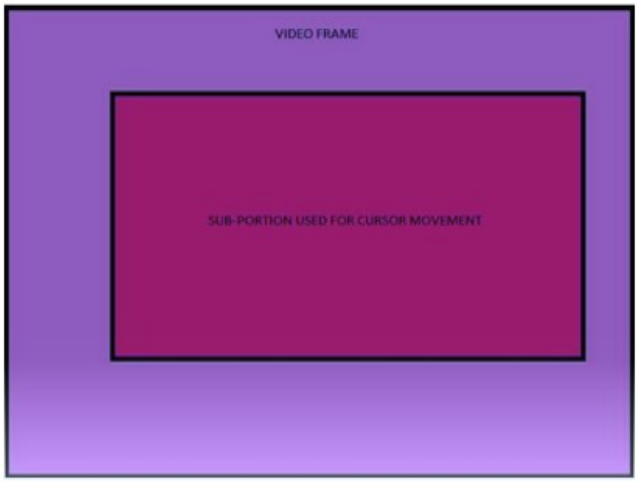
and efficient interaction with the system.

I. Cursor Control: The culmination of the methodology involves the precise control of the cursor based on the tracked hand movements. The system utilizes the extracted hand information and corresponding gestures to perform cursor control operations, such as cursor movement, left-click, right-click, and other mouse functionalities.

In summary, the proposed methodology comprises four main components: color detection, color contour extraction, hand tracking, and gesture recognition. Through the integration of these components, the system achieves accurate hand gesture recognition and precise cursor control, enhancing the overall user experience and interaction with the computer system.

V. PROPOSED SYSTEM

This section presents an in-depth implementation of the hand gesture recognition system, focusing on the conversion of captured video into HSV format, color calibration, fingertip extraction, noise removal, center localization, cursor positioning, and action determination.

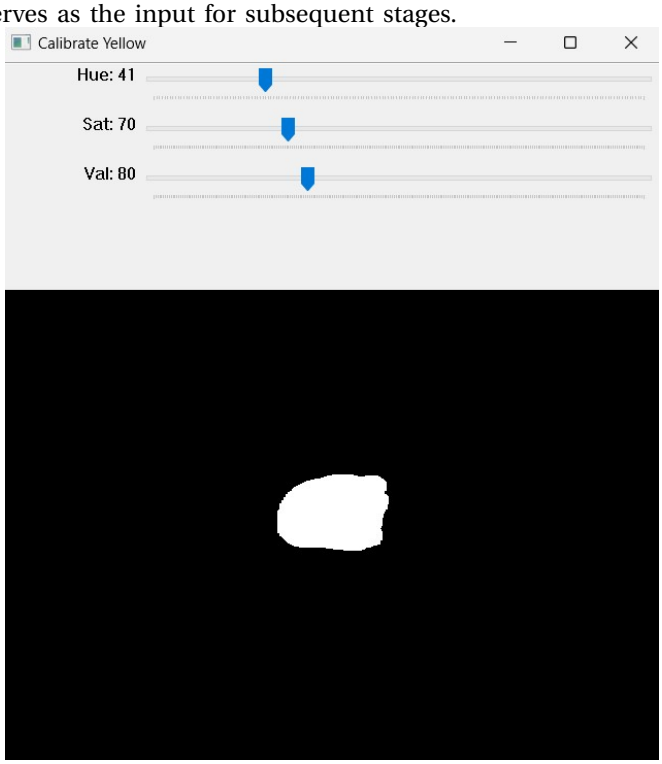


The following steps outline the implementation process:

A. Video Conversion and Color Calibration: The initial step involves converting the captured video into HSV format, facilitating accurate color analysis. To ensure optimal performance, the user has the option to individually calibrate the color ranges for three fingers. This calibration process, carried out at the beginning of the program by invoking the calibrateColor() function, enables personalized color range settings. Alternatively, default settings can be utilized.

