# A Game Using Eye, Finger, & Mouse Tracking

Faris Durrani
afarisdurrani@gatech.edu
Georgia Institute of Technology

Rishabh Ghora
rishabhghora@gatech.edu
Georgia Institute of Technology

## ABSTRACT

This paper considers the application of eye movements to user interfaces, mainly as an input to user interfaces allowing for a new premise to control applications with less need to physically touch any buttons or screens. The paper first introduces the topic of eye-tracking and how such an implementation on user interfaces would be beneficial. Next, we review previous literature on similar subjects in order to clarify what research has already developed in this field and how we will use findings from other papers to enhance our system. Finally, we implemented an application that combines eye movements with finger gestures as user inputs to showcase the feasibility of having an application that is concurrently considering both forms of input.

## 1 INTRODUCTION AND MOTIVATION

Since the introduction of direct manipulation by Shneiderman ([28], cited in [26]), the common interface has been based in 2D Graphical User Interface (GUI) with the the button being a fundamental element of interaction. However, the limitations of the button, i.e. the need to move the cursor and press the button especially by people with limited capability whether that may be for medical reasons or high-performance tasks, prove the need to extend the functionality of the button beyond the physical touch. Taking into consideration the increased presence of various forms electronic devices in everyday lives, and the increased communication and social media, it is necessary to use these technology to increase the control modes of our computers whether for patients with neuro-motor disabilities [20], virtual reality systems, or video games (eye tracking 1993). This project aims to explore how eye gaze, blinks, and other nontraditional input methods can enhance the basic concept of a button. Such implementations would add to the multimodality of user interface and allows for more complex instructions from the user, in addition to increasing accessibility to those who have limited neuro-motor functions. We explore this possibility by creating a simple game using inexpensive eye-tracking device and input processing software.

## 2 LITERATURE REVIEW

Research on nontraditional user input methods have been long in progress with the creation of the unique prosthetic device Erica devised by Thomas et al. [11]. Erica works by accepting input directly from the human eye using a camera that detects near-infrared light from the eyes and allowing invoking a corresponding menu option based on the position of where the eye looks at. Similarly, Yamada and Fukuda [35] designed an eye word processor controlled by eye movements by victims of amyotrophic lateral sclerosis (ALS). Their design used photo-transistors and infra-red LED placed opposite the eye, detecting horizontal movements. However, Kaufman et al. [14] argued these methods are too expensive and described instead an inexpensive eye movement controlled user interface based on electro-oculography (EOG) rather than the expensive reflectance based methods. Being inexpensive, the system is applicable for many virtual reality systems and video games, as well as for people with limited muscular control. EOG depends on the corneoretinal potential that creates an electrical field in front of the head, which changes as the eyeballs move. The signal quality of the EOG output, though noisy, could be improved with straightforward signal processing steps and a notch filter. To finally extract the salient events such as eye fixations, saccades, and post-saccadic oscillations (PSO), Goltz et al. [8] have come up with a simple and fast automatic solution for eye-gaze analysis based on supervised neural network learning.

These advances in eye tracking technology enable complementary eye trackers integrated into the default display or attached as peripheral devices ([16]; [4], cited in [26]). For instance, Lupu et al. [21] proposed a new

technique to be used by patients with neuro-motor disabilities to control a web browser by means of eye tracking. However, with slight modifications, the authors noted this can be used by healthy subjects as well. The infrared web camera acquires eye images and sends them to the computer where these are processed using the pupil detection algorithm implemented to determine the eye gaze input (called the *eye tracking mouse*). In order to provide the low running time to obtain the stable cursor movement without lag between the gaze direction and cursor movement, the first two out of four levels of the application were written in C++, with the rest in .NET. The application determines the cursor position variation within a radius, and after the dwell time has passed, if the cursor has not moved from that radius, the application generates a click event. Additionally, a matrix is used to implement a grid for fixed snap points, or adaptive snap points. This type of adaptive grid offers easier element selection for the user although it requires additional computing and instructions [17]. Our project will include these aspects to aid in user control and provide a smooth transition when trying to select certain buttons.

