University of Nottingham
UK | CHINA | MALAYSIA

EEE Department of Electrical and Electronic Engineering

EEEE3001

Final Year Individual Project Dissertation

AI Control of Fish Behaviour Using Acoustic Arrays

AUTHOR: Mr. Jake Smejko

ID NUMBER: 20488824

SUPERVISOR: Dr. Kevin Webb

MODERATOR: Dr. Peter Christopher

DATE: 8th May 2025

This third year project Dissertation is submitted in part fulfilment of the requirements of the degree of Bachelor of Engineering.

Contents

1 Introduction

1.1 Overview

A newly introduced vertebrate fish, *Danionella cerebrum (DC)*, has been established at the University of Nottingham. It offers the advantage of optical transparency in body and through a lack of bony brain plates, enabling non-invasive, lifelong brain studies using microscopic imaging. Although *DC* shares 85% of its DNA with its close relative, *Danio rerio*, its breeding habits have proven to be unusually unpredictable [1]. Without the ability to stimulate their reproduction, a steady populus of this breed of fish within their place of study can be difficult to maintain.

To combat this, Dr. Rob Wilkinson, assistant professor of developmental genetics at the University of Nottingham, proposes to identify the behavioural cues exhibited by DC during procreation using AI, hoping to recreate these circumstances and produce more reliable mating cycles. An approach to this is the use of a neural network to analyse patterns in fish behaviour, particularly those that lead to egg-laying. By identifying these behavioural triggers, researchers may be able to recreate the stimuli that reliably promote reproduction, ensuring the stability of laboratories' *DC* populus.

DC live in muddy water and communicate sonically, producing sounds exceeding 140dB [2]. These acoustic communications will be recorded in tandem with video footage, allowing researchers to categorise behaviours based on their corresponding vocalisations. A neural network will then parameterise the vocal patterns that frequently precede egg-laying in hope that hours of observational work can be accurately summarised in minutes.

Commercial fish-tank suppliers, such as Techniplast, are moving toward automation in lab facilities. If the study's findings can reliably induce egg-laying, digitised population control could be implemented in products such as the Tritone XS, which only offer automatic feeding as of now. Not only could the final product sell well; industrially available facilitation of *DC* could encourage their study, potentially accelerating progress in developmental genetics.

Figure 1 (left): Danionella Cerebrum within the premise of Dr. Rob Wilkinson's lab.

Figure 2 (right): Observation setup for DC.

1.2 Specification

1.2.1 The Scope of the Objectives

An iterative process will be used in the development of the solution to achieve functionality that is useful to the laboratory from its foremost achievements onwards. Contingencies and importances to the success of the project are portrayed using a set of keywords in **bold**. Their meanings have additional circumstantial importance to this (section 1.2). These are outlined in table 1:

Table 1: Circumstantial meanings of keywords arising in this section

Keyword	Circumstantial importance
Required/Must	Absolutely necessary to the project's success
Should	Additional features that dramatically improve the quality of the result
Could	Contingent on **required** goals, enhancements that are doable in this project's scope
Would	Contingent on goals that **could** be acquired, these are developments that would be made if the study was pursued outside of this project's scope.

1.2.2 Objectives

Fundamentally, a system that presents surveillance footage, of an individual fish tank, to the client, is **required**. Enabling the fast-forwarding of recorded video will allow events that took place outside of in-person lab access increases the number of egg lays and their preceding events that can be viewed.

Capable of serving the key details of hours of surveillance, video made available to the client **should** also be run through analyses including image localisation, visualising bounding boxes around specimen that have been detected by neural networks. Where the clarity of the human eye is lacking in camera footage, a neural network model specialised in pinpointing the fish will support the viewing experience.

A tracking solution **could** be devised with methods inbuilt to the detection model's library. Combination of two orthogonal views imagining the fishes' three-dimensional locations emulate a natural view of their positions, providing insight into whom is close to one another at any given instant. Frame by frame quantification of this distance using the pixels in each view could indicate their average distances spent between each other over prolonged periods.

All information will be viewable during the development of the software; however, a user interface **should** be provided through a webpage on the local host for clarity and scalability to multiple viewers. Releasing this on a real domain would be possible after the project if the device proved to be useful to the client.

Communication produced by the fish **could** be captured using devices that act as hydrophones. Because real hydrophones are outside of the project's budget, an experiment **must** be pursued, studying the utilisation of piezoelectric devices in the acquisition of underwater audio. Where a successful prototype is conceived, three-dimensional triangulation of the audio cues' origin **should** be executed for DC using time-distance of arrival (TDOA) reversal.

Given the presence of the above information, it **should** be condensed into arrays and flattened, before being passed through a regressive neural network model in hopes of detecting correlations between DC's behaviour and events where they are found to lay eggs. An extension that **could** be solved by the project includes the self-supervised training of this neural network until a humanly perceivable behavioural trigger is observed at the model's output, which **would** probably extend beyond the time restraint of the project.

Where the conditions of the model's output are applied to the fish in the study and the perceived trigger can be proven to induce procreation within the populus, the reputability of both the observational and recreation of the observed behaviour **would** be evaluated and potentially marketed as a true solution to DC's unreliable breeding habits. The patent and commercialisation of the combined solution **would** ensue, and the sale pitched to potential buyers such as Techniplast and similar corporations.

2 Design and Implementation

2.1 System Modelling

2.1.1 High-Level Plan

A top-down view of the solution was constructed foremost to draw out exactly what needed to be achieved. In its simplest form, reproduction could be caused by observable behaviour, meaning that the analysis of audio and video footage within their fish tanks could reinforce the stakeholder's proposition.

DC were introduced to the University because their unique cranial structure enables their research to be non-invasive [1], in the spirit of this non-invasive approach, the profiling of their behaviour here will aim only to watch and learn from a safe distance, through audio and video footage. Neural networks 'learn' their biases through backpropagation, which adjusts their biases proportionately to the error in their predictions [3], meaning that their output must be quantifiable.

Through restricts in the lab's in-person accessibility, electronic devices can pursue DC's observation, serving information the end user wishes to view at a glance by storing running totals of simple analyses such as occurrences of physical contact between specimens to the device's hard drive.

Devised is a solution that relies on cause and effect in the fish's observable behaviour. With luck, the details producing the answer dataset would combine to form the cause, leaving only quantification of the effect at the model's input: eggs being laid. An image detection model will be trained to spot these within the photos in real time, with high confidence, so that the time at which they're first detected can be used as a reference point to chunk all prior events, by a duration which could sensibly withhold the proceedings which invoked the egg lay, say an hour, such that the most recent chunk can be flagged as containing the cause and everything prior can be marked as "not the cause". This binary input feature is less versatile than a timeseries forecasting model which doesn't have to go by such a strong assumption, but with so many output features and potentially very weak correlations in the fishes' behaviour, the model requires harsh thresholding using as many human assumptions as possible before the data enters the network.

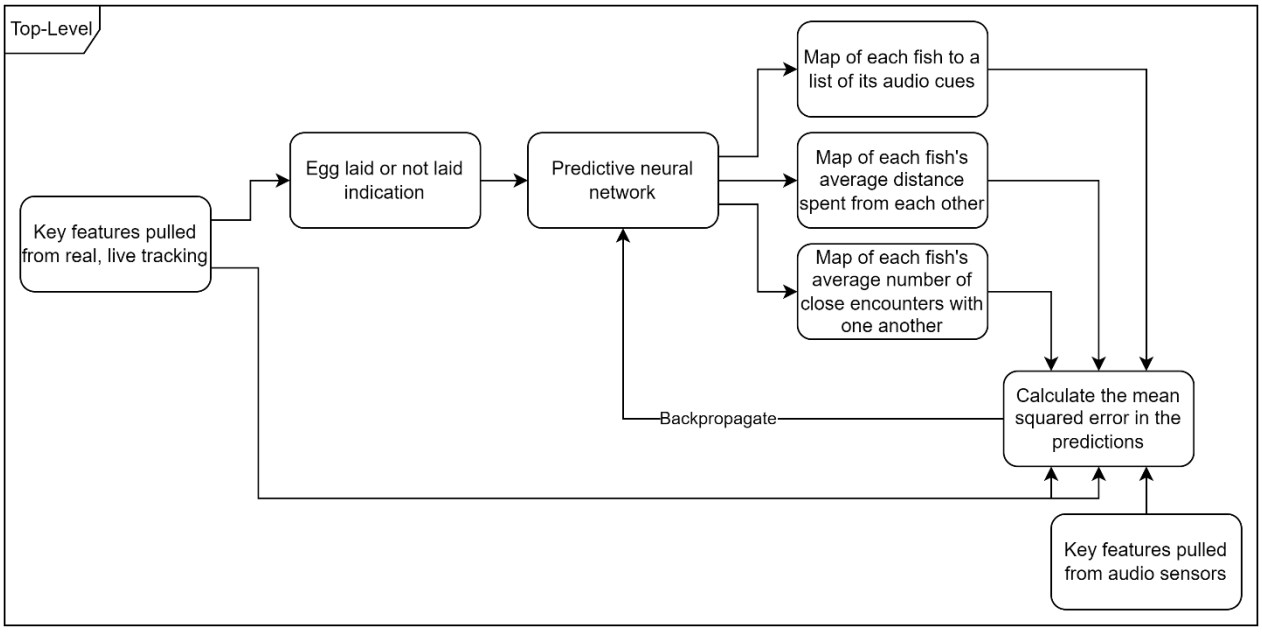


Figure 3: Highest level whole system overview. white boxes (all here, some in future diagrams) represent components which entail more work at a lower level than they describe, these implementations are yet to be discussed.

2.1.2 Two-Dimensional Tracking

Programmatical classes, member variables and functions will be written in italics. Function names will have an added () at the end of their name to distinguish them.

2.1.2.1 Overview

With it clear that the binary flag of cause/no cause is going to produce an output which indicates classifiable behaviour demonstrated by the fish, these classes must be derived next to inform the methods of acquisition for the audio and video footage. Metrics useful to both researchers and the predictive m2.1.odel include relational data between each specimen, hallmarks of which are the average distances from fish to fish, rate and durations of contact between specimen and their speed when approaching one another, all measurable using tracking.

Frame-by-frame, objects present in images can be pinpointed by neural networks designed to perform the localisation of classes which they are trained to detect. Powerful libraries encapsulate this principle, offering means of providing one's own dataset of training and methods that produce the pixelwise (x, y) coordinates of objects within its BGR formatted arguments.

Video taken from a single camera can be processed using image detection models that can easily be taken from existing libraries which encapsulate all the heavy lifting. Two dimensional photos can be annotated with boxes that represent the x and y pixelwise location of the object found within and the class that the object might belong to.

YOLO has become a central real-time object detection system for robotics, driverless cars, and video monitoring applications [4] and are easily accessible through the Ultralytics Python library with methods that pinpoint objects in images in a single function call.

YOLO instances also provide a method named *track()* which also places unique IDs on detected objects under a member *track_id*. In video feed, the IDs are maintained across frames by linking new inferences to their previous ID by their class and position. It is assumed that the closest object of this class in the last frame to that in the current frame is the same object.

If there are excessive durations between frames during tracking, objects of the same class may be able to switch position, particularly in the situation of the fish tank, where all objects come under the same class. It is important that the frame rate is adequate to prevent this and that the model is reliable so that objects aren't misinterpreted or unidentified.

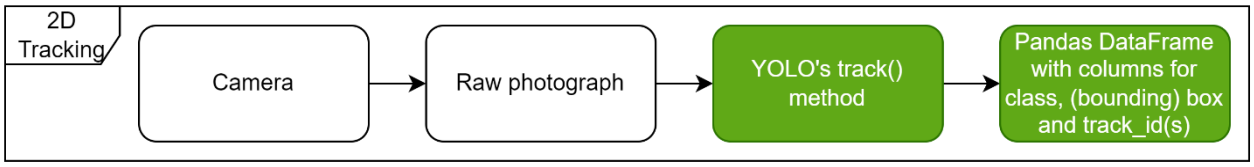


Figure 4: Flowchart summary of 2D tracking. Green boxes represent stages which are as simple to implement as they describe.

2.1.2.2 Using YOLOv8 with OpenCV

OpenCV is an open-source computer vision library. In this instance, algorithms that are useful to the project include OpenCV's *imshow*() and *waitKey*() functions, which can produce BGR images stored as NumPy arrays on the screen through demonstration of YOLOv8. NumPy's *ndarray* class are highly optimised on a C backend with operator overloads that expose elementwise mathematical operations highly efficiently and succinctly in Python's high-level language; acting as the base class for OpenCV's matrices which are common across nearly all OpenCV algorithms used here.

A laptop's webcam was opened using an OpenCV *VideoCapure* object and passing "yolov8n.pt" to the constructor of Ultralytics' YOLO class automatically prompted the download of a YOLO version 8 neural network with 3.2 million pretrained parameters, having been trained on a Collection of Common Objects, COCO dataset, which includes a "person" class that can localise people in images. Exercising the 2D Tracking algorithm outlined in figure 4 now in a Python can produce performance metrics such as inference speed.

YOLO was adapted to track by replacing use of the YOLO model's functor with *track()* in the example program, passing *persist=True*. The flag to persist indicates that the frames are being analysed over multiple calls, instead of batching the frames or encoding them as video before making the function call. With this, assigned IDs would be truly unique to their corresponding objects.

With this adaptation, a laptop's webcam was opened using an OpenCV *VideoCapure* object, before an infinite loop used this to capture a frame, pass it to YOLO's *track()* and display the results. The YOLO model in question was a nano example of version 8, meaning that it had less than a 20th of the neurons that are present in the extra-large model, accelerating inference. This was observed in the test; however, accuracy will be highly important in the real implementation, meaning that benchmarking of these inference times was important.

2.1.2.3 Inference Times and Accuracy

A running total of the execution duration and the number of iterations of the infinite loop were divided to provide insight into the average inference time for each model size. This process must be serialised with several demanding tasks in the final product which could mean consultation with table 1 if the fish begin to travel too much distance between frames (distance is directly proportional to time if the fish travel at constant speed).

Running inference was a laptop housing a ten-core, 2.3GHz processor and a discrete graphics processing unit (GPU) with 6GB of dedicated video memory (VRAM). The model downsized all frames gathered from the webcam to the 640x640px resolution that it was trained on, producing the rows for inference time and framerate in table 2.

Table 2: YOLO model size against inference time when running on the chosen PC for the project's processing stage.

No. Parameters (M)	**3.2**	**11.2**	**25.9**	**43.7**	**68.2**
Inference time (ms)	40	40	47	49	60
Framerate (FPS)	25	25	21	20	17
mAP^{val} [5]	37.3	44.9	50.2	52.9	53.9

Mean average precision (mAP^{val}) statistics were gathered from Ultralytics' website [5]. A model with higher mAP^{val} is more likely to identify, equally, correctly identify, an instance of a given class within an image.

2.1.3 Three-Dimensional Tracking with Two Cameras

2.1.3.1 Justification

Section 2.1.2 has proven successful in 2D tracking, however, without depth perception, the real distance between detected objects could be larger than expected as objects vary in distance down the normal of the photo. Cameras have obtainable parameters regarding their perspectives, granting insight to how objects grow and shrink within frames as they travel closer and further from camera, which rely on a powerful algorithm to omit changes in the fishes' orientation as they change direction, which isn't feasible on the generic equipment available to the budget, which includes motion blur in video feed, causing imprecise bounding box estimations.

Shooting canon images with width or height parallel to the original camera's normal could introduce a second 2D tracker which will triangulate with the reference tracker to form 3D coordinates. This leaves only the need for a common denominator between trackers for which the separate sets of detections will be combined, as each will operate separately when using YOLO. This can be achieved by matching a sparse collection of features (feature-based approaches) or by directly minimising the difference in image intensity values (direct approaches [6].

2.1.3.2 Limitations

A feature-based approach would rely on unique details between specimen such as colouration which could classify the instances of DC into separate YOLO classes by eye at the time of the image detection model's creation, however, granted the fish are identical, a more likely approach would be direct. A direct approach assumes that the inter-frame motion is very small [6].

A commonality must be identified across the cannon frames before tracking can maintain the mapping between trackers. Orthogonal cameras in any axes of the tank are bound to share a single axis, being the tank's height when each is at ground level. Fish will be able to swim at the same height which compromises the basic concept, however, some immunity can be granted from this: Objects at different heights in the frames prior to their interception can be remembered by their two external tracker IDs, meaning that if they do cross physical heights with another detection, the coordinates under these IDs will still represent that of the same specimen.

Another issue is the cameras' perspectives: depicted in figures 1 & 2, at the same water depth, fish towards the back of the tank could be less pixels high in the detection than those right in front of the camera, increasing the threshold for which the y-coordinate of bounding boxes in each frame should be recognised as "the same". A high threshold increases the chance of two separate specimen being incorrectly mapped to each other's external ID in the other cannon frame. This would mean that one of the three axes in their 3D projection is incorrect. If the camera is far away, the relative distance between the front and back of the fish tank becomes very small, minimising the threshold for more reliable tracking.

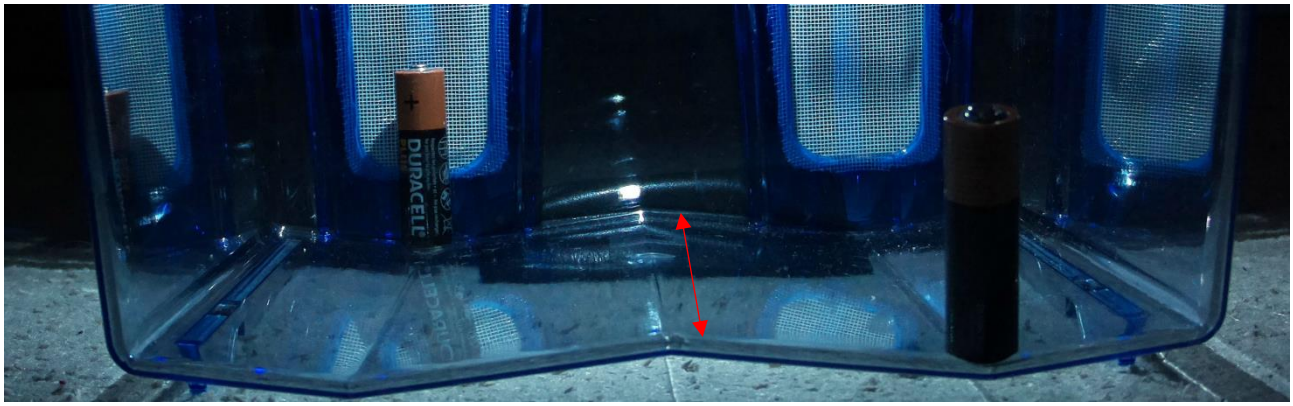


Figure 5: A photo taken through a 5X telescopic lens from 600mm. Although the subjects (represented by batteries) are on level ground, the closest one appears at pixels lower down in the frame. The threshold at which objects in the frame can be assumed to be at the same height, indicated by the red arrow, is large (360 out of 2160 pixels, or ~17% of the frame). Both batteries are 50mm in height.