B. Fingertip Extraction: Based on the color calibrations, the program extracts the three fingertips from the video using the cv2.inRange() function. By applying appropriate color filters, only the desired fingertips are isolated, forming the basis for subsequent processing steps. This filtering mechanism ensures the extraction of relevant hand features while mitigating the influence of background elements.

C. Noise Removal: To enhance the quality of the video feed and reduce unwanted noise, a two-step morphological operation comprising erosion and dilation is employed. This process effectively eliminates artifacts and enhances the clarity of the extracted hand regions. The resultant noise-filtered image, referred to as the mask, serves as the input for subsequent stages.



D. Center Localization: The next stage involves locating the centers of each of the three fingertips. This process encompasses the following steps:

1. Finding Contours: Contours relevant to the color range of each fingertip are identified in the mask image. This step isolates the regions of interest and forms the foundation for subsequent center localization.

2. Area Filtering: To discard contours associated with irrelevant areas, an area filter is applied. Only contours with significant area are retained, ensuring the focus on the fingertips and discarding spurious elements.

3. Center Calculation: Among the remaining contours, the largest contour is selected, and the method of moments is applied to determine its center. This centroid calculation yields precise fingertip positions, facilitating subsequent cursor control and gesture recognition.

E. Cursor Positioning: The positioning of the cursor on the screen is influenced by several factors, and the following techniques are employed to optimize user comfort and accuracy:

1. Frame Mapping: Considering the resolution of the captured video (typically 640x480 pixels) and the target display screen (1920x1080 pixels), a linear mapping strategy is employed. To accommodate right-handed users and reduce strain on the wrist, a rectangular sub-portion biased towards the right and upper regions of the frame is considered. This sub-portion, measuring 480x270 pixels,

is then linearly mapped to the screen with a scaling factor of 4, ensuring improved usability and ergonomics.

2. Shake Reduction: Vibrations in the hand and noise captured by the webcam can lead to cursor instability. To address this issue, a differential position allocation approach is implemented. By comparing the new center with the previous cursor position, the system determines the magnitude of movement. If the difference is below 5 pixels, it is primarily attributed to noise, and the new cursor position is biased towards the previous position. However, if the difference exceeds a threshold, it signifies intentional movement, and the new cursor position is set close to the new center. This strategy minimizes cursor shakiness and enhances positioning accuracy.

F. Action Determination: The three localized centers are utilized to determine the appropriate action to be performed based on their relative positions. This decision-making process is implemented in the chooseAction() function, which analyzes the spatial configuration of the centers and assigns corresponding actions.

G. Action Execution: The performAction() function, utilizing the PyAutoGUI library, executes the determined action.

Here's a pseudocode implementation of a simple CNN algorithm:

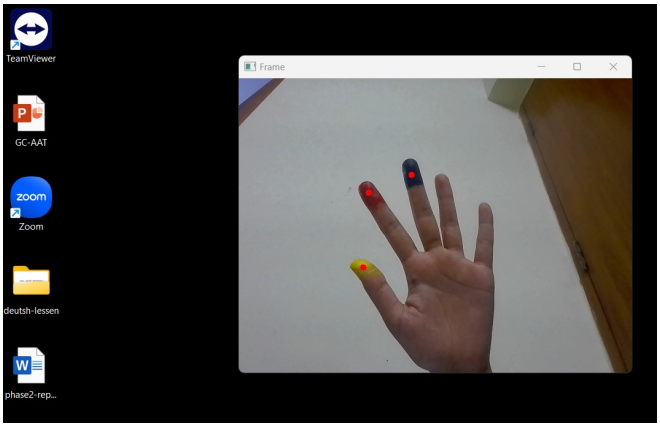
Algorithm 1: Computer vision algorithm for object detection and tracking using OpenCV. It uses color-based segmentation and contour analysis techniques to detect and track objects of different colors in a video stream.

