Gaze Controlled Underwater Remotely Operated Vehicle (ROV) to Improve Accessibility in Maritime Robotics

Vanessa Barth*, Van Jones†, Avishka Samuel Athapaththu‡ and Leigh McCue§
Department of Mechanical Engineering, George Mason University
Fairfax, USA
Email: *vbarth@gmu.edu, †vjones20@gmu.edu, ‡aathapat@gmu.edu, §lmccuewe@gmu.edu

OCEANS 2023 - Limerick | 979-8-3503-3226-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/OCEANSLimerick52467.2023.10244706

***Abstract*—Underwater remotely operated vehicles (ROVs) are commonly operated with joystick-style controllers, but users with upper limb disabilities may find using a joystick either difficult or impossible due to partial or complete loss of function of their hands. Eye gaze control is a method that allows individuals to control a vehicle, device, or application without the use of their hands. It is often intended for those with a disability, but the method can also be used by those without a disability for hands-free operation. Past work on vehicle gaze control has focused primarily on land and aerial vehicles, and has yet to be extended into the field of maritime robotics. This paper discusses a method for gaze controlled underwater ROVs based on ArduSub and MAVLink, with transferability to other vehicle types using ArduPilot. The result of this method is a single script that provides the ability to control a tethered ROV via eye gaze in the directions of 'up,' 'down,' 'left yaw,' 'right yaw,' 'forward,' and 'reverse,' in addition to controlling the vehicle's lights and arming/disarming the vehicle. In-water demonstrations show successful operation of a gaze controlled ROV with no perceptible time delay.**

***Index Terms*—ROVs, maritime robotics, accessible control, eye tracking, gaze control, ArduPilot**

I. Introduction

Even modern autonomous vehicles typically require some degree of human-in-the-loop or human-on-the-loop control. Eye gaze controllers [1] have been quality of life enablers for people with mobility challenges. Recent advances driven by the gaming and e-sports industries have significantly reduced costs of eye gaze controllers [2]. At present, this has resulted in a rich range of applications for eye tracking and eye gaze control, from serving individuals with speech and motor impairments [3], [4] to telepresence [5] and teleoperated [6] robots to unmanned aerial vehicle (UAV) control [7] and immersion [8]. The aim of this work is to promote accessibility in the use of unmanned maritime vehicles for environmental, coastal, and biological research regardless of the scientist user's physical constraints.

In 2022, 12% of the US civilian noninstitutional population reported having a disability [9]. For persons with a disability who are aged 16-64, the employment to population ratio was 34.8% and the unemployment rate was 8.2% [9]. In comparison, for persons without a disability who were aged 16-64, the employment to population ratio was 74.4% and the unemployment rate was 3.5% [9]. The aim of this work is to provide an alternate method of control, that would facilitate use of remotely operated vehicles (ROVs) for individuals with upper limb disabilities and participate in fields where they are used. Furthermore, we recognize that decreasing barriers to access with intuitive control methods improves usability by all.

Eye gaze control is a method that allows individuals to control a vehicle, device, or application without the use of their hands. It is often intended for those with a disability, but the method can also be used by those without a disability for hands-free operation. Gaze control is implemented using an eye tracker, offered as a webcam, screen-based, or wearable device, and eye tracking algorithms. A study by Møllenbach *et al.* [10] describes the different eye movements and gaze interactions involved in gaze control. Eye movements, fixations, saccades, and smooth pursuits, combined with static, dynamic, or absent graphic display objects (GDOs), lead to the various gaze interactions that enable gaze control. Fixations are when the eyes are still and fixed on a feature, whereas saccades are the movements from one fixation to the next [10]. Smooth pursuits are when the eyes are following a moving target [10].

Past work on vehicle gaze control has focused primarily on land and aerial vehicles, such as mobile robots [5], [6], [11], wheelchairs [12], and drones [7], [13] and has yet to be extended substantively into the field of maritime robotics. For webcam or screen-based eye trackers, a control screen is either overlaid atop a first-person video feed, as in [6], [11], [13], or in the case of [12], where the user is in the wheelchair, there is no video feed and only the control screen is present. The four aforementioned works couple fixations with static GDOs, and served as guidance for the proposed method to be discussed later.