Additionally, Rivu et al. [26] proposed the *GazeButton* which takes the advantage of a physical button to aid in selection of items on the screen. The concept extends the default button mode with advanced gaze-based interactions for custom-made text reading, writing, and editing on a multitouch tablet device. Where the default button enables an atomic action, adding gaze allows the interaction to be more expressive by taking into account where the user looks while issuing the manual input ([24], cited in [26]). The GazeButton provides three input states in addition to the default interaction state, where the user interface (UI) is controlled solely by normal touch mode. For example, if the user is looking at a text field, physically pressing the button places the text cursor at the visual position. If the user is looking at a key on the keyboard, the user would type that key. If the user looks and touches the same physical button, the system triggers a distinct action, e.g., provide a system-level menu. This addition of a physical button minimises the learning curve and improves the grip and reach issue, i.e. the hand holding the tablet limiting the interaction. Indeed, gaze and touch interactions have been investigated by Stellmach and Dachselt [29], who looked at how gaze can be used as an interaction technique for mobile devices to zoom and pan while browsing images on a large screen. Turner et al. also applied gaze and touch to the context of transferring content between a local touchscreen and remote display ([32], [33], [34]). From these, we see the advantages of adding a touch component to improve our project's future accessibility.

Further, research in eye typing or gaze typing, first introduced in the 1970s for disabled people ([22], cited in [26]) have made typing using eye movements fast and accurate [26]. Dasher [31] is one of the fastest eye-typing applications today, operated by continuous gaze pointing gestures between letters. EyeSwipe is another example of eye typing that works by scanning the gaze line [19]. In addition, dwell-based gaze interaction software like pEyeWrite that uses a pie menu shape as a text pad has also found its way to eye-typing [10].

There are other nontraditional input methods as well. For example, using an immersive visual environment in combination with a mobile eye tracking system, proposed by Schrom-Feiertag et al. [27]. The authors argued that body orientation and movement have a strong impact on eye tracking patterns in spatial navigation tasks and hence, uses a mobile eye tracking device. There is also the detecting of voluntary blinks and interprets them as control commands [18], moving the tongue to push buttons on the screen (mouth screen reading) [21], and of course, voice commands like one proposed by Ferracani et al. [7] for immersive experiences in augmented museums.

The first aspect of our project relies on developing a program that will be able to track a user's eye movements. This will be done by developing a state of the art machine learning model which will essentially be a neural network built through known computer vision techniques that will be able to identify human eyes and track their movement.

There has been a lot of research in the world of object detection and more particularly how deep learning can assist with the task of object detection. In our case in particular we are less interested of identifying objects such as a dog in a picture but rather need to identify the bounding boxes of human eyes in order to track their movement. We also want to accomplish this real time with the Camera rather than a prerecorded video. Zhao et al. compared deep learning architectures to more traditional object detection algorithms before the rise of deep learning and, in the case of face detection, observed that most of the classic methods perform with

similar results and are outperformed by CNN based methods by a significant margin [36]. However, a major limitation of their review is the ability to detect small objects. The authors point out that to improve localization accuracy on small objects under partial occlusions, it is necessary to modify network architectures by introducing scale adaptation, 3D object detection, and Video Object detection.

More recently, there has been a lot of work done with Video Object detection and cloud services such as AWS and Google Cloud Platform offer object detection services so that companies do not need to develop their own models but can simply pay for a service and feed it images or videos to accomplish object detection [1]. The task of video object detection is extremely important for self driving cars and thus there has been extensive research done on it. Perreault et al. worked at taking advantage of the similarity between video frames to produce better detections by introducing a novel video object detection architecture that allows a network to share feature maps between nearby frames and a feature fusion module that learns to merge feature maps to enhance them [23]. We can leverage some of these techniques for real time object detection as well by creating short clips or videos and merging features in order for more accurate detection of eye movements.

Another technique was presented by Redmon et al. where the task of object detection was reframed as a regression problem to spatially separated bounding boxes and associated class probabilities [25]. This technique is more relevant to our project as it attempts to create multiple bounding boxes on an image or one frame in our continuous video that is occurring in real time, and associate class probabilities to all of the boxes outputting the highest probability for human eyes.