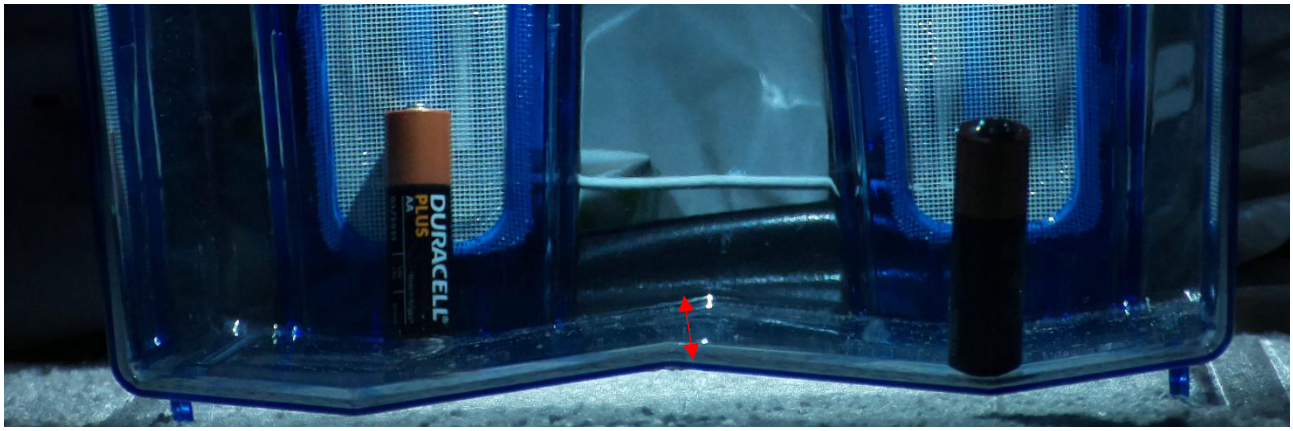


Figure 6: A photo taken through a 10X telescopic lens from 1800mm or 3x the distance of figure 1. The batteries appear at similarly low pixels in the frame, meaning that the threshold, indicated by the red arrow, is low (just 80 out of 2160 pixels, or ~3.7% of the frame). The ground remains equally level as well as the camera angle. Both batteries are still 50mm in height.

Lastly, overlapping fish in one perspective could hide the specimen appearing behind its companion, meaning that the coordinates derived by said camera are no longer apparent, alongside the external ID maintained by its respective YOLO model. Introduction of a third camera viewing into the top of the fish tank would share a common x-axis with the x-axis of one camera's view and a y-axis with the x-axis of the other camera's view. This would mean that the fish are more likely to be present in at least two views, reinforcing the depth perception of the final product.

2.1.3.3 Design Summary

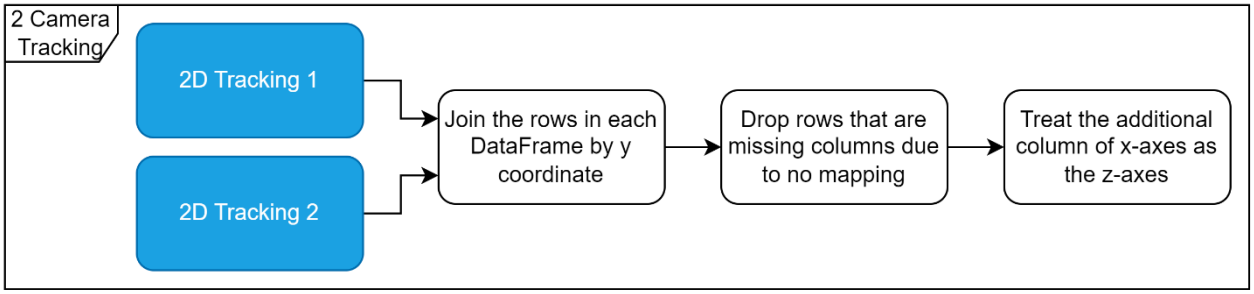


Figure 7: Flowchart to summarise the 3D triangulation of two 2D trackers. Blue components implement stages that have already been described. Here, the "2D Tracking" instances are described in figure 2.

2.1.3.4 Proof-of-Concept

Combination of the axes present in each camera view's detection was demonstrated using an LED as a subject. Small and distinct, the component was likely to consistently be pinpointed by the image detection model, making it ideal in proof of the concept at hand.

A glass rod was used to position the LED within the confinements of an empty fish tank by hand, subject to two orthogonally placed cameras which recorded as the LED was "randomly" translated within the premise of the tank, subject to human error in the randomicity of motion direction and speed.

Figure 8: A glass rod attached to a green LED using blue tac.

A dataset of 200 frames captured by the cameras was created using hand-drawn bounding boxes, and a detection model was trained on this (more on this process in section 2.2.1.2). A small amount of noise was introduced using markings on a whiteboard in the background to enhance the noise immunity of the trained model. Examples of the photographs passed into the model are displayed in figures 9 & 10.

Figure 9 (left): A photograph caught by the camera viewing the depth of the tank.

Figure 10 (right): A photograph caught by the camera viewing width of the tank.

The resulting neural network was applied in the instantiation of two independent YOLO objects for two-dimensional tracking of each camera's view. The algorithm then produced a single 3D coordinate for each canon couple of frames, which was plotted to Matplotlib axes in real time.

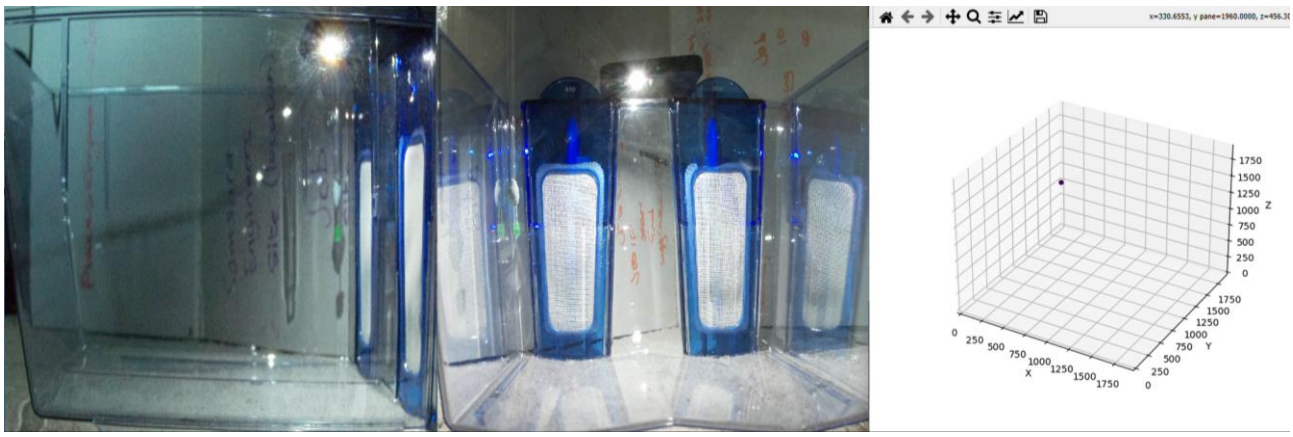


Figure 11: Wide view (left), deep view (middle) and the Matplotlib scatter showing the LED's 3D location (right)

An additional calibration stage took place beforehand, discussed in section 2.2.1.4, solely ensuring that the coordinates filled as much of the frame as possible in each view, allowing the scatter graph's axis limits to better reflect the pixelwise locations. The full video is available in the supplementary material; a very low framerate can be observed. A combination of the extra-large YOLO model, high resolution images (2048x2048px) and Matplotlib's insistence in running on the main thread all account for this.

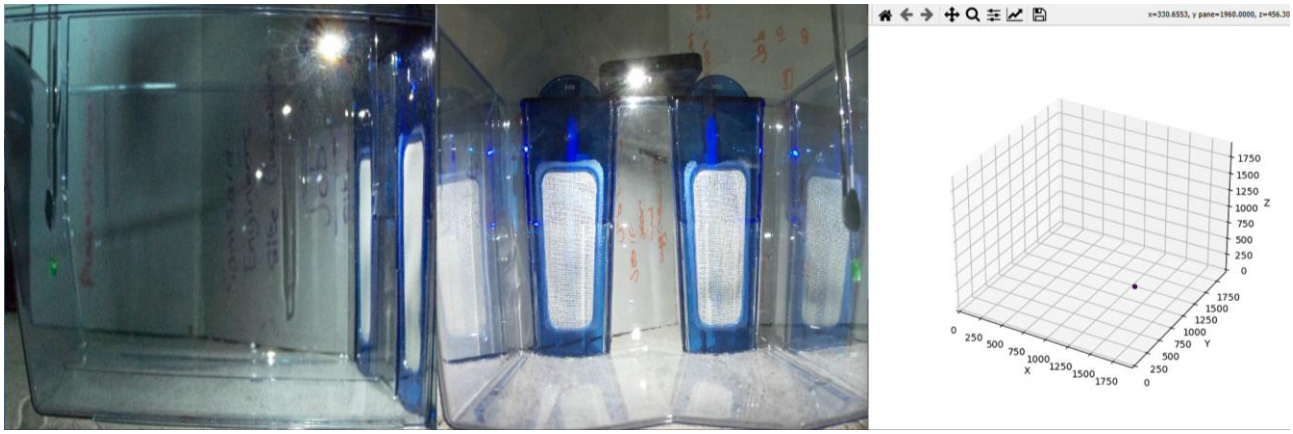


Figure 12: Later in the video, at a similar height, the LED is moved to the opposite edge of the tank. Rightmost tracks in the deep view appear at large x-coordinates, while rightmost tracks in the deep view appear at large y-coordinates.

2.1.4 Tracking Audio

2.1.4.1 Justification

Tracking data for the fish is unlikely to solve the problem alone, rather, it may act as a stepping stone in identifying the root of each communication, given the clicking occurring within the tank can be digitised. Hydrophones are cumbersome and priced from roughly £160 when generally viewing available products online, meaning that the communication between fish should be tapped into some other way. A likely contender would be piezoelectric sensors. Piezoelectricity is a phenomenon exhibited by non-centrosymmetric crystals whereby an electric polarization (i.e. charge) is induced in the material upon the application of a stress [7]. The clicking of DC is produced by a dedicated muscle that rapidly beats a drumming cartilage against the swim bladder [2], which may be enough to stimulate the charges in a submerged piezoelectric sensor.

Clicking sequences could prompt reactions such as egg laying in DC, however, monitoring only vocalisations or visible behaviour at any one time would show no correlation, which is why localisation of the sound could also be used in tandem with the tracker to associate clicks with the individuals whom they originate from. Triangulation of the time-distance of arrival (TDOA) of sound is a widely used method of localisation [8] and could be applied within the fish tank through the introduction of another three piezoelectric sensors as receivers. The relative time taken for the sound to arrive at each sensor with respect to a reference sensor indicates the point of origin because the sounds will register sooner on sensors that are closer in distance.

2.1.4.2 Limitations

In water, sound travels at near 1500 metres per second [9], meaning that it travels DC's body length in 8 microseconds. Capturing the TDOA within this accuracy would require checking for the sound's arrival this frequently – a polling system operating at around 125KHz. ADCs that are compatible with the RPi are costly, a solution to which is polling the digital GPIO pins for the TDOA, which can be performed more rapidly and then packaged with a single ADC conversion containing the audio's amplitude-against-time profile. At the risk of smoothing out clicks that occur very close together, a capacitor could also be used in between the amplifier and ADC where the transience of the amplifier is insufficient to ensure that each impulse's duration spans a single ADC conversion.

2.1.4.3 Solution

Four piezoelectric sensors dedicated to TDOA will pass their signal into a comparator which will scale the signal to a logical 1 on the controller's GPIO pins, which will be polled sequentially at a rate which renders the lack of unison in the checks negligible. When a logical 1 is received, a dictionary of the pin number against the number of cycles of checks that occur before, or until, the reference pin then receives a 1 will be formed. A parallel thread will begin to poll the ADC at its maximum rate for a set duration in hopes to acquire the entire time spectrum of the smoothed communication, before the dictionary and signal are packed and sent over the network to a more powerful PC for analysis. An infinite cycle will then see the system begin to wait for its next logical 1.

At the more powerful PC, the data will be interrogated to find the frequency domain of the signal, calculate its origin and the average magnitude of the vocalisations. The calculated origin can then be labelled as having been produced by the tracker's closest specimen's ID at that given time, forming valuable data for analysts and the predictive neural network which can backpropagate the peak frequency and average magnitude during training.

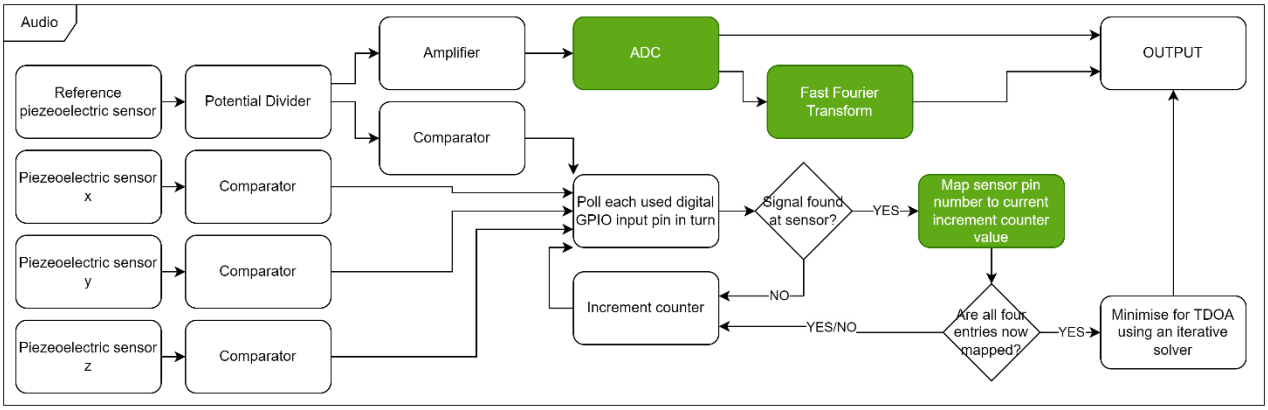
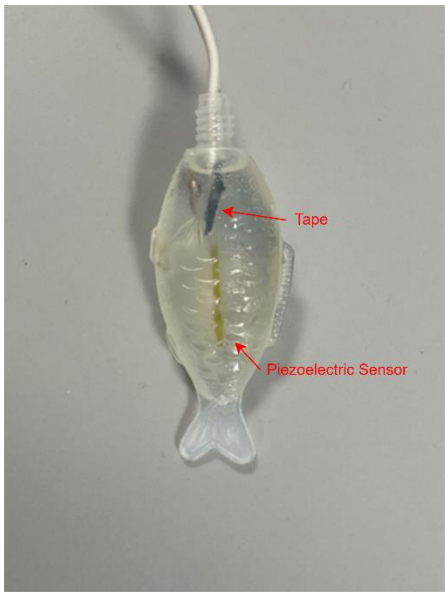


Figure 13: Flowchart summarising the audio capture and analysis.

2.1.4.4 Piezoelectric Signals

Piezoelectric devices vary in shape and size, with large disc shaped that often specialise as buzzers and bimorph sensors that can be very small, meaning that there is very little material capable of flexing and producing voltage, at the advantage of easily being encompassed within waterproof casing. Two solutions are at hand: piezoelectric buzzers on the outside of the tank and submerging piezoelectric sensors entombed in protective casing, neither of which can directly be tested within DC's lab due to the lack of measuring equipment such as oscilloscopes.

To gauge the performance of each variation, an emulation of the fishes' circumstance was developed by encasing a piezoelectric sensor and applying a small amount of alternating current through it to produce a buzz. A sealable container was then used to submerge the device in water while passing the safety inspection for the lab, which is harsh on liquids, allowing access to the signal generator and oscilloscope.



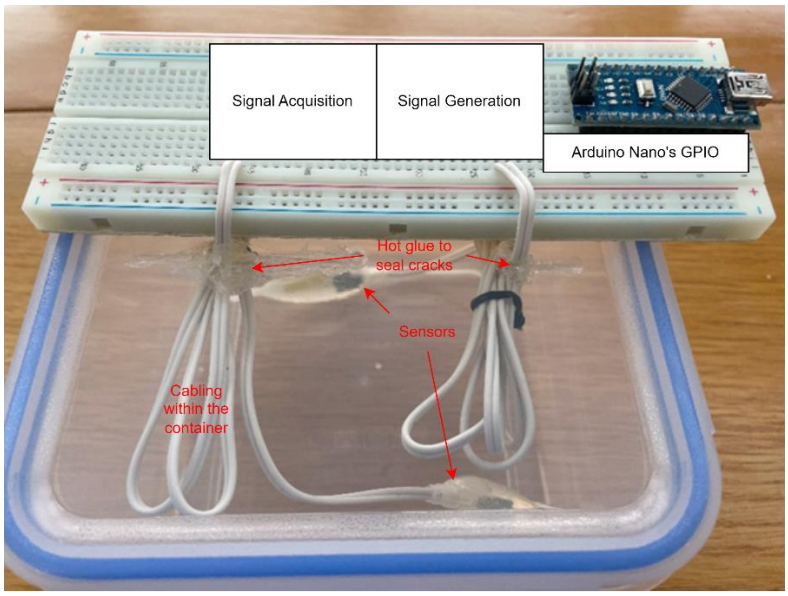


Figure 14 (left): A piezoelectric sensor cast in epoxy resin within a soy sauce packet for sushi..

Figure 15 (right): A repurposed container, modified to allow the sensors' wires through the lid but not water.

Bimorph sensors used in this project arrived from RS components with connecting wires of a few millimetres in length. Extensions were added to these by soldering the end of 2-core, non-coaxial cables closer to 700mm to the wire ends, before the chance of short-circuiting the exposed areas was reduced by wrapping one in tape.

The sensor and a centimetre of the protected cabling were then pressed into the entrance of a 3ml plastic packet that would normally be used to serve soy sauce in sushi restaurants, which would waterproof the electrical device. Glue would ensure that water would not enter the packet and acoustically couple the sensor with the packaging's surroundings, meaning that the chosen adhesive must be electrically insulating and have high compressive strength.

Epoxy resins are using in various fields including insulating materials and encapsulating and packaging materials for electronic devices [10]. When combined with a hardener, a curing reaction is caused which changes its physical properties. A 2:1 ratio between the adhesive and its hardener produced a solution viscous enough to be injected into the soy sauce packet while the wires were also present at the opening. Positioning the components within the packet beforehand ensured that the fragile, exposed copper connecting the wire to the sensor would not be pressed together when submerged.

48 hours was allowed for the glue to set before an oscilloscope was directly connected to the cable ends to prove that an electrical signal was produced when the device was knocked against. The lid of a standard container, present in figure 15 was then punctured twice, allowing the cables of two sensors to pass through it. One sensor would be used to produce a vibration, in hopes that the other would sense the signal.

A minimal length of the sensors' cables was exposed through the lid, meaning that once the signal acquisition is in place for the final product, freeing the sensors from the lid by cutting the cables would expend a minimal amount of their length. Hot glue was then used to seal any remaining gaps in the lid to reduce the chance of water escaping the container.

Once the sensors were submerged within the container, an oscilloscope was used with DC coupling to probe the signal at the end of the cable. A roughly 40mV peak-to-peak trace at precisely 50Hz was observed. This is the national grid's frequency which suggests that the noise of appliances within the lab was being picked up. Once a signal generator was used to induce a vibration in the water using the other sensor, superposed on the 50Hz signal was the much weaker trace of the vibration.

