Fly Robot Interface (Neural)

NeuRob

Department of Bioengineering

Imperial College London

A Project Report Submitted in
Partial Fulfilment of the

MEng —– Degree

Word count:4970

Supervisor:
Professor Holger Krapp

Department of Bioengineering
h.g.krapp@imperial.ac.uk

April, 2025

Fly Robot Interface (Neural)

Zeyuan Xin, Yuichiro Minamikawa, Yi Zhang, Changyu Hu, Shuchang Zhang, Wenjun Jiao, and Badriyah Islam

Department of Bioengineering, Imperial College London, UK

Abstract

Bioinspired technology has the capabilities to improve technological applications in autonomous robotic systems based on biological design principles that are more energy-efficient and adaptive compared to current engineering approaches. The Neural Fly-Robot Interface (FRI) investigates the use of neural outputs of *Calliphora vicina* (blowfly), a highly efficient organism in flight and gaze stabilization, to control a two-wheeled robot for real-time collision avoidance. This involved recording real-time extracellular signals of the identified H1-cell in the motion vision pathway of the fly, utilizing its capabilities of processing optical flow to control the trajectory of a robot. The FRI was successfully able to repeat experiments previously done by Prof. Holger Krapp and Dr. Jiaqi Huang, having the mounted blowfly accurately control a robot that avoids collision with the walls of an experimental arena. A more compact and robust 3D-printed robot was also designed and built to improve the reliability of the FRI. Additionally, we explored the feasibility of a dual H1-cell approach to the collision avoidance mechanism, showing its potential to be translated to the FRI. This approach is still novel but will be crucial in understanding a more complete model of binocular insect vision, enabling collision-free navigation of arbitrary environments.

1 Introduction

1.1 Background and Motivation

The study of bio-hybrid systems has gained momentum due to their potential to improve sensorimotor control mechanisms by leveraging biological principles. Nature is energy-efficient, flexible, and capable of complex control— traits that have not been fully captured by engineered robotic systems. [1, 2] Bioinspired systems can bring solutions to the energy-intensive demands of conventional maneuvering technologies, such as artificial intelligence, radar, and camera-based systems. For example, event cameras, inspired by biological vision, which asynchronously measures per-pixel brightness changes instead of capturing images at a fixed rate, have been successfully implemented in modern robotics. [3]

The blowfly, *Calliphora vicina*, is an exemplary model for efficient sensorimotor control, capable of navigating complex environments smoothly while minimizing energy expenditure. [4, 5] This contrasts with engineered autonomous robots, which require much greater amounts of energy, with computation and sensors already accounting for 50% of its energy consumption. [6] Moreover, optical flow techniques in conventional systems struggle with ambiguity in motion estimation, especially in scenes that lack visual contrast or when objects are moving along the same axis. This reduces depth perception reliability and makes it difficult for drones to navigate cluttered or low-contrast environments. Environmental factors such as lighting variation, reflective surfaces, and motion blur further exacerbate these limitations, making conventional systems not fully reliable in real-world, unstructured settings.

By contrast, the blowfly's H1-cell solves many of these

problems through evolutionary tuning, a neuron known for its association with collision avoidance. The H1-cell selectively responds to horizontal back-to-front motion, filtering out irrelevant noise and firing a spike rate that is inversely proportional to distance for a certain turning radii, responding to collision-related trajectories of motion. [7, 8] The cell response properties have led to studies that exploit it to provide a biohybrid low power solution for real-time obstacle avoidance. [9] Furthermore, unlike algorithm-based systems that often require stereo cameras or Light Detection and Ranging (LiDAR) for depth estimation, [10] the fly uses optical flow to infer relative motion and obstacle proximity, highlighting the efficiency of neural encoding over brute-force.

However, lack of bilateral neural integration in previous studies limits the performance of the bionic system. Most work has been done in monocular recordings, only capturing a portion of the visual system, where often binocular processing is crucial for a complete understanding of operations in arbitrary visual environments. [11]

1.2 Project Overview

The Fly-Robot Interface (FRI) project seeks to utilize the blowfly's efficient collision avoidance mechanisms by translating the neural encoding of motion into robotic control strategies. This work is built upon the research by Holger G. Krapp and Jiaqi Huang, who investigated insect sensorimotor control and bioinspired robotics. Central to this is the H1-cell, an interneuron part of the Lobula Plate Tangential Cell (LPTC) network, sensitive to back-to-front horizontal motion, a reliable cue for nearby obstacles during a yaw rotation. [12] It fires proportionally to the proximity of obstacles, acting as an indicator of time-to-collision.

This single-cell computation is remarkably efficient: it exploits optic flow statistics and filters irrelevant background motion, firing in response to motion trajectories that indicate a collision risk. [8] Moreover, H1-cells inhibit contralateral homologous neurons via synaptic connections, enhancing directional sensitivity and energy-efficient neural coding. [12] The behavior of the H1-cell can be further understood by the video in **Appendix D**.

To understand and exploit this biological mechanism in robotic systems, extracellular H1-cell recordings are recorded and processed with a neural interface, which turns it into motor command signals that control the wheels of the FRI and initiates a collision avoidance when a certain threshold of the spike rate is exceeded.

Early implementations relied on single H1-cell recordings, restricting the robot to simple wall-following behaviours. In contrast, the FRI project incorporated bilateral neural integration to better model insect vision. Since the two H1-cells are predominantly monocular with minimal binocular overlap, integrating signals from both cells achieves an enlarged visual field to capture more information, allowing for collision avoidance in more complex environments. This bilateral approach aligns with evidence that flies compare optic flow inputs from left and right H1-cells to refine collision vector estimation. [7]

1.3 Project Objectives

This project aimed to use the blowfly's motion vision pathway to control a robot and further advance the capabilities of the neural FRI. The specific objectives of this study were:

- To successfully repeat previous experiments that control robot movements using live extracellular recordings of H1-cells. Specifically, we aim to achieve an obstacle avoidance accuracy of at least 90%
- Design a more robust, stable and lighter robot for better manoeuvring
- Integrate dual H1-cell recordings to the FRI. Bilateral recordings help understand the problem of motion disambiguation in dynamic environments

2 Methods

The FRI consists of 4 main components: **(i) Dissection and H1-cell Signal Collection**, **(ii) Neural Interface Signal Processing**, **(iii) Robot**, and **(iv) Trajectory Analysis**. Each component is described in detail in **Sections 2.1-2.4**, followed by overall testing and data collection in **Section 2.5**.

2.1 Dissection and H1-Cell Signal Recording

High-quality H1-cell recordings are crucial for the FRI. A good H1-cell recording has a signal-to-noise ratio (SNR) of 2:1. Precise dissection and recording techniques were followed to ensure signal quality and consistency. Detailed description of procedures is provided in **Appendix C**. [13]

For the ease of dissection and minimizing noise from muscle activity, the fly is tethered. The fly head is fixed to the fly holder, ensuring that the pseudopupils

(**Figure 1**) of both eyes are aligned horizontally. Care is taken to avoid obstructing the eyes or trachea as this blocks vision and breath.

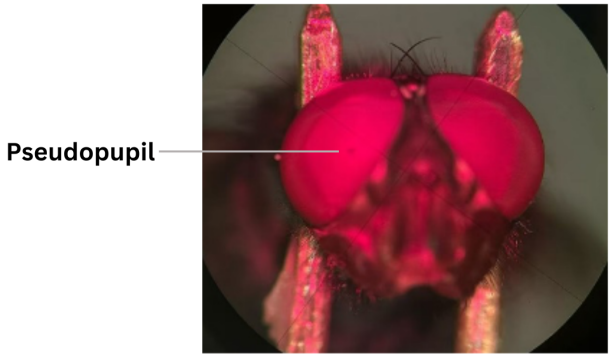


Figure 1: Image taken during dissection and signal collection. Pseudopupil of the blowfly, visible as the black spot in the middle of the eye.

The head is carefully dissected to expose 3 tracheas which aid in electrode positioning. Tungsten microelectrodes with optimized impedance (∼3 MΩ) are used. [14] The H1-cell is typically located in the outer region of the middle trachea. Under a microscope, the position of the electrode was carefully adjusted using micromanipulators. Once the electrode tip is immersed in brain fluid, it is connected to the oscilloscope and linked to a speaker to translate voltage changes into acoustic signals, allowing for real-time auditory feedback of cellular activity.

2.2 Neural Interface Signal Processing

Live extracellular H1-cell recordings were processed in real-time by a custom-built neural interface based on the RP2040 Zero microcontroller (Raspberry Pi Foundation, UK). Additionally, for collecting data to analyze offline (open-loop experiments), a data acquisition card (NI USB-6215, National Instruments Corporation, Austin, TX, USA) connected to a PC hard disk was used. The raw signals are sampled at 20kHz. [13] A flowchart illustrating the signal processing pipeline is shown in **Figure 2**, with code provided in **Appendix H**.

The voltage trace was processed through a two-stage thresholding algorithm. The first stage involves thresholding voltages corresponding to action potentials fired during back-to-front optic flow patterns.

With single H1-cell recordings, a single threshold of 2.8 V was used, which can be adjusted to account for variations electrode placements. Depending on the recorded signal, either rising-edge or falling-edge

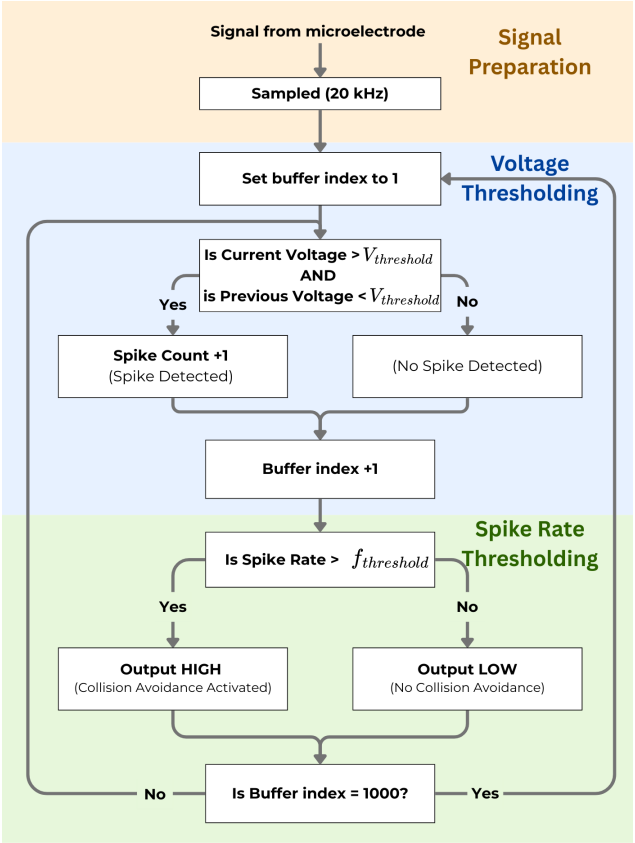


Figure 2: Flowchart showing the algorithm for two-step thresholding for single H1-cell processing in the RP2040 Zero neural interface. Top (Orange) – Signal being sampled from the microelectrode for processing by the neural interface. Middle (Blue) – Spike Detection, involving a voltage threshold applied to the recorded raw signal. Bottom (Green) – Spike Rate Threshold, involving a threshold on the frequency of spikes and determining whether a collision maneuver is triggered.

threshold crossings were used to detect upwards or downwards peaks respectively. Spike rate was calculated in 50 ms non-overlapping windows, corresponding to the sampling buffer of 1000 samples at a 20 kHz sampling rate as shown in Equations (1).

$$\begin{aligned} \text{Spike Rate} &= \frac{\text{Spikes in last 50 ms}}{\text{50 ms}} \times 1000 \\ &= \text{Spike Count} \times 20 \end{aligned} \tag{1}$$

With dual H1-cell recordings, we introduce two methods that analyze the extracellular signals from the left and right H1-cells simultaneously : **(i) single-electrode multi-threshold classification**, **(ii) dual-electrode synchronized recordings**. These methods are evaluated in terms of their capacity to resolve spike patterns and suitability for downstream FRI robotic control. The first approach applies two thresholds to specific amplitudes to identify action potentials

from different H1-cells. This is tested both with non-overlapping and overlapping signals, with a band-pass filter to enhance the characteristics of signals. [15] The second approach uses two electrodes to record the two H1-cells separately, effectively conducting two single-cell experiments simultaneously.

Spike rate is inversely proportional to distance between the fly and an obstacle; thus, a high spike rate indicates the FRI being close to a wall. When the spike rate reaches a 220 Hz threshold, which corresponds to the frequency of back-to-front motion triggered action potentials, the RP2040 Zero releases a HIGH Boolean signal to the master Raspberry Pi unit, indicating that the robot must actuate a collision avoidance.

To test the signal collection and neural interface prior to completing closed-loop experiments, electrophysiological signals with manually induced back-to-front motion were recorded from the blowfly using the data acquisition card. For dual H1-cell experiments, RP2040 output signals were recorded as an alternative to arena testing and trajectory plotting. The dual-cell spike-sorting algorithm was implemented on pre-recorded data and executed on the RP2040, which processed the input in real-time and generated Boolean output signals on four general-purpose I/O (GPIO) pins to give an indication of when collision avoidance would be triggered.

2.3 Robot

2.3.1 Robot Design

A new 3D-printed robot was developed to replace a previous version constructed with laser-cut wood. The robot has four main parts: the ArUco Marker Holder, the Signal Collection Platform, the Signal Processing Platform, and the Robot Motor Control Platform, as shown in **Figure 3**.

The ArUco marker holder uses a triangular support column to minimize visual obstruction to the blowfly. The marker is a 10 × 10 mm square, centered in the field of camera view. The signal collection platform includes four 1:7 Z-shaped damping structures. The signal processing and motor control platforms form a sealed and integrated unit. The assembled robot has a total mass of 1093 g and a height of approximately 30 cm. Manufacturing details for each component and technical drawings are provided in **Appendix E** and **Appendix N** respectively.

A Raspberry Pi 3B serves as the control unit, receiving input from the neural interface and sending commands to L298N motor drivers connected to TT motors. The system is powered by a portable battery. Control code and schematics are in **Appendix I** and **Appendix O** respectively.

2.3.2 Dynamic Test

To evaluate the performance of the robot in different control frequencies and durations, a dynamic test was conducted. Control frequency refers to how frequently the robot changes direction. Each trial consisted of 4 alternating rotations (clockwise and counterclockwise), with a 0.5-second pause between each directional change. Rotation speed was adjusted using different Pulse Width Modulation (PWM) values. The tested durations were: 2, 1.5, 1, 0.5, 0.4, 0.2, 0.1, 0.0625, and 0.05 seconds. Each duration represents how long the robot maintains one rotational direction before switching. Video footage was recorded to capture the robot's motion. The robot's angular velocity (ω_{out}) over time was extracted through frame-by-frame video analysis and stored in CSV format. The cumulative rotation angle per directional change was computed and plotted to the system's ability to follow rapid command signals. The resulting data were used to assess response characteristics, including delay, gain, cut-off frequency, and resonance behavior. The analysis and angle plotting code are in **Appendix K** and **Appendix L** respectively.

2.4 Trajectory Plotting

To test the performance of the FRI, the trajectory is recorded and plotted for analysis and comparison with extracellular signals. The trajectory is plotted by identifying the robot ArUco marker's location frame-by-frame with OpenCV identifiers. To visualize the motion path, arrowed lines connecting consecutive positions are drawn on a dynamically updating Matplotlib quiver plot. A plot of the trajectory can be seen in **Figure 10**, with the code in **Appendix M**.

2.5 Testing and Data Collection

The robot is placed in a 2 m x 2 m arena with black-and-white striped columns(1.5 cm wide, 50 cm high) to provide wide-field motion stimulation to the H1-cell during the oscillatory trajectory of the FRI. [13] A metal baseplate is used to minimize electromagnetic interference. A camera is placed vertically above the arena in the center to record trajectories of the FRI as described in Section 4.5.