This paper presents a method for gaze control of underwater ROVs using a BlueROV2 [14], Tobii Pro Fusion eye tracker [15], ArduSub [16], MAVLink [17], and Python's multiprocessing module [18]. This approach is transferable to other vehicle types that use ArduPilot [19]. A python script was created which captures eye gaze data using the eye tracker, allowing the user to control the lights, arm/disarm the ROV, and send manual control commands to the vehicle.

The authors wish to express their gratitude for financial support of this work under NSF grant CMMI 2135619.

II. Implemented System

The implemented system leverages a BlueROV2 and existing software, ArduSub and QGroundControl (QGC) [20], while incorporating an eye tracker and the gaze control script, the main contribution of this work, to the user side. ArduSub is the vehicle's autopilot software and QGC is the ground station where the user can manage the vehicle, plan missions, monitor the video feed, etc. A diagram of the system overview can be seen in Fig. 1. On the user side, the eye tracker collects and sends gaze data to the python script, wherein the correct control message is identified and sent to QGC. QGC then sends that control information to the Raspberry Pi on the ROV, which eventually becomes a voltage signal sent to the thrusters and propels the vehicle in the intended direction. A more detailed diagram of the BlueROV2's software components and their interactions can be found at [21]. Further discussion on the eye tracker, gaze control script, and gaze data will be in Sections II-A, II-B, and II-C, respectively.

A. Eye Tracker

The system hardware is comprised of a BlueROV2, Tobii Pro Fusion eye tracker, and 14" laptop running Windows OS. A BlueROV2 from Blue Robotics (R1 version with a Pixhawk and Companion Software [22]) was used as the vehicle platform due to its modularity and open-source software. Among the options for an eye tracker, the Tobii Pro Fusion was selected for its range of sampling frequencies, high precision and accuracy, and price point. A summary of key specifications for the Tobii Pro Fusion and other systems considered is provided in Table I. The main software components include QGC, ArduSub, pymavlink [23], Python 3.8, Python's multiprocessing module, and Tobii Pro Software Development Kit (SDK) [24].

B. Gaze Control Script

To make the ROV capable of gaze control, three software solutions were evaluated; altering ArduSub, adding to ArduSub, or creating a stand-alone script. To mitigate risks associated with editing or adding to ArduSub, and its intricate set of supporting libraries and files, it was decided that a stand-alone script rather than modifying ArduSub would be optimal. Furthermore, creation of a standalone script enables dissemination of this work by relaying a single file rather than describing embedded edits.

Because MAVLink is already used for communication between QGC and the ROV, the script was written in Python so that the python implementation of MAVLink, pymavlink, could be used to communicate to the vehicle. To establish communication, a new mavproxy server at port 14660 was added to the ROV's mavproxy settings and another endpoint was created in the script. The new server was required to be at an unused port otherwise there would be conflicts between QGC and the script. Additionally, pymavlink has a library of functions that were utilized, such as one that sends manual control messages that mimic input from the joystick [33]. These messages are sent to QGC based on where the user is looking on the screen, ultimately "tricking" QGC to believe that these inputs are coming from a joystick. There is a failsafe in QGC where if a manual control message is not received at least every five seconds, then manual control is lost and the vehicle is disarmed. Due to this, every space of the screen was used for sending commands, otherwise looking in a non-command section of the screen could trigger the failsafe. The control screen, represented in Fig. 2, includes commands for yaw, forward/reverse, up/down, and lights brighter/dimmer. Additional commands could be incorporated, such as ones for camera tilt or gripper arm control (if installed), but it can become increasingly difficult for users to remember where each command is located on the screen. For this reason, only the commands that are essential for ROV operation were included. The script was written and tested in three stages, discussed in Section III-A.