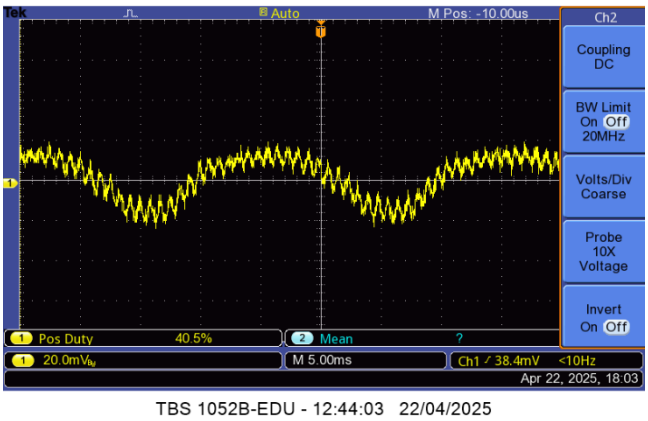


Figure 16: A 50Hz signal with the target signal superposed on it.

2.1.4.5 Signal Acquisition

Digital GPIO is to be used for the acquisition of these signals, meaning that a preset threshold must be decided to which the detections will register as a logical high. Filtering must be introduced so that only impulses created by the fish will arrive at the comparator as the imposition of noise may falsely register as high signals.

Equally, because piezoelectric sensors create alternating current, for a particular impulse, the received signal cannot be guaranteed to be either above or below ground. At the comparator, negative voltages can be misinterpreted as high signals with disregard to the threshold, meaning that noise will slip through without amendment. A solution to this is to rectify the signal so it is always positive.

Signals never exceed the threshold of the diodes available in the lab, so it is reasonable that the rectifier is active using op-amps. If the signal is applied to the inverting input at one and the non-inverting input of another, each with no bias, then the combined outputs will be the rectified signal which can also be amplified.

LTSpice was used to demonstrate the problem at hand and derive a theoretical solution: the noise is ambience from the whole room that the tank is placed in, so the superposition of a single sensor in a separate tank with no fish might leave only that which is desired. Outlined in figures 17 & 18 is the key principle:

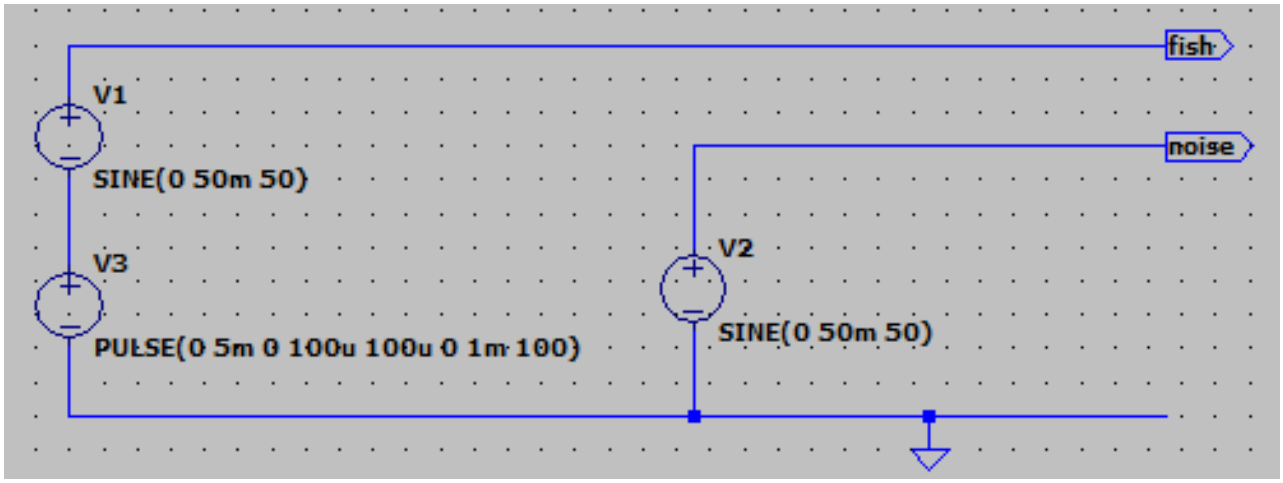


Figure 17: Subnets that emulate noise within the fish tank (V1), the impulses of the fish (V3) and that which might be attained from a separate, fishless tank (V2)

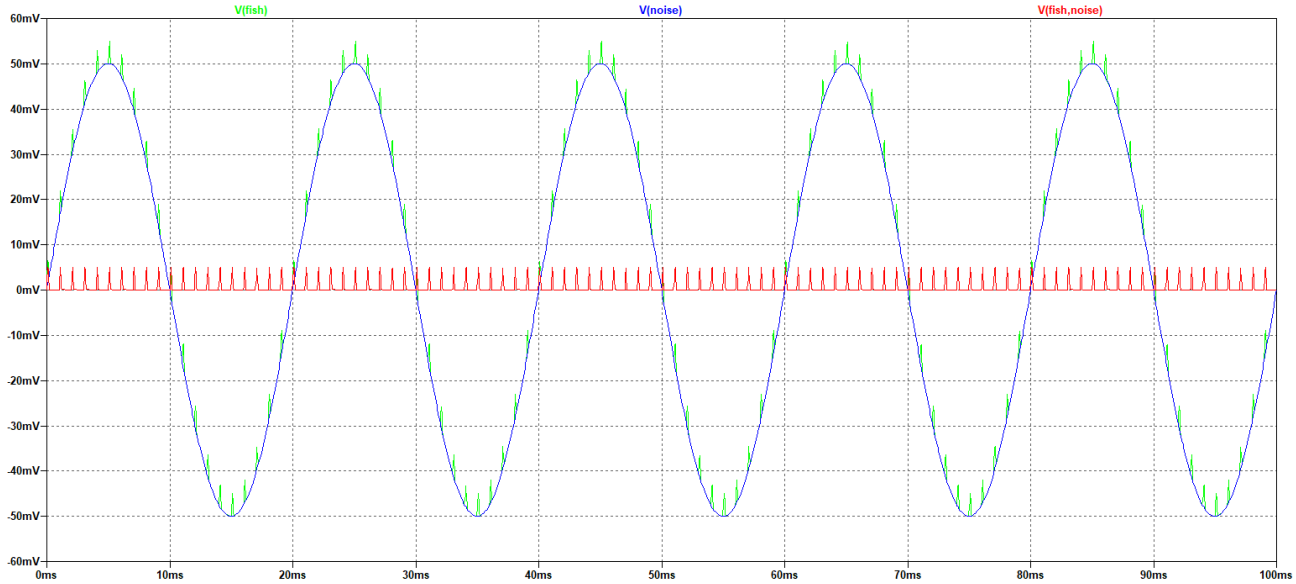


Figure 18: The signal from the fish tank (green), the signal from the empty fish tank (blue) and the potential difference between these two subnets (red)

Depicted is an ideal circumstance where the noise is of equal magnitude at each sensor. In real life, this may not be the case. Equally, the potential difference in the two signals has no reference to the rest of the circuit's ground. To acquire this difference with respect to the rest of the circuit, a differential amplifier is to be used.

A large gain was chosen so that the difference was large relative to the original signal. With the clicking at the non-inverting input, the noise then only produced an offset as shown in figure 19:

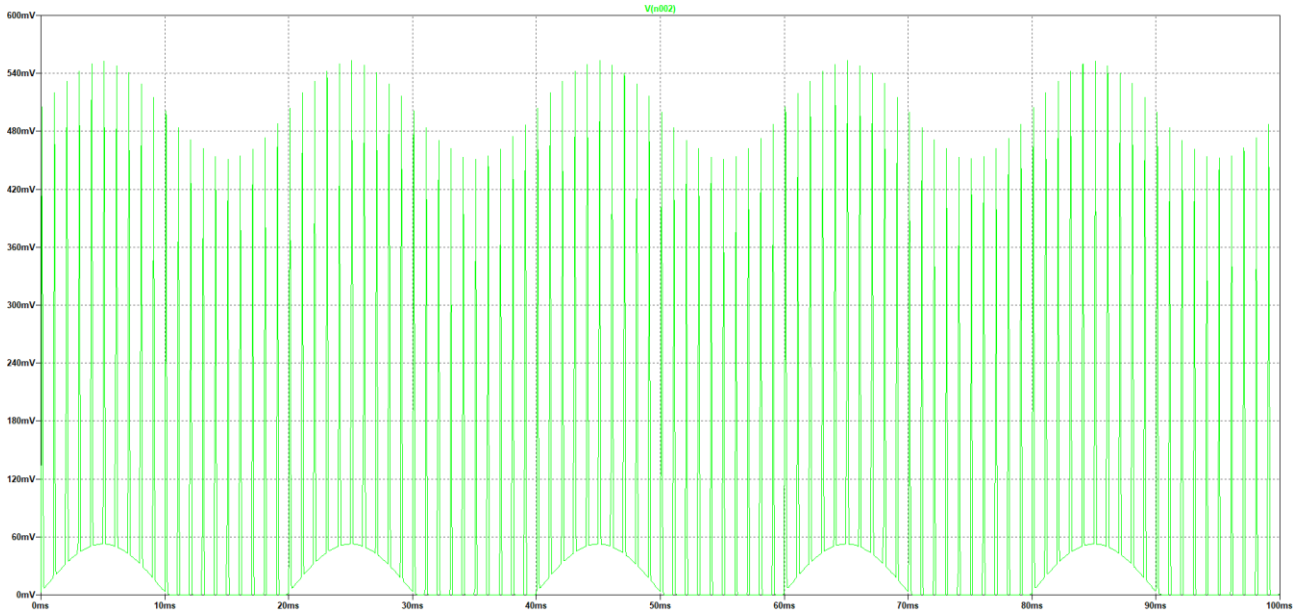
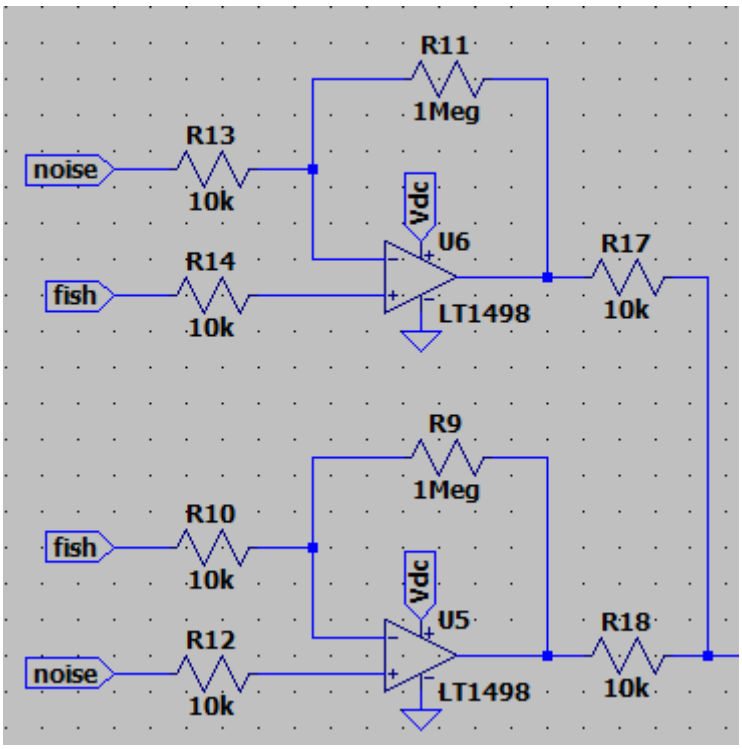


Figure 19: Simulated trace of the differential amplification of the two signals.

A carefully chosen comparator threshold will then trigger for each click occurring in the original signal. Next, Negative voltages in the original signal must be considered, as the lower power supply rail at ground will cut this off. To amend this, a second, identical amplifier is introduced where the signal is now at the inverting input. This will scale that which is lost in the initial circuit to its relative positive amplification. Summing the two signals will then produce a fully rectified output. This is better demonstrated when the original signal is also oscillating.



It is important to note that adjusting R13 and R14 allows for the scaling of each signal's influence on the result, meaning that if the noise is greater at one sensor than the other, these can be fine-tuned for minimal noise. The same applies to R10 and R12.

$$V_{out} = \frac{R11}{R13}V_{noise} + \frac{R11}{R14}V_{fish}$$

Due to discrepancies in component tolerances, R17 and R18 can then be adjusted to equivalate and summate the two outputs.

$$V_{sum} = \frac{R17 * V_{U6} + R18 * V_{U5}}{R17 + R18}$$

This summation may need to be performed through an additional differential amplifier in practice but remains as shown in figure 20 to outline the fundamental principle.

Figure 20: The devised setup for noise omission and rectification.

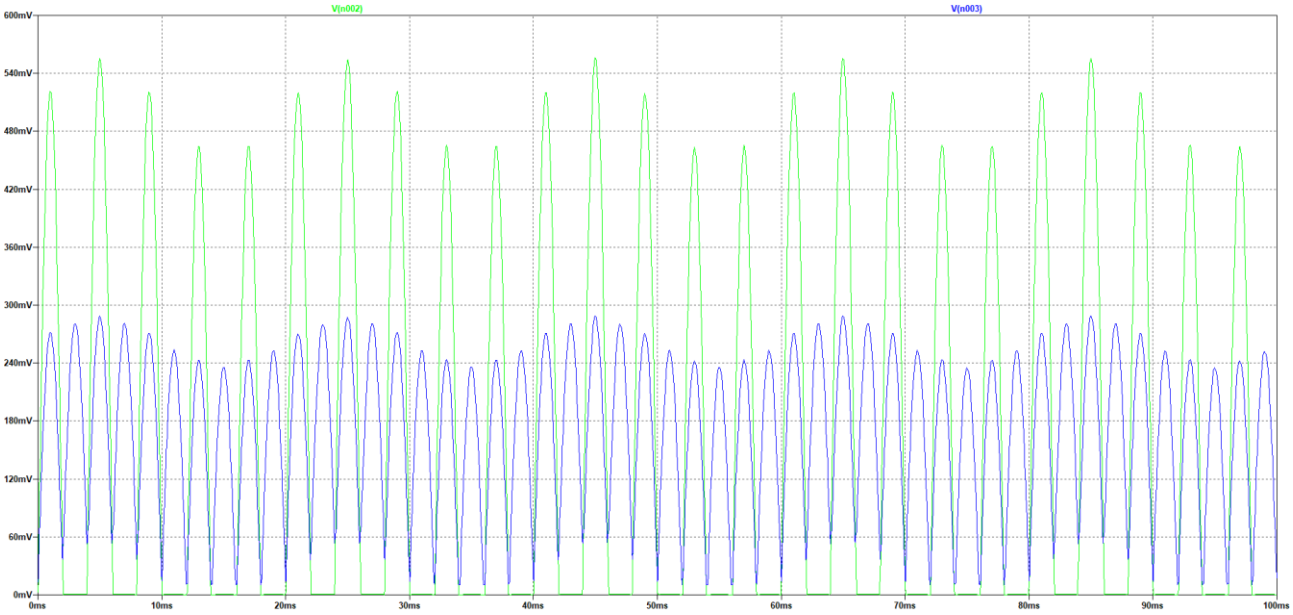


Figure 21: The output of just one differential amplifier (green) and the fully rectified signal (blue)

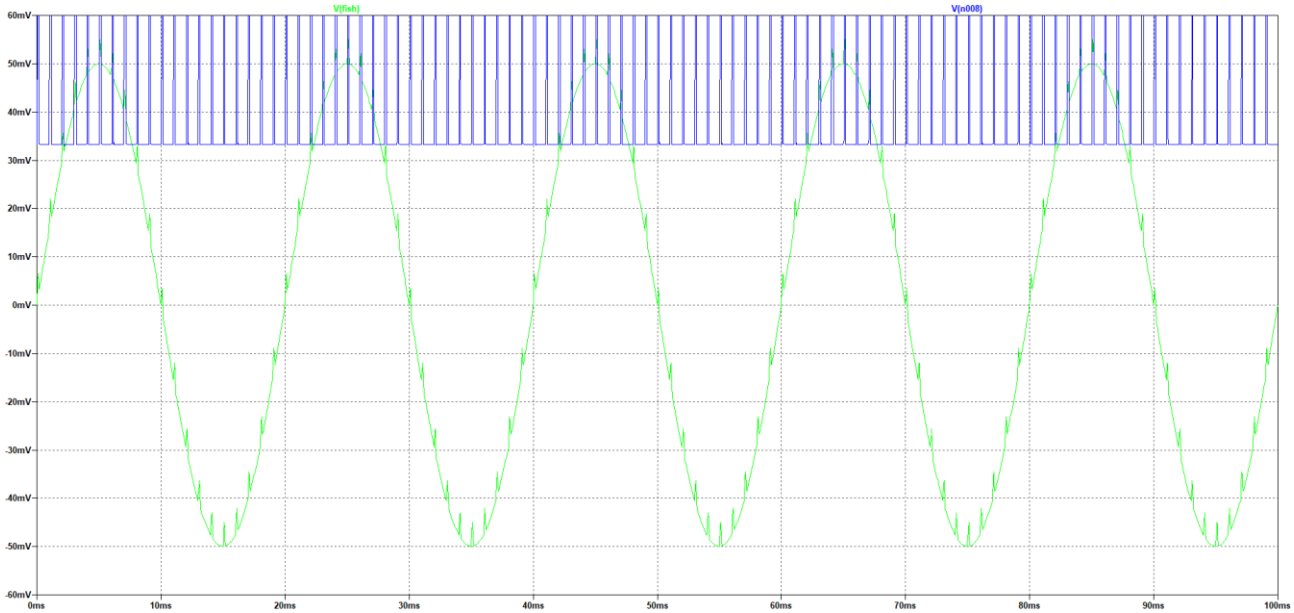


Figure 22: The input (green) and output (blue) of the whole process (blue pulses that go beyond the y-axis limits all peak at 5V).

Figure 15's blue trace depicts what might be seen at the RPi's digital GPIO pins after passing a comparator in addition to the derived circuit for the inputs shown in green, concluding the effectiveness of the simulated circuit.

2.1.5 Required Hardware

Aforementioned equipment is summarised in table 3:

Table 3: A summary of all aforementioned equipment

Item	Quantity	Application
Camera	2-3	Enables the three-dimensional tracking discussed in section 2.1.1.2. A third camera grants additional immunity to the loss of a specimen's track.
Camera Lens	0 or 3	Positioning the cameras further from the subject could increase the tracker's reliability (section 2.1.1.2)
Piezoelectric Sensor	4-5	Sound triangulation requires a reference sensor then a single sensor in each dimension x, y and z. The signal from the reference sensor may be split amongst the digital and ADC pins using potential a potential divider before amplification to remove the need for a fifth.
Comparator	4	Scales the piezoelectric signal to register on the controller's digital GPIO pins at the compromise of its detail.
Operational Amplifier	1	Scales the piezoelectric signal so for better resolution at the ADC while maintaining its detail.
Microcontroller or Single-Board Computer (SBC)	1-3	If two cameras are used, a single controller can read both cameras, given they are close enough together. Introduction of another camera would require an additional controller. Where more distance is required between cameras, a controller will be required for each.
Personal Computer	1	Real-time image processing with a highly accurate model requires powerful hardware [CITE HERE].