There have been some other recent developments in the field of object detection that can relate to our project in general. Beery et al. demonstrated boosting object detection performance on the current frame by using an attention-based approach that allows the model to index into a long term memory bank constructed from previous frames and aggregate contextual features [5]. Now there is tradeoff between speed and accuracy and we do not want to introduce complexity to the model if achieving greater accuracy takes longer running time as we are trying to track eye movements in real time and use them as user inputs for an interface. Huang et al. presents a detector that achieves real time speed and can be deployed on a mobile device [9], which is closely related to the system we want to build. Related to complexity, more intricate and layered neural networks will increase the running time for predictions for which Tan et al. have proposed various methods to improve efficiency including a weighted bi-directional feature pyramid network and a compound scaling method for images [30]. All of these techniques will be useful in developing a fast and efficient eye tracker.

Now to narrow down the task of object detection to eye movement tracking and using such movement to dictate user inputs in an application have its own set of challenges. As pointed out by Jacob et al., for real time use, the problem with eye tracking is to find appropriate ways to respond judiciously to eye movement input, and avoid over-responding; which is not nearly as straightforward as responding to well-defined, intentional mouse or keyboard input [12]. With that being said, these questions were risen very long ago before the rise of neural networks and developments in machine learning. In fact there has been plenty of research of doing object detection quickly in low level systems. In fact Kassner et al. released a product that comprises of a light-weight headset with high-resolution cameras, an open source software framework for mobile eye tracking, as well as a graphical user interface to playback and visualize video and gaze data [13]. In addition, most popular computer vision libraries such as OpenCV leverage most of the recent advancements made in the machine learning community allowing developers to easily build applications that rely on models for object detection. Thus, we are confident we will be able to develop an accurate eye tracking model that will be used to define user input in applications.

# 3 PROPOSED SOLUTION

Our solution will be in the form of a spacecraft game that is trying to defeat the evil ships shooting at it. The goal would be to survive as long as possible. Eye movement will be combined with more complex user interface inputs such as finger gestures to control the layout or influencing the course of the game itself. One example could be inspired by the *Fruit Ninja* game, where the player could 'slice' incoming asteroids to save the ship should there be an incoming one. This allows for the player to attack two fronts–using their eyes and hands–simultaneously. Manipulating the number of fingers

as well as incorporating the already ubiquitous mouse could add much more to the controls as well. Overall, through this project we will demonstrate how eye, finger, and mouse movements could be used as inputs to user interfaces and discuss the benefits and challenges of this approach.

# 4 IMPLEMENTATION

The final game implementation employed most of the proposed user interface solutions to give a complex yet intuitive control to the game and the spaceship. The Pygame Python package (v. 2.1.0) was used ubiquitously as the main platform where this game was coded from. The main control comes from the eye tracking program, followed by peripheral controls using the fingers and mouse to either defeat obstacles, change views, or simply to toggle a button.

## 4.1 Eye Tracking Input

As stated before, we utilized common methods in open source projects that involve object detection for our Eye Tracking system. To start, we get frames from the camera of the device running our program using the OpenCV VideoCapture function [3]. We then run these frames through dlib's frontal face detector [2] which returns rectangular objects of faces detected in the frame. Next, we pass these detected facial objects through a pretrained facial keypoints detector presented originally by Kazemi et al [15] which estimates the face's landmark positions directly from sparse subset of pixel intensities, achieving super-realtime performance with high quality predictions. Theoretically, more complex CNN models could improve facial landmark positioning in terms of pinpoint accuracy but would lose out on performance in terms of speed, making it unreasonable to use for a game application. Next, we take the region indicated by the landmark position coordinates and isolate the eye by masking the region and bitwise not the rest of the image with the solid color black. The next step involves taking the isolated eye images and prepossessing them by applying bilateral filter, erosion, and threshold based binarization using OpenCV all of which are common filtering processing in the world of computer vision. We then detects the iris from the preprocessed image using OpenCV findCountours and estimates the position of the iris by calculating the centroid. The thresholds for the filters are also optimized for each person based on their pupil size and the device camera frames.