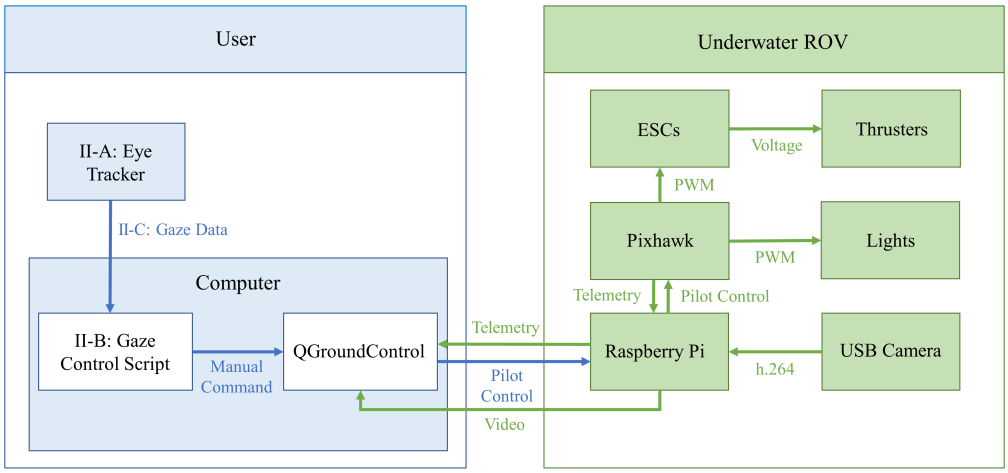

Fig. 1. Overview of the gaze control system with corresponding sub-sections identified.

TABLE I
COMPARISON OF SCREEN-BASED EYE TRACKERS.

Eye Tracker	Brand	Sampling Frequency (Hz)	Precision (°)	Accuracy (°)	OS
GP3 [25], [26]	Gazepoint	60 or 150	-[a]	0.5-1	Windows
Fusion [15]	Tobii Pro	30, 60, 120, or 250	0.2	0.3	Windows, Mac, or Linux
XO [27]	Smart Eye	30 or 60	-	-	Windows
Aurora [28], [29]	Smart Eye	30, 60, 120, or 250	0.1	0.3	Windows
Pro [30]	Smart Eye	60 or 120	-	0.5	Windows
Encore [31]	Eyegaze	50	-	<0.4	Windows
Advantage [32]	Eyegaze	50	-	<0.4	Windows

[a] "-" means the data was unavailable.

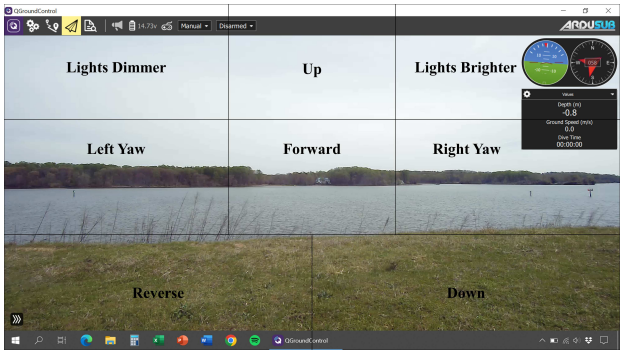

Fig. 2. QGC window overlaid with the eight commands.

C. Gaze Data

To get data on the location of the user's gaze, the Tobii Pro SDK (version 1.10.1) was used. The SDK has libraries of pre-existing functions that enabled connecting to the eye tracker, starting data collection, and stopping data collection. A significant constraint with the eye tracker and SDK was that while collecting gaze data, the script could not progress. To resolve this issue for real-time use, multiprocessing was used so that gaze data could be collected and used as control inputs essentially simultaneously.

Three separate processes were created; the first for logging, the second for collecting gaze data, and the third for vehicle control, referred to herein as the logging process, gaze process, and vehicle process, respectively. The logging process is created to help the user with testing and debugging processes, and is initiated first amongst the three processes. In the gaze process, launched second, the eye tracker is connected, begins collecting data, and relays the collected data through a queue to the vehicle process. The vehicle process does not begin until the gaze process has collected the first data array. This delay is necessary so as to ensure there is data available in the queue for the vehicle process.

The eye tracker can collect a wide range of data types, such as gaze origin, gaze point, pupil diameter, pupil validity, and time stamps. The data collected for use in this system was the gaze point on the screen and the pupil validity for the left and right eyes. The gaze point is a set of x and y coordinates from zero to one, where the upper left corner of the screen is (0,0) and the lower right corner is (1,1) [34]. The pupil validity checks whether the pupil data is valid (e.g. if each eye is open), and is either a zero for invalid (eye closed) or one for valid (eye open and pupil discernible) [35]. In this implementation, pupil validity is used to arm/disarm the vehicle such that a left eye closure or right eye closure can be used to arm or disarm the vehicle.