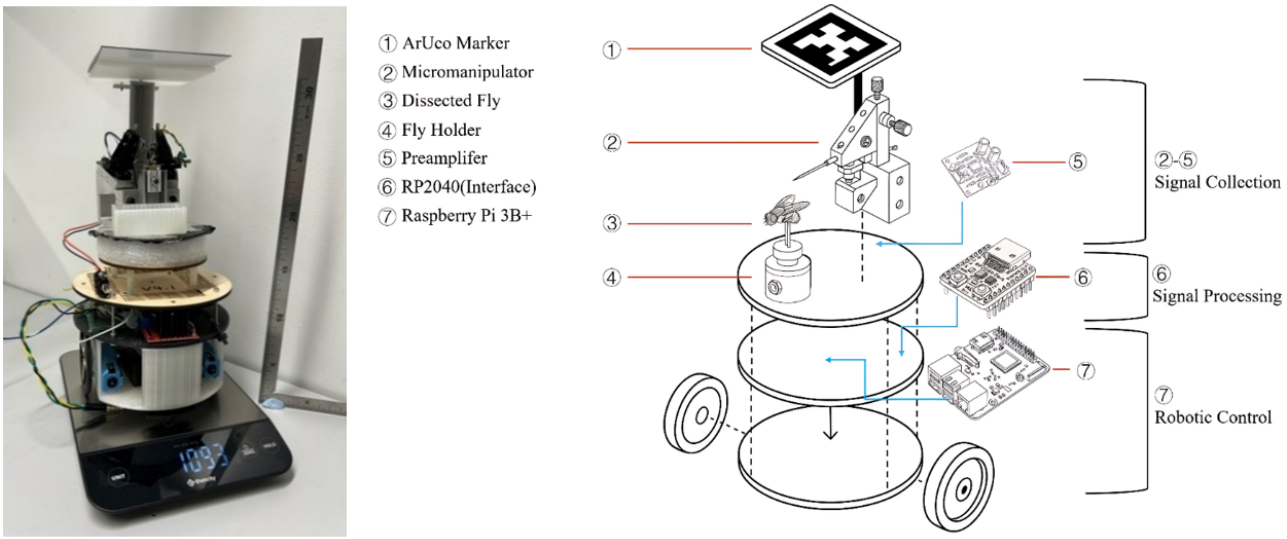


Figure 3: FRI robot setup. Left is a photograph of the FRI robot, with a weight of 1093g and height of 30cm. Right is a sketch of the different components of the FRI, showing signal collection (2-5), signal processing neural interface (6) and chip for robotic control (7). All components are contained within for compactness.

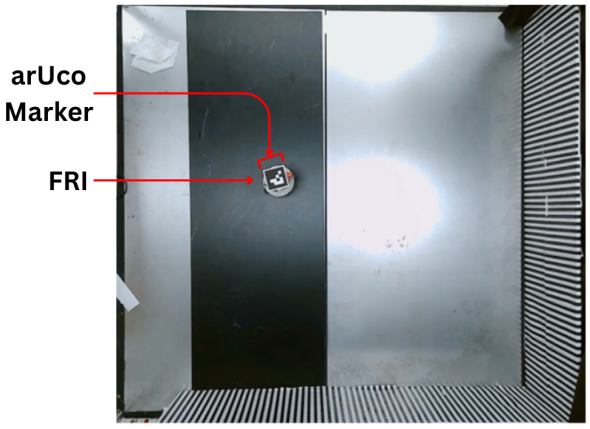


Figure 4: Testing arena for the FRI robot from the perspective of the recording camera (vertically above). The FRI robot can be seen in the middle-left, with the arUco marker being the most visible part. Striped walls can also be seen in the bottom and right walls.

For comparing trajectories with the electrophysiological signals, the data acquisition card is connected to the FRI. The FRI is intentionally biased to move towards the left, and the bottom-right corner of the arena is slightly illuminated. The robots inherent oscillatory motion makes it biased to the wall, stimulating back-to-front motion. Each trial lasted 10 to 30 seconds. For comparison, the Instantaneous Spike Rate (ISR) from the electrophysiological recordings is plotted along with the distance-to-wall of the robot. Angle-with-wall and distance-to-wall are plotted together for examining the significance on angle with the trajectory.

The performance of the FRI are evaluated by the following metrics: (i) collision avoidance success rate, as shown in (2) (ii) trajectory patterns (iii) correlation analysis between ISR, and trajectory variables (distance to wall and orientation angle) of the collected data and an ideal result. (iv) absolute error of the correlation in (iii).

$$\text{Accuracy} = \frac{\text{Number of Successful Trials}}{\text{Total Number of Trials}} \quad (2)$$

2.6 Ethics Statement

The choice of the blowfly is intentional – it is biologically relatively simple, and most importantly, of minimal ethical constraints, making constant experimenting feasible. While there are no legal ethical restrictions on working with insects or a clear scientific consensus on whether insects are sentient, [16] our experiments still adopt techniques that minimize unnecessary harm and pain to the insect. Furthermore, these experiments are a part of the advancement of technology and energy-efficiency midst the global energy and climate change crisis, for the greater good.

3 Results

3.1 Signal Collection and Neural Interface

With both single and dual H1-cell recordings, an SNR of greater than 2:1 was achieved. Comparison of obtained signals (burst) with steady signals (tonic) show that the electrophysiological signals are statistically significant, with the inter-spike interval (ISI) histogram and probability distribution shown in **Figure 5**. ISI distributions were computed from extracellular spike trains. Probability density estimates (PDE) were obtained using kernel density estimation.

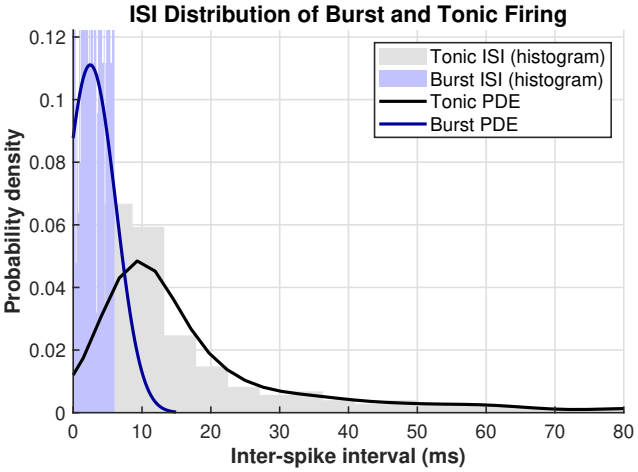


Figure 5: Distribution of ISIs for burst (blue) and tonic (grey) firing. Histogram bars represent the empirical distributions. Burst and tonic ISI PDEs had peaks seperated by 6 ms.

3.1.1 Single H1-Cell

As illustrated in **Figure 6**, extracellular recordings (6.a) from the H1-cell revealed distinct high-amplitude spike events in response to back-to-front motion stimuli. These spikes, detected by threshold crossings (6.c), were temporally aligned with stimulus onset and were used to compute the ISR. Notably, the ISR exceeded the 200 Hz threshold during periods of strong motion input, compared to the mean rate of 76Hz, as shown in (6.a) and (6.d), triggering collision avoidance commands via the neural interface (6.e).

3.1.2 Dual H1-Cell

Single Electrode Multi-Threshold The single-electrode, dual-threshold method enabled accurate ISR calculation for both non-overlapping and overlapping dual H1-cell recordings, as shown in **Figure 7**. The non-overlapping signal employed valley detection due to the negative spike shape, while the overlapping signals used peak detection.

The bilateral voltage traces show two distinct spike amplitudes corresponding to the left and right H1-cell. Spike detection plots (7.c and 7.d) confirmed precise temporal alignment of detected spikes with voltage fluctuations in the raw voltage exceeding the respective thresholds. ISR analysis (7.e) of non-overlapping signal showed asymmetric activity patterns between the two H1-cells, with temporal offsets to each eye. The spike rates ranged between 110-190 Hz across trial, indicating stable decoding performance. During overlapping conditions, the dual-threshold method maintained reliable separation of left and right H1-cell activity, as evidenced by distinguishible ISR profiles (7.f).

To evaluate the algorithm in an open-loop experiment, we performed GPIO analysis using non-overlapping signals to evaluate the utility of dual H1-cell recordings for robotic control. **Figure 8** presents the results, the top panel (8.a) shows the raw voltage trace fed into the RP2040 Zero. GPIO0 (8.b) and GPIO2 (8.d) capture spike detection signals generated from high and low voltage thresholds, corresponding to left and right H1-cell activity, respectively. The Boolean collision avoidance output from the microcontroller (8.c) and (8.e) mirrors the expected spike rate patterns for bilateral H1-cell activity in **Figure 7.c**. Boolean outputs on GPIO1 and GPIO3 signal threshold crossings, serving as real-time collision avoidance triggers sent to the Raspberry Pi controller.

Dual Electrode Synchronised Recordings Synthetic dual-channel signals were generated based on a single H1-cell recording to evaluate the robustness of the proposed detection approach. Each channel produced a stable spike event after filtering and thresholding. The ISR analysis revealed that the left H1-cell showed sustained activity levels above a 100 Hz threshold for extended periods, while the ISR of right H1-cell remained below that threshold during the same period. The resulting spike detection effectively simulated dual-cell behavior, exhibiting phase-shifted responses. The time asymmetric setup allowed for precise comparison of temporal dynamics between the two channels, with spike timing delay estimated at 2 s. The signals and neural interface output for the dual-electrode approach can be found in **Appendix G**.

3.2 Robot Dynamic Test

In the dynamic test (**Figure 9**), repetitive rotations produced a characteristic 'W'-shaped curve. By identifying the y-values at the local peaks and averaging them, we obtained the approximate rotation angle for

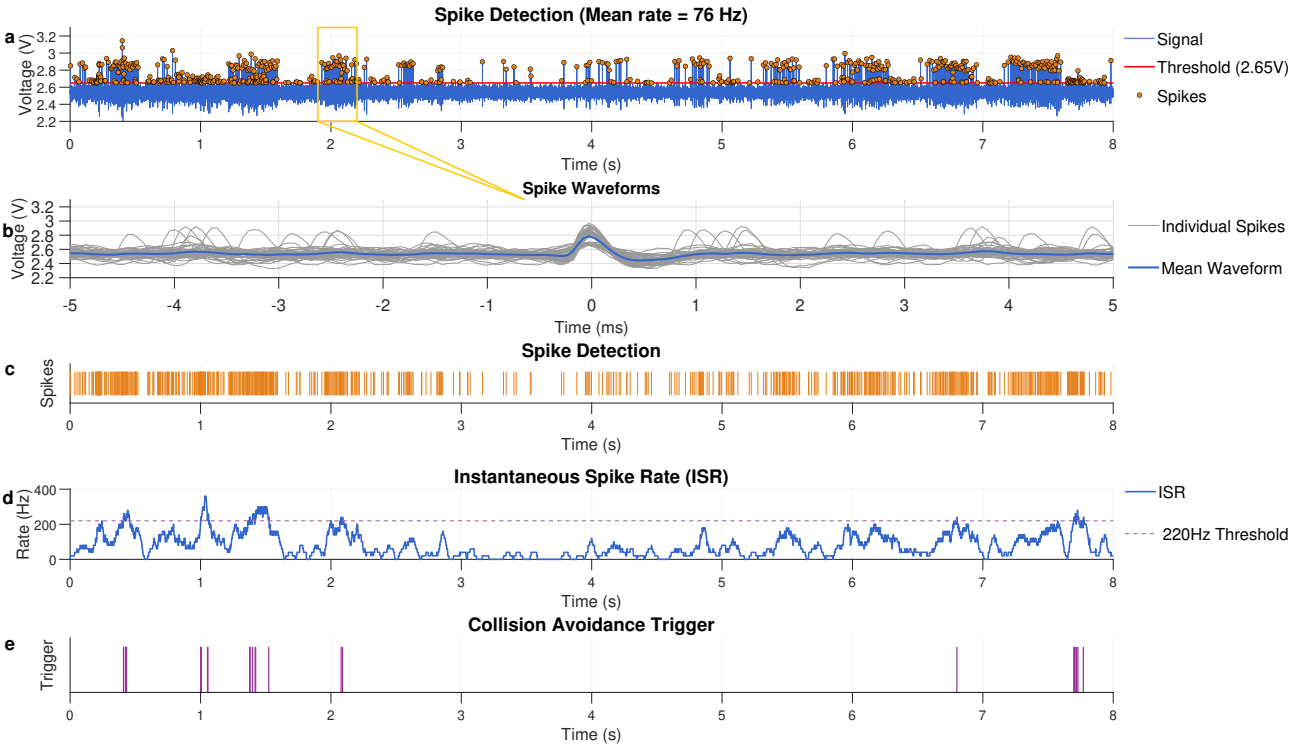


Figure 6: Raw single H1-cell recordings and outputs of the neural interface by offline processing. (a) 8 second raw extracellular voltage trace of the blowfly with back-to-front stimulus applied occasionally. Orange dots indicate detected spikes that exceed the 2.65 V detection threshold (red line) (b) Spike waveforms of 100 samples of H1-cell back-to-front motion response spike.(c) Spike detection, corresponding to when the voltage trace exceeds the threshold. (d) ISR computed with the 50 ms sliding window and a 200 Hz threshold (purple line) (e) Collision avoidance triggers (purple bars) indicate when the spike rate exceeds the spike rate threshold.

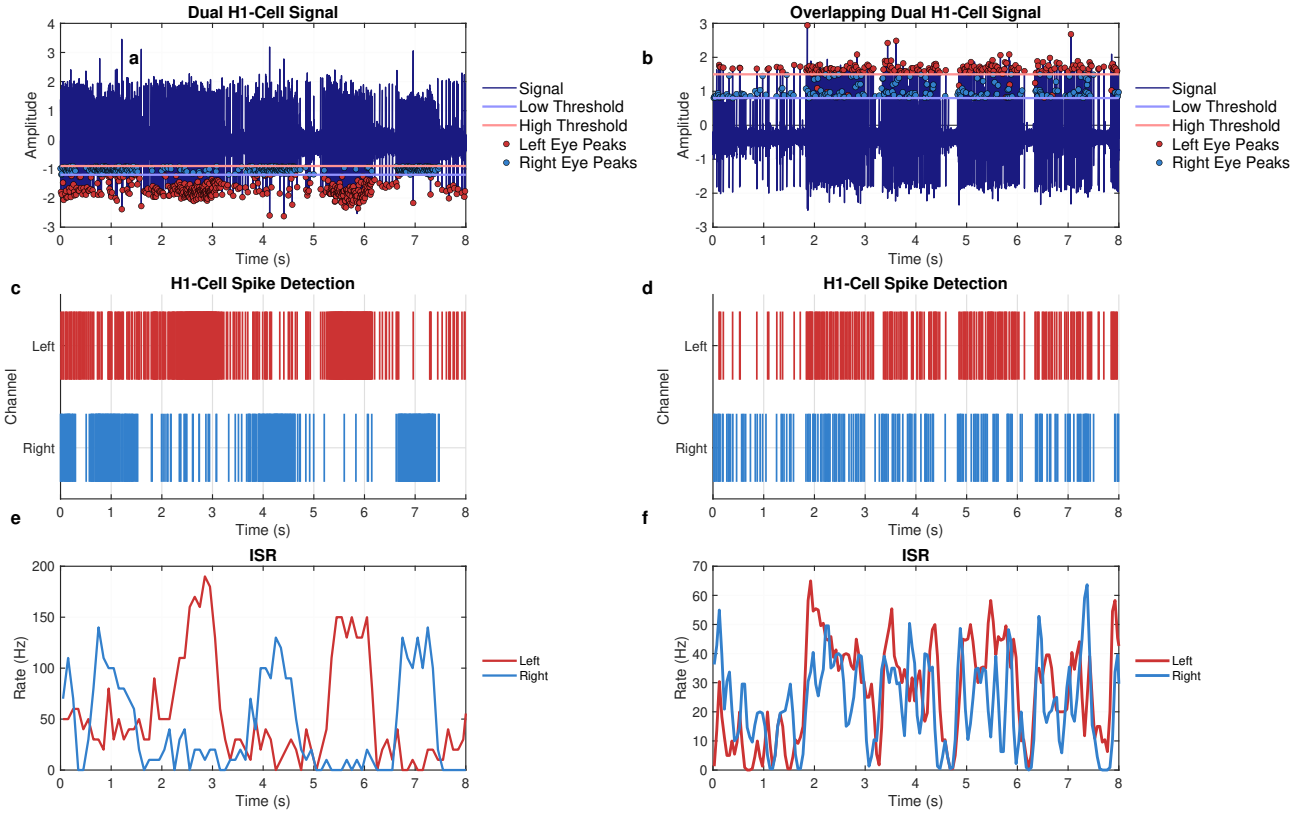


Figure 7: Raw dual H1-cell recordings and outputs of the neural interface by offline processing. Left are the non-overlapping dual H1-cell neural signal, right are the overlapping signal. (a) & (b) 10 second bandpass filtered extracellular voltage trace of the blowfly with back-to-front stimulus applied occasionally. Red dots indicate detected spikes that exceed the high threshold (red line), corresponding to left H1-cells. Blue dots indicate detected spikes that exceed the low threshold (blue line) but do not exceed the high threshold (red line), corresponding to right H1-cells. (c) & (d) Spike detection of left (red) and right (blue) H1-cells, corresponding to when the voltage trace satisfies the corresponding threshold. (e)& (f) ISR for left (red) and right (blue) H1-cells computed with the 100 ms sliding window.