2.1.6 Photography Performance Profiling

2.1.6.1 Camera Resolutions & Lenses

A device must be in place to drive any cameras that are involved in the project, which also places the device within DC's lab. The specification states that surveillance footage of DC should always be accessible to the client, meaning the footage must be served to a separate device which might host the GUI. PCs with processing power capable of real-time image detection are outside of the project's budget and it would be unreasonable to fixate such a multipurpose device on a single task, leading to the dedication of an individual single-board computer (SBC), to the acquisition of sensor data, before transferring it to another computer for analysis and access to the results. The means of transfer are discussed in section 2.1.7.2.

Foremostly, the camera and controller must be compatible. Previously undertaken work saw the Raspberry Pi (RPi) 3A+ drive an RPi ZeroCam as part of a real-time computer vision solution for autonomous driving. This combination of components and documentation made available by [11] proves reputable such that the project has been undertaken annually by the University of Nottingham's cohort of first-year Electrical and Electronic Engineering (EEE) undergraduate students since 2018.

Unfortunately, the ZeroCam shoots photos at just 320x240px. Given an ideal circumstance where a side of the fish tank encompassed the entire frame shot by the ZeroCam, at just 12mm in length [2], DC are a $\frac{12mm}{275mm} = 4.4\%$ of the length of the tank, meaning they would accommodate just $320 * 4.4\% \approx 14$ columns of pixels and even fewer rows, which is unlikely to grant an image detection model enough information to be able to form a reliable class for the fish. A much higher resolution is required at the expense of higher GPU demand on the chosen client of the RPi's images. Available is the RPi HQ camera capable of shooting frames at 4056x3040 pixels which would resolve each fish with $4056 * 4.4\% \approx 177$ columns of pixels, entailing smaller detail on the fish. Two of these were ordered alongside their corresponding lenses. Whether or not the cameras were to be distant from the tanks, the lenses could focus the cameras on the tank, ensuring there would be less wasted data (i.e. the fish tank's surroundings) per frame, should the cameras be positioned further from the subject.

Inward investment was made by Dr. Rob Wilkinson as a stakeholder in the project through the order of two RPi HQ cameras, their corresponding lenses, lens caps, cabling and tripods to mount them. Once the cameras are monitoring DC and serving this surveillance to a device outside of the lab, he might immediately see this accessibility as enhanced quality of life.

2.1.6.2 Server Memory Sizes & Clock Speeds

RPi SBCs that can interface two cameras simultaneously are the RPi Compute Module 4 (CM4) and RPi 5: the CM4 houses a 2-lane CSI and a 4-lane CSI port [12] while the RPi 5 has two 4-lane MIPI transceivers [13]. Important factors which distinguish the devices are backwards compatibility with prior OS versions and access to a hardware H.264 codec.

Video compression could be required in the transfer of frames captured by the RPi, as the bandwidth of raw video stream would likely surpass limits imposed by solutions such as Wi-Fi. H.264 encoding can achieve up to a 50% improvement in bit-rate efficiency compared to previous standards and was adopted by many application standards [14]. It is clearly stated in [12] that the CM4's H.264 codec can encode full HD (1920x1080px) video at 30FPS, while [13] neglects to mention this. Equally, the official RPi store makes it clear that the latest version of RPi OS is needed for the RPi 5.

With it clear that the CM4 is preferable for the task at hand, an amount of RAM was to be selected at purchase, with options of 1, 2, 4 and 8GB. When the RPi HQ cameras arrived, an RPi Model 3A+ was setup on its legacy 32-bit operating system to profile the amount of memory that will be required on the chosen SBC for the dual camera setup. [11] was implemented in the lightweight C/C++ editor, Codeblocks, before an experiment saw the size of the captured frames increase incrementally alongside the amount of memory that had to be designated to the GPU. Figure 23 outlines this process:

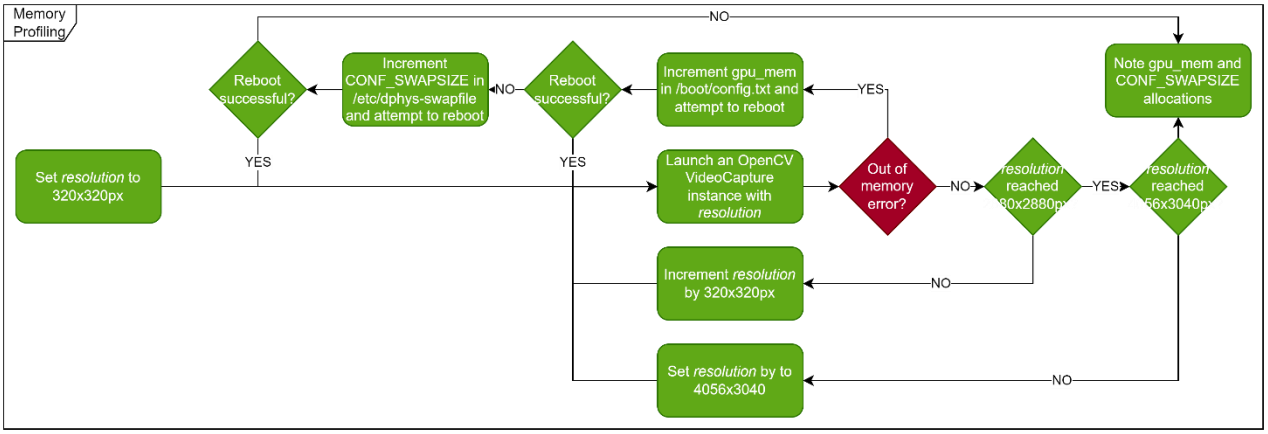


Figure 23: Incremental process to assert the VRAM and RAM required for the chosen SBC.

Figure 23's process saw 196MB of GPU memory finally accommodate OpenCV shooting frames at 4056x3040px, which would see cannon frames require $196 * 2 = 392MB$ of RAM for the GPU alone, which would leave the 1GB model $1024 - 392 = 632MB$ for alternative processes including running the OS. If H.264 encoding is required in the future, it is possible that an additional $392MB$of VRAM would be reserved for the codec, making the 2GB model a cost-effective choice that provided overhead for the encoding.

Wi-Fi and eMMC were selected as optional extras so that the images could be transferred to the main PC over the network without requiring ethernet in the laboratory, while eMMC provided greater longevity over SD cards for non-volatile storage [15] and removed the need for SD adapters being at hand.

2.1.7 Network Protocol Speeds and Reliability

2.1.7.1 TCP Using 0MQ

A single 4K JPEG image occupies around four megabytes. Sending dozens of these over the network each second could easily see the network begin to drop data packets when overloaded, which can manifest as large fragments in the image instead of the true pixels or high delays and low framerates due to the dropping of entire frames. A couple of key transmission protocols and their libraries were analysed so that the solution which delivers the greatest quantity of artefact-free frames per unit time will be at play in the final product, namely 0MQ, and GStreamer.

0MQ's application produces a unique solution because it purely encapsulates the sockets which the data is sent across, while others additionally organise the camera pipeline. Section 2.2.1.2 arranged the camera pipeline using OpenCV, which seamlessly integrated with 0MQ because the MatLike instance produced by VideoCapture's *read*() method is immediately compatible with OpenCV's *imencode*() then the 0MQ socket. Being sent over TCP, there was also assurance that every packet would arrive whole at the recipient.

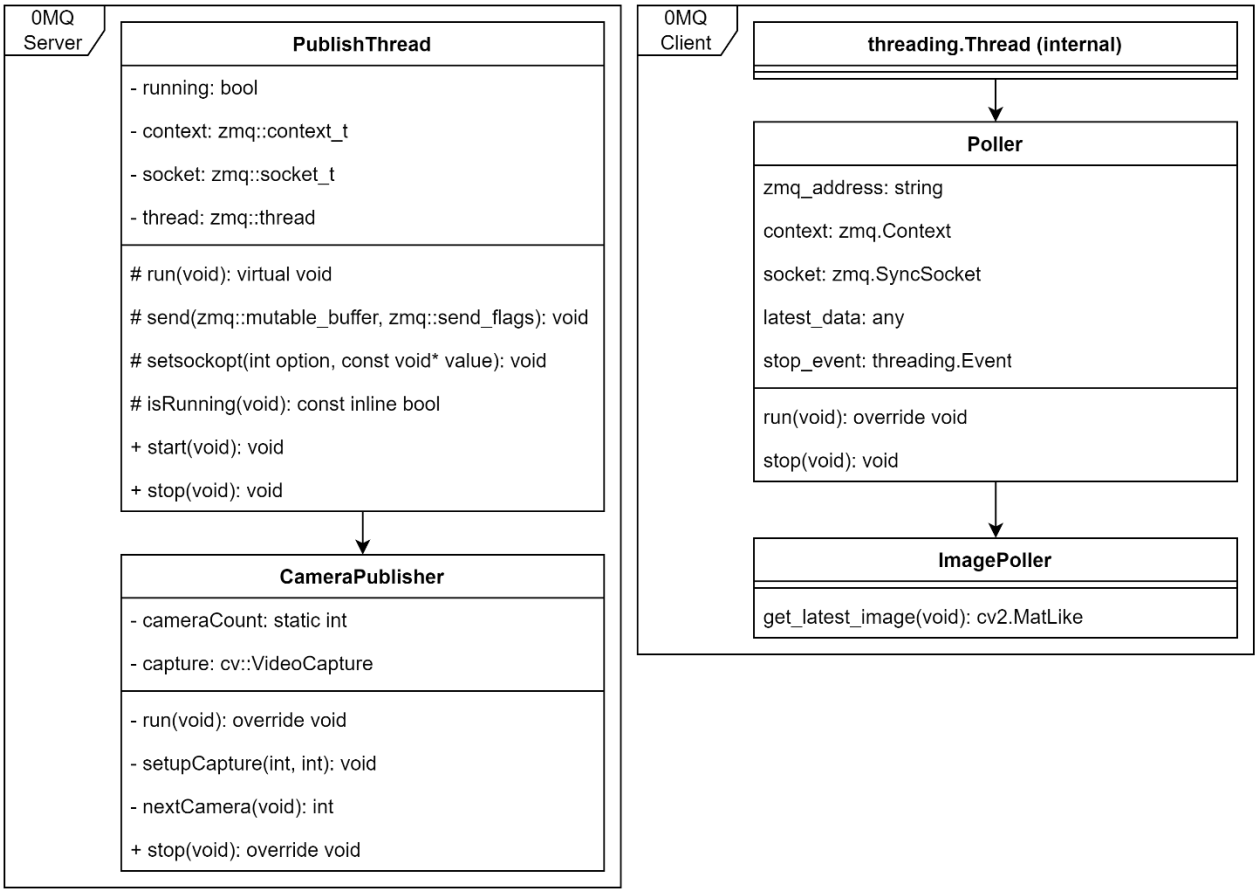


Figure 24: Unified Modelling Language (UML) class diagrams representing the server (left) and client (right) side of the 0MQ solution. The client was written in Python, so all variables are within public scope.

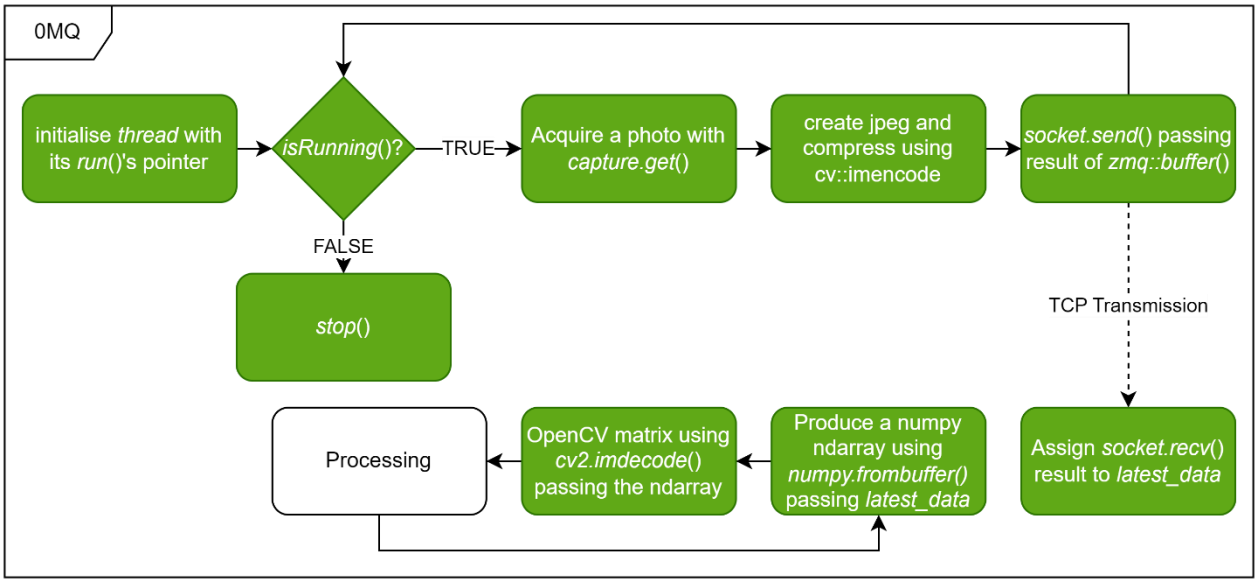


Figure 25: Flowchart summarising the steps in the 0MQ solution.

2.1.7.2 RTP/UDP Using GStreamer

GStreamer and FFmpeg are both open-source libraries which encapsulate the transmission of video feed over RTP/UDP at a high level. Command lines can be executed with several arguments that dictate the preprocessing steps of the execution. These are called pipelines, with varying syntaxes across libraries.

OpenCV can use GStreamer on its backend but must be built with GStreamer support to be compatible. Having been proven to work separately, OpenCV's "sources" directory was set as the source of a CMake build.

After configuration, GStreamer was unfound although in the system path, meaning that the paths were manually supplied. Reconfiguring the project then indicated that GStreamer was available alongside FFmpeg before the project was generated and built in Visual Studio.

Under the build folder were several Python versions in the Python directory. Of use was the .pyd file found under "python-3.10"; copied into the Anaconda Navigator kernel's site-packages directory for use alongside libraries such as YOLO. GStreamer's availability could then be confirmed using *cv2.getBuildInformation()*.

OpenCV's (cv2 namespace) *VideoCapture* was then initialised with a string as its first argument and an enumerate flag *cv2.CAP_GSTREAMER*, meaning that the GStreamer backend would treat the string as a pipeline which contained the following elements:

Table 4: Recipient GStreamer pipeline

Pipeline Element	Reason
udpsrc port=5000	Tells GStreamer that the information is arriving via the UDP protocol on port 5000
application/x-rtp, encoding-name=H264	The source is using H.264 codec. These are wrapped in RTP packets for network streaming
rtph264depay	Extracts the raw H.264 bitstream from the RTP packets
avdec_h264	Decodes frames from the raw bitstream
videoconvert	Performs the conversion from the decoded frame to the desired format
video/x-raw, format=BGR	Explicitly ensures that videoconvert will use the BGR colourspace for compatibility with OpenCV
appsink sync=false	Delivers the frame to the *VideoCapture* object. Sync=false delivers the frame as fast as possible instead of awaiting its timestamp. This gives the rest of the program more time to process the frame.

A hardware H.264 codec was part of the specification when selecting a suitable SBC for the project in section 2.1.6.2. C++ on the server side was then tasked with the launch of parallel threads again with GStreamer's native C API in place of OpenCV and 0MQ.

A namespace was developed to encapsulate the parallel launching of multiple pipelines at once. This included two functions: *gCamera()* and *launchGThreads().* Arguments specify the network port to use, width and height of the image frames and finally a *std::vector<int>* defining the camera indices under */dev/video* to launch.

An instance of GStreamer's *GstElement* was assigned to the value returned by the execution of *gst_parse_launch*(), passing the pipeline as a string/character array for C typing, alongside a pointer to a null pointer of type *GError*. If the pipeline fails to launch, *gst_parse_launch*() returns *null* and the *GError* instance is assigned, meaning that checking against this allows the message member of this to be printed to the console for debugging. The full process is outlined in figure 26, which launches the pipeline defined in table 5.

Table 5: Server side GStreamer pipeline

Pipeline Element	Reason
v4l2src device=/dev/video *deviceNum*	Video4Linux2 is a framework on Linux based operating systems like RPi OS using the legacy camera stack. The device property dynamically specifies the path to the camera's driver by concatenating the variable *deviceNum*
video/x-raw, width=*width*, height=*height*, framerate=10/1	Specifies that the camera will capture raw video at 10 frames per second, which is the maximum for the HQ camera at full resolution and adequate for the solution. Dynamic selection of resolution by passing *width* and *height* enhances scalability
videoconvert	Performs the conversion from the raw frame to the desired format
x264enc tune=zerolatency bitrate=*bitrate* speed-preset=ultrafast	Encodes the frame using H.264, tuned to minimise encoding latency by disabling slow conversion techniques. Dynamic selection of bitrate adjusts the compression rate of the video based on the network capacity, while the speed preset to ultrafast minimises the CPU load of the compression.
rtph264pay	Packages the encoded data in RTP packets for network streaming.
udpsink host=*host* port=*port*	Sends the RTP stream over UDP to *host,* the IPv4 address of the PC client, over the port specified by *port*

Where the *x264enc* element is defined, the hardware encoder can be specified instead using *v4l2h264enc* in its place. It is recommended that in sequence with this, an element *! video/x-h264, level=4 !* takes the place of the arguments previously passed to *x264enc*. This tells the pipeline that the RPi's hardware encoder can encode up to 1920x1080px video at 30FPS as defined in [12].

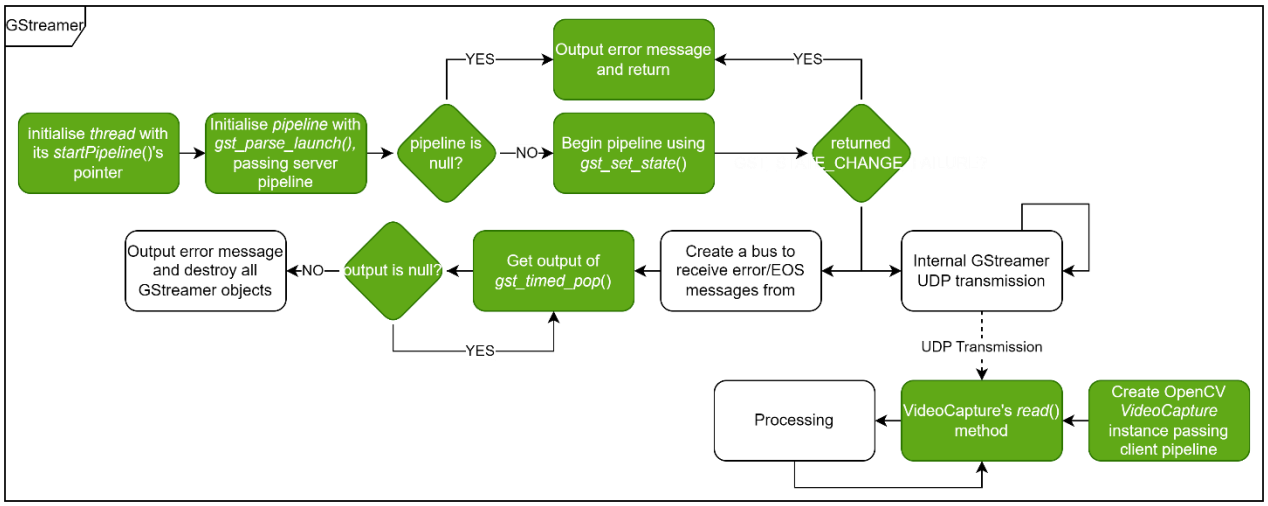


Figure 26: Summary of the GStreamer solution

2.1.7.3 0MQ Versus GStreamer

0MQ was the original plan for streaming over the local network as the relatively short distance the data travels was not expected to be the solution's bottleneck. The network in which it was tested, however, uses powerlines for transmission over ethernet and a hub which have been found to be very slow. To combat this, a solution which entails the transmission of only the pixels which change colour between frames would reduce the level of compression required, better preserving the original photograph, which led to consideration of GStreamer's implementation.