Once the gaze data is retrieved from the queue in the vehicle process, the gaze point for the left and right eyes are averaged together. The pupil validity is then used to check for a left eye closure or a right eye closure for arming or disarming, respectively. Since the eye tracker is collecting data at 60 Hz frequency, an eye closure is one that lasts for one second or more. The duration of a blink ranges from 200-400 ms and thus would not accidentally trigger an arm/disarm command. Once the vehicle is armed, the average gaze points are used to determine which command section the eyes are in (see Fig. 2), and the corresponding control message is sent to the vehicle. A step-by-step overview of the Gaze and Vehicle Processes is below.

Gaze Process

1. Find eye tracker
2. Connect to eye tracker
3. Start data collection at 60 Hz frequency
   Left gaze point on screen = (x_i, y_i)
   Right gaze point on screen = (x_j, y_j)
   Left/Right pupil validity = 0 or 1
4. Eye tracker collecting data = True
5. Data placed in an array
   data = [left gaze point, right gaze point, left pupil validity, right pupil validity]
6. Array put into the queue
7. Repeat steps 3, 5, & 6 until the 'space' key is pressed
8. Stop data collection

Vehicle Process

1. Create the connection to the vehicle
2. Retrieve a heartbeat message from the vehicle
3. Check the message for correct flight mode

a) Switch to manual mode if not currently in it

4. Bring QGC window to the front
5. Grab the data from the queue
6. Average the gaze points for the left and right eyes
   $x_{avg} = (x_i + x_j)/2$
   $y_{avg} = (y_i + y_j)/2$
7. Send a heartbeat to the vehicle at 1 Hz frequency
8. Check for lights brighter or dimmer command
9. If disarmed
   a) Check for arming command: left eye closure for $\geq$ 1 sec
10. If armed
    a) Check for disarming command: right eye closure for $\geq$ 1 sec
    b) Check for left yaw, right yaw, forward, reverse, up, and down commands
11. Repeat steps 5-10 until the 'space' key is pressed
12. Disarm vehicle if currently armed

After the processes, including the logging process, are stopped, they are joined and the script terminates. Joining each process ensures that it is safely shut down before the script is terminated. The main block of code waits for each process to complete all code, log all messages, and remove all data from the queue, and then resumes to terminate the script. Although not completely hands free due to the space key press for stopping the script, the keyboard press was chosen instead of eye input, such as a double eye closure or double blink, in order to prevent accidental interruption of the script. The interaction between the main block of code and the three processes is presented in Fig. 3.

III. Testing

Testing was conducted in three stages; bench testing, laboratory testing, and open water testing. Bench testing was primarily focused on code verification, whereas laboratory and open water testing assessed in-water performance. Laboratory testing permitted testing the system in an enclosed water tank, while open water testing was performed to test the system in a real-world setting.

A. Bench Testing

The gaze control script was written and tested in three stages: (1) connection to and manipulation of data from the eye tracker, (2) connection and relay of commands to the vehicle, and (3) facilitating arming/disarming of the vehicle. Testing of the eye tracker connection confirmed that data could be obtained from the eye tracker and manipulated. Testing the connection to the ROV ensured that a heartbeat data stream could be regularly sent to the vehicle and that messages could be received by the top-side computer. Testing the arming/disarming of the vehicle provided a dry land verification that the two prior steps were successfully integrated, as arming/disarming is controlled by a left or right eye closure, which requires a connection to the vehicle and data from the eye tracker. As noted previously, while a spacebar is used as a tactile input to terminate the script, being able to arm/disarm the vehicle with the eyes optimized usage of the eye tracker and reduced delays that would be inherent in using two control input sources. Furthermore, arming/disarming by eye gaze is a way for the user to confirm that command messages are successfully received by the vehicle.