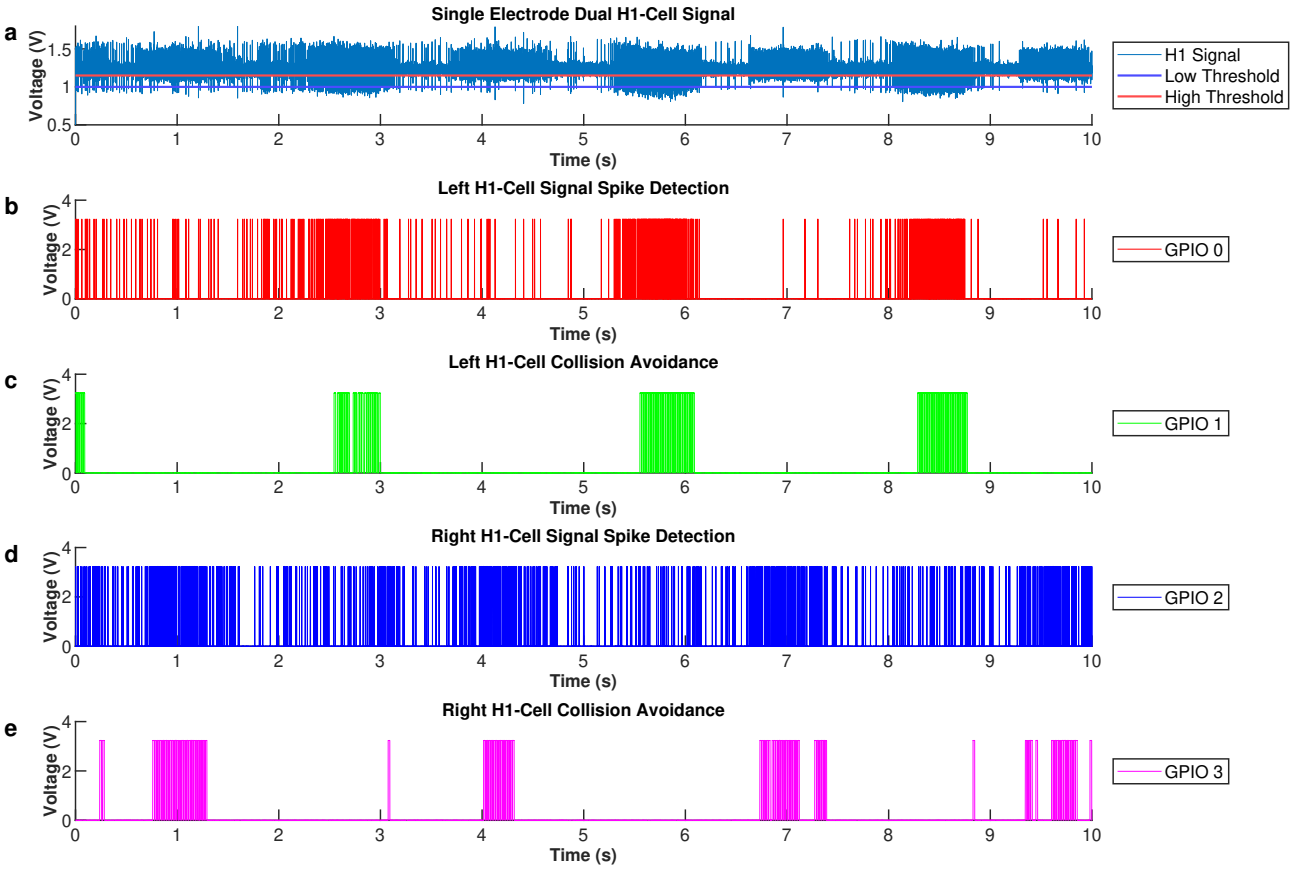


Figure 8: Raw non-overlapping dual H1-cell electrophysiological recordings and outputs of real-time processing done on RP2040 Zero neural interface. (a) 10 second extracellular voltage trace of the blowfly with back-to-front stimulus applied occasionally. The lower threshold is shown in blue, and the higher threshold in red. (b) GPIO 0 Signal from the RP2040, which corresponds to spike detection of left H1-cells when the voltage trace is lower than 1V () (b) GPIO 1 is the output of the RP2040 Neural Interface for left H1-cell processing. Green bars indicate when collision avoidance is triggered by the left H1-cell, corresponding to when the spike rate frequency exceeds 220Hz (c) GPIO 2 Signal from the RP2040, which corresponds to spike detection of right H1-cells when the voltage trace is between 1.1V and 1.15 V (d) GPIO 3 is the output of the RP2040 Neural Interface for right H1-cell processing. Purple bars indicate when collision avoidance is triggered by the right H1-cell, corresponding to when the spike rate frequency exceeds 220Hz. 3.3 V spikes correspond to the pin outputs of the RP2040

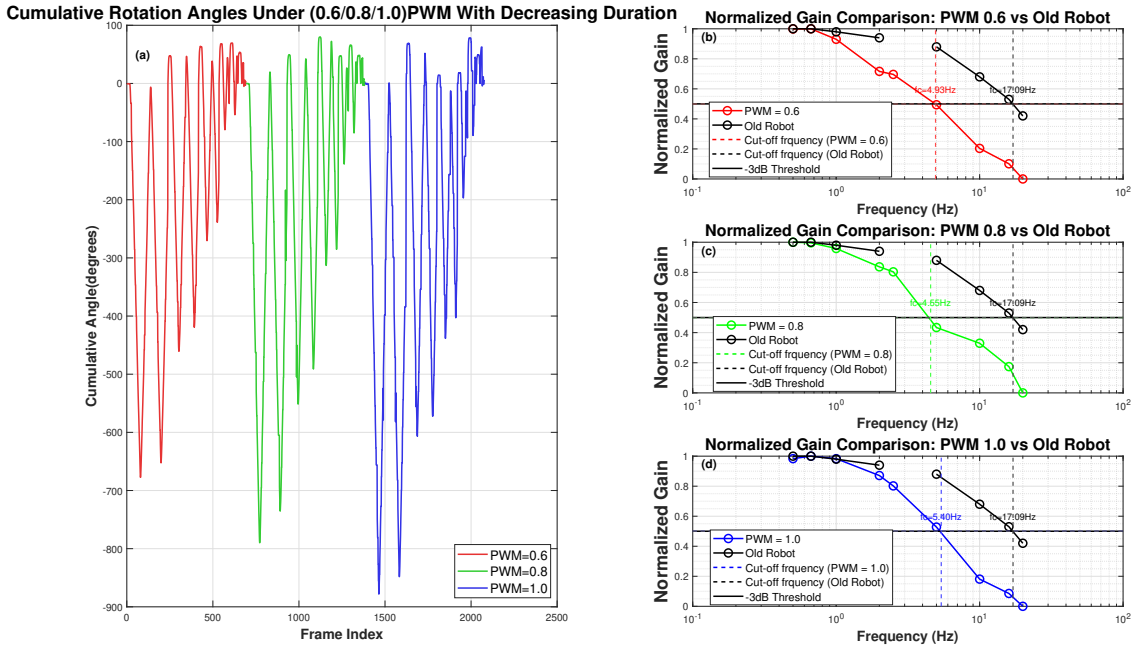


Figure 9: The robotics dynamic test result. (a) The cumulative rotation angle under different PWM rates for different time durations. The three test segments (corresponding to different PWM values) are shown sequentially without overlap to allow clear comparison. (b-d) Magnitude Bode plot of the robot dynamic under 0.6, 0.8, 1.0 PWM. Results from the wooden laser-cut version of the robot are shown in black for reference.

each combination of PWM and time duration. Detailed values are provided in Appendix F. The gain in the Bode plots was normalized using equation (3), resulting in a range from 0 to 1. Under the three PWM duty cycles (0.6, 0.8, and 1.0), the measured cutoff frequencies of the new robot were 4.93 Hz, 4.55 Hz, and 5.40 Hz, respectively. The highest cutoff frequency occurred at PWM = 1.0, indicating that at this setting, the robot achieved a stronger dynamic response and improved ability to follow higher-frequency input signals.

$$\text{Normalized Gain} = \frac{\text{Gain - Min(Gain)}}{\text{Max(Gain) - Min(Gain)}} \qquad (3)$$

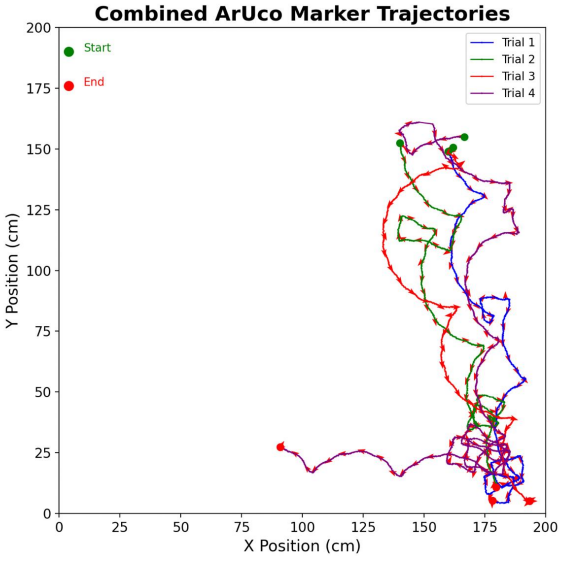


Figure 10: Trajectories of the FRI over 4 successful trials. Green dot indicates start point, dot indicates end point.

3.3 Closed-Loop Experiment

The 4 trials in **Figure 10** show trajectories of closed-loop FRI experiments, with the arena walls outlined according to the real physical space. During all 4 trials, the FRI consistently avoided collision with the arena walls and adjusted its orientation accordingly.

4 Discussion

The project aimed to demonstrate a single H1-cell driven collision avoidance system while initiating dual

H1-cell signal recording for potential integration into the FRI. In all 4 closed-loop trials, the robot successfully avoided potential wall contact. In parallel, two dual H1-cell recording strategies and algorithms were developed and tested, providing a potential for bilateral integration in future FRI implementations.

4.1 Signal Collection

4.2 Single H1-Cell Closed-Loop Experiments

Offline signal analysis confirmed that back-to-front motion stimuli increased spike rates in targeted H1-cells, supporting the notion that this neuron encodes motion cues relevant for obstacle avoidance. Trajectory analyses and spike pattern alignment (**Figure 11a and 11b**) further validated the functional targeting of the H1 cell. This supports the fundamental mechanism of this project — that the H1-cell can be used to control the trajectory of a robot.

From the 4 trials of closed-loop experiments in **Figure 10**, we can observe that the robot demonstrates consistent collision avoidance by the smooth turning trajectories and absence of contact with the arena wall. While the accuracy of collision avoidance is 100% in this case, further trials and data collection were necessary to solidify these results for statistical significance.

Prior unofficial trials had collisions of the FRI, indicating potential issues in either the fly's visual motion processing or a component of the FRI. As the blowfly is known to be an "almost perfect encoder" of vision, the likelihood of failure originating from the neural processing of fly is low. The intrinsic biological noise like photon noise, [17] synaptic noise and ion-channel noise from the fly are relative negligible compared to FRI noise sources and human error. Electromagnetic and mechanical noise associated with the arena and FRI are likely to be the main source. Shielding improvements (metal-based arena floor) and mechanical dampening (in the updated robot design) significantly improved signal quality. Trials before these improvements yielded unusable signals and constantly collided into walls.

One concern is the changing of direction of the robot in Trial 3 at t = 4.8 s despite the spike rate not crossing the predefined threshold (Figure 11a). Further experimentation must be done to see the significance of an event like this. These discrepancies may result from system latency, as shown by the 34ms delay in **Figure 12**, combined with hardware noise and angular approach effects. It is also likely that the observed collision avoidance was a result of the robot's natural oscillatory motion rather than spike rate.

Collision avoidance events were found to be strongly influenced by the robot's orientation relative to the wall. Comparison of distance-to-wall and angle-with-wall (**Figure 11b**) show that direction turning events are often met with the robot facing the wall (90°). Comparison of spike rates and angle-with-wall (**Figure 11c**) reveal that high spike rates occur when the FRI appoached the wall head-on. This directional dependence suggests that lateral or oblique approaches generate weaker back-to-front motion signals, likely insufficient to elevate the H1-cell spike rate beyond the 220 Hz threshold required to trigger avoidance. This observation aligns with the known directional selectivity of the H1-cell, which preferentially responds to motion along a specific horizontal axis. Cross-correlation analysis between spike rate and orientation angle (Figure 11d) revealed a peak at approximately 0.70 s, indicating a consistent lag between angular alignment and elevated neural activity. This delay may reflect a combination of biological integration time and system-level hardware latency (**Figure 12**).

Furthermore, the directional asymmetry observed in spike-triggered outputs may be partially explained by the use of the left H1-cell in all experiments. Visual motion originating from the left hemifield (i.e., wall approached from the left side) may not fall within the optimal receptive field of the recorded neuron, reducing the likelihood of a threshold-crossing response. This highlights a key limitation of monocular motion decoding: spatial sensitivity is restricted to a single visual hemifield. As a result, approaches from the contralateral side may be underrepresented or entirely missed. This shows the importance of incorporating bilateral H1-cell recordings, which would enable more symmetric encoding of optic flow across the full visual field and allow the system to respond effectively to motion from all directions.

As shown in **Figure 13**, comparisons between actual spike rate responses and an idealized model based on robot distance and angle revealed systematic deviations. Although general trends were captured by the model (linear decrease with angle and distance), the error distribution was non-uniform, indicating that spike responses were influenced by additional, nonlinear factors—likely including robot orientation, angular velocity, and subtle asymmetries in visual input or circuit timing.

The most significant limitation of this study was the limited number of closed-loop trials. While preliminary results are promising, further data collection under varied conditions is necessary to validate. Importantly, the observed angular sensitivity and motion preference of the single H1-cell system strongly motivate the transition to a dual H1-cell interface, providing a more complete visual field representation and reduce direction-dependent performance variability.