1: Video stream Detected and tracked objects' positions
2: ObjectDetectionAndTrackingvideoStream Initialize color ranges for blue, yellow, and red objects Initialize object centroids and cursor position Start video capture from the video stream
3: **while** *video capture is active* **do**
4: Read a frame from the video stream Apply color-based segmentation to detect blue objects Find contours and analyze them Calculate centroid of blue objects Apply color-based segmentation to detect yellow objects Find contours and analyze them Calculate position of yellow objects Apply color-based segmentation to detect red objects Find contours and analyze them Calculate centroid of red objects Calculate cursor position based on detected object positions Select actions based on the detected objects Perform actions using the cursor Display the processed frame with object positions Check for user input or termination condition
5: Stop video capture

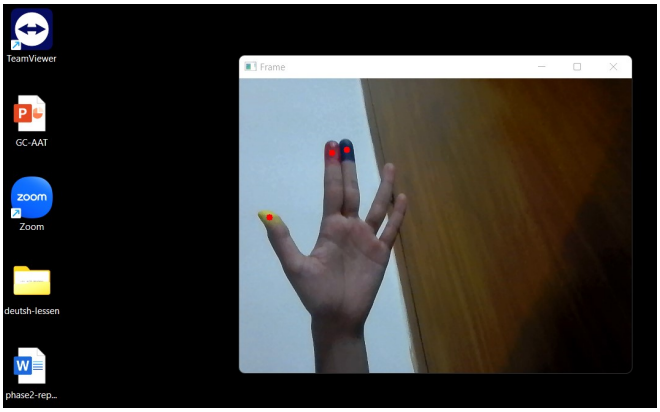
VI. WORK DONE AND RESULTS ANALYSIS

A. Gesture-Controlled Mouse

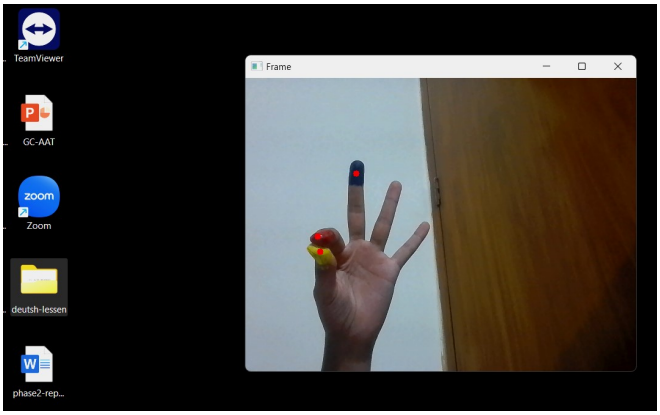
1) Free Cursor Movement: This gesture moves the cursor to the desired location.



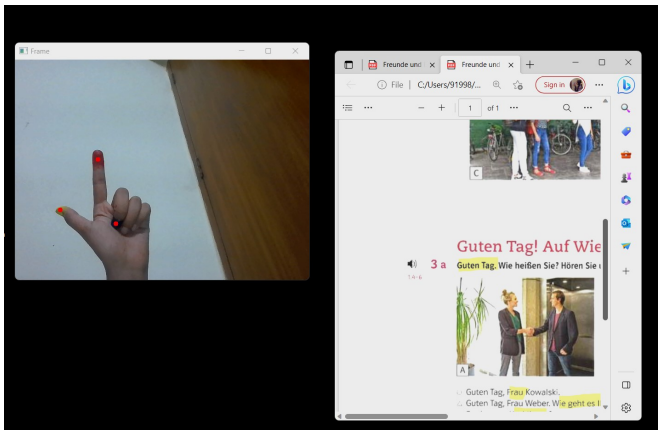
2) Left Click: Gesture for single left click. It Lets the user to perform the left click operation.



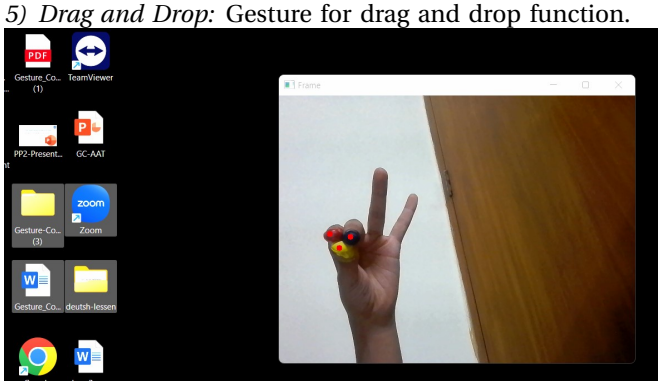
3) Right Click: Gesture for single right click It Lets the user to perform the right click operation.