This process is implemented in our code by creating Pupil, Eye, and GazeTracking objects. A GazeTraking object is initialized with the continuing while loop of our webcam feed and constantly refreshes new frames from the video feed. Each refresh causes a new analyze which resets the left and right Eye objects within the GazeTracking object. Each Eye object also contains a Pupil object. Now, left and right eye coordinates are calculated by taking the origin Eye coordinates, which are the min x and y coordinate of the isolated eye image added to the centroid pupil coordinates of that eye. A horizontal ratio is calculated considering the positions of both eyes' pupils and their x center coordinate. The horizontal ratio is a number between 0.0 and 1.0 that indicates the horizontal direction of the gaze where the extreme right is 0.0, the center is 0.5 and the extreme left is 1.0. Similarly, a vertical ratio is calculated considering the positions of both eyes pupils and their y center coordinate. The vertical ratio is a number between 0.0 and 1.0 that indicates the vertical direction of the gaze where the extreme top is 0.0, the center is 0.5 and the extreme bottom is 1.0. These attributes of the GazeTracking object allow for the eye movement of a user to be used as input for an application.

## 4.2 Eye Tracking for Ship Coordinates

With the input of the right eye normalized coordinates, the normalized coordinates is converted to a scaled coordinate commensurate with the window width and height, which shall be where the main spaceship points to and shoots at. Such angle where the ship must rotate to (counterclockwise from the vertical axis being $270 + a + b$ degrees in Figure 1) is calculated through simple trigonometry. The spaceship is always located in the center of the screen.

The added difficulty of implementation comes from the evil ship's rotation which must always point towards, and shoots at, the main spaceship. Referring to Figure 1, what must be entered into Pygame is the coordinates of the evil ship and its angle of rotation $c$. The evil ships are initialized randomly on the screen, so the coordinates are derived from its current coordinate plus the speed of the evil ships. The ships move such that if the

main ship points to them, the separating distance gets smaller and vice versa. This is to give the impression of running into them or away from them in space. To find $c$, we find that the following set of mathematical equations to be true in Figure 1 when the evil ship is in Quadrant I of the screen:

- $b = \arctan \frac{Q}{P}$
- $c = 180 - (90 - b) = 90 + b$

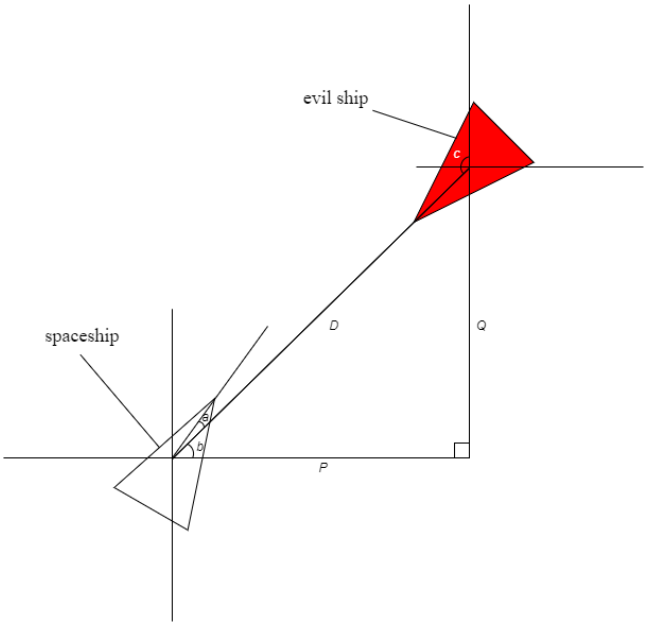


**Figure 1: Calculating the angle of rotation of enemy ships**

## 4.3 1-Finger Gestures

The concept of a one-finger gesture was expanded further to increase the utility of this user interface input. An input may be a click to a button or, as this game implements it, drawing a specific shape that is interpreted as a recognizable shape present in its template library. A "9-Square recognizer" was used for the system to interpret these simple hand-drawn shapes. Such a recognizer that supported digital ink and natural ink gestures was developed at Xerox PARC for use on the Liveboard system and has been used in meetings over a period of several years [6]. We took on the idea and built our own simple 9-square recognizer whose basic algorithm can be summarized as the following:

(1) Continuously record the coordinates of the single-finger gesture as long as the finger does not lift above the screen.
(2) Compute the minimum bounding box of said coordinates
(3) Divide the bounding box into 9 equal sized rectangles
(4) Mark where the gesture passes through
(5) Compare this gesture with the templates. Such templates may mark a 'square' as requiring a gesture present, left blank, or does not matter.

This algorithm can easily be expanded to accommodate roughly $3^9$ possible templates but for this game, five templates are used. In the event the finger gesture does not match any templates, the user is required to input the gesture again. Additionally, the program draws the gestures made to aid in user feedback.

### 4.3.1 *Left (<).*

The first template is equivalent to the less-than symbol ("<"). See Figure 2 for demonstration. Taking advantage of the human intuition to associate this bracket to pointing to the left, this application will pause the game and open a Settings panel that stays on the left part of the screen, shadowing over the rest of the screen covered with a translucent Pygame surface. When paused, most interactions would stop except for the shimmering stars and asteroid rotations to remind the user the game is still running well. Within the Settings panel, one setting option is provided for the user to customize their spaceship's bullet color to a set of given colors. The setting button changes color to the current color chosen to aid in user's color choice.

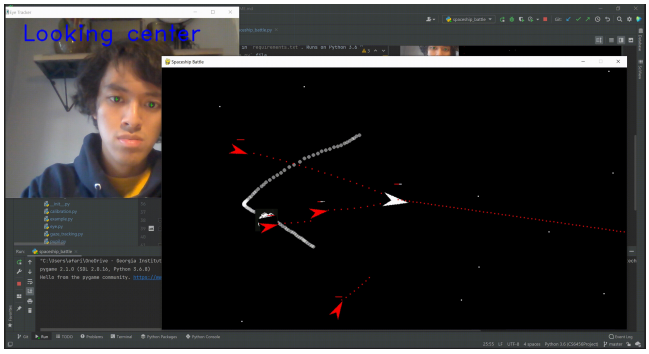


**Figure 2: Drawing a '<' to open Settings panel**

#### 4.3.2 *Right (>).*

The second template is one equivalent to the more-than symbol (“>”). Like the previous template, this takes the advantage of human intuition to open a panel on the right. This Request Support panel allows the user to select a number of friends, decorated as blue ships, to come to their support to defeat the evil ships until they ran out of health or until they are out of screen range. The current implementation means these friends are randomly generated and operated but in higher implementations, this can be expanded to retrieve a list of online friends, e.g., on Facebook and have the friends control the ally ships; in essence, this creates a multiplayer game. For added feedback, the button decorations change when hovered upon or selected.

#### 4.3.3 *Up (∧).*

The third template is equivalent to the wedge symbol (“ ∧ ”). This opens the Market panel filled with randomly generated items with randomly generated prices (within their predetermined ranges based on intrinsic value). The user may select the items to buy and sell. With random prices, this means the user can trade items–buying at low prices and selling at higher prices to make a profit. These money can be used to buy shields and upgrade their bullet power. The inventory button changes color depending on whether the user is selling or buying to the inventory. Red if the inventory is decreasing or green if increasing. The change in cash also changes color depending if the user is gaining or losing money from these trades. See Figure 3 for demonstration.