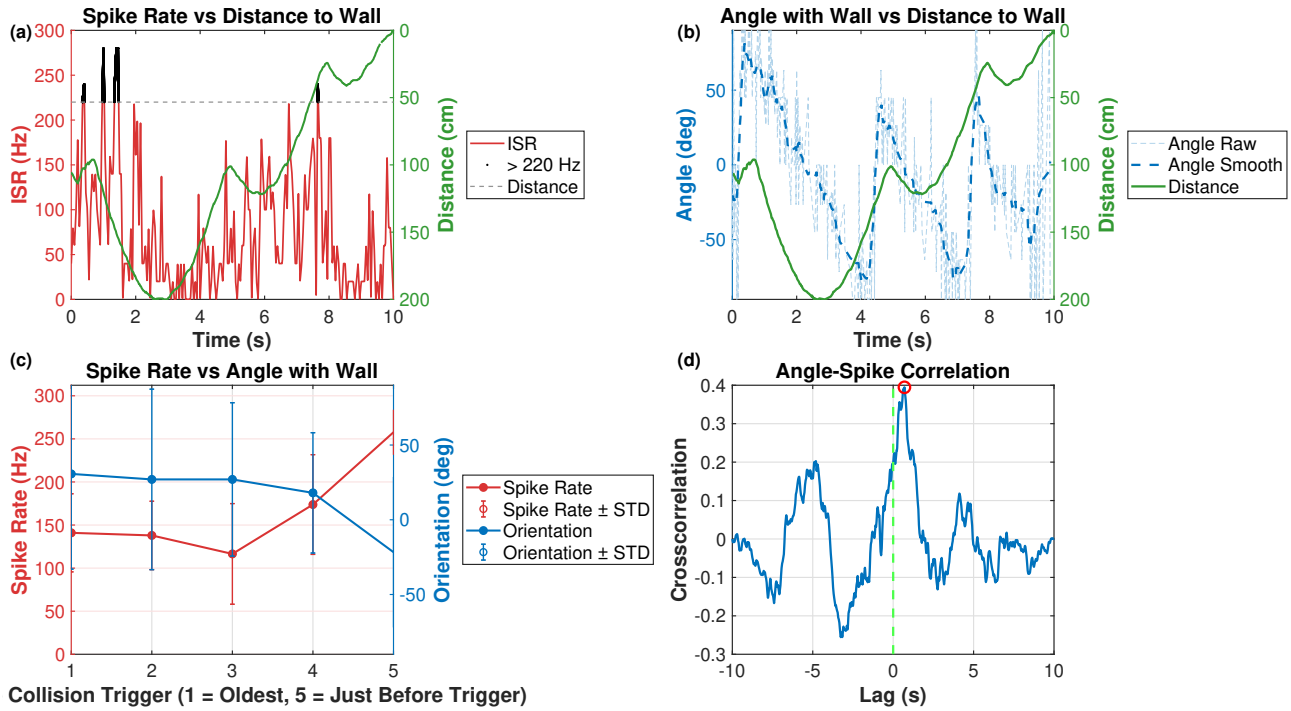


Figure 11: Spike rate (ISR) and angle dynamics in relation to wall proximity and collision-triggering events. (a) ISR, (red) over time, with a collision threshold (220 Hz, dashed) and time points exceeding this threshold (black). The green trace shows distance from the wall. Higher spike rates are observed when the animal is closer to the wall. (b) Orientation angles relative to the wall, plotted with distance to the wall (green). The smoothed angle (blue dash) trajectory reveals coordinated head alignment approaching collision. (c) Averaged ISR (red, left y-axis) and orientation (blue, right y-axis) within 5-time bins before ISR collision triggers. Shaded bars denote standard deviations. (d) Cross-correlation between angle and spike rate time series, showing a peak at 0.7008 s lag.

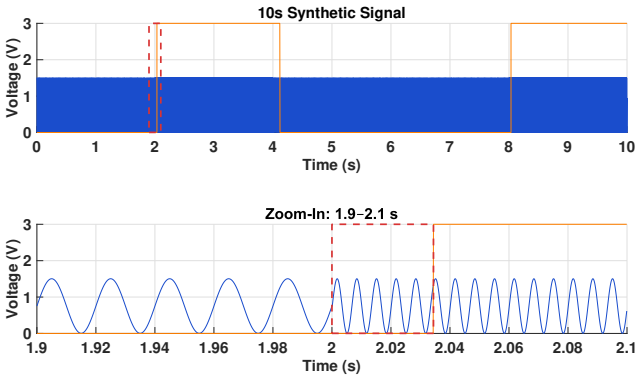


Figure 12: Synthetic stimulation and delay in recorded neural signals. Top: A 10-second synthetic signal composed of alternating sine wave blocks (100Hz and 300Hz) causing spike detection when the spike rate over 220 Hz. Bottom: Zoomed-in view (1.9–2.1 s) highlights a clear delay between the onset of the gating signal (orange) and the corresponding response in the recorded signal (blue), indicating a system-level latency (0.03441s).

4.3 Dual H1-Cell Closed-Loop Experiments

To investigate the feasibility of bilateral visual motion encoding using both H1-cells, two recording strategies were tested: (i) single-electrode recording with spike sorting, and (ii) dual-electrode recording, with one electrode targeting each H1-cell independently.

Single-electrode recordings required highly precise dissection to capture activity from both H1-cells. Spike separation was achieved through a dual-threshold algorithm based on amplitude differentiation. High-amplitude spikes were attributed to the left H1-cell and low-amplitude spikes to the right H1-cell, as confirmed by bimodal distributions in the raw voltage traces (**Figure 7a**). This allowed for effective spike separation with minimal computational load. The resulting ISRs demonstrated stable and distinct spike rate patterns from both cells, supporting effective signal segregation even in overlapping conditions.

Despite the increased complexity of signal separation, real-time neural-to-behavioural processing using the RP2040 Zero was successful (**Figure 8**). ISR computa-

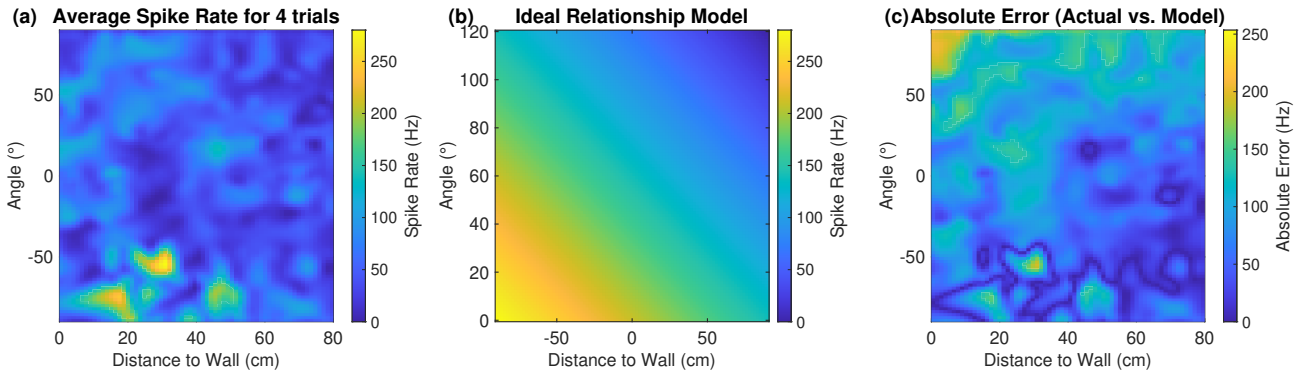


Figure 13: Comparison between actual spike rate responses and an ideal model in distance-angle space. a, Averaged spike rate responses (0–280 Hz) from four trials, shown as a function of the angle of robrelative to a nearby wall and its distance to the wall. b, A simplified model assuming a linear decrease in spike rate with increasing angle and distance. c, Absolute error between the actual and model-derived spike rates. The nonuniform distribution of errors reveals that the recorded neural responses deviate systematically from the idealized geometric prediction (94.01Hz).

tions were performed onboard, and output traces from GPIO1 and GPIO3 (**Figures 8c and 8e**) signaled collision avoidance commands at times of back-to-front motion stimuli, essentially mimicking closed-loop decision events. The alignment between ISR dynamics and GPIO command outputs demonstrated that the RP2040 is capable of decoding dual-channel H1 activity in real-time and responding with appropriate behavioral triggers.

Interestingly, phase differences were observed between the two ISR signals. While the oscillation frequencies remained similar, their temporal alignment varied, likely reflecting directionally selective motion encoding——a key feature of H1-cell dynamics. These findings are consistent with the known orientation tuning properties of lobula plate tangential cells, which support lateralized visual processing for turning behavior. [18, 19]

However, bilateral H1-cell input introduces additional complexity due to the expanded visual field and the directional tuning of each neuron. Motion signals are encoded with respect to the receptive field and optic flow geometry for each eye, resulting in non-linear integration across the visual hemifields. This includes angular velocity, yaw axis orientation, and interocular phase offsets, suggesting that future dual-cell systems must account for these spatial-temporal interactions when designing trajectory control algorithms. In addition, the Boolean GPIO signals offer limited resolution and flexibility for more complex behaviors, such as graded turning based on spike rate magnitude or temporal dynamics. Future implementations may benefit from a more refined communication protocol between the neural interface and the robotic controller—potentially integrating analog ISR transmission or higher-resolution event encoding.

As a backup strategy, the dual-electrode configuration presents a promising alternative. It simplifies the dissection procedure and eliminates the need for spike sorting as each H1-cell is recorded independently, while the computational framework remains similar to single H1-cell experiments.

4.4 Issues with Neural Interface

One limitation of the system was related to the neural interface hardware, as the RP2040 Zero lacked a built-in user interface. To address this, LED system and a buzzer was implemented to provide real-time visual and auditory feedback corresponding to neural activity and collision avoidance events. However, a critical fault was observed, caused by a broken wire between the neural interface and the master Raspberry Pi. This disrupted the closed-loop function of the FRI and highlighted the fragility of the original setup. In response, a more robust, compact robotic design was introduced (see Section 2.4.1), which reduce vulnerability to physical damage, particularly during collisions.

4.5 Robot Design

The new design prioritizes robustness, modularity and minimization of noise. All hardware components are contained within the robot to minimize the exposure of hardware and wires to reduce the risk of impact damage. This was a response to the failure of the FRI which was due to faulty wires, as mentioned in Section 4.1, which are likely to have been caused by collisions by previous 3rd Year Group Project experiments.

From a structural perspective, an integrated single-unit

printing approach helps further reduce weight and assembly complexity. To mitigate noise from the robot itself, the motor platform has been redesigned into an enclosed box-like structure, which helps prevent the amplification and propagation of noise. However, this required more materials and a much longer printing time. In addition, the four 1:7 Z-shaped damping areas in the signal collection section helped distribute the vibration generates from the motor in the lower part of the robot, overall minimizing the noise disturbance. Without the box-like structure of the motor platform and the damping areas, the real-time signals collected with the fly onboard the robot had a SNR less than 2:1, which was unsatisfactory for the experiment. In the context of code, upgrades from RPI.GPIO to gpiozero ensured safer motor current regulation.

The dynamic tests reveal that the PLA-based robot has a lower cut-off frequency than the previous wooden robot, suggesting faster performance degradation under dynamic disturbances. The wooden robot maintains better stability across broader operating conditions due to its better low-frequency response. This may be attributed to increased friction between the wheels and the PLA body, evidence by surface scratches, which reduce motor efficiency and introduces control disturbances. Additionally, wood's fibrous homogenous structure provides effective vibration damping through energy dissipation, [20] while PLA's layered structure with a low internal damping amplifies vibration transmission, increasing resonance risks. These factors—friction, damping deficiency, and structural weakness—explain the PLA robot's inferior dynamic performance and elevated instability risks. However, this does not affect the performance of obstacle avoidance maneuvering and overall the better compactness and robustness increased the reliability of the FRI.

4.6 Improvements

The main limitation in this project was the little time available. More data collection and experimentation would've given a better understanding of the FRI. Furthermore, testing across different obstacle configurations could allow us to investigate for optic flow field superposition interference.

The dual H1-cell architecture addresses a key limitation identified in earlier single-cell experiments: direction-dependent sensitivity. The right H1-cell alone showed reduced responsiveness to wall approaches from right side. By integrating signals from both left and right H1-cells, the robot could theoretically detect motion from a broader angular range and generate more symmetrical responses to environmental stimuli. Although closed-loop robotic trials using dual H1-cell input were not completed due to time constraints, the combined evidence from offline spike detection and real-time GPIO triggering supports the feasibility of this system. Future work will involve full integration of this dual-input system into the FRI, where bilateral ISR streams will simultaneously drive motor responses, enabling more refined and directionally balanced obstacle avoidance. However, a lot more research still has to be done for further downstream applications.

The newly fabricated robot demonstrates improved robustness and compactness, helping improve the reliability of the FRI even after multiple collisions. However, its reliance on a heavy power bank significantly increases its mass. Replacing it with a lightweight LiPo battery requires a buck converter for voltage regulation, though LiPo's fire/explosion risks from shorts or physical damage demand careful handling. Future priorities involve securing the battery position to prevent movement, replacing nuts with 3D-printed straw tubes for simplified assembly, and implementing compensatory hole positioning to offset printing inaccuracies. In addition, other materials would be explored to find the best frequency response.

4.7 Conclusion

In summary, we successfully replicated the single H1-cell FRI collision avoidance experiment in the arena using real-time extracellular signals from the fly. We built a robot that demonstrated better stability with less noise, making signal fidelity much higher, contributing to the high accuracy. Additionally, we explored the possibility of implementing real-time dual H1-cell recordings to the FRI. We found the efficacy of the algorithms and its feasability for downstream applications, helping to advance the field of biological sensorimotor principles and create further progress to bioinspired robotics and maneuvering.

References

[1] Alexander Borst, Jürgen Haag, and Alex S. Mauss. How fly neurons compute the direction of visual motion. *Journal of Comparative Physiology. A, Neuroethology, Sensory, Neural, and Behavioral Physiology*, 206(2):109–124, March 2020.

[2] Auke J. Ijspeert. Biorobotics: Using robots to emulate and investigate agile locomotion. *Science*, 346(6206):196–203, October 2014.

[3] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison,

Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, January 2022.

[4] Roland Kern, Norbert Boeddeker, Laura Dittmar, and Martin Egelhaaf. Blowfly flight characteristics are shaped by environmental features and controlled by optic flow information. *Journal of Experimental Biology*, 215(14):2501–2514, July 2012.

[5] Jiaqi V. Huang and Holger G. Krapp. Fly H1-cell distance estimation in a monocular virtual reality environment. In Fabian Meder, Alexander Hunt, Laura Margheri, Anna Mura, and Barbara Mazzolai, editors, *Biomimetic and Biohybrid Systems*, pages 325–337, Cham, 2023. Springer Nature Switzerland.

[6] Mingyu Wu, Che Fai Yeong, Eileen Lee Ming Su, William Holderbaum, and Chenguang Yang. A review on energy efficiency in autonomous mobile robots. *Robotic Intelligence and Automation*, 43(6):648–668, September 2023.

[7] Holger G. Krapp, Roland Hengstenberg, and Martin Egelhaaf. Binocular Contributions to Optic Flow Processing in the Fly Visual System. *Journal of Neurophysiology*, 85(2):724–734, February 2001.

[8] M. O. Franz and H. G. Krapp. Wide-field, motion-sensitive neurons and matched filters for optic flow fields. *Biological Cybernetics*, 83(3):185–197, September 2000.

[9] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. Vision-based control of near-obstacle flight. *Autonomous Robots*, 27(3):201–219, October 2009.

[10] Julien R. Serres and Franck Ruffier. Optic flow-based collision-free strategies: From insects to robots. *Arthropod Structure & Development*, 46(5):703–717, September 2017.

[11] Matthew M. Parsons, Holger G. Krapp, and Simon B. Laughlin. Sensor fusion in identified visual interneurons. *Current Biology*, 20(7):624–628, April 2010.

[12] K. Hausen. Functional characterization and anatomical identification of motion sensitive neurons in the lobula plate of the blowfly calliphora erythrocephala. *Zeitschrift für Naturforschung C*, 31(9-10):629–634, October 1976.

[13] Jiaqi V Huang, Yiran Wei, and Holger G Krapp. A biohybrid fly-robot interface system that performs active collision avoidance. *Bioinspiration & Biomimetics*, 14(6):065001, September 2019.

[14] Scott F Lempka, Matthew D Johnson, Michael A Moffitt, Kevin J Otto, Daryl R Kipke, and Cameron C McIntyre. Theoretical analysis of intracortical microelectrode recordings. *Journal of Neural Engineering*, 8(4):045006, July 2011.