With both solutions then at hand, table 4 was drawn up to compare the advantages and disadvantages of each. Please note that the headings refer to their aforementioned implementations rather than the libraries themselves.

Table 6: Advantages/disadvantages of the two solutions and their importance

Target	0MQ	GStreamer	Importance
Framerate	Depends wholly on packet size	Always met	Fish can't travel too far between frames.
Delay	Scales heavily with resolution	Very low	Lens calibration
Resolution	Always met	Always met	Detection reliability
Colour	Reduces with compression rate	Improves with bitrate	Detection reliability
Reliability	Always met	Size/chance of artifacts increase with bitrate	Artifacts may cover subjects
Scalability	Could be optimised beyond what is presented by expert software developers. Modular design keeps the solution concise albeit bulky	Limited to the elements made available by the API. Lower modularity but minimal quantity of modules	Futureproofing

Red highlighting emphasises disadvantages; green vice versa. Under the **importance** heading, red highlighting represents high importance while green can be overlooked.

Overall, the notable trade-off between the solutions is reliability vs. framerate, so each must be quantified to conclude the most suited solution to the final product. This is where the *processing* module in flowcharts 9 & 10 is implemented. The same network and demand conditions were used for each test.

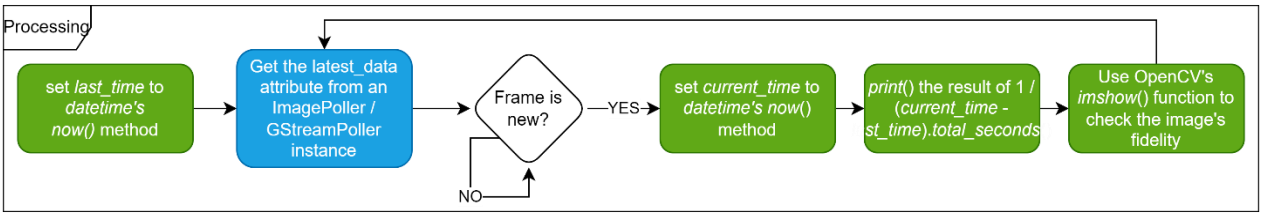


Figure 27: Outline of the "processing" module referred to in figures 9 & 10; for framerate acquisition

Both cameras were posed to face a stopwatch with a refresh rate of 60Hz, ensuring difference between current and prior frames. Each solution was run until 420 photos were stored to the hard disk – a saving process which is negligible compared to the transfer rate of frames, allowing the review of the quality of images produced. Of these, the GStreamer CPU H.264 encoded solution's photos arrived with excessive artifacts in 18 or 4.3% instances of pictures in the dataset. Every image in the 0MQ solution arrived whole with high quality. Average framerates were also recorded over six resolutions for each implementation to derive figure 28:

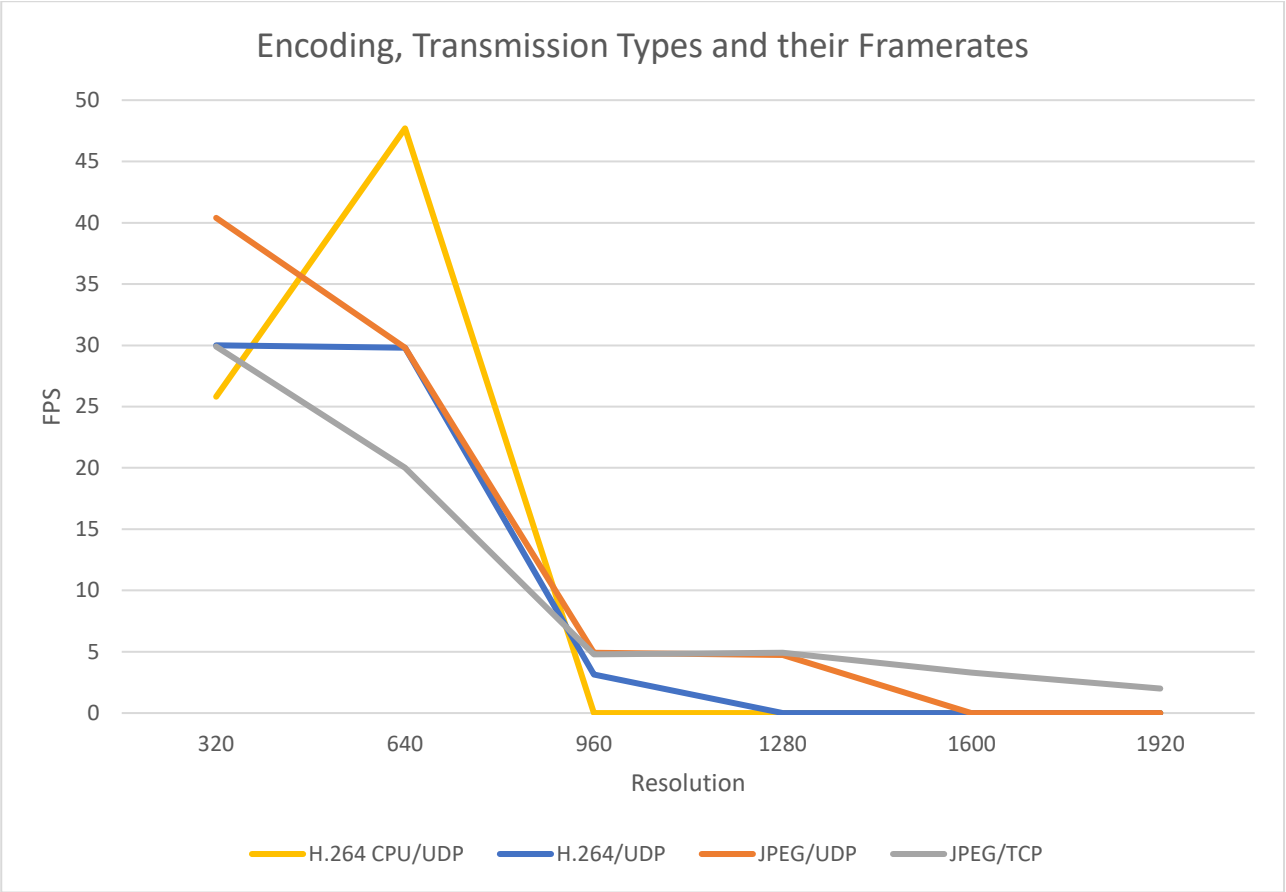


Figure 28:The framerate achieved by encoding types and their transmission protocols for different resolutions.

When encoding using the CPU, the flag speed-preset=ultrafast was passed to the encoder element of the pipeline. All other settings produced less than one frame per second. Resulting video at 640x640px included black pixels at the bottom of the frame, which may be because the frame was pushed through the pipeline before it was complete due to strict timestamping with the ultrafast speed preset.

Between 320x320px and 640x640px, H.264 and JPEG/UDP proved to reliably support greater framerates than the TCP solution, before reaching a trough below the TCP solution's rate of transfer at resolutions greater than 1280x1280px. Assuming the use of the "highest framerate solution" across all resolutions, the maximum data transfer rate can be quantified for each resolution as shown in figure 29.

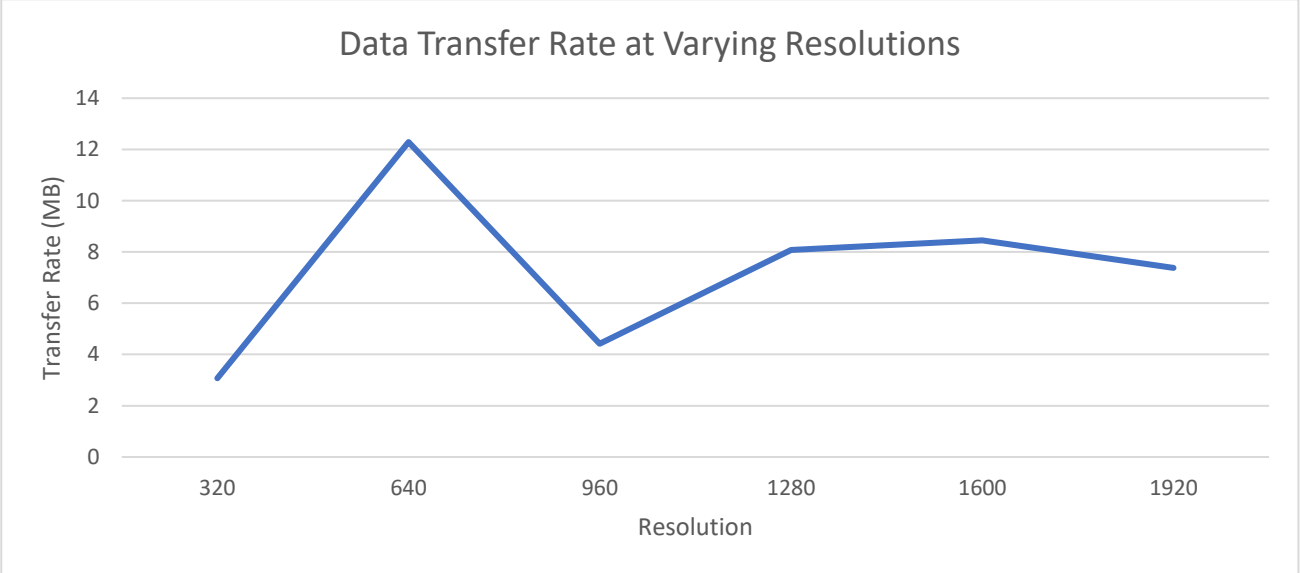


Figure 29: Achievable data transfer rates when using the most optimal solution for the given resolution

2.2 System Development

2.2.1 Adapting the Tracker

2.2.1.1 Balancing Resolution and Framerate

Section 2.1.7.3 concluded that the nonlinear degradation of framerate with increasing resolution presents two extreme circumstances: a high framerate can be achieved between 320x320px and 640x640px, with a severe drop for resolutions beyond this, however, JPEG/TCP's framerate reached a plateau such that at 1600x1600px the data transfer rate was reasonable. The deciding factor is the rate at which DC travel within their tank. When writing frame-by frame at 2048x2048px (frame write time can be deemed negligible compared to transfer rate), consecutive frames written to a directory, apparent in figures X & X were set apart by a duration such that a direct approach of tracking, even with the human eye, is insufficient to predict which specimen is which between frames.

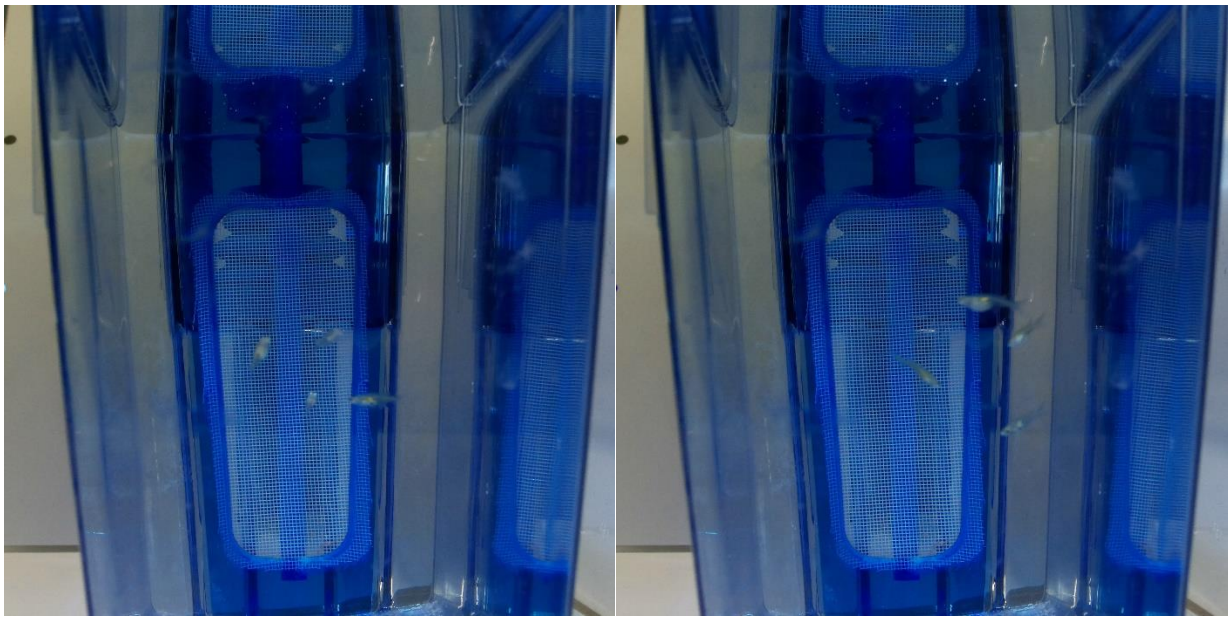


Figure 30 (left): DC in one frame and the next consecutive frame when recording at 2048x2048px (not to scale)

2.2.1.2 Training the Image Detection Model

Bounding boxes were manually drawn onto a dataset of one hundred photos from each of the two orthogonal cameras, taken 5 seconds apart from each other, capturing four fish in a single tank. The delay increased the likelihood that the fish would move or change orientation in each frame, increasing the variety of a set number of annotations which were to be performed. Because time was not of essence in this instance, frames were captured at a resolution of 2048x2048px, which could be downscaled at the dataset's creation if a lower resolution is required, enhancing the versatility of the images gathered in a single sitting.

YOLO version 8 expects a .yaml formatted file which outlines the file paths, which can be relative to the file's location, to three separate directories each containing a set of images: train, validation and testing sections. The training set is backpropagated through the model to adjust its biases. At the end of each epoch, the validation set evaluates the model's performance, providing accuracy metrics which decide the epoch where the model performed with the most precision. Providing a test set is optional: this is unseen data which is used after training to grant the fairest evaluation of the model's performance. Here, performance metrics will be used to estimate how effective the model will be as part of the final product, meaning this set made up X% of the final dataset.

It is assumed that each image's corresponding annotations are in its neighbouring directory, which take the shape of a .txt file with columns separated by spaces (' ') that represent the class ID of objects present in the image, their bounding boxes' normalised pixelwise centre x and y coordinates; width and height - in that order. Each entry is separated by newlines ('\n').

A dataset produced from one hundred unique images can train a model to a recall rate on unseen data of more than 90% [CITE]. Once Annotating 100 images could be time consuming as it is a manual process, which

is where a program or application which visualises the drawing of boxes onto the immediate image and maps these to the annotation file would optimise the process. This exact functionality is available for free on [16]: a web application specialised in the creation of computer vision applications. Within the workspace created on [16], enhancements including shading were available to the annotated images, producing duplicates of the original images that are lightened and darkened to create a more extensive; light-immune dataset. When exported in the YOLOv8 format, the .yaml file and expected file structure were already zipped together.

Training the model was the next task for which a dedicated Python script was developed. This made the process more repeatable than running YOLO from the command line. The functionality is outlined in figure 14:

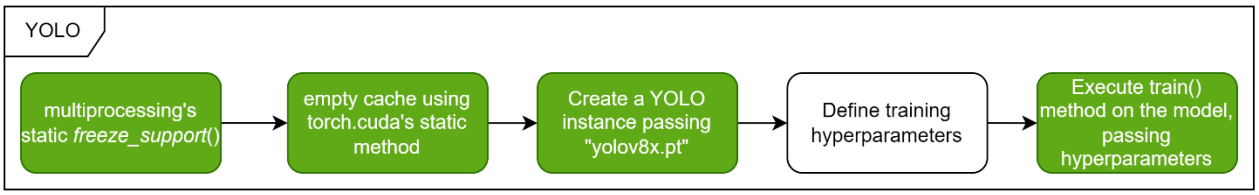


Figure 31: A process to setup and perform the training of a YOLO model

Hyperparameters are the most important part of *train()*'s execution, alongside parameters which define the "project" and the model's "name". These ensure that information about the model is stored to the disk, including the model itself. The "project" is the name of the directory which iterations of the model's training will be stored to. Hyperparameters have default values assigned by Ultralytic's developers, which provide a reasonable starting point [17].

For each iteration of training, an index is incremented and appended to the end of the "name" which stows the model at its "best" and last epoch of an individual *train*() execution. For further refinement, "last" can be trained further using more refined hyperparameters in hope of producing an even more effective "best" model, which is to be used in the real application.

Granted section 2.2.1.1, the initial training run used images of 640x640px, in anticipation of significantly higher framerates being required. Dependant on the network transmission of frames, Section 2.1.2.3 predicted that the maximum achievable detection rate was 25 FPS, which makes the detection algorithm the bottleneck on the program's framerate. It is important that as frames are stored to the hard drive as part of a video, the framerate is accurately predetermined, otherwise the interval between when the frame is appended to the end of the video would mismatch its timestamp at playback, causing a fast-forward/slow-motion effect.

Overhead involved in the queuing of data between threads during processing could easily unsettle the projected 25FPS performance granted by models between 3.2 and 11.2 million parameters, which deems 20FPS a more reasonable target, which includes models with up to 25.9 million parameters in the consideration of model sizes at implementation.

A model was trained at 640x640px for each model size with the testing results displayed in table 7. Notable metrics are:

- Box: The error in the model's predicted position and size of objects. A lower error would increase the chance of the box's centre from each perspective to land closer to its expected 3D mapping.
- Recall: The proportion of objects were recognised at all by the model – it is important that fish are identified as much as possible to reduce the likelihood of two fish going unidentified by the tracker at the same time as the fish would then be indistinguishable when reallocated to the tracker.
- Precision: The proportion of detected objects that were correctly identified – reflections and the background could be misidentified as DC. Where these land on the same epipolar line, it could be mapped to a 3D position in place of the real specimen. A greater precision reduces the likelihood of these misidentifications.

- mAP50 (Mean average precision at IoU=0.5): the precision of the model if the detections are only correct when they overlap the ground truth box (drawn from dataset annotations) by more than half.
- mAP50-95 (mean Average Precision averaged over 10 IoU thresholds from 0.50 to 0.95): the average precision of the model if the detections are only deemed correct when they overlap the ground truth box by a value increasing by 0.05, starting from 0.5, over 10 increments.

Table 7: Performance metrics derived from the testing portion of the dataset at the end of the training process

No. Parameters (M)	Box	Recall	mAP50	mAP50-95	Total Inference Time (ms)
3.01	0.970	0.925	0.939	0.564	5.30
11.1	0.970	0.900	0.944	0.550	16.0
25.8	0.990	0.875	0.959	0.571	22.8
43.6	0.948	0.950	0.972	0.563	45.8

Naming conventions used by YOLOv8 expected parameter counts as stated on [5], however, it is visible that the parameters for given models are lower, which resulted in lower inference times. The "medium" model (25.8M parameters) was projected a framerate greater than 40FPS, which lead to consideration of a larger model (43.6M parameters) which added the additional row, which presented a framerate just greater than 21FPS, leaving minimal headroom for other tasks in the application.