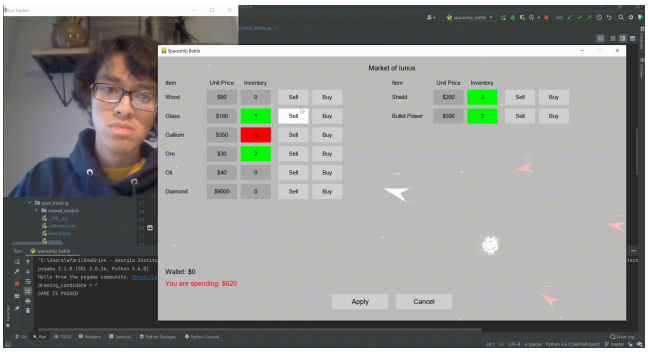


**Figure 3: User can buy and sell items on the market, changing the button colors**

#### 4.3.4 *Shield (◯).*

The fourth is an approximate circle (“ ◯ ”). Drawing a circle will temporarily activate a shield around the spaceship, which halts any health damages to the ship while the shield is up. The ship must have at least one inventory of a shield item bought from the Market for this function to be activated.

#### 4.3.5 *Strike (/).*

The fifth and final one-finger gesture template is a backslash (“/”). Drawing this strike symbol over an incoming asteroid will destroy the asteroid. See Figure 4 for demonstration. This design takes inspiration from the *Fruit Ninja* game where the player slices fruits with their fingers. This added functionality complements the eye tracking software which allows the user to control the direction of the ship using the eyes in an effort to aim the gun to the evil ships and destroy asteroid obstacles using their fingers. In further implementations, this can be expanded to have much higher difficulty with more asteroids/enemy ships and the capability to destroy both asteroids and enemies with both finger and eye inputs.

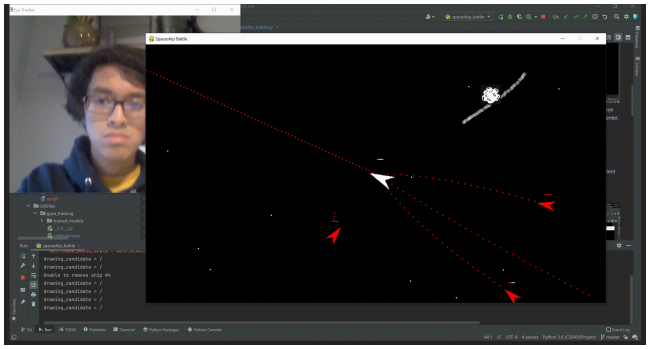

**Figure 4: Striking the asteroids with a finger destroys them**

## 4.4 4-Finger Gesture

A 4-finger gesture is detected by keeping track of the number of fingers, using the finger IDs, currently on screen.

#### 4.4.1 *Slide to change view.*

By sliding 4 fingers on the screen from the right to left, the user can change the view of the ship from a third-person view to a first-person view. See Figure 5 for

demonstration. In the first view, the main spaceship is always centered at the bottom middle part of the screen to give the impression of looking into space from the ship's cockpit. Additionally, ship icons are squashed vertically to give the impression the enemy ships are pointing at you.

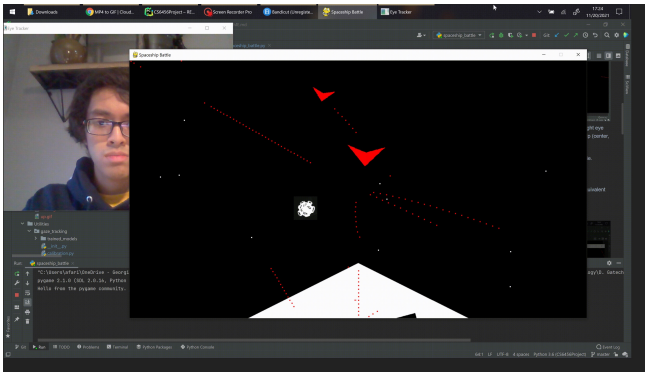

**Figure 5: An alternative first-person view after sliding 4 fingers to the left**

This view is derived from the original third-person view where only the environment in the 180 degree frontal view of the cockpit are rendered on screen. The 2-dimensional Euclidean distance from the spaceship to enemy ships are calculated to give the absolute distance between them. The coordinates of the ships in third-person view are used to calculate the angle deviation of the enemy ships from the cockpit. Together with the angle and Euclidean distance, the new coordinates in the first-person view is rendered for the enemy ships. Looking in Figure 6, we see the following set of mathematical equations are true if the evil ship is located in the right half of the main cockpit view.