[15] Rodrigo Quian Quiroga. Spike sorting. *Current Biology*, 22(2):R45–R46, January 2012.

[16] Matilda Gibbons, Andrew Crump, Meghan Barrett, Sajedeh Sarlak, Jonathan Birch, and Lars Chittka. Chapter three - can insects feel pain? A review of the neural and behavioural evidence. In Russell Jurenka, editor, *Advances in Insect Physiology*, volume 63, pages 155–229. Academic Press, January 2022.

[17] Jan Grewe, Jutta Kretzberg, Anne-Kathrin Warzecha, and Martin Egelhaaf. Impact of photon noise on the reliability of a motion-sensitive neuron in the fly's visual system. *The Journal of Neuroscience*, 23(34):10776–10783, November 2003.

[18] Holger G. Krapp and Roland Hengstenberg. Estimation of self-motion by optic flow processing in single visual interneurons. *Nature*, 384(6608):463–466, December 1996.

[19] Klaus Hausen. Motion sensitive interneurons in the optomotor system of the fly. *Biological Cybernetics*, 45(2):143–156, September 1982.

[20] Novel wood-based functional material with vibration and noise reduction | ACS applied materials & interfaces. https://pubs.acs.org/doi/10.1021/acsami.3c06508?utm_source

5 Appendix

5.1 Appendix A: Acronym Nomenclature

Acronym	Definition
FRI	Fly Robot Interface
GPIO	General-Purpose Input/Output
ISR	Instantaneous Spike Rate
LiDAR	Light Detection and Ranging
PWM	Pulse Width Modulation
SNR	Signal-to-Noise Ratio

5.2 Appendix B: Project Management

5.2.1 Departures from Project Planning

Initially, the plan was to divide the project into subgroups for different components. While this structure worked well initially, the high interdependency between subgroups meant that frequent collaboration was necessary. Furthermore, varying schedules of individual members meant that there was often a need to move around members. As a result, the subgroups were less defined as the project moved on. To ensure that organization was not being compromised however, we elected a lead for each task for communication and management responsibilities, as shown below.

Member	Role
Yi Zhang	Robotics Design
Wenjun Jiao	Dynamic Test
Zeyuan Xin	Neural Interface
Yuichiro Minamikawa	Pitch Leader
Changyu Hu	Image Processing
Shuchang Zhang	Dissection
Badriyah Islam	Background Research

Secondly, although we were ahead of schedule in getting individual components of the project to function, we encountered delays when attempting to integrate the system as a whole. This was ultimately traced back to a faulty connection wire, which caused us to fall slightly behind in the data collection and testing stages. This resulted in a tighter timeline towards the end of the project, however with flexibility and effective communication, our group was successfully able to complete the project.

5.2.2 Project Management Lessons

1. Importance of Communication within the Group With the project having multiple parts, it was natural to form subgroups and work on completely different areas, based on skills and interest. While this was initially effective, there was a very high dependency between subgroups which required consistent updates to ensure alignment and progress. For example, for the robot group to test with live data, this was dependent on results of the dissection and neural interface group. There were often where one subgroup faced difficulties and thus conducting complete FRI experiments would be challenging. To address this, we implemented a lab logbook, where all progress, changes, and questions were recorded. This ensured that all members understood the progress and issues were quickly addressed by discussing during supervisor meetings.

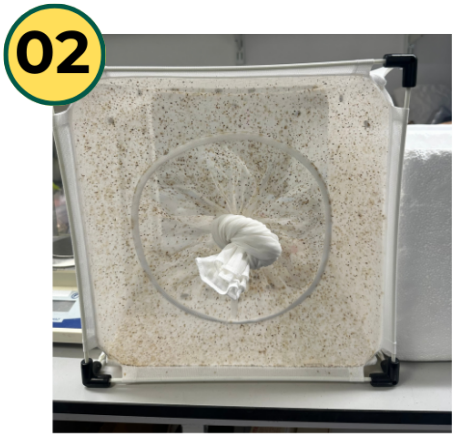
2. Purpose of Meetings Regular meetings were held from the outset, with the frequency remaining constant. However, the outcomes of these meetings grew significantly as the project progressed. A crucial part of meetings is to update progress and plan moving forward. However, more important is having a clear agenda for the meeting and ensuring all those points are cleared by the end led to better momentum and outcomes. Rather than spending an hour discussing what needs to be done for the next week, more effective was to pre-prepare beforehand what needs to be discussed and spending the hour discussing how to solve these. For instance, during the project mind map session, we decided that completing the task together in the meeting was more efficient than just allocating tasks, doing it separately and ultimately deferring it. In addition, making the most out of supervisor meetings by having questions noted throughout the week was deemed effective. By adopting this approach, our meetings became more focused and productive, actively creating output and eliminating redundant communication throughout the week.

3.Adaptability and Contingency Planning Although the project had been successfully completed in the past, we faced challenges such as issues with the neural interface and a collapsing robot. In addition, with deadlines from other modules, availability of our members varied throughout the year. Hence, instead of fixed roles and subgroups, we opted for a more flexible group structure, to allow members to move around to different subgroups when facing challenges. In addition, each member was a lead for a specific role or task, to high-level organization tasks, with every member contributing. This adaptability enabled us to redesign and manufacture a completely new robot, debug the neural interface and work on the project pitch presentation simultaneously. Despite the unexpected nature of these challenges, our ability to adjust and communicate effectively allowed us to incorporate these changes into the project timeline successfully.

5.3 Appendix C: Dissection Guide

A Guide on preparing Fly for Dissection

Prepare the ice bath and a lid with double sided tap under the microscope



Capture the fly in the test tube. It's easier to aim for the flies at the top.

Place faced down into the ice bath, allowing some ice into the test tube and wait for the fly to stop moving around

If the fly continues to move. Fill a cup with water and ice, and pour the content of the test tube (ice and fly) into the cup.

Add some more ice on top and the fly will soon slow down

Using tweezers, grab the fly's leg and place onto the lid under the microscope pushing the wings down firmly

07

Using the hot wax, place a drop on the side of the wing, sealing the wing to the body. This prevents the fly from moving. Do this on both sides

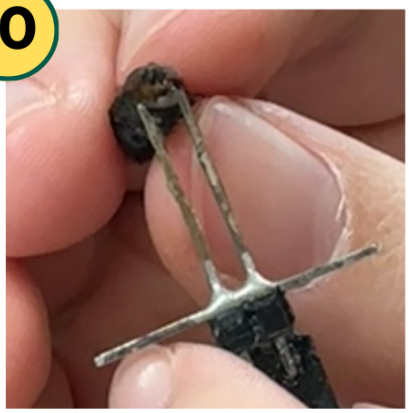
08

Using the microscope, take a closer look at the fly. Holding the leg with forceps, use the scissors to clip the legs as close to the body as possible.

09

Use the hot wax to seal the holes by the legs of the fly to prevent drying out. Only use small droplets.

10



Place the fly's head directly centred on the holder, being careful to not remove the fly's head.

11

Push the body slightly to expose the back of the neck.

12

Use hot wax to seal either side of the fly's head to the holder

5.4 Appendix D: Videos

Video Description	Link
H1-Cell Back-to-front Behaviour	https://youtu.be/tAFEsI0npeU
Trajectory Video T7	https://youtu.be/zrm-TPD97To
Trajectory Video T8	https://youtu.be/z51T_oZpx3U
Trajectory Video T9	https://youtu.be/ZuaGfyNw4c4
Trajectory Video T10	https://youtu.be/NuDsnk3uSPk

5.5 Appendix E: Table of Robot Materials

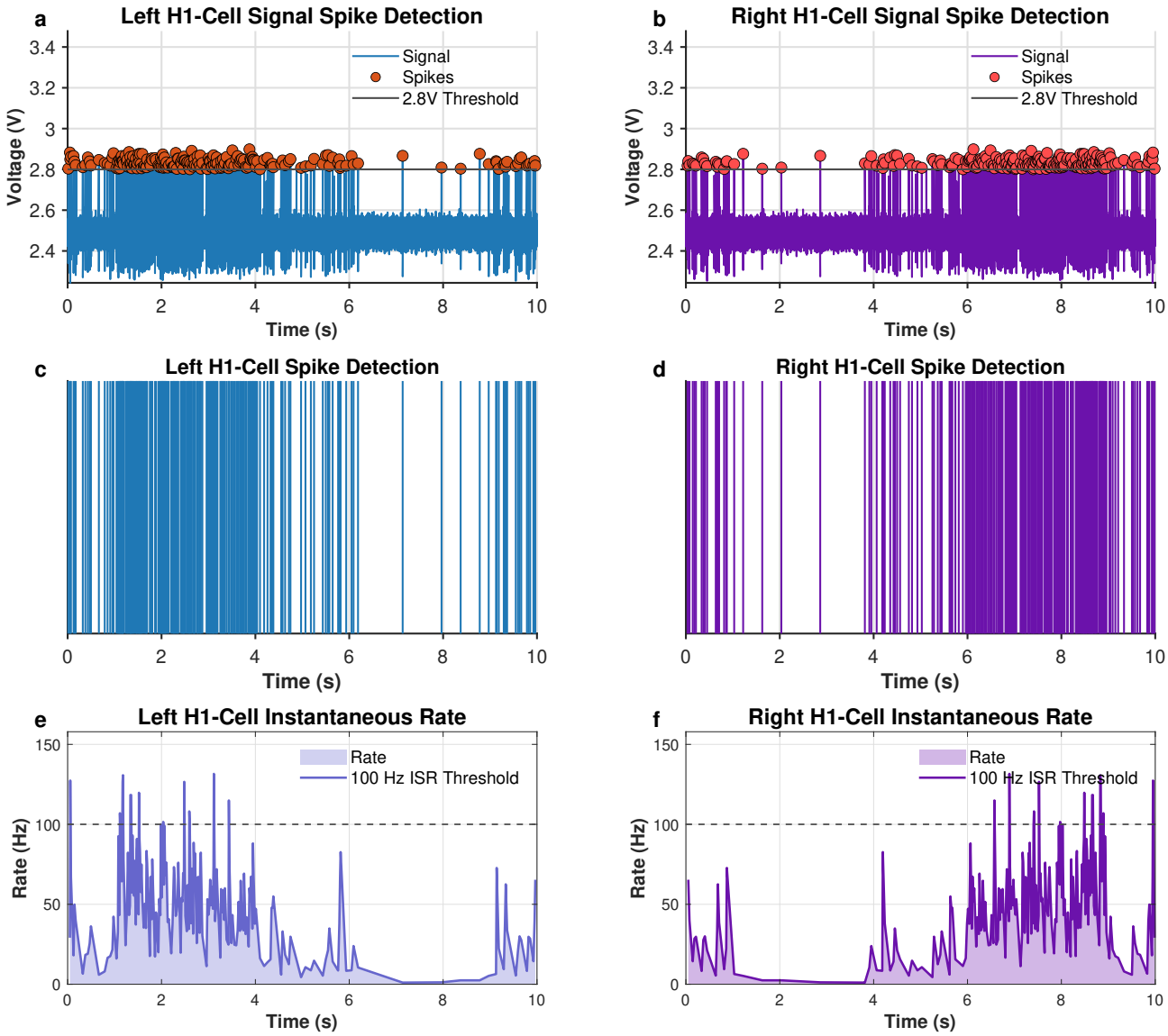
Part	3D Printer / Laser Cutting	Material	Filament Density	Filament Pattern	Temperature
Aruco Marker Holder	Bambu Lab A1 3D Printer (0.4 mm nozzle)	PLA	15%	Grid	Nozzle: 215℃ Bed: 60℃
Hardware Platform					
Motor Platform			50%		
Wheel	Laser Cutting	Wood	N/A	N/A	N/A
Signal Collection Platform					

Table 1: Fabrication methods and parameters for various system components.

5.6 Appendix F: Dynamic Test Result

PWM	Duration	2	1.5	1	0.5	0.4	0.2	0.1	0.0625	0.05
Frequency (1/T)		0.5	0.667	1	2	2.5	5	10	16	20
Angular Velocity (rad/s)		ω_{in}								
Input Angle		0.5	0.667	1	2	2.5	5	10	16	20
0.6	Output angle (degree)	673.48	506.3	320.995	135.385	106.53	43.775	15.105	7.945	5.175
	Normalized Gain	0.999	1	0.9301	0.7164	0.6963	0.4941	0.2035	0.101	0
	Cut-off Frequency	4.93Hz								
	Resonance Frequency	0.67Hz								
0.8	Output angle (degree)	799.035	596.725	386.865	174.95	135.95	45.59	19.64	9.34	4.835
	Normalized Gain	1	0.9958	0.9584	0.8367	0.8035	0.4338	0.3295	0.1743	0
	Cut-off Frequency	4.65Hz								
	Resonance Frequency	0.5Hz								
1	Output angle (degree)	878.195	666.56	439.015	199.635	150.55	55.78	15.735	7.74	4.69
	Normalized Gain	0.9835	1	0.9832	0.871	0.802	0.5277	0.1812	0.0856	0
	Cut-off Frequency	5.40Hz								
	Resonance Frequency	0.67Hz								
0.6 (Old Robot)	Output angle (degree)	815.1	612.39	398.68	192.75	/	71.44	27.57	13.62	8.56
	Normalized Gain	1	1	0.98	0.94	/	0.88	0.68	0.53	0.42
	Cut-off Frequency	17.09Hz								
	Resonance Frequency	0.50Hz								