At the worst recall rate of 87.5% when using the "medium" model, there is a $1 - 0.875^4 \approx 41\%$ chance of at least one fish being missed per frame.

2.2.1.3 Majority Vote System

It is assumed that individual DC are indistinguishable from each other, aside from distinctions between gender. Here, cameras may not even capture the fish at a high enough resolution to be able to make this judgement, which means that when captured in an image, the fish can only be told apart by maintaining an internal digital ID throughout the program's lifetime. If fish are lost 10% of the time, there would be less than a half chance of the fish having the same ID as it started with by the 7th frame ($log_{0.9}(0.5)$).

A second camera provides another perspective on fish that might be missing in the other, which would allow a mapping of two independent external IDs to a single internal ID to be maintained in this instance. This would mean that both trackers would have to lose track of the same specimen for the internal mapping to finally become uncertain. This can be improved further by allowing any one external ID to be updated in the mapping per iteration when it can be affirmed that it is the same object as seen by the other tracker, meaning that both trackers would have to lose sight of the fish simultaneously to permanently lost its track. The chance of this becomes $0.1^2 = 0.01\ or\ 1\%$ per specimen. In an ideal circumstance, any ID has a half chance of having been maintained for $log_{0.99}(0.5) > 68$ frames.

Given the capacity of IDs that can be present at any one time, more immunity can be granted to the loss of a track: the next ID that is found by the tracker could be assigned to IDs that were previously lost. Now, if only one object is truly lost at any one time, it is guaranteed to be placed at its correct ID again once it is redetected. Now, the loss would only persist if two tracks were truly missing at any one time, which means that with a $1 - 0.99^4 \approx 4\%$ chance of a truly missing specimen per frame, the chance of two missing at a time would be $0.04^2 = 0.16\%$, which gives each ID $log_{0.9984}(0.5) > 446$ frames before its chance of being the same as when it was first registered falls below half. At 20FPS, this would take under 23 seconds.

Points 1 to 6 below outline a scenario to depict the algorithm's decision making:

1. To begin each iteration of the process, localisations within a set threshold of rows from each other in each frame are paired and listed, meaning that for two fish in both frames that are close to each other, four entries will be made, entering the "heightwise" YOLO tracker ID pairs as 00, 01, 10 and

11, where the first number is the ID presented by the first tracker in one orthogonal view and the second number is from the next tracker.

2. IDs in each frame are removed from the entries once they are assigned an internal pair, these are concluded using the previous internal mapping. If there is no prior mapping, then a first in first out method is used to choose the current pairs. This is likely to be inaccurate for the first few iterations, for example, chosen pairs would be 00 and 11 in this instance. A strong assumption is made here.
3. It can then be imagined that the fish separate and are no longer at similar heights. The next set of pairs might look like 01 and 10, because the IDs within each YOLO instance might not represent the same specimen – this deems the previous map incorrect, meaning that when the comparison is made using the majority vote, there are no absolute matches, however, the 0 in 01 matches the 0 in 00 and the 1 in 01 matches the 1 in 11. FIFO is again the chosen method, meaning that the internal mapping will reassign its first two indices to 01 and 10, fixing the tracking. The assumption has been amended.
4. If one of the fish travel quickly in one dimension, the camera's tracker might lose it, meaning that the new pairs could look like X1 and 10. 10 will immediately be registered and the 1 in X1 still matches the 1 in 01, meaning that the 01 will become X1 – it is important to note that although a 3D depiction of the fish's position is no longer available, it remains internally mapped to the 0^{th} index. The fish in the 1^{st} index is correctly tracked, while the 0^{th} fish's location only has a 2D projection.
5. Now, imagine the fish come close again. The new set of pairs will look like 20, 21, 10 and 11 again. Because the previous mapping is X1 and 10, 10 will be maintained due to an absolute match. This now makes 1 unavailable in the first tracker, meaning that the mapping cannot be of the form 1X. A previous mapping is X1, forcing the 11 pair out of the question and amending the 0^{th} mapping to 21, which must be the specimen that was lost in the prior frame. Although two hinderances were posed to the tracker in this instance, we can be sure that the internal 0^{th} and 1^{st} 3D tracks are intact.
6. Using the known number of specimens in the tank, if one is lost in both views, an XX in its internal mapping will find no matches with the other tracks found in the next frame but the majority vote may match up the rest of the fish, in this case, the unassigned mapping will be correctly appended against its previous ID. If one fish is completely lost at a time, it can be found. Only if numerous fish are lost in both cameras at once is their track permanently lost.

Additional immunity can be granted to point 6 by introducing a direct approach of 3D tracking (currently the direct-approach tracking is performed on two separate 2D detection models). When mappings are truly lost, their last known 3D coordinates could be stored, meaning that on the next iteration, if they are detected again, prior mappings would be maintained and those that were lost could be assigned their "new" ID based on the lowest distance between the coordinate formed by merging / triangulating the sets of 2D tracks and the previous ID's position. Now, the fish would have to swap positions in between the loss of detection, which at 20FPS would likely be just 50 milliseconds, which is extremely unlikely to happen.

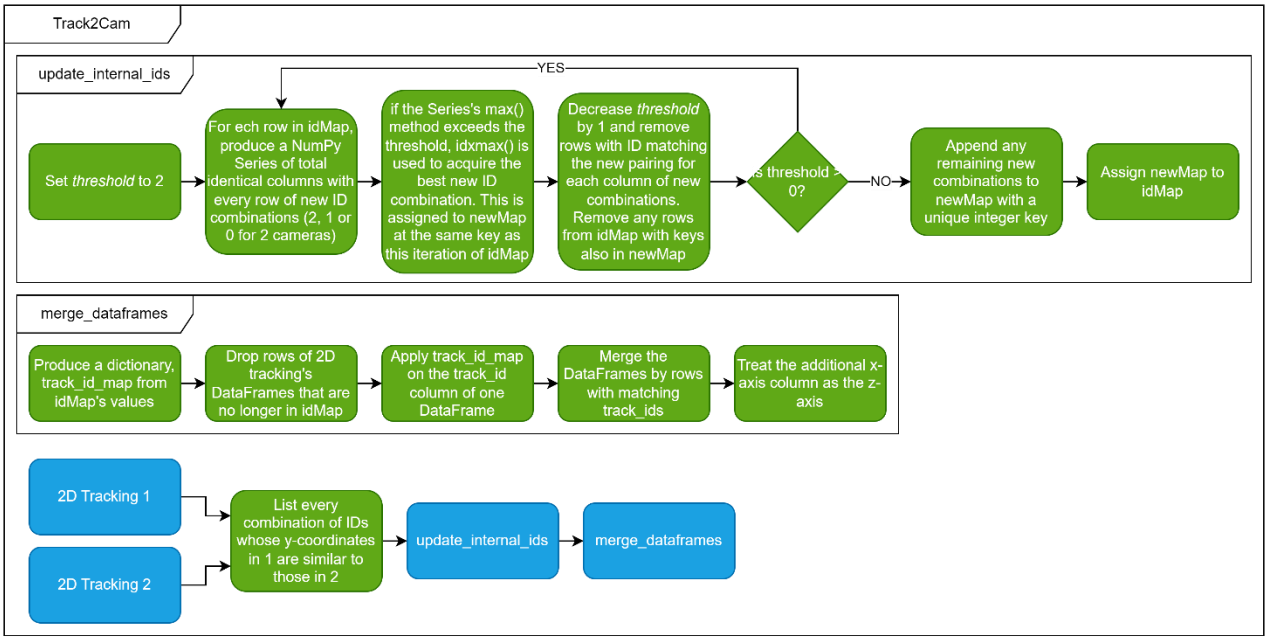


Figure 32: Outline of the three-dimensional tracking algorithm.

2.2.1.4 Triangulating 2D Tracks into 3D

A constant ratio between the frame and tank's bounds ensures that pixelwise coordinates can be placed back into the real world using their linear relationship, requiring the human calibration of the images being processed to assert that the tank's edges are placed at the frame's edges. This is easy to portray to the technician through cropping of the images at hand.

During calibration, the frames will be cropped using variables gathered from dynamically adjusted trackbars, taking the offsets provided by the user and scaling the reduced frame back to its original resolution. This will provide the technician with a sample of where the objects in the frame will now be localised by the tracking algorithm.

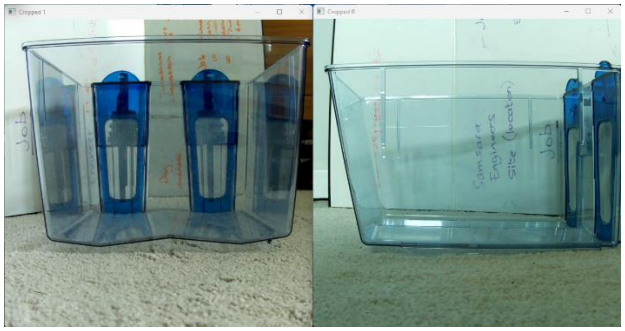


Figure 34 (two on the left): Each view before calibration

Figure 33 (two on the right): Each view after calibration

Behind what is viewable, the original image is what is run through image detection to preserve the aspect ratio; better matching the dataset the model is trained on, before the pixelwise coordinates of the bounding boxes are finally adjusted using the same offsets sampled by the calibration steps.

Calibrated sets of 2D coordinates are then run through the Track2Cam process to produce pixelwise x, y and z-coordinates for detected objects. Granted the assumption that these are the finalised coordinates of the object, a solid estimate of its 3D position is granted. Continually plotting the location of an example subject which traversed the boundary of the tank to a scatter graph created though means discussed in section 2.2.3.3 produced figures 35, 36 and 37:

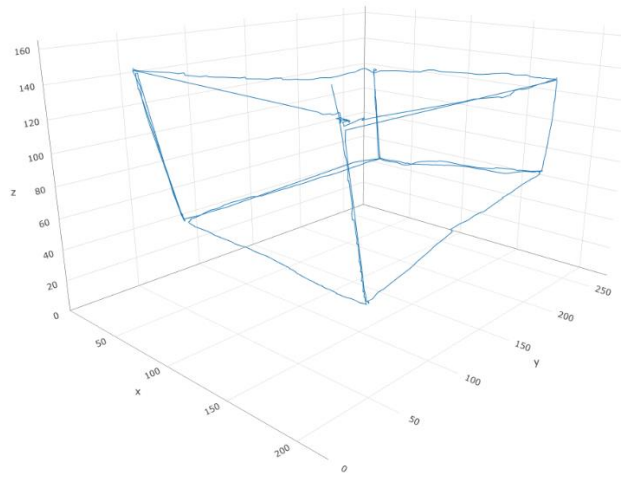


Figure 36 (left): Scatter graph of an example object's track after traversing each of the fish tank's edge without calibration

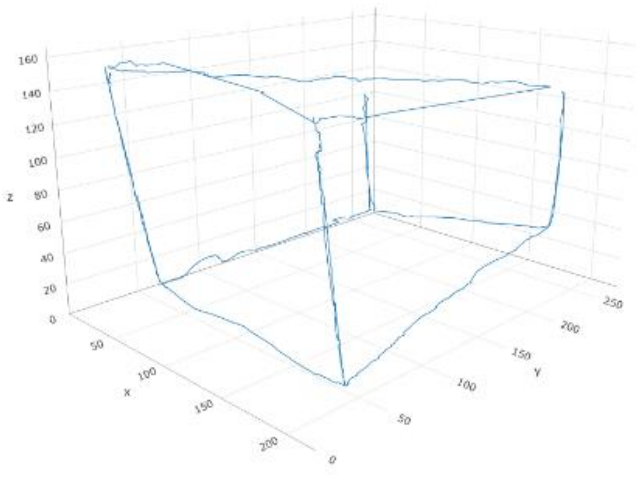


Figure 35 (right): Scatter graph of an example object's track after traversing each of the tank's edges with calibration

Figure 38(left): The fish tank (approximately 220x275x165mm^3)

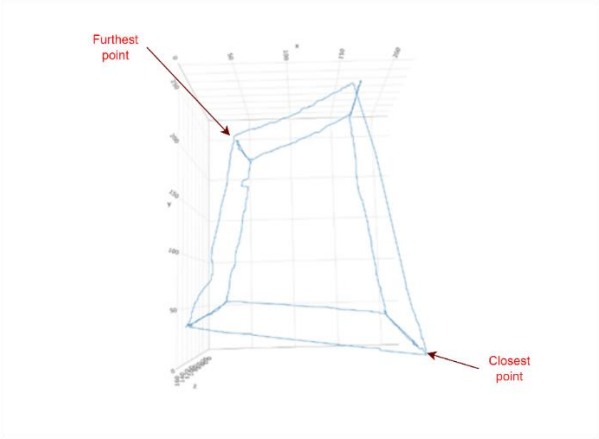


Figure 37 (right): Skew of points as they stray further from each camera

Figures 35-38 and depict the program's 3D depiction of the tank before and after calibration. Both axes are of identical sizes, meaning that in the calibrated instance the axes better depict the tank's boundaries. Nooks in the drawing are due to human error. Another error entails the cameras' field of views. The tank's panes furthest from the camera appear smaller in the frame, meaning that fewer pixels are traversed by the LED as it draws the bounds, forming a relationship between x and y as the axes are traversed, creating skew in the final representation.

Projection matrices describe this relationship by combining the translation and rotation of each camera with the intrinsic properties of their lenses. Calibration routines accessible through routines provide the different parameters that form this matrix, meaning that they were to be plugged into a Python script which would perform the required processes in turn.

First, the intrinsic and extrinsic parameters can be acquired from OpenCV's *calibrateCamera*() method. The matrix is the second value unpacked from the returned tuple, which is of the form…

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

…where f_x and f_y define the focal length and c_x and c_y define the optical centres of the camera. To call *calibrateCamera*(), a set of real-world information and the camera's corresponding perception of this needs acquiring, a process which OpenCV walk through using their *findChessboardCorners*(). Invoking this with a photo taken by the camera as an argument will use a detection method to locate the pixelwise corners of the

board's checkers and find the relation to the actual edge length of each, which must additionally be provided. The returned tuple then contains the parameters of this relationship.

For ease of use, the additional step of plotting the detected corners back onto the board on a successful detection was copied from the documentation at [18]. A set of these points was formed by analysing the real-time camera footage at the time of calibration. The *LatestImages* class developed as part of section 2.1.3 offers the collection of these images with a call of its functor. A while loop ensured that at least 49 pairs of points were acquired before passing them as an argument to *calibrateCamera*().

Although *calibrateCamera*() returns a full projection matrix, its rotational and translational vectors (extrinsic matrix) show no relation between the two cameras, meaning that only the intrinsic parameters are useful. The fundamental matrix of the cameras can be acquired using OpenCV's *findFundamentalMat*(), which uses a set of points that locate the same object in both frames: creating a set of points being a use case for the current tracker and its LED detection, as the LED is small, leaving low margin for error in the predicted bounding boxes, and only one object present per pair of detections to remove the risk of confusing the ID pairs.

OpenCV's *recoverPose*() extracts the extrinsic matrix based on the fundamental matrix and the set of points that formed it. The product of each camera's intrinsic matrices and the fundamental matrix should form the essential matrix beforehand to match the input argument required by the function call. Multiplying a manually created extrinsic matrix with no rotation or translation with the first intrinsic matrix then produced the first cameras projection matrix. The other was created by horizontally stacking the recovered rotational and translational vectors before multiplying this by the second intrinsic parameters.

Using *triangulatePoints*() to then combine points that were matched by the tracker produced figures 40 & 39:

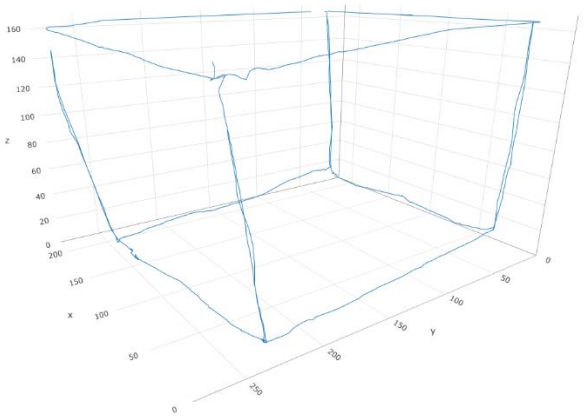


Figure 40: Triangulated projections of the tank's bounds

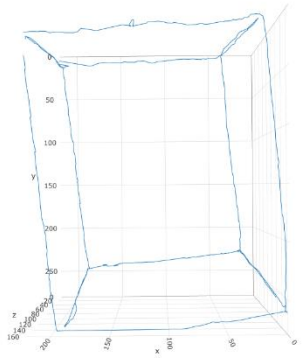


Figure 39: Birdseye view of the triangulated boundaries showcasing no skew.

Points that strayed from the cameras no longer had gradients towards the centre of focus, which indicated that the use of this triangulation before deciding which points are within the water depth threshold where they are considered potential matches, could reduce the size of said threshold. Using epipolar geometry, if x and x′ are corresponding image points, then $x'^{T} \times F \times x = 0$ [19]. Headroom should be left for imperfect calibration, but it will be significantly less than when there was no calibration.

2.2.2 Submerging Piezoelectric Sensors with Danionella Cerebrum

A vast amount of noise in the signals acquired from the piezoelectric sensors was the result of the bunched cabling due to the length required when reaching the maximum depth of the real fish tank. To combat this, conductive tape was wrapped around each cable and grounded to short-circuit any interference. Figure 41 is an example of the amended configuration.

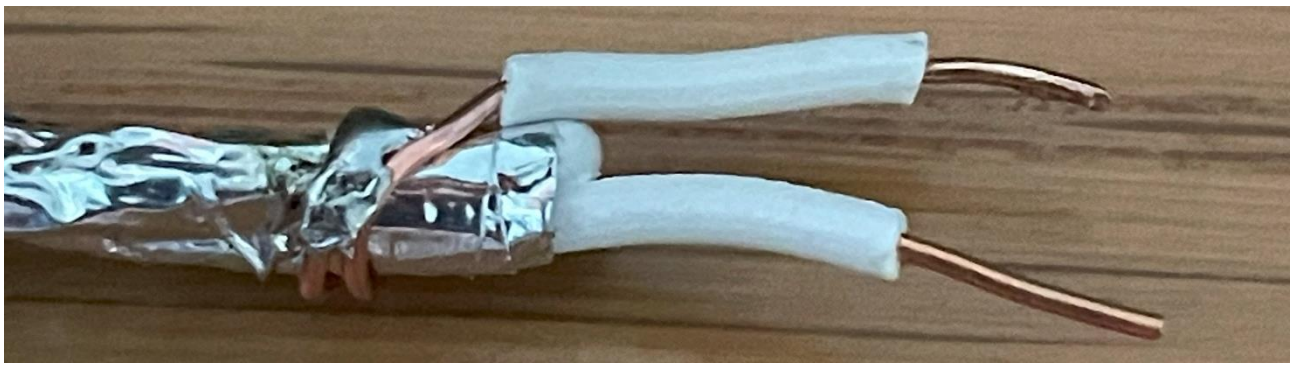


Figure 41: The end of a cable when grounding the electrical interference. One wire has been wrapped and soldered against the tape; this should be connected to the rest of the circuit's ground when used.