- $D = \sqrt{P^2 + Q^2} = E$
- $d = 90 - a$
- $e = 180 - (90 - d) = 90 + d$
- $S = E \sin d$
- $R = E \cos d$

## 5 CONCLUSION

Overall, we have implemented an application that uses eye movement and finger gestures as user input. We have demonstrated that these two input types in concurrence is feasible as modern machine learning techniques have allowed for real time accurate eye position

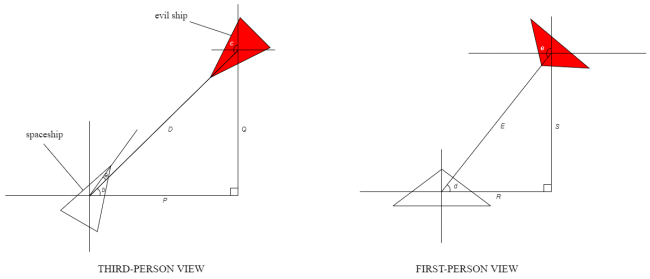


**Figure 6: Deriving the new coordinates of the enemy ships in first-person view**

predictions. Future work involves expanding the domain interest beyond a fast paced application such as a game, as we have personally found concentrating on the game and eye movements in a quick manner can be quite difficult.

## REFERENCES

[1] Amazon rekognition. https://aws.amazon.com/rekognition/?blog-cards.sort-by=item.additionalFields.createdDate&blog-cards.sort-order=desc. Retrieved September 27, 2021.
[2] dlib. http://dlib.net/python/index.html. Retrieved November 21, 2021.
[3] Opencv videocapture. https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html. Retrieved November 21, 2021.
[4] Peripherals. Available at https://gaming.tobii.com/products/peripherals/. Retrieved 30 September 2021.
[5] Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, and Jonathan Huang. Long term temporal context for per-camera object detection. *CoRR*, abs/1912.03538, 2019.
[6] W. Keith Edwards. 10-recognizers, Aug 2021.
[7] Andrea Ferracani, Marco Faustino, Gabriele Xavier Giannini, Lea Landucci, and Alberto Del Bimbo. Natural experiences in museums through virtual reality and voice commands. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, page 1233–1234, New York, NY, USA, 2017. Association for Computing Machinery.
[8] Jonas Goltz, Michael Grossberg, and Ronak Etemadpour. Exploring simple neural network architectures for eye movement classification. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research amp; Applications*, ETRA '19, New York, NY, USA, 2019. Association for Computing Machinery.
[9] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.
[10] Anke Huckauf and Mario H. Urbina. Gazing with peyes: Towards a universal input for various applications. In *Proceedings of the 2008 Symposium on Eye Tracking Research amp; Applications*, ETRA '08, page 51–54, New York, NY, USA, 2008. Association for Computing Machinery.

[11] T.E. Hutchinson, K.P. White, W.N. Martin, K.C. Reichert, and L.A. Frey. Human-computer interaction using eye-gaze input. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1527–1534, 1989.

[12] Robert Jacob and Keith Karn. *Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises*, volume 2, pages 573–605. 01 2003.

[13] Moritz Kassner, William Patera, and Andreas Bulling. Pupil: An open source platform for pervasive eye tracking and mobile gaze-based interaction. *CoRR*, abs/1405.0006, 2014.

[14] A.E. Kaufman, A. Bandopadhay, and B.D. Shaviv. An eye tracking computer user interface. In *Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium*, pages 120–121, 1993.

[15] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.

[16] Mohamed Khamis, Florian Alt, and Andreas Bulling. The past, present, and future of gaze-enabled handheld mobile devices: Survey and lessons learned. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '18, New York, NY, USA, 2018. Association for Computing Machinery.

[17] Elizabeth S. Kim, Adam Naples, Giuliana Vaccarino Gearty, Quan Wang, Seth Wallace, Carla Wall, Jennifer Kowitt, Linda Friedlaender, Brian Reichow, Fred Volkmar, Frederick Shic, and Michael Perlmutter. Development of an untethered, mobile, low-cost head-mounted eye tracker. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '14, page 247–250, New York, NY, USA, 2014. Association for Computing Machinery.