5.7 Appendix G: Dual-electrode Dual H1-cell Signal Processing Results



5.8 Appendix H: Neural Interface Code

5.8.1 Single H1-Cell

```
#include <Arduino.h>
extern "C" {
#include "hardware/adc.h"
}
#include <hardware/timer.h>  // Timer Related Operations
#include <hardware/irq.h>    // Disruption related operation
#include <stdio.h>           // standard input and output library
#include "pico/stdlib.h"     // standard Raspberry Pi Pico library
#include "hardware/gpio.h"   // GPIO related operation library
// #include "hardware/adc.h"   // ADC related libbbrary
#include <Adafruit_NeoPixel.h>  // Third-party libraries for controlling LEDs
#define ALARM_NUM 1             // Timer number
#define ALARM_IRQ TIMER_IRQ_1   // Define the interrupt source as timer interrupt 1
#define ALARM_FREQ 1000         // Timer frequency (used for test led)
#define PIN 16                  // LED test footnote
#define NUMPIXELS 1             // Number of LED that been used

const int inputPin = 29;  //  G P I O 2 8 correspond  to  A D C 2 read  analoge input
const int NIDAQ_AI1 = 0;  //  G P I O 0 spike  rate detection output
const int NIDAQ_AI2 = 1;  //  G P I O 1 spike  rate signal output
const int NIDAQ_AI3 = 2;
const int NIDAQ_AI4 = 3;
const float thresholdVoltage = 2.65;  // action potential voltage threshold**

int buffer[1000] = { 0 };   // Moving windows, used to store the result of spike
    detection
int bufferIndex = 0;        // Current buffer index
float sensorValue = 0;      // Sensor reading value
float voltage = 0;          // The voltage value after convertion
float previousVoltage = 0;  // Previous/last voltage value
int spikeRate;              // spike rate
int spikeCount = 0;         // number of spike
uint32_t alarmPeriod;       // Timer interrupt period
int debounce_flag = 0;      // debounce sign.  Debounce: When you press the button in
     any machine,
                            //There are vibration when two metal sheet collide,
                               microcomputer will detect multiple squre wave that
                               should not exist in the result(noise)
int debounce_counter = 0;   // debounce counter
int LED_BUILTIN = 0;
volatile bool spikeHigh = 0;
volatile uint32_t outputCounter =0;
const uint32_t Duration = 200;

Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);  // Initialize LED
enum { NONE,
       RED };         // LED colour enumeration
int ledColor = NONE;  // recent LED color

//Timer initialization function
static void alarm_in_us_arm(uint32_t delay_us) {
  uint64_t target = timer_hw->timerawl + delay_us;  // Calculate target time
  timer_hw->alarm[ALARM_NUM] = (uint32_t)target;    // Setting the timer trigger time
}
static void alarm_in_us(uint32_t delay_us) {
  irq_set_exclusive_handler(ALARM_IRQ, alarm_irq);            // Set the IRQ
      function
  irq_set_enabled(ALARM_IRQ, true);                           // Enable IRQ
```

```
  timer_hw->alarm[ALARM_NUM] = timer_hw->timerawl + delay_us;  // Time for Timer
      trigger
  hw_set_bits(&timer_hw->inte, 1u << ALARM_NUM);               // Enable specific
      timer IRQ
}


void setup() {
  pinMode(NIDAQ_AI1, OUTPUT);
  pinMode(NIDAQ_AI2, OUTPUT);
  // for (int i=0 ; i<10000; i++){
  //    digitalWrite(0, HIGH);
  //    digitalWrite(1, LOW);
  //    delay(5000);
  //    digitalWrite(0, LOW);
  //    digitalWrite(1, HIGH);
  //    delay(5000);
  // }
  analogReadResolution(12);  // Set ADC resolution to 12bit

  pinMode(29, INPUT);
  //pinMode(2, OUTPUT);            // Configure GPIO2 for output mode
  //digitalWrite(NIDAQ_AI3, LOW);          // Set GPIO2 to low
  //pinMode(NIDAQ_AI4, OUTPUT);            // Configure GPIO3 for output mode
  pinMode(LED_BUILTIN, OUTPUT);  // Configure the on-board LEDs for output mode
  alarmPeriod = 50;              // Timer interrupt period of 50 microseconds
  alarm_in_us(alarmPeriod);      // Set timer interrupt
  pixels.begin();                // Initialize the led
  // pixels.clear();
  // pixels.show();
  adc_init();                          // Initialize ADC hardware
  adc_gpio_init(inputPin);             // Configure GPIO28 as ADC pinout
  adc_select_input(3);                 // Select ADC2 as input
  // gpio_init(NIDAQ_AI1);                // Initialize GPIO0
  // gpio_set_dir(NIDAQ_AI1, GPIO_OUT);  // Set GPIO0 as output
  // gpio_init(NIDAQ_AI2);                // Initialize GPIO1
  // gpio_set_dir(NIDAQ_AI2, GPIO_OUT);  // Set GPIO1 as output
}



//ISR
static void alarm_irq(void) {
  hw_clear_bits(&timer_hw->intr, 1u << ALARM_NUM);  // clear interrupt sign
  alarm_in_us_arm(alarmPeriod);                     // reset interrupt time
  sensorValue = adc_read();                         // Read ADC value

  float voltage = sensorValue * (3.3 / 4095);  // Convert the value of ADC to voltage
  digitalWrite(NIDAQ_AI1, LOW);                // Default close output 0

  //Spike detection

  if (voltage > thresholdVoltage && previousVoltage <= thresholdVoltage) {
    // if (!debounce_counter) {

    digitalWrite(NIDAQ_AI1, HIGH);
    buffer[bufferIndex] = 1;  // debounce, avoid continuous spikes
    //   debounce_counter = 50;
    // }
  } else {
    buffer[bufferIndex] = 0;  // No detection of spike
    digitalWrite(NIDAQ_AI1, LOW);
  }
```

```
  // if (debounce_counter > 0) {
  //     debounce_counter--;

  // }
  // debounce
  // if (debounce_flag == 1) {
  //   debounce_counter++;
  //     if (debounce_counter == 50) { // Ensure no detection of spike during 50
      smapling period
  //         debounce_flag = 0;
  //     }
  // }

  previousVoltage = voltage;
  bufferIndex = (bufferIndex + 1) % 1000;
  spikeCount = 0;
  for (int i = 0; i < 1000; i++) {
    spikeCount += buffer[i];
  }
  // Calculate spike rate over the buffer's period
  spikeRate = spikeCount * 20;  // 20000 samples per second


  if (spikeRate > 200) {
    if (!spikeHigh) {
      spikeHigh = true;
      outputCounter = Duration;
      digitalWrite(NIDAQ_AI2, HIGH);  // Set HIGH once
    }
} else {
  if (outputCounter > 0) {
    outputCounter--;
    if (outputCounter == 0) {
      digitalWrite(NIDAQ_AI2, LOW);
      spikeHigh = false;
    }
  }
}

  // digitalWrite(3, digitalRead(3)^1);
}

void loop() {
  // Main loop is empty, all logic is handled in interrupts
}
```

5.8.2 Dual H1-Cell

```
#include <Arduino.h>
extern "C" {
#include "hardware/adc.h"
}
#include <hardware/timer.h>
#include <hardware/irq.h>
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include <Adafruit_NeoPixel.h>
#include <math.h>

#define ALARM_NUM 1
```

```
#define ALARM_IRQ TIMER_IRQ_1
#define ALARM_FREQ 1000
#define PIN 16
#define NUMPIXELS 1
// #define FIR_FL 51 // The length of FIR filter
#define FS 20000
#define buffersize 1000

const int inputPin = 29; //ADC Input Pin
const int NIDAQ_AI1 = 0;
const int NIDAQ_AI2 = 1;
const int NIDAQ_AI3 = 2;
const int NIDAQ_AI4 = 3;
const float threshold_low = 1;
const float threshold_high = 1.15;
int sumhigh = 0;
int sumlow = 0;

int HighBuffer[buffersize] = {0};
int LowBuffer[buffersize] = {0};
int cIndex = 0;
int high = 0;
int low = 0;

float sensorValue = 0;
// float voltage = 0;
float previousVoltage = 0;
float currentVoltage = 0;
static bool firstRun = true;

// float fir_h[FIR_FL]; //Coefficient of FIR
// float fir_buffer[FIR_FL]; //Filter input data buffer
// volatile int fir_index = 0;

int spikeRate;
uint32_t alarmPeriod;
int LED_BUILTIN = 6;
volatile bool spikeHigh = 0;
volatile uint32_t outputCounter = 0;
const uint32_t Duration = 200;

Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
enum { NONE, RED };
int ledColor = NONE;

// // FIR Filter Coefficients (Bandpass 300Hz - 3000Hz)
// void init_fir(void) {
//     int M = FIR_FL - 1;
//     float omega1 = 2.0f * 3.14159265358979323846 * 300.0f / FS;
//     float omega2 = 2.0f * 3.14159265358979323846 * 3000.0f / FS;
//     float sumh = 0.0f;
//     for (int n = 0; n <= M; n++) {
//         float h_val;
//         if (n == M / 2) {
//             h_val = (omega2 - omega1) / 3.14159265358979323846;
//         } else {
//             h_val = (sinf(omega2 * (n - M / 2)) - sinf(omega1 * (n - M / 2))) /
    (3.14159265358979323846 * (n - M / 2));
//         }

//         float w = 0.54f - 0.46f * cosf(2.0f * 3.14159265358979323846 * n / M);
//         h_val *= w;
//         fir_h[n] = h_val;
```

```
//         sumh += h_val;
//     }

//     for (int n = 0; n < FIR_FL; n++) {
//         fir_h[n] /= sumh;
//     }
// }

static void alarm_in_us_arm(uint32_t delay_us) {
    uint64_t target = timer_hw->timerawl + delay_us;
    timer_hw->alarm[ALARM_NUM] = (uint32_t)target;
}

static void alarm_in_us(uint32_t delay_us) {
    irq_set_exclusive_handler(ALARM_IRQ, alarm_irq);
    irq_set_enabled(ALARM_IRQ, true);
    timer_hw->alarm[ALARM_NUM] = timer_hw->timerawl + delay_us;
    hw_set_bits(&timer_hw->inte, 1u << ALARM_NUM);
}

static void alarm_irq(void) {
    hw_clear_bits(&timer_hw->intr, 1u << ALARM_NUM);
    alarm_in_us_arm(alarmPeriod);
    high = 0;
    low = 0;
    sensorValue = adc_read();
    float voltage = sensorValue * (3.3 / 4095);
    // float norm = 2.0f*(voltage/3.3f) - 1.0f;
    // fir_buffer[fir_index] = norm;
    // fir_index = (fir_index + 1) % FIR_FL;

    // float norm = 2.0f*(voltage/3.3f) - 1.0f;
    // fir_buffer[fir_index] = norm;
    // fir_index = (fir_index + 1) % FIR_FL;

    // float filtered_signal = 0.0f;
    // int idx = fir_index;
    // for(int n = 0; n< FIR_FL;n++){
    //   filtered_signal += fir_h[n]*fir_buffer[idx];
    //   idx = (idx + 1) %FIR_FL;
    // }
    if (firstRun) {
    previousVoltage = voltage;
    currentVoltage = voltage;
    firstRun = false;
    }
    else {

    if (previousVoltage > currentVoltage && voltage >= currentVoltage) {
    if (currentVoltage < threshold_low) {
        digitalWrite(NIDAQ_AI1, HIGH);
        digitalWrite(NIDAQ_AI3, LOW);
        high = 1;
        low = 0;
    } else if (currentVoltage > threshold_low && currentVoltage < threshold_high) {
        digitalWrite(NIDAQ_AI1, LOW);
        digitalWrite(NIDAQ_AI3, HIGH);
        high = 0;
        low = 1;
    } else {
        digitalWrite(NIDAQ_AI1, LOW);
        digitalWrite(NIDAQ_AI3, LOW);
        high = 0;
```

```
        low = 0;
    }
} else {
    digitalWrite(NIDAQ_AI1, LOW);
    digitalWrite(NIDAQ_AI3, LOW);
    high = 0;
    low = 0;
}
    previousVoltage = currentVoltage;
    currentVoltage = voltage;

    HighBuffer[cIndex] = high;
    LowBuffer[cIndex] = low;
    cIndex = (cIndex + 1) % buffersize;
    sumhigh = 0;
    sumlow = 0;

    for (int i = 0; i < buffersize; ++i) {
        sumhigh += HighBuffer[i];
        sumlow += LowBuffer[i];
    }
    int highrate = sumhigh * (FS / buffersize);  // spike/s
    int lowrate = sumlow * (FS / buffersize);
    bool highExceeds = (highrate > 220);
    bool lowExceeds  = (lowrate > 220);

    if (outputCounter == 0) {
    if (highExceeds || lowExceeds) {
        outputCounter = Duration;
        spikeHigh = true;

        if (highExceeds && lowExceeds) {
            digitalWrite(NIDAQ_AI2, HIGH);
            digitalWrite(NIDAQ_AI4, HIGH);
        } else if (highExceeds) {
            digitalWrite(NIDAQ_AI2, HIGH);
            digitalWrite(NIDAQ_AI4, LOW);
        } else if (lowExceeds) {
            digitalWrite(NIDAQ_AI2, LOW);
            digitalWrite(NIDAQ_AI4, HIGH);
        }
    }
  } else {
      outputCounter--;
      if (outputCounter == 0) {
          spikeHigh = false;
          digitalWrite(NIDAQ_AI2, LOW);
          digitalWrite(NIDAQ_AI4, LOW);
      }
  }
    }
}
void setup() {
    pinMode(NIDAQ_AI1, OUTPUT);
    pinMode(NIDAQ_AI2, OUTPUT);
    pinMode(NIDAQ_AI3, OUTPUT);
    pinMode(NIDAQ_AI4, OUTPUT);
    // for (int i=0 ; i<10000; i++){
    //  digitalWrite(NIDAQ_AI1, HIGH);
    //  digitalWrite(NIDAQ_AI2, LOW);
    //  digitalWrite(NIDAQ_AI3, HIGH);
    //  digitalWrite(NIDAQ_AI4, LOW);
    //  delay(5000);
```

```
    //  digitalWrite(NIDAQ_AI1, LOW);
    //  digitalWrite(NIDAQ_AI2, HIGH);
    //  digitalWrite(NIDAQ_AI3, LOW);
    //  digitalWrite(NIDAQ_AI4, HIGH );
    //  delay(5000);
    // }
    analogReadResolution(12);
    pinMode(inputPin, INPUT);
    pinMode(LED_BUILTIN, OUTPUT);

    alarmPeriod = 50;
    alarm_in_us(alarmPeriod);

    pixels.begin();
    adc_init();
    adc_gpio_init(inputPin);
    adc_select_input(3);

    // gpio_init(NIDAQ_AI1);
    // gpio_set_dir(NIDAQ_AI1, GPIO_OUT);
    // gpio_init(NIDAQ_AI2);
    // gpio_set_dir(NIDAQ_AI2, GPIO_OUT);
    // gpio_init(NIDAQ_AI3);
    // gpio_set_dir(NIDAQ_AI3, GPIO_OUT);
    // gpio_init(NIDAQ_AI4);
    // gpio_set_dir(NIDAQ_AI4, GPIO_OUT);
    // init_fir();

}

void loop() {

}
```

5.9 Appendix I: Robot Control Code

```
from gpiozero import Robot, PWMOutputDevice, DigitalInputDevice
from time import sleep, time
import keyboard

# Define H-bridge pin numbers on Pi (BCM numbering)
robot = Robot(left=(20, 26), right=(23, 24))

# Define PWM pin numbers for both motors (BCM numbering)
enable_left = PWMOutputDevice(12)  # BCM 12 (PWM for left motor)
enable_right = PWMOutputDevice(13)  # BCM 13 (PWM for right motor)

# Define sensor pins (BCM numbering)
left_sensor = DigitalInputDevice(8)   # BCM 8 (Physical pin 24)
right_sensor = DigitalInputDevice(7)  # BCM 7 (Physical pin 26)

def turn_left():
    enable_left.value = 0.4
    enable_right.value = 0.4
    robot.left()

def turn_left2():
    enable_left.value = 0.0
    enable_right.value = 0.3
    robot.forward()
```

```
def turn_right():
    enable_left.value = 0.4
    enable_right.value = 0.4
    robot.right()

def turn_right2():
    enable_left.value = 0.3
    enable_right.value = 0.0
    robot.forward()

def move_forward():
    enable_left.value = 0.5
    enable_right.value = 0.5
    robot.forward()

def stop():
    enable_left.value = 0
    enable_right.value = 0
    robot.stop()

def manual_command():
    if keyboard.is_pressed('a'):
        return 'a'
    elif keyboard.is_pressed('d'):
        return 'd'
    elif keyboard.is_pressed('w'):
        return 'w'
    elif keyboard.is_pressed('s'):
        return 's'
    elif keyboard.is_pressed('z'):
        return 'z'
    elif keyboard.is_pressed('c'):
        return 'c'
    return ''

def manual_program(command):
    if command == 'a':
        turn_left()
        print('Turning left')
    elif command == 'd':
        turn_right()
        print('Turning right')
    elif command == 'w':
        move_forward()
        print('Going forward')
    elif command == 'z':
        turn_left2()
        print('Turning Left Manually')
    elif command == 'c':
        turn_right2()
        print('Turning Right Manually')
    elif command == 's':
        stop()
        print('Stop')

def main():
    LR = 0
    TR = 0
    TL = 0
    print('Entering main program.')
    is_auto_mode = False
```

```
    while True:
        if keyboard.is_pressed('p'):
            is_auto_mode = True
            sleep(1)
            print("Auto mode activated")
        elif keyboard.is_pressed('e'):
            is_auto_mode = False
            stop()
            print("Stopped in auto mode")
            sleep(0.5)
        elif keyboard.is_pressed('q'):
            print("Exiting program")
            break

        if is_auto_mode:
            if left_sensor.value:
                turn_right()
                print(f"Left sensor: {left_sensor.value}")
                sleep(0.25)
                print("Collision detected on left, turning right")
            elif right_sensor.value:
                turn_left()
                print(f"Right sensor: {right_sensor.value}")
                sleep(0.25)
                print("Collision detected on right, turning left")
            else:
                if LR == 0:
                    TL += 1
                    print(TL)
                    enable_left.value = 0.3
                    enable_right.value = 0.0
                    robot.forward()
                    sleep(0.01)
                if LR == 1:
                    TR += 1
                    print(TR)
                    enable_left.value = 0.0
                    enable_right.value = 0.3
                    robot.forward()
                    sleep(0.01)
                if TL >= 4 and LR == 0:
                    LR = 1
                    TR = 0
                if TR >= 2 and LR == 1:
                    LR = 0
                    TL = 0
        else:
            char = manual_command()
            manual_program(char)
        sleep(0.1)

    robot.stop()
    enable_left.close()
    enable_right.close()
    left_sensor.close()
    right_sensor.close()
    print('Program ended.')

if __name__ == '__main__':
    main()
```