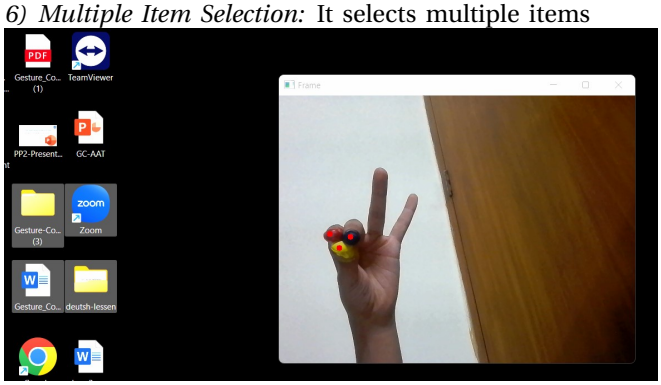
4) Scrolling Up/Down: Dynamic Gestures for scrolling action. The speed of scroll is proportional to the distance moved by pinch gesture from start point. Scrolls are controlled by movements shown respectively.



5) Drag and Drop: Gesture for drag and drop function.



6) Multiple Item Selection: It selects multiple items



VII. PROSPECTIVE IMPLEMENTATION PLAN

Future Implementation Plan for Kalman Filter Integration in Computer Vision Algorithm for Object Detection and Tracking using OpenCV is to enhance the accuracy and robustness of the computer vision algorithm for object detection and tracking using OpenCV using subsequent development strategy the following future implementation plan is proposed to incorporate the Kalman Filter:

Research and Analysis: Conduct a thorough study to understand the theoretical foundations and practical applications of the Kalman Filter in computer vision tasks. 1.Algorithm Modification: Modify the existing computer vision algorithm to integrate the Kalman Filter, ensuring seamless compatibility with OpenCV. 2.Data Fusion and Tracking: Utilize the Kalman Filter to fuse information from multiple sources, improving object detection and tracking accuracy in dynamic environments. 3.Parameter Optimization: Fine-tune the Kalman Filter parameters through systematic experimentation, striking a balance between tracking accuracy and responsiveness. 4.Performance Evaluation: Evaluate the integrated Kalman Filter's effectiveness by comparing performance metrics against the baseline algorithm. 5.Integration and Deployment: Seamlessly integrate the Kalman Filter into the existing computer vision system, ensuring compatibility, maintainability, and scalability.

By integrating the Kalman Filter, we aim to enhance object detection and tracking accuracy, reduce uncertainty, and improve adaptability to challenging environmental conditions. This implementation plan represents a significant step towards advancing computer vision systems in various applications such as surveillance, robotics, and autonomous vehicles.

Algorithm 2: Kalman Filter

[1] Initialize the state vector: $\hat{\mathbf{x}}_0$ Initialize the state covariance matrix: $\mathbf{P}_0$ **while** *New measurements available* **do**

Predict the state estimate: $\hat{\mathbf{x}}_{k|k-1} = \mathbf{A}\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}\mathbf{u}_k$ Predict the error covariance: $\mathbf{P}_{k|k-1} = \mathbf{A}\mathbf{P}_{k-1|k-1}\mathbf{A}^T + \mathbf{Q}$ Calculate the Kalman gain: $\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R})^{-1}$ Update the state estimate: $\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1})$ Update the error covariance: $\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k|k-1}$

VIII. CONCLUSIONS

In conclusion, a Python-based vision-based cursor control system utilizing hand gestures was developed, leveraging the OpenCV library. This system exhibited the capability to track the user's hand and manipulate the cursor accordingly, presenting itself as a potential substitute for the conventional computer mouse. Nonetheless, certain limitations impede its complete replacement of the mouse. Notably, the system's dependence on a well-lit environment restricts its usability in outdoor settings or poorly illuminated conditions commonly encountered during computer usage. To enhance the accuracy of hand gesture recognition, the integration of the Template Matching method with a machine learning classifier could have been pursued, albeit at the expense of additional implementation time. Moreover, the cursor's instability posed challenges for precise movements, necessitating improvements in stability through the incorporation of a Kalman filter. However, time constraints prevented the implementation of this enhancement. Despite these limitations, the system successfully executed various operations utilizing diverse gestures, yielding satisfactory outcomes.