[18] Aleksandra Królak and Paweł Strumiłło. Eye-blink detection system for human–computer interaction. *Universal Access in the Information Society*, 11(4):409–419, November 2012.

[19] Andrew Kurauchi, Wenxin Feng, Ajjen Joshi, Carlos Morimoto, and Margrit Betke. Eyeswipe: Dwell-free text entry using gaze paths. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 1952–1956, New York, NY, USA, 2016. Association for Computing Machinery.

[20] Min Lin and Bin Li. A wireless eog-based human computer interface. In *2010 3rd International Conference on Biomedical Engineering and Informatics*, volume 5, pages 1794–1796, 2010.

[21] Robert Gabriel Lupu, Radu Gabriel Bozomitu, Alexandru Păsărică, and Cristian Rotariu. Eye tracking user interface for internet access used in assistive technology. In *2017 E-Health and Bioengineering Conference (EHB)*, pages 659–662, 2017.

[22] Päivi Majaranta and Kari-Jouko Räihä. Twenty years of eye typing: Systems and design issues. In *Proceedings of the 2002 Symposium on Eye Tracking Research amp; Applications*, ETRA '02, page 15–22, New York, NY, USA, 2002. Association for Computing Machinery.

[23] Hughes Perreault, Guillaume-Alexandre Bilodeau, Nicolas Saunier, and Maguelonne Héritier. Ffavod: Feature fusion architecture for video object detection, 2021.

[24] Ken Pfeuffer, Jason Alexander, Ming Ki Chong, and Hans Gellersen. Gaze-touch: Combining gaze with multi-touch for interaction on the same surface. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 509–518, New York, NY, USA, 2014. Association for Computing Machinery.

[25] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[26] Sheikh Rivu, Yasmeen Abdrabou, Thomas Mayer, Ken Pfeuffer, and Florian Alt. Gazebutton: Enhancing buttons with eye gaze interactions. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research amp; Applications*, ETRA '19, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Helmut Schrom-Feiertag, Volker Settgast, and Stefan Seer. Evaluation of indoor guidance systems using eye tracking in an immersive virtual environment. *Spatial Cognition & Computation*, 17(1-2):163–183, 2017.

[28] Shneiderman. Direct manipulation: A step beyond programming languages. *Computer*, 16(8):57–69, 1983.

[29] Sophie Stellmach and Raimund Dachselt. Look amp; touch: Gaze-supported target acquisition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 2981–2990, New York, NY, USA, 2012. Association for Computing Machinery.

[30] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *CoRR*, abs/1911.09070, 2019.

[31] Outi Tuisku, Päivi Majaranta, Poika Isokoski, and Kari-Jouko Räihä. Now dasher! dash away!: longitudinal study of fast text entry by eye gaze. In *ETRA '08*, 2008.

[32] Jayson Turner, Jason Alexander, Andreas Bulling, and Hans Gellersen. Gaze+rst: Integrating gaze and multitouch for remote rotate-scale-translate tasks. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 4179–4188, New York, NY, USA, 2015. Association for Computing Machinery.

[33] Jayson Turner, Andreas Bulling, Jason Alexander, and Hans Gellersen. Cross-device gaze-supported point-to-point content transfer. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '14, page 19–26, New York, NY, USA, 2014. Association for Computing Machinery.

[34] Jayson Turner, Andreas Bulling, and Hans Gellersen. Combining gaze with manual interaction to extend physical reach. In *Proceedings of the 1st International Workshop on Pervasive Eye Tracking amp; Mobile Eye-Based Interaction*, PETMEI '11, page 33–36, New York, NY, USA, 2011. Association for Computing Machinery.

[35] Mitsuho Yamada. Eye word processor (ewp) and peripheral controller for the als patient. *IEE Proceedings A (Physical Science, Measurement and Instrumentation, Management and Education, Reviews)*, 134:328–330(2), April 1987.

[36] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018.