5.10 Appendix J: Dynamic Test Code

```
from gpiozero import Robot, PWMOutputDevice
from time import sleep,time

# define H-bridge pin number on pi
robot = Robot(left=(26,20), right=(24,23))

# define PWM pin number for both motor
enable2 = PWMOutputDevice(13)
enable = PWMOutputDevice(12)

# loop durations for testing
duration = [2, 1.5, 1, 0.5, 0.4, 0.2,0.1,0.0625, 0.05]

# PWM range for testing
pwm_values = [0.6, 0.8, 1.0]

timestamps=[]

for pwm_value in pwm_values:
    enable.value = pwm_value
    enable2.value = pwm_value

    print(f'Testing with PWM value: {pwm_value}')

    # for robot's motion for each duration
    for i in duration:
       start_time = time()
       robot.right()
       sleep(i)
       robot.left()
       sleep(i)
       robot.right()
       sleep(i)
       robot.left()
       sleep(i)
       robot.right()
       sleep(i)
       robot.left()
       sleep(i)
       end_time = time()
       robot.stop()
       sleep(1)
       timestamps.append((start_time,end_time))
    # Give a 10-second pause to allow for data recording
    print(f"Pausing for 11 seconds after testing PWM value: {pwm_value}")
    robot.stop()
    sleep(1)

with open("timestamps.txt","w") as f:
    for start,end in timestamps:
       f.write(f"{start},{end}\n")

print("Timestamps saved.")
```

5.11 Appendix K: Dynamic Test Video Processing Code

```
    import cv2
import numpy as np
import os
```

```
import csv
from cv2 import aruco
from datetime import datetime

# --- CONFIGURATION ---
VIDEO_PATH = r"C:\Users\Lenovo\Desktop\DT3\VIDEO.mp4"
TIMESTAMPS_FILE = r"C:\Users\Lenovo\Desktop\DT3\timestamps.txt"
OUTPUT_CSV = "rotation_angles_with_phases.csv"
DEBUG_MODE = True  # Enable detailed processing logs
SHOW_DETECTION = True  # Show real-time detection preview

# --- TIMESTAMP PROCESSING ---
timestamps = []
with open(TIMESTAMPS_FILE, "r") as f:
    for line in f:
        start, end = map(float, line.strip().split(","))
        timestamps.append((start, end))

# --- VIDEO TIME SYNCHRONIZATION ---
try:
    # Extract timestamp from filename (format: video_YYYYMMDD_HHMMSS.mp4)
    filename = os.path.basename(VIDEO_PATH)
    time_str = filename.split("_")[1] + "_" + filename.split("_")[2].split(".")[0]
    video_start_time = datetime.strptime(time_str, "%Y%m%d_%H%M%S").timestamp()
except Exception as e:
    video_start_time = timestamps[0][0]  # Fallback to first timestamp
    print(f"Warning: Using default video start time {video_start_time}")

# --- VIDEO CAPTURE INITIALIZATION ---
cap = cv2.VideoCapture(VIDEO_PATH)
if not cap.isOpened():
    raise RuntimeError(f"Failed to open video file: {VIDEO_PATH}")

# --- ARUCO DETECTOR CONFIGURATION ---
aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_50)
params = aruco.DetectorParameters()
detector = aruco.ArucoDetector(aruco_dict, params)

# --- DATA STORAGE ---
phase_data = {i: [] for i in range(len(timestamps))}
current_phase = None
prev_phase = None
prev_angle = None
frame_counter = 0

# --- FRAME PROCESSING LOOP ---
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Calculate current absolute time
    frame_pts = cap.get(cv2.CAP_PROP_POS_MSEC) / 1000
    current_time = video_start_time + frame_pts

    # Detect current phase
    current_phase = next((i for i, (s, e) in enumerate(timestamps) if s <=
        current_time <= e), None)

    if current_phase is None:
        if DEBUG_MODE:
            print(f"Frame {frame_counter}: No phase matched for time {current_time:.3
                f}s")
```

```
        continue

    # Handle phase transition
    if current_phase != prev_phase:
        if DEBUG_MODE:
            print(f"\n=== Entering phase {current_phase} ===")
            print(f"Start: {timestamps[current_phase][0]:.3f}s")
            print(f"End: {timestamps[current_phase][1]:.3f}s")
        prev_angle = None  # Reset angle tracking
        prev_phase = current_phase

    # Marker detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = detector.detectMarkers(gray)

    # Visual feedback
    if SHOW_DETECTION:
        debug_frame = cv2.aruco.drawDetectedMarkers(frame.copy(), corners, ids)
        cv2.imshow('Detection Preview', debug_frame)
        cv2.waitKey(1)

    if ids is not None and len(ids) > 0:
        # Process first detected marker
        marker = corners[0][0]
        origin = marker[0]
        center = np.mean(marker, axis=0)

        # Maintain original coordinate system (Y-axis down)
        dx = center[0] - origin[0]
        dy = center[1] - origin[1]
        current_angle = np.arctan2(dy, dx)

        # Handle first frame in phase
        if prev_angle is None:
            phase_data[current_phase].append(0.0)
            prev_angle = current_angle
            if DEBUG_MODE:
                print(f"Frame {frame_counter}: Initial angle = {np.rad2deg(
                    current_angle):.1f}  ")
            continue

        # Calculate angle difference
        angle_diff = np.rad2deg(current_angle - prev_angle)
        angle_diff = (angle_diff + 180) % 360 - 180  # Normalize to [-180, 180]

        phase_data[current_phase].append(round(angle_diff, 2))
        prev_angle = current_angle

        if DEBUG_MODE:
            print(f"Frame {frame_counter}:    = {angle_diff:.2f}   | ", end='')
    else:
        phase_data[current_phase].append(0.0)
        if DEBUG_MODE:
            print(f"Frame {frame_counter}: No marker | ", end='')

    frame_counter += 1

cap.release()
cv2.destroyAllWindows()

# --- DATA EXPORT ---
max_length = max(len(v) for v in phase_data.values())
pwm_map = [0.6] * 9 + [0.8] * 9 + [1.0] * 9
```

```
duration_map = [2, 1.5, 1, 0.5, 0.4, 0.2, 0.1, 0.0625, 0.05] * 3

with open(OUTPUT_CSV, 'w', newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['Phase', 'PWM', 'Duration(s)'] + [f'Angle_{i}' for i in range(
        max_length)])

    for phase in phase_data:
        writer.writerow([
            phase,
            pwm_map[phase],
            duration_map[phase],
            *phase_data[phase],
            *[np.nan] * (max_length - len(phase_data[phase]))
        ])

print(f"Data saved to: {OUTPUT_CSV}")
```

5.12 Appendix L: Angle Plotting Code

```
    clc; clear; close all;

%% Read CSV file
filename = 'rotation_angles_with_phases.csv';
data = readtable(filename);

%% Manually define duration order (2s -> 0.05s)
specified_durations = [2, 1.5, 1, 0.5, 0.4, 0.2, 0.1, 0.0625, 0.05]; % Custom order

%% Automatically get duration column name
duration_col_name = data.Properties.VariableNames{contains(data.Properties.
    VariableNames, 'Duration')};

%% Parameter definitions
pwm_values = unique(data.PWM);
phase_colors = [0.9 0.2 0.2;  % Red: 0.6 PWM
                0.2 0.8 0.2;  % Green: 0.8 PWM
                0.2 0.2 0.9]; % Blue: 1.0 PWM

%% Create 3x3 subplot layout
figure('Position', [100, 100, 1200, 900]);
for dur_idx = 1:length(specified_durations)
    current_duration = specified_durations(dur_idx);
    subplot(3, 3, dur_idx); % Maintain specified order

    % Check if duration exists in data
    if ~any(abs(data.(duration_col_name) - current_duration) < 1e-6)
        fprintf('Skipping missing duration: %.4f s\n', current_duration);
        continue;
    end

    hold on;
    grid on;
    box on;

    % Title and labels
    title(sprintf('Duration: %.3f s', current_duration), 'FontSize', 10);
    xlabel('Frame Index', 'FontSize', 9);
    ylabel('Cumulative Angle (degrees)', 'FontSize', 9);

    % Plot all PWM values
```

```
    for pwm_idx = 1:length(pwm_values)
        current_pwm = pwm_values(pwm_idx);

        % Filter data with floating point tolerance
        mask = (abs(data.(duration_col_name) - current_duration) < 1e-6) & ...
               (data.PWM == current_pwm);

        if sum(mask) == 0
            continue;
        end

        % Process angle data
        angle_data = data{mask, 4:end};
        valid_angles = angle_data(~isnan(angle_data));
        cumulative_angles = cumsum(valid_angles);
        frame_indices = 1:length(cumulative_angles);

        % Plot curve
        plot(frame_indices, cumulative_angles, ...
            'Color', phase_colors(pwm_idx, :), ...
            'LineWidth', 1.5, ...
            'DisplayName', sprintf('PWM=%.1f', current_pwm));
    end

    % Set axis limits
    ylim_current = ylim;
    ylim([ylim_current(1)-5, ylim_current(2)+5]);

    % Add legend to top-right corner
    legend('Location', 'northeast', 'FontSize', 8); % Changed to 'northeast'
end

%% Add main title
annotation('textbox', [0.1, 0.94, 0.8, 0.05], ...
    'String', 'Cumulative Rotation Angles Under PWM with 0.6/0.8/1.0 For Different
        Duration', ...
    'FontSize', 14, ...
    'FontWeight', 'bold', ...
    'HorizontalAlignment', 'center', ...
    'EdgeColor', 'none');

%% Save image
print('cumulative_rotation_angles_2.png', '-dpng', '-r300');
```

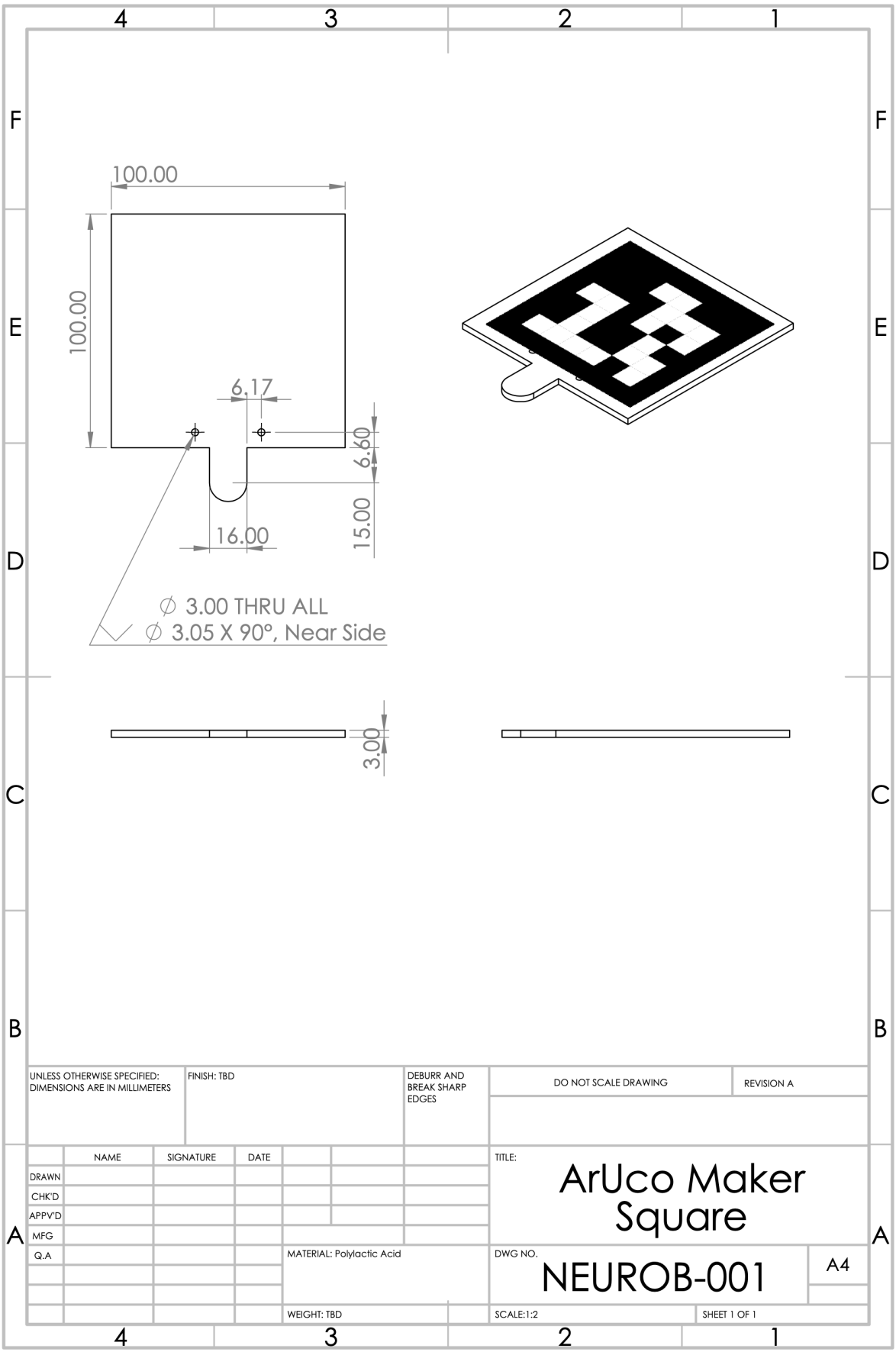
5.13 Appendix M: Trajectory Plotting Code

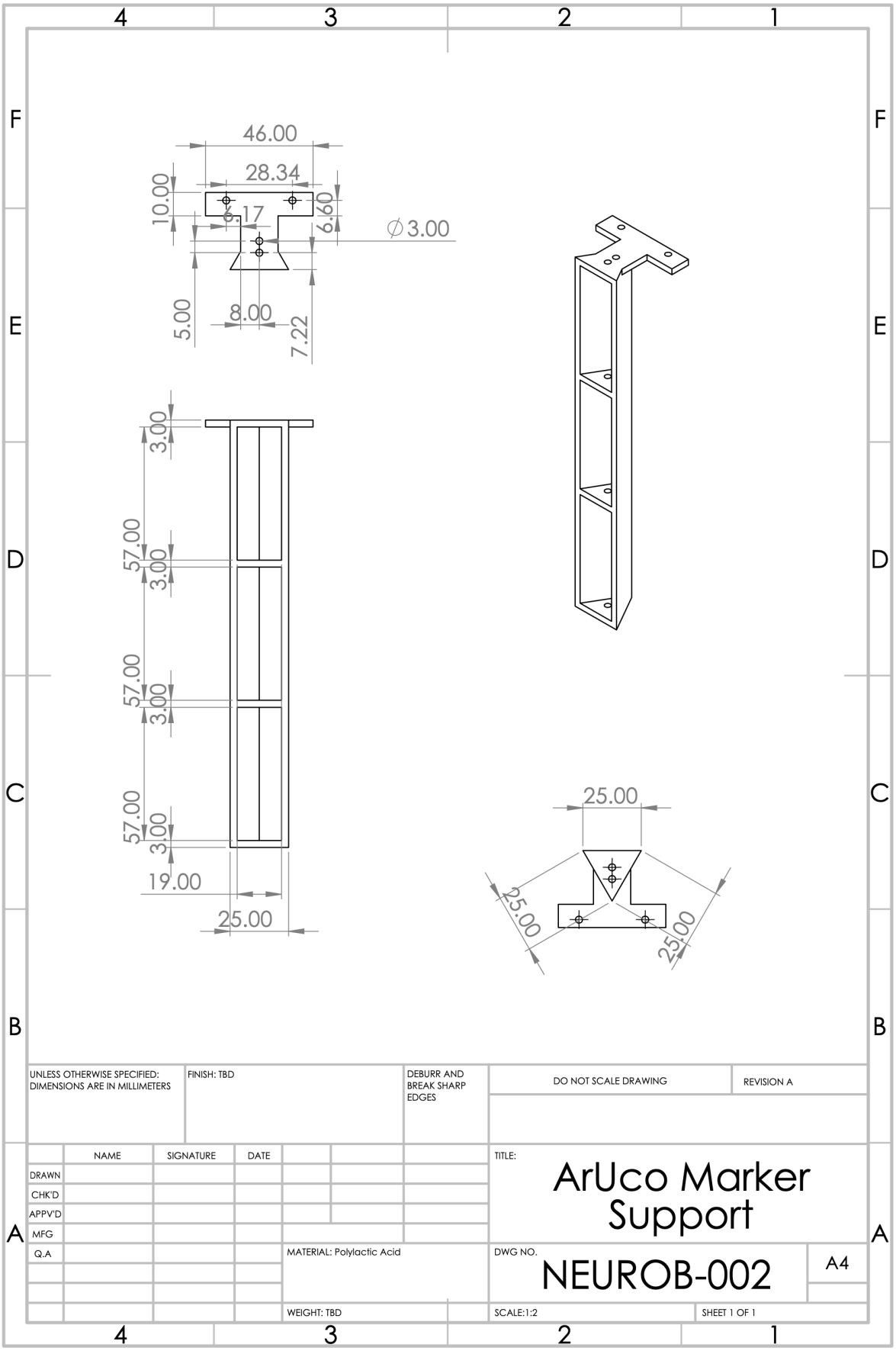
```
import numpy as np
import cv2
from cv2 import aruco
import matplotlib.pyplot as plt

# Load the video file
cap = cv2.VideoCapture("/Users/changyuhu/Desktop/Video_2025.4.2/T10_new.mp4")

# Define ArUco dictionary and detection parameters
aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_50)
parameters = aruco.DetectorParameters()

# Dummy camera matrix and zero distortion coefficients
matrix_coefficients = np.eye(3)
distortion_coefficients = np.zeros((5, 1))
```