ACKNOWLEDGMENT

We would like to thank Miss Sunayana for her valuable comments suggestions. We would also like to thank the Department of Computer Science and Engineering, B.M.S. College of Engineering for providing us with opportunity to encourage us to write this paper.

REFERENCES

[1] Tsang, W.-W. M., Kong-Pang Pun. (2005). A finger-tracking virtual mouse realized in an embedded system. 2005 International Symposium on Intelligent Signal Processing and Communication Systems. doi:10.1109/ispacs.2005.1595526.

[2] Tsai, T.-H., Huang, C.-C., Zhang, K.-L. (2015). Embedded virtual mouse system by using hand gesture recognition. 2015 IEEE International Conference on Consumer Electronics - Taiwan. doi:10.1109/icce-tw.2015.7216939 10.1109/icce-tw.2015.7216939.

[3] Roh, M.-C., Huh, S.-J., Lee, S.-W. (2009). A Virtual Mouse interface based on Two-layered Bayesian Network. 2009 Workshop on Applications of Computer Vision (WACV). doi:10.1109/wacv.2009.5403082 10.1109/wacv.2009.5403082.

[4] Li Wensheng, Deng Chunjian, Lv Yi. (2010). Implementation of virtual mouse based on machine vision. The 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding. doi:10.1109/icacia.2010.5709921 10.1109/icacia.2010.5709921.

[5] Choi, O., Son, Y.-J., Lim, H., Ahn, S. C. (2018). Co-recognition of multiple fingertips for tabletop human-projector interaction. IEEE Transactions on Multimedia, 1–1. doi:10.1109/tmm.2018.2880608.

[6] Jyothilakshmi P, Rekha, K. R., Nataraj, K. R. (2015). A framework for human- machine interaction using Depthmap and compactness. 2015 International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT). doi:10.1109/erect.2015.7499060.

[7] S. Vasanthagokul, K. Vijaya Guru Kamakshi, Gaurab Mudbhari, T. Chithrakumar, "Virtual Mouse to Enhance User Experience and Increase Accessibility", 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA), pp.1266-1271, 2022, doi:10.1109/ICIRCA54612.2022.9985625.

[8] Shajideen, S. M. S., Preetha, V. H. (2018). Hand Gestures - Virtual Mouse for Human Computer Interaction. 2018 International Conference on Smart Systems and Inventive Technology (ICS-SIT). doi:10.1109/icssit.2018.8748401.

[9] Henzen, A., Nohama, P. (2016). Adaptable virtual keyboard and mouse for people with special needs. 2016 Future Technologies Conference (FTC). doi:10.1109/ftc.2016.7821782.

[10] Reddy, V. V., Dhyanchand, T., Krishna, G. V., Maheshwaram, S. (2020). Virtual Mouse Control Using Colored Finger Tips and Hand Gesture Recognition. 2020 IEEE-HYDCON. doi:10.1109/hydcon48903.2020.9242677 .

[11] Shetty, M., Daniel, C. A., Bhatkar, M. K., Lopes, O. P. (2020). Virtual Mouse Using Object Tracking. 2020 5th International Conference on Communication and Electronics Systems (IC-CES). doi:10.1109/icces48766.2020.9137854.

[12] Xu, G., Wang, Y., Feng, X. (2009). A Robust Low Cost Virtual Mouse Based on Face Tracking. 2009 Chinese Conference on Pattern Recogni-tion. doi:10.1109/ccpr.2009.5344072