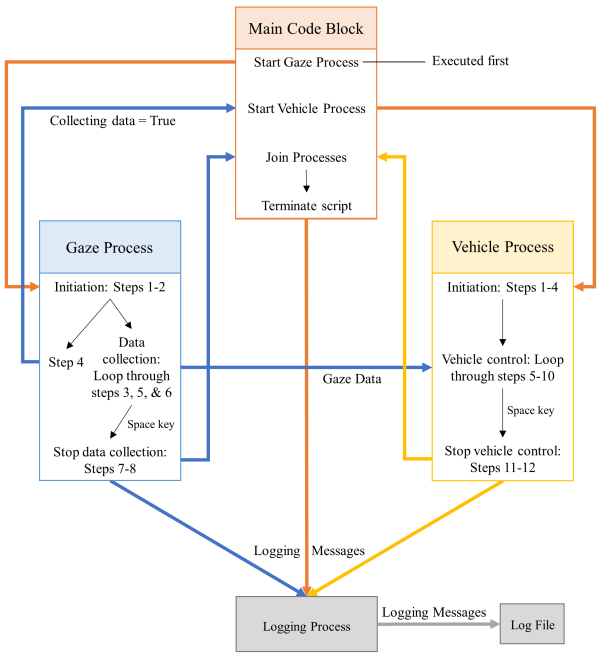

Fig. 3. Overview of the interaction between the main code block and the three processes.

B. Laboratory Testing

After bench testing, laboratory testing was conducted in the 3.8 m diameter water tank at the maritime robotics laboratory located at George Mason University's (GMU) College of Engineering and Computing 'Innovation Drive' facility. Tests were conducted over a three-week period. During this testing phase, fine-tuning of the script focused upon identifying the optimal motor speed, i.e. one that is not so fast that the vehicle overshoots the intended position or so slow that it becomes difficult to keep the gaze on the intended position. This was determined iteratively based on visual feedback using the code developer as the test subject and their determination of what felt most intuitive and natural as testing progressed. As discussed in Section 4.0, this is an area ripe for future research with human subjects testing. The time from when the eyes look in a command section versus when the vehicle moves was not calculated, but there is no perceptible time delay. Video showing gaze control of the ROV can be found in [36] with a screenshot from the video in Fig. 4.

C. Open Water Testing

Following successful laboratory testing, the system was tested in open waters in order to obtain practical results. The system was loaded onto a Parker 2530 boat and tested in Mallows Bay, approximately 45 minutes from GMU's Potomac Science Center. Mallows Bay, a ship graveyard for WWI era wooden ships, is located on the Maryland side of the Potomac River [37]. The original goal of this test was to navigate to a sunken wreck, obtain video footage of the outside of it, and navigate back to the boat. After arriving and anchoring the boat, the BlueROV2 was launched and testing began. Unfortunately, water conditions were not favorable, as seen in Fig. 5, with strong currents and water clarity limited to approximately six inches of visibility.

Therefore, the test plan pivoted to imaging the host boat, in essence emulating an underwater inspection activity, as shown in Fig. 6 where one can clearly assess the state of the boat's submerged trim tabs. The gaze control worked as intended lending evidence that this system would provide a viable option for underwater inspection and in other scenarios like environmental, coastal, and biological research. As discussed further in Section IV, there are opportunities to refine the visual display to improve usability.

IV. Conclusions and Future Work

In this paper, in order to improve accessibility in maritime robotics, the design and testing of a gaze control system was discussed. The implemented system uses a BlueROV2, Tobii Pro eye tracker, and a laptop running Windows 10. A script was created that utilizes pymavlink, Tobii Pro SDK, and multiprocessing in order to retrieve eye gaze data and send commands to the ROV according to the location of the user's gaze on the screen. To overcome the script pausing while collecting data, separate processes were created for data collection and vehicle control.

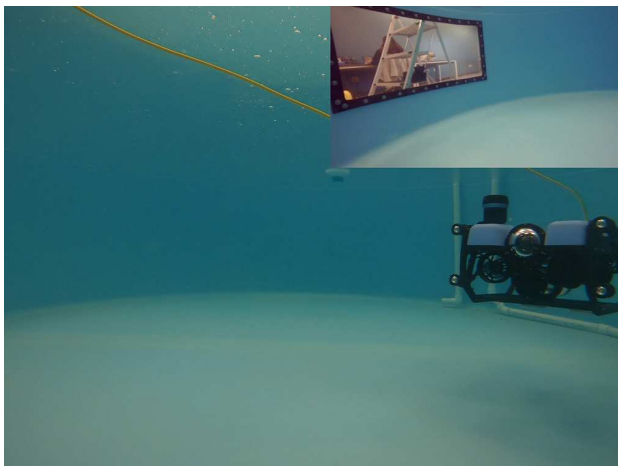

Fig. 4. Testing of gaze control using a BlueROV2.

Fig. 5. Testing of gaze control using a BlueROV2 in Mallows Bay.

Fig. 6. Underwater image of Parker 2530 trim tab.