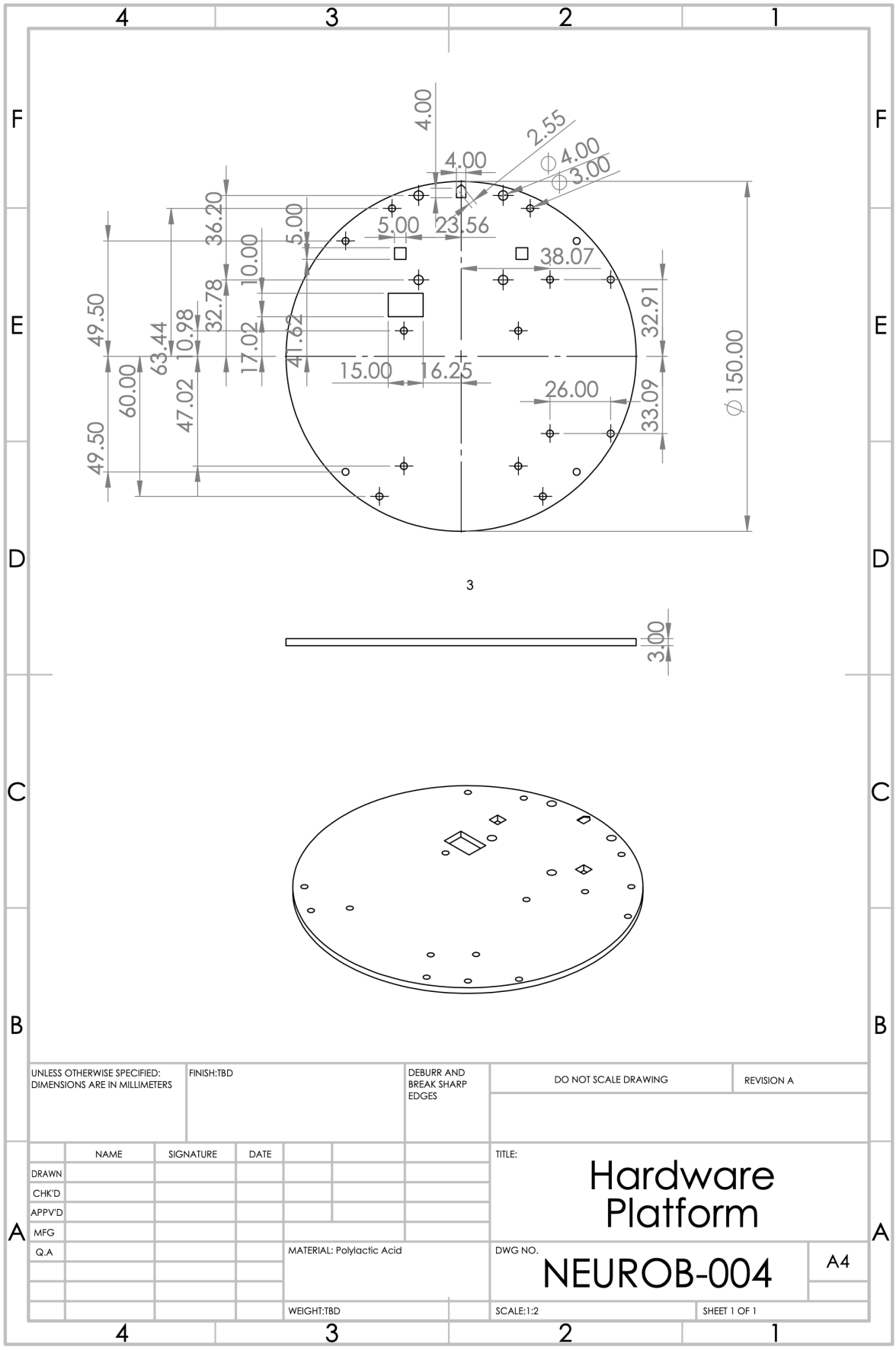
```
# Dictionary to store marker trajectories
trajectories = {}

# Arena conversion settings
x_min, x_max = 520, 1444
y_min, y_max = 61, 1021
arena_cm = 200.0

# Set up Matplotlib for real-time plotting
plt.ion()
fig, ax = plt.subplots()
ax.set_xlabel("X Position (cm)", fontsize=12)
ax.set_ylabel("Y Position (cm)", fontsize=12)
ax.set_title("ArUco Marker Trajectory with Real-Time Direction", fontsize=16)

# Loop through video frames
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("End of video or error.")
        break

    frame_height = frame.shape[0]
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)

    if ids is not None:
        for i in range(len(ids)):
            marker_id = int(ids[i][0])
            center = tuple(np.mean(corners[i][0], axis=0).astype(int))
            center = (center[0], frame_height - center[1])  # Flip y

            if marker_id not in trajectories:
                trajectories[marker_id] = []

            trajectories[marker_id].append(center)

            # Draw trajectory on video frame
            for j in range(1, len(trajectories[marker_id])):
                cv2.line(frame, trajectories[marker_id][j - 1], trajectories[
                    marker_id][j], (0, 255, 0), 2)

            # Pose estimation (optional)
            rvec, tvec, _ = aruco.estimatePoseSingleMarkers(corners[i], 0.02,
                matrix_coefficients, distortion_coefficients)
            rvec = rvec.reshape(-1, 1, 3)
            tvec = tvec.reshape(-1, 1, 3)
            aruco.drawDetectedMarkers(frame, corners)
            cv2.drawFrameAxes(frame, matrix_coefficients, distortion_coefficients,
                rvec, tvec, 0.01)

            # === Real-Time Plotting ===
            ax.cla()
            ax.set_xlabel("X Position (cm)", fontsize=12)
            ax.set_ylabel("Y Position (cm)", fontsize=12)
            ax.set_title("ArUco Marker Trajectory with Real-Time Direction", fontsize
                =16)

            x_vals, y_vals = zip(*trajectories[marker_id])
            x_vals = np.array(x_vals)
            y_vals = np.array(y_vals)
```

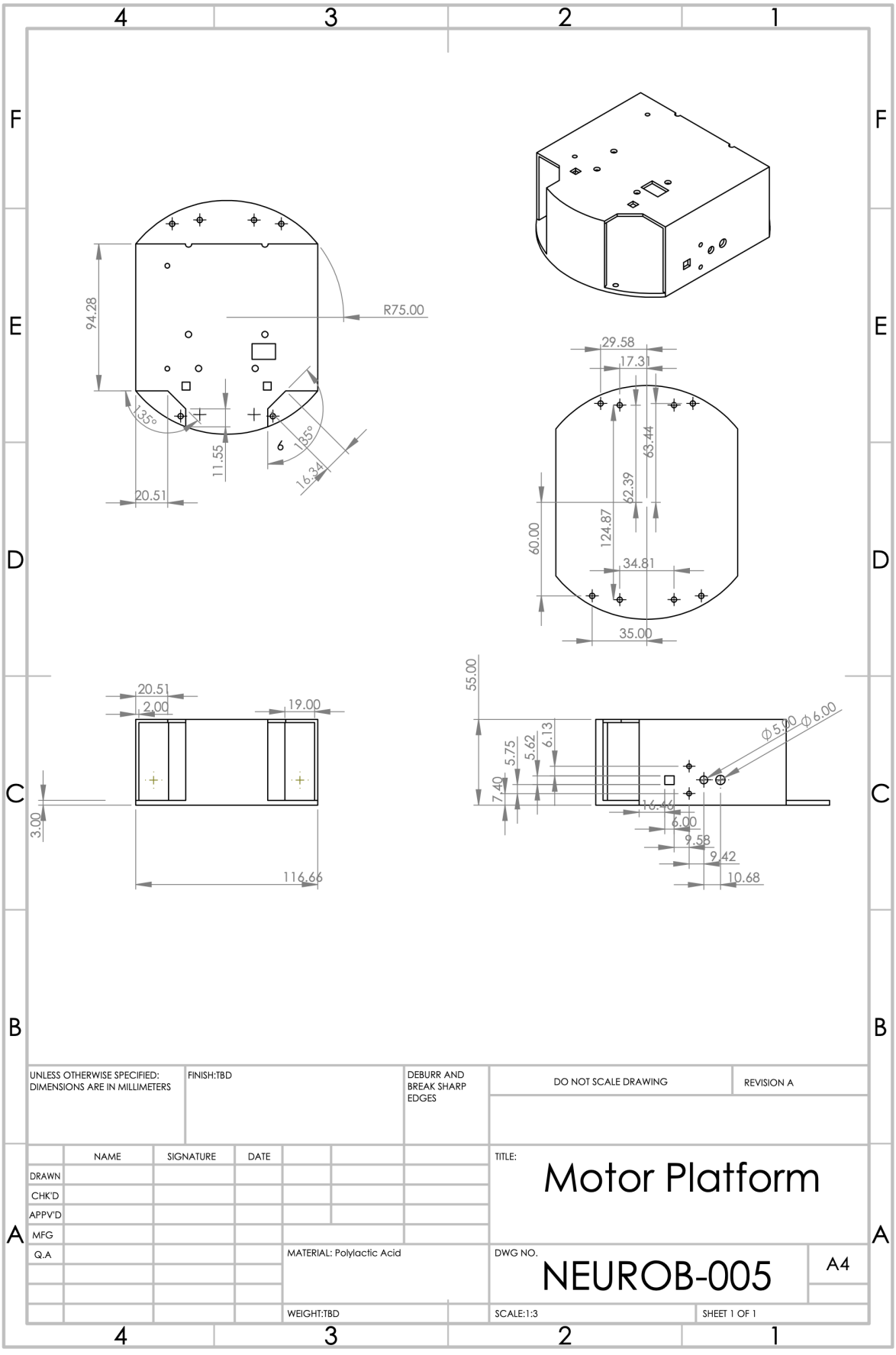
```
            # Scale to 0  200   cm
            x_cm = (x_vals - x_min) / (x_max - x_min) * arena_cm
            y_cm = (y_vals - y_min) / (y_max - y_min) * arena_cm

            # Plot trajectory
            ax.plot(x_cm, y_cm, marker='o', color='blue', markersize=0.05, linewidth
                =1, linestyle='-')

            # Start and end points
            start = (x_cm[0], y_cm[0])
            end = (x_cm[-1], y_cm[-1])
            ax.plot(start[0], start[1], 'go', markersize=8)
            ax.plot(end[0], end[1], 'ro', markersize=8)

            # Legend-like markers
            ax.plot(0.02, 0.95, 'o', color='green', markersize=8, transform=ax.
                transAxes)
            ax.text(0.05, 0.95, 'Start', transform=ax.transAxes, fontsize=10, color='
                green', verticalalignment='center')
            ax.plot(0.02, 0.88, 'o', color='red', markersize=8, transform=ax.
                transAxes)
            ax.text(0.05, 0.88, 'End', transform=ax.transAxes, fontsize=10, color='
                red', verticalalignment='center')

            # Direction arrows
            if len(x_cm) > 1:
                dx = np.diff(x_cm)
                dy = np.diff(y_cm)
                x_mid = x_cm[:-1]
                y_mid = y_cm[:-1]

                magnitude = np.sqrt(dx ** 2 + dy ** 2)
                magnitude[magnitude == 0] = 1
                dx_unit = dx / magnitude
                dy_unit = dy / magnitude

                sampled_idx = np.arange(0, len(dx), 15)
                ax.quiver(
                    x_mid[sampled_idx], y_mid[sampled_idx],
                    dx_unit[sampled_idx], dy_unit[sampled_idx],
                    angles='xy', scale_units='xy', scale=0.3,
                    width=0.01, headwidth=4, headlength=6, headaxislength=5,
                    color='red'
                )

            ax.set_xlim(0, 200)
            ax.set_ylim(0, 200)
            ax.set_aspect('equal', 'box')
            ax.tick_params(axis='both', labelsize=10)

            plt.pause(0.01)

    # Show video with OpenCV
    cv2.imshow('ArUco Marker Tracking', frame)

    if cv2.waitKey(3) & 0xFF == ord('q'):
        break

# Cleanup
cap.release()
cv2.destroyAllWindows()
plt.ioff()
plt.show()
```

5.14 Appendix N: Technical Drawings




4
3
2
1
F
E
D
C
B
A
46.00
28.34
10.00
6.17
6.60
∅3.00
5.00
8.00
7.22
3.00
57.00
3.00
57.00
3.00
57.00
3.00
19.00
25.00
25.00
25.00
25.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
FINISH: TBD
DEBURR AND
BREAK SHARP
EDGES
DO NOT SCALE DRAWING
REVISION A
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
ArUco Marker
Support
MATERIAL: Polylactic Acid
DWG NO.
NEUROB-002
A4
WEIGHT: TBD
SCALE:1:2
SHEET 1 OF 1


4.00
2.55
⌀4.00
⌀3.00
4.00
5.00
23.56
38.07
36.20
5.00
10.00
32.78
10.98
17.02
41.62
49.50
63.44
60.00
47.02
49.50
15.00
16.25
26.00
32.91
33.09
⌀150.00
3
3.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
FINISH:TBD
DEBURR AND
BREAK SHARP
EDGES
DO NOT SCALE DRAWING
REVISION A
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
Hardware
Platform
MATERIAL: Polylactic Acid
DWG NO.
NEUROB-004
A4
WEIGHT:TBD
SCALE:1:2
SHEET 1 OF 1


4
3
2
1
F
E
D
C
B
A
94.28
R75.00
135°
11.55
6
135°
16.34
20.51
29.58
17.31
63.44
62.39
124.87
60.00
34.81
35.00
20.51
2.00
19.00
3.00
116.66
55.00
5.75
5.62
6.13
7.40
ϕ5.00
ϕ6.00
16.46
6.00
9.58
9.42
10.68
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
FINISH:TBD
DEBURR AND
BREAK SHARP
EDGES
DO NOT SCALE DRAWING
REVISION A
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
Motor Platform
MATERIAL: Polylactic Acid
DWG NO.
NEUROB-005
A4
WEIGHT:TBD
SCALE:1:3
SHEET 1 OF 1

5.15 Appendix O: Hardware Schematics