Signals acquired from an idle tank of water had significantly smaller amplitude when read from sensors with the tape on rather than off. Oscilloscope traces in figures 42 & 43 attest to this improvement. This meant that the reference signal proposed in section 2.1.4.3 was no longer needed, allowing one amplifier to be setup as a generic inverting amplifier and the other non-inverting, these minor adjustments produced the circuit in figure 44.

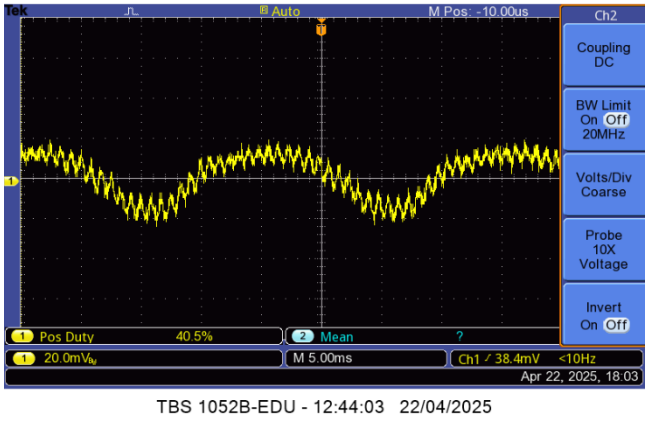


Figure 43 (left): Signal previously acquired from the sensor

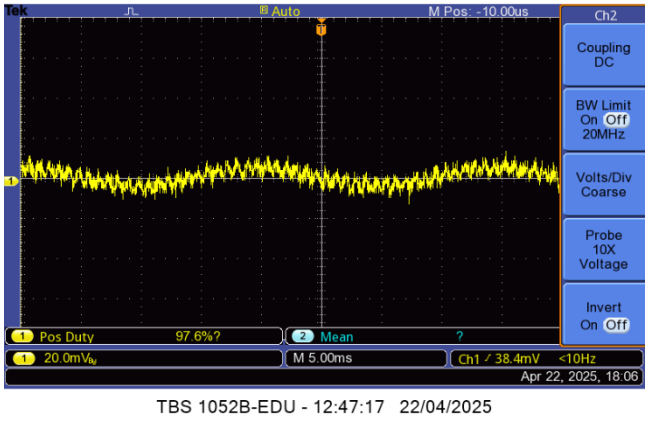


Figure 42 (right): Signal acquired from a sensor with the upgrade

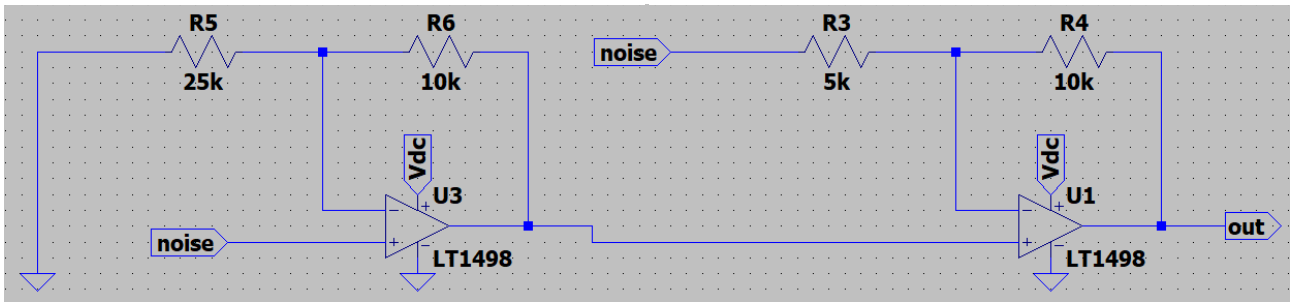


Figure 44: Active rectifier without the subtraction of a reference signal. The noise subnet is being used as an example input to this circuit.

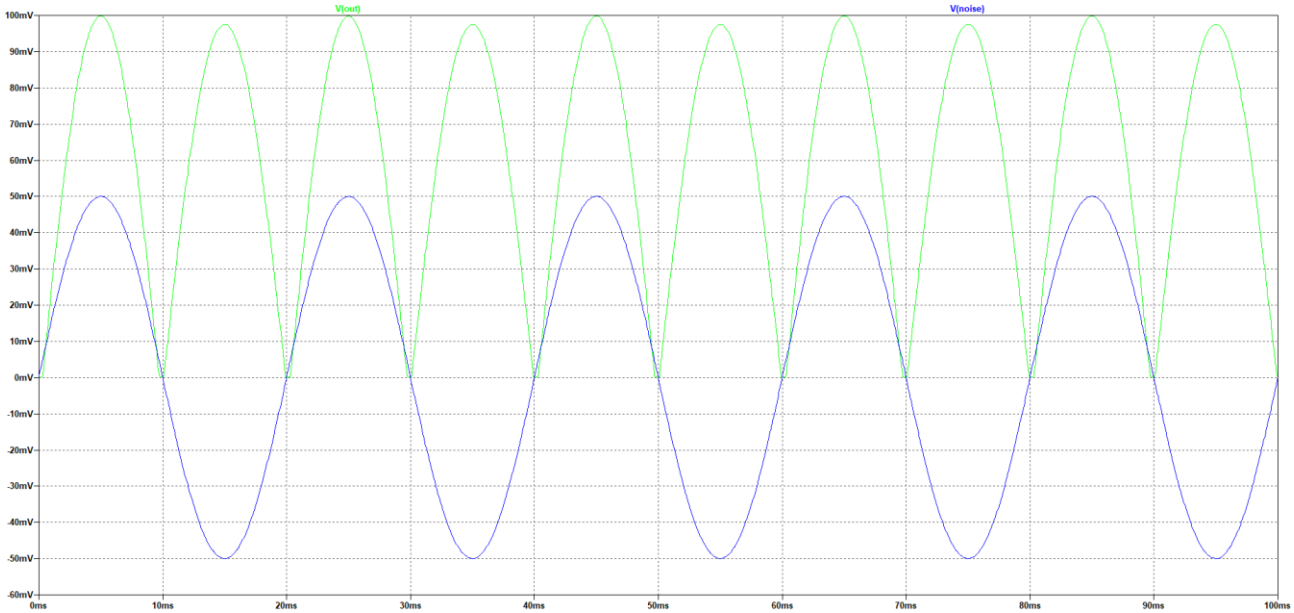


Figure 45: The rectifier's input (blue) and output (green). R3, R4, R5 and R6 can be adjusted to increase the gain in the final solution.

2.2.3 Frontend Web App

Results acquired by the study were to be presented by a frontend which was to be accessible from outside of the lab, allowing surveillance of DC without confronting the security protocol and discomfort faced when entering it in person. If the RPi was setup to serve video to the local network in the initial idea, the app's host could be positioned anywhere with access to the University of Nottingham's Eduroam connection, process the live feed and serve the results.

In the sandbox network setup with only the RPi and host PC, the network's range is limited to the Pi's Wi-Fi strength which can only penetrate one or two drywalls. This restricts the choice of host PC not only to one with a powerful GPU for real-time image detection, but also one with a Wi-Fi driver. Its position may also be inconvenient to access; defeating part of the solution's purpose.

If the application were networked, there is the possibility that the results acquired by the powerful PC could be forwarded onto HTTP. Now acting as a server, it could run the python analyses on the raw data on one of its Wi-Fi adapter's IPv4 ports before serving the results it over its ethernet adapter if placed in a room with access to this. A parallel task, or computer, could then host a web application such as React to place this data on a GUI. When made accessible on a real domain, it is plausible that the results would be viewable from anywhere. Here, the focus is on the development environment running on the localhost, advertising the capacity for future production builds and releases.

2.2.3.1 React & Vite

Components will be referred to within <> for succinct reading.

React is a library which modularises the user interface using components. Under the hood, components are JavaScript functions that make use of JSX: a syntax extension which allows HTML-like code to be developed within their files. The return values of these are React's adaptations of HTML elements, meaning that components can equally be implemented as their own UI elements in a similar manner to normal HTML tags.

Vite manages packages in a manner that automatically organises dependencies and source code to rapidly start the development server from cold. Creating a Vite project with the TypeScript template creates a React project with additional strong type checking, improving the scalability and rate of development due to enhanced IntelliSense and runtime errors instead of hiding failed variable assignments.

React router allows the app to be navigated using weblinks, which meets generic users' expectations when browsing the web and separates independent topics of data into pages. Children of the API's <*RouterProvider*> component will be subject to routing, meaning that page elements should be provided with

their desired path. This is achieved by creating objects which define these attributes and listing them in a *createBrowserRouter*() call, the return value of which plugs directly into the <*RouterProvider*>'s router attribute.

Pushed from the <*RouterProvider*> will be the root path, "/", which should implement the API's <*Outlet*> which always returns the element at the current path if it's defined. This means that elements in the root page will still be apparent alongside whatever appears at the outlet, allowing utilities like sidebars to seamlessly persist as links are traversed.

2.2.3.2 Surveillance Page & HLS

Observable behaviour in DC is a core takeaway in this project, leading to its accessory in the web app. Real-time video streaming from the lab will allow the specimens' keeper to check on them in the current moment, however, more importantly, a backlog of surveillance would allow the human review of key events after they have occurred. For example, if eggs were found in the tank in the morning, the footage available to the app could rewind to watch the lay and its preceding events.

GStreamer is already used at the transport layer between the RPi and web server. Already installed, it was also introduced as part of the forwarding of the surveillance to the web server. HLS (HTTP Live Streaming) is a communication protocol that can be multiplexed by a GStreamer element on the server and demultiplexed by the Hls JavaScript library on the front end. An OpenCV *VideoWriter* instance should be used to launch the GStreamer backend for writing the files while automating the required conversion from NumPy arrays; the format the frames are processed in.

Table 8: GStreamer pipeline for HLS file generation

Pipeline Element	Reason
appsrc	Provides OpenCV with a buffer to feed raw frames to
videoconvert	Converts the frames to a format compatible with the next pipeline elements
x264enc	Uses H.264 encoding to minimise the video file size
h264parse	Ensures that the stream is formatted correctly
mpegtsmux	Multiplexer that packs the stream into an MPEG Transport Stream, used for HLS
hlssink location={dir}/segment_%0{digits}d.ts playlist-location={dir}/playlist.m3u8 target-duration={file_duration} max-files={file_count} playlist-length={file_count}	Splits the MPEG-TS stream into smaller segments and an m3u8 file to manage these as a playlist. More discussion of parameters to come.

Injecting parameters into the pipeline in a function call makes their adjustment easier, as the sheer size of the files could exceed the remaining space on the server's hard drive if not tested first, a process which first entailed gathering a one-hour long playlist and observing its properties.

Both streams were recorded a directory under a common parent containing no other data, meaning that the directory size would represent the size of the playlists, providing insight to the sheer hard disk demand of keeping long backlog of video. *file_duration* was then set to 15 seconds, meaning that the viewer can be as close as 15 seconds behind real-time and *file_count* was set to 400, allowing a total of 6000 seconds of video - 100 minutes. A clock was setup in front of both cameras so that the timestamping of the video could be confirmed to work, while also exercising the predictive-coded frames that come between key frames.

After filling the playlist, the video was validated using the web app to rewind to the beginning and the directory size was confirmed to be 2.16GB, estimating 1.3GB per hour, which makes a 24-hour backlog able to fit on a standard 32GB USB drive.

Each pipeline was launched on a parallel thread due to the time consumption applied by H.264 encoding when performed on a CPU, meaning that their data was provided in a thread safe manner. Instances of the queue module's "Queue" class provide *put*() and *get*() methods for communicating data amongst threads, meaning that once the main thread performed image detection, the resulting plots were *put*() on the encoders' threads for processing.

At the receiving end, a component was developed to act as a normal HTML video element; with HLS demultiplexing capability, which is native to Apple's browser Safari but not all. An instance of the *Hls* class can be created with methods to load the source, attach the media to HTML video elements and setup callbacks for events such as segments' parsing.

React's *forwardRef*() exposes this component's DOM node to the parent which allows functionality such as *play*() and *pause*() on the returned video element to be controlled by the container. The component is declared as a constant instance of the value returned by *forwardRef*() when executed with a lambda function containing the component's functionality. This functionality is another lambda function executed by the *useEffect*() hook when the "src" dependency, a property of the component, changes. The video's source file prop outlined in figure 46:

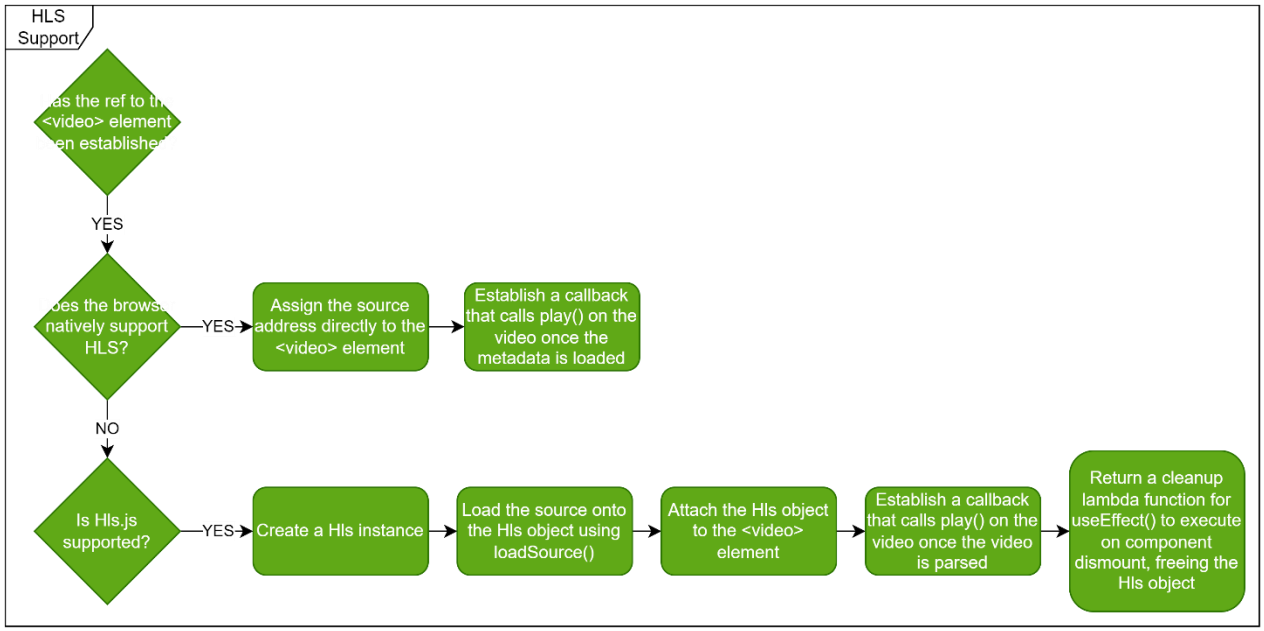


Figure 46: HLS enabler for JavaScript browsers

2.2.3.3 Plotting with Plotly

An in-person view of DC's tank grants a better perspective of their situation than separate orthogonal views. The fish are almost identical which means that viewing the fish tank mainly provides insight to their location, which is information that was already calculated in section 2.2.1.4, making the next task serving the data.

Plotly provides a JavaScript open-source graphing library which was introduced for the display of this data in a new component, <*Scatter3D*> which returns a <*div*> who's reference is setup using the *useEffect*() hook. When no dependencies are passed to *useEffect()*, it executes only as the component mounts, meaning that axis limits can be set in stone to represent each dimension of the tank in millimetres. Plotly's static functions accept an instance of the containing HTML element's reference which they edit according to the chosen function and its arguments.

Plotly's Layout interface contains attributes such as the title, axis ranges and the element's margins and is accepted as an optional argument to a *newPlot*() call which instantiates the chart within its container

element. The fish tanks' dimensions are 220x275x165mm^3 which will be represented on the chart once the 3D coordinates have been scaled in each dimension, which makes the "scene" member of the Layout object as follows:

Table 9: Setting the size of the 3D scatter graph's axes

	value	
key	range	autorange
xaxis	[0,220]	false
yaxis	[0, 275]	false
zaxis	[0, 165]	false

Data was to arrive as a property of the component, making the plot usable with any data arrival method. In this case, the containing component was setup to use provide <*Scatter3d*> with information that arrived over WebSocket. The component managed its own state, a mapping of IDs that had previously arrived to a Boolean flag. The flag would determine whether the coordinate should be appended to the IDs prior tracks as a line graph or replace its previous position as just a live location. Displaying historical data for all fish simultaneously could make tracks hard to follow as they intertwine, however, when looking by eye, the option to toggle these lines could make patterns more obvious. The containing component would be called <*LivePlot3d*>.

2.2.3.4 WebSocket & React Redux Toolkit

Data was then to be livestreamed to <*LivePlot3d*> using a WebSocket, allowing the server to prompt communication to the client. An instance of the WebSocket API was created passing the address of the server on the localhost before its instance's *onmessage*() property was assigned to a handler's pointer. This handler was created in a *useEffect*() hook with no dependencies in *App.tsx*: the app's entry point, meaning that the ongoing connection to the server would not be reestablished as lower-level components such as LivePlot3d are dismounted as the user navigates the page.

Passing the live feed from *App.tsx* to <*Scatter3d*> as props would need forwarding through the <*RootLayout*>, <*Outlet*>, <*View3d*> page and <*LivePlot3d*>, which is not succinct or efficient. To combat this, React Redux Toolkit provides centralised state management out of the box with its open-source API. A single store encapsulates what their documentation regards as "global" state within the province of a <*Provider*> component who's "store" prop points to it.

It was noticed that if the user was adjusting their view of the graph as an update was received, the adjustment would be reset. To combat this, a Boolean prop "pause" was introduced to enable the containing component to temporarily stop the plot from updating if the user desires to adjust their view. Checking this attribute before updating the plot was then performed in the *onmessage*() handler, which was importantly redrawn on state changes to depend on the newest value of "pause".

2.2.4 The Product's Frame

2.2.4.1 Freestanding Cameras

Correct positioning of the cameras affected the entire surveillance solution's accuracy. Circumstances that are more like that which image detection models are trained will realise better accuracy than when the perspective and lighting is changed. Ensuring that these variables are as controlled as possible could improve the solution's repeatability.

Results acquired thus far required the careful placement and recalibration of the cameras' positions and lenses. Each camera and lens were ordered alongside a small tripod which can place them as high as the walls of a fish tank. For demonstration purposes, a circumstance was created to place the tripods on level ground with a tank, which is not sustainable for longer periods of time as it removes the pump from the water.