Preliminary testing reported on here includes bench, confined laboratory in-water testing, and open water testing. Of note, in open water testing it was observed that should the ROV driver wish to refer to the artificial horizon included in the QGC software (visible in Fig. 2), they would inadvertently relay a control command. This may result in guidance that the system be utilized solely in clear water, where the user is not reliant on an artificial horizon.

Selecting the BlueROV2 as the vehicle platform was partially due to its open-source software, ArduSub, which is part of the ArduPilot project. This allows the system to be transferrable to other vehicles with ArduPilot based software, such as surface boats running ArduRover or aerial vehicles running ArduCopter or ArduPlane. Additionally, communication using pymavlink lets additional endpoints to be created, such as endpoints to additional vehicles, which would allow for multi-vehicle control. Lastly, the use of a script allows for less experienced developers to alter the script themselves, such as when support for a different eye tracker needs to be added, instead of trying to alter ArduSub.

Future work for the implemented system includes conducting a user study comparing joystick control to gaze control. Ease of operation, comfortability, optimal command speed, and number of errors would be assessed and compared to joystick control to determine accuracy and device preference. Feedback from participants will be used to identify shortcomings of the system and areas for improvement. In addition, gaze control will be tested on a surface boat to prove the transferability of the system. This will lead into work on multi-vehicle control where a fleet of autonomous maritime vehicles supported by machine learning will be following a gaze-controlled vehicle, such as the BlueROV2.

REFERENCES

[1] The eyegaze edge. Eyegaze. [Online]. Available: https://eyegaze.com/products/eyegaze-edge/

[2] Tobii eye tracker 5. Tobii. [Online]. Available: https://gaming.tobii.com/product/eye-tracker-5/

[3] V. K. Sharma, K. Saluja, V. Mollyn, and P. Biswas, "Eye gaze controlled robotic arm for persons with severe speech and motor impairment," in *ACM Symposium on Eye Tracking Research and Applications*, ser. ETRA '20 Full Papers. Association for Computing Machinery, Jun. 2020, pp. 1–9. [Online]. Available: https://doi.org/10.1145/3379155.3391324

[4] Y.-S. L.-K. Cio, M. Raison, C. L. Ménard, and S. Achiche, "Proof of concept of an assistive robotic arm control using artificial stereovision and eye-tracking," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 27, no. 12, pp. 2344–2352, Dec. 2019. [Online]. Available: https://doi.org/10.1109/TNSRE.2019.2950619

[5] G. Zhang, J. P. Hansen, and K. Minakata, "Hand- and gaze-control of telepresence robots," in *Proc. of the 11th ACM Symposium on Eye Tracking Research & Applications*, ser. ETRA '19. Association for Computing Machinery, Jun. 2019, pp. 1–8. [Online]. Available: https://doi.org/10.1145/3317956.3318149

[6] H. O. Latif, N. Sherkat, and A. Lotfi, "Teleoperation through eye gaze (telegaze): A multimodal approach," in *Proc. 2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec. 2009, pp. 711–716. [Online]. Available: https://doi.org/10.1109/ROBIO.2009.5420585

[7] J. P. Hansen, A. Alapetite, I. S. MacKenzie, and E. Møllenbach, "The use of gaze to control drones," in *Proc. of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '14. Association for Computing Machinery, Mar. 2014, p. 27–34. [Online]. Available: https://doi.org/10.1145/2578153.2578156

[8] P. K. B. N., A. Balasubramanyam, A. K. Patil, C. B., and Y. H. Chai, "Gazeguide: An eye-gaze-guided active immersive uav camera," *Appl. Sci.*, vol. 10, no. 5, 2020. [Online]. Available: https://doi.org/10.3390/app10051668

[9] "Persons with a Disability: Labor Force Characteristics — 2022," Bureau of Labor Statistics, Feb. 2023, News Release. [Online]. Available: https://www.bls.gov/news.release/pdf/disabl.pdf

[10] E. Møllenbach, J. P. Hansen, and M. Lillholm, "Eye movements in gaze interaction," *J. Eye Mov. Res.*, vol. 6, no. 2, pp. 1–15, May 2013. [Online]. Available: https://doi.org/10.16910/jemr.6.2.1

[11] M. Tall, J. P. Hansen, A. Alapetite, D. W. Hansen, J. S. Agustin, E. Møllenbach, and H. H. Skovsgaard, "Gaze-controlled driving," in *Proc. CHI '09 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '09, Apr. 2009, pp. 4387–4392. [Online]. Available: https://doi.org/10.1145/1520340.1520671

[12] Y. K. Meena, H. Cecotti, K. Wong-Lin, and G. Prasad, "A multimodal interface to resolve the midas-touch problem in gaze controlled wheelchair," in *Proc. 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2017, pp. 905–908. [Online]. Available: https://doi.org/10.1109/EMBC.2017.8036971

[13] A. Kogawa, M. Onda, and Y. Kai, "Development of a remote-controlled drone system by using only eye movements: Design of a control screen considering operability and microsaccades," *J. Robot. Mechatron.*, vol. 33, no. 2, pp. 301–312, Apr. 2021. [Online]. Available: https://doi.org/10.20965/jrm.2021.p0301

[14] BlueROV2. Blue Robotics. [Online]. Available: https://bluerobotics.com/store/rov/bluerov2/

[15] Tobii Pro Fusion. Tobii Pro. [Online]. Available: https://www.tobii.com/products/eye-trackers/screen-based/tobii-pro-fusion

[16] ArduSub: Overview. Blue Robotics. [Online]. Available: https://www.ardusub.com/

[17] MAVLink Developer Guide. Dronecode Project. [Online]. Available: https://mavlink.io/en/

[18] multiprocessing - Process-based parallelism. Python Software Foundation. [Online]. Available: https://docs.python.org/3.8/library/multiprocessing.html

[19] ArduPilot. ArduPilot Dev Team. [Online]. Available: https://ardupilot.org/

[20] QGroundControl. Dronecode Project. [Online]. Available: http://qgroundcontrol.com/

[21] Software Components. Blue Robotics. [Online]. Available: https://www.ardusub.com/developers/software-components.html

[22] Companion Computer Software. Blue Robotics. [Online]. Available: https://www.ardusub.com/introduction/required-software/companion-computer-software.html

[23] Pymavlink. Blue Robotics. [Online]. Available: https://www.ardusub.com/developers/pymavlink.html

[24] Tobii Pro SDK. Tobii Pro. [Online]. Available: https://www.tobii.com/products/software/applications-and-developer-kits/tobii-pro-sdk/tobii-pro-sdk

[25] GP3 Eye Tracker. Gazepoint. [Online]. Available: https://www.gazept.com/product/gazepoint-gp3-eye-tracker/

[26] GP3 HD Eye Tracker 150 Hz. Gazepoint. [Online]. Available: https://www.gazept.com/product/gp3hd/

[27] XO. Smart Eye. [Online]. Available: https://smarteye.se/xo/

[28] Aurora. Smart Eye. [Online]. Available: https://smarteye.se/aurora/

[29] "Aurora," Smart Eye, Tech. Spec. Sheet. [Online]. Available: https://2372627.fs1.hubspotusercontent-na1.net/hubfs/2372627/Smart%20Eye%20PDF%20Files/Smart%20Eye%20Aurora%20%E2%80%94%20Product%20Sheet.pdf

[30] "Smart Eye Pro," Smart Eye, Tech. Spec. Sheet. [Online]. Available: https://smarteye.se/wp-content/uploads/2022/11/Smart-Eye-Pro_tech-spec_updated.pdf

[31] Encore. Eyegaze. [Online]. Available: https://eyegaze.tech/products/encore/

[32] Eyegaze Advantage®. Eyegaze. [Online]. Available: https://eyegaze.tech/products/advantage/

[33] MAVLink Common Message Set. Dronecode Project. [Online]. Available: https://mavlink.io/en/messages/common.html#MANUAL_CONTROL

[34] Coordinate systems. Tobii Pro. [Online]. Available: https://developer.tobiipro.com/commonconcepts/coordinatesystems.html

[35] Python – SDK reference guide. Tobii Pro. [Online]. Available: https://developer.tobiipro.com/python/python-sdk-reference-guide.html

[36] V. Barth. (2022, Dec.) Eye Gaze Control of a BlueROV2. Fairfax, VA, USA. Video. [Online]. Available: https://www.youtube.com/watch?v=C-Ql4g1nJ-Y

[37] Shipwrecks. NOAA. [Online]. Available: https://sanctuaries.noaa.gov/mallows-potomac/education/shipwrecks.html