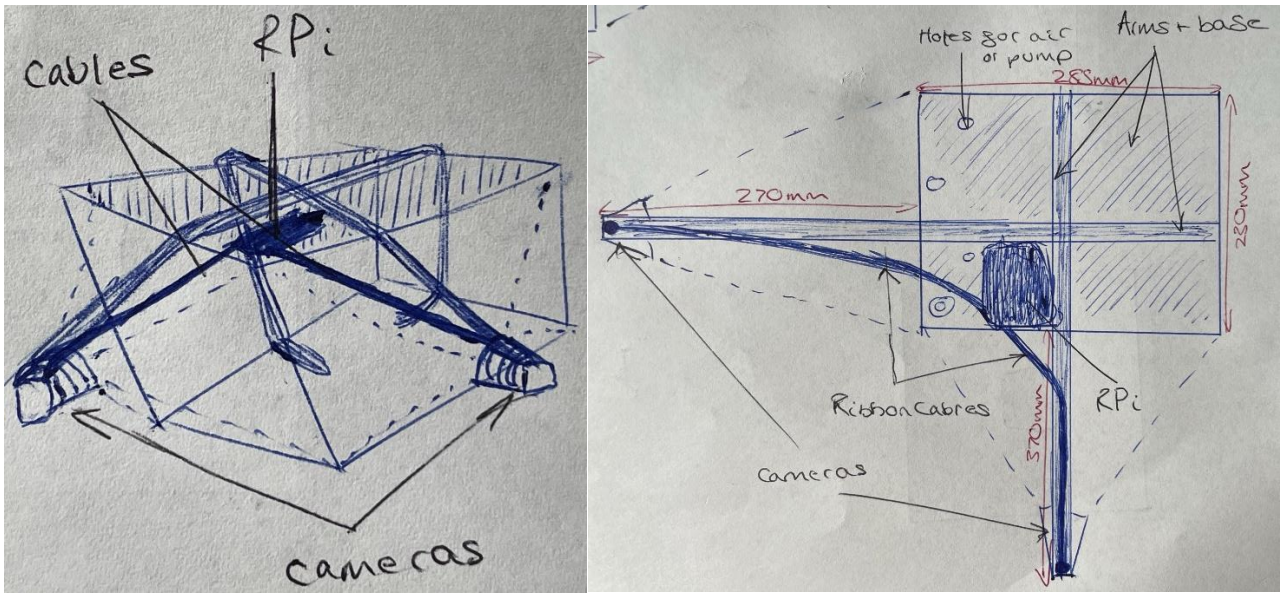
Figure 47: A mechanism housing the fish tanks while monitoring and pumping their water by Techniplast.

A more permanent solution is needed longer term, particularly as the cables connecting the cameras to the RPi are always taut due to their maximum length being reached, which primarily results from the ribbon wires being exposed at the opposite side of their plug when accommodating both cameras.

2.2.4.2 CAD Skeleton

A permanent solution to this problem could be 3D printed, and its premise is simple: a frame which sits on the roof of the tank, housing the RPi on top with an arm extending outwards for each camera. The cameras can be connected to the end of each arm upside down, omitting tripods and orienting their plugs so that the ribbon wire can run flat against the arm.

Arms extending from the base should position the cameras with a view of the breadth of the tank. This will ensure there are no blind spots in the final view but also place a larger moment on the arm as it grows in length, meaning that although the base that is to sit on the tank lid can be thin, the arms should be thicker to withstand this. The arms will extend beyond the centre point of the base where they will meet and follow the tank's perimeter until below its floor. This will ensure that when elevated on its mount as in figure 47, the weight of the cameras don't tip the frame.



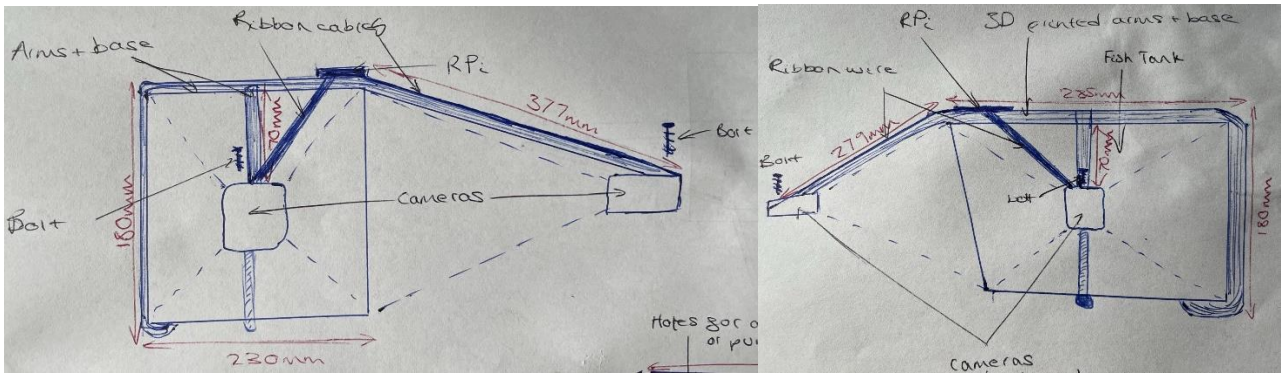


Figure 48: The frame upon a fish tank (top left), its plan view (top right), front view (bottom left) and side view (bottom right). Not to scale. The diagram quality was hindered by the lack of a tablet type device to draw immediately onto, also, pencil did not show well in photographs, deteriorating the quality of shading through use of pens.

2.2.4.3 Bespoke Lighting

Hollowing out the centre point of the frame where it is in contact with the tank's roof would leave room for a lighting solution that could evenly distribute light using a diffuser film. Currently, light isn't well dispersed and there is contrast between fish in light and dark areas, which can make general viewing of the footage more difficult and means the detection model must be generalised. If the lighting circumstances were guaranteed in frames across sessions, less light immunity would be required, and a highly specialised model could be more reliable.

2.2.5 Combining Every Module

With the video surveillance, 3D tracker and web application at play, a high-level overview of how these modules work together can be drawn up in a UML sequence diagram.

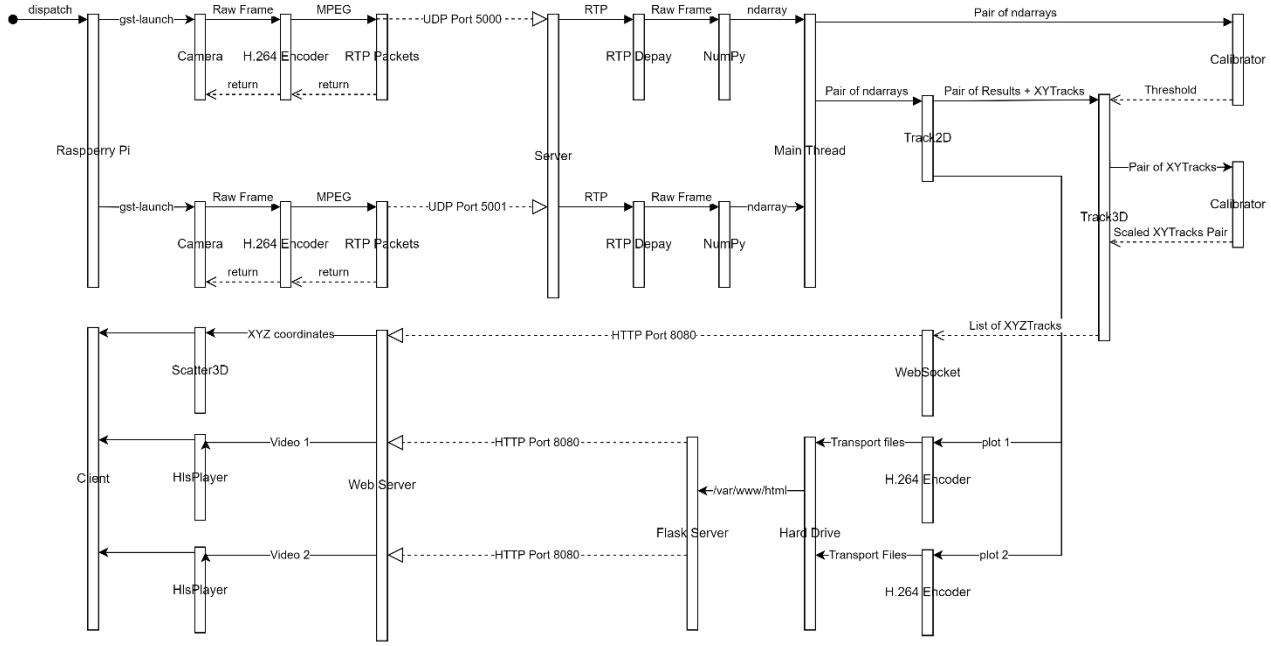


Figure 49: Highest level overview of the formulation and display of tracking results and surveillance footage using a UML sequence.

3 Final System Testing and Validation

3.1 Summaries

3.1.1 Specification Summary

Section 1.2 is an in-depth specification. To concisely layout these objectives, a summary is provided below:

1. Video surveillance system, making occurrences in the lab viewable after they occurred.
2. A conclusive decision of whether piezoelectric sensors would pick up DC's vocalisations.
3. Image detection on the recorded video, making the location of DC more apparent.
4. A user interface available through the web browser.
5. Triangulate TDOA measurements to locate the source.
6. Track the fishes' live location in 3D
7. Gather and quantify information that can be taken from the tracking and audio cues.
8. Setup the unsupervised training of a neural network for the prediction of DC's behaviour.

3.1.2 Modelling Summary

Section 2.1 provides insight to how each module of the product performed independently from the rest of the solution. A summary is provided here to make metrics used in comparison points easier to access.

1. Video could be made available from dual cameras simultaneously, at the host PC over a networked solution, each at 640x640px @ 20FPS.
2. It was predicted that the elimination of noise would see the waterproofed piezoelectric sensor models successfully detect DC's vocalisations.
3. An image detection model was trained on a similarly sized object, and a single class, like how all instances of DC will all come under the same class. Metrics associated with this model showed that it had a recall rate of 0.994 or completely missed the model just 0.6% of the time.
4. A 3D tracker was proven to be able to track objects by concatenating its current coordinate in two orthogonal 2D trackers. Objects were to be separated by at least 17% of the frames captured by the camera in the y-axis to be safe from confusion of their identities.

3.2 Testing Setup

Careful encapsulation of each module meant that in the program's main thread, their classes were instantiated and launched before the main loop called into their methods. Classes that were launched in the end-product included:

- *LatestResults* – combines two GStreamer pipelines (section 2.1.7.2) listening on separate ports for the H.264 stream produced by the RPi, safely making the data available to the main thread.
- *Track2D* (section 2.1.2) – implements a YOLO model of choice with a bespoke tracking algorithm that uses an entirely direct approach of tracking.
- *Track3D* (section 2.2.1) – accepts a pair of the most track IDs and their coordinates resulting from Track2D calls, returning the triangulated 3D coordinates.
- *HLSEncoder* (section 2.2.3.2) – two objects were created, launching separate threads for the HLS encoding of a playlist for the video from each camera view after plotting the YOLO results to the frame.
- *WebSocketServer* (section 2.2.3.3) – launches a new thread for the registration and deregistration of clients listening to WebSocket on the localhost on port 8765, exposing thread-safe queuing of data to be parsed as JSON and sent to all listeners.

- *Flask* (section 2.2.3.2)– a WSGI application which serves a directory over HTTP. Here, it serves the HLS playlist files to the frontend.
- *FPSCounter* (section 2.2.1.1) – Registers the time of the first frame's arrival, before forming a running total of new frames across both cameras. When the program cleanly exits (by pressing Esc with an OpenCV window in focus), it divides the execution duration by the running frame total and prints the average framerate across both cameras to the console.

Another instance of the code editor was then launched to run the frontend web app, before opening the hosted webpage's development environment on the localhost. Submodule tests previously performed were then repeated with this additional demand.

Prepared to launch the cameras and use its hardware H.264 encoder for both streams, the RPi was setup at the midpoint between the two cameras where the ribbon cables connecting them were at their least taut as shown in figure 50.

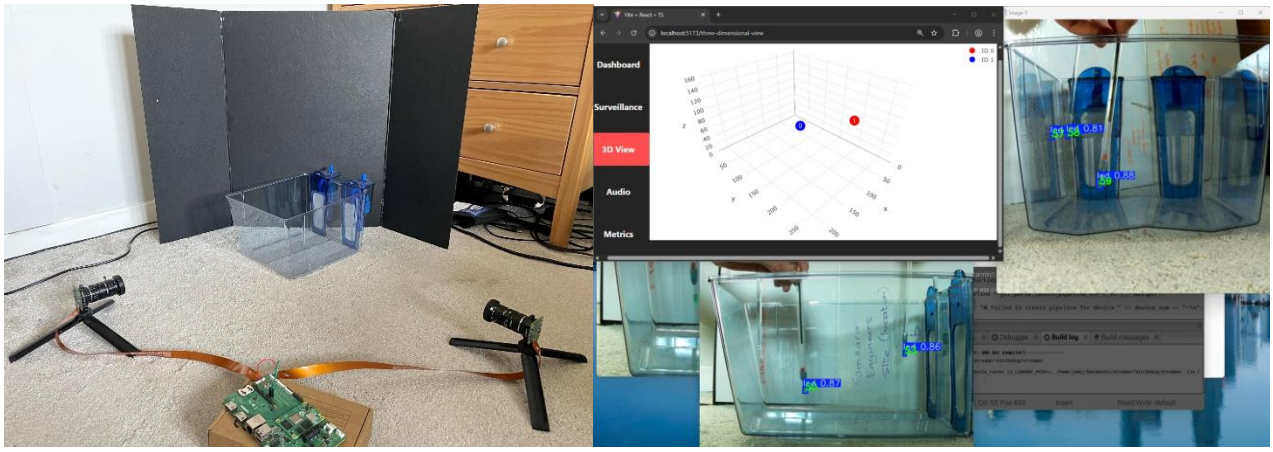


Figure 50 (left): Orthogonal views of the fish tank

Figure 51 (right): Orthogonal views of the fish tank Triangulation of orthogonal 2D tracks

3.3 Results

Background tasks on both computers were minimised and when the host PC's Python code was launched, OpenCV windows that launched as part of the host PC's debugging arrived exactly as they did during modelling, however, playback on the webpage appeared to be in fast-forward, which indicated that frames were arriving more slowly than the specified framerate. The video was left to accumulate for 100 minutes before being confirmed to be rewindable and cleanly exiting the program. The framerate read from the console was **18.78FPS**.

Track2D was running image detection using the model trained on an LED as in the modelling section, demonstrating the newly developed triangulation system which produced figure 51. Using the fundamental matrix to calculate how close points are to the same epipolar line across cameras would decrease the threshold where fish would have to be at different y-coordinates in the frame to be distinguished. The threshold was gradually decreased until the 3D track for a single object was no longer apparent. It was found that thresholds below 0.015 would occasionally exclude the object due to imperfections in the calibration routing, meaning as low as **1.5%** of the total pixels in the y-axis are the threshold where DC at this similar water depth for the tracker to possibly mistakenly swap identifications for objects of the same class. If the fish tank filled the entire camera field of view, this would suggest that the fish should be within **2.55 millimetres** of each other before a risk of mistaking the fish for one another is posed to the tracker.

A detection model trained on DC instead of the LED was able to **recall up to 95% of the fish** in its testing dataset. When the footage is observed by eye, this is a substantial achievement given how elusive DC can be in photographs. Section 2.2.1.3 estimated that at a minimum of 90% recall rate, the current tracker design would have a **50% chance of the ID that an object is first assigned still being associated with the same object after the 446th frame**.

Frames saved as part of the dataset of DC were analysed by the detection model trained on the fish in a bespoke algorithm designed to analyse the average pixelwise height that separates DC, granting insight to how often the fish enter the threshold of 1.5% of the frame. The labels correlogram produced by Ultralytics during model training grants insight to the average locations of location of objects and is provided in the appendix section 2 as a reference for the algorithm's input data. Figure 52 outlines the algorithm:

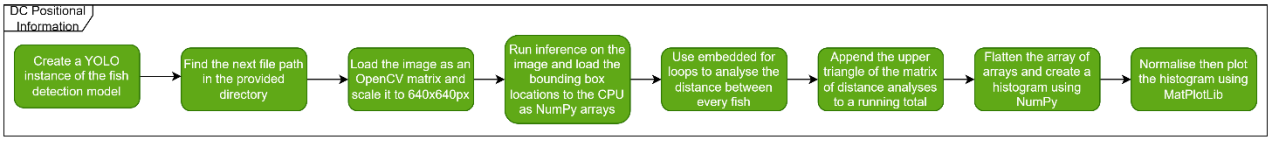


Figure 52: Algorithm for the analysis of DC's positioning from each other.

Results were displayed using a normalised histogram of DC's average distance apart from each other, which is available in the appendix and the distances are measured as a proportion of the frame.

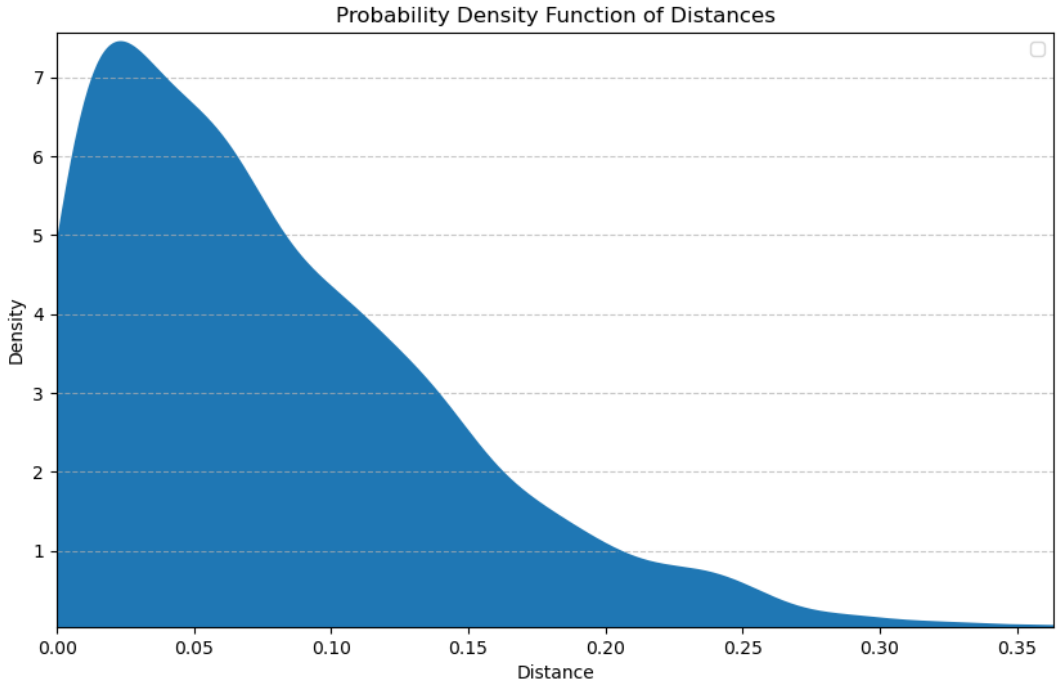


Figure 53: Gaussian distribution of the difference in water depth between DC at any one time.

A Gaussian distribution was formed from the data and is shown in figure 53. The area beneath the graph between distances of 0 and 0.015% estimates the likelihood of the fish being in this range. Using the trapezium rule at these points with densities of roughly 4.8 and 5.28 respectively estimates this as being 7.56%.

Assuming that this is another condition which must be met for tracking ID's to be confused, there is now a $0.16\% \times 7.56\% = 0.0112\%$ chance of a single ID swap per frame, giving $log_{0.99988}(0.5) > 6188$ frames until there is a half chance of any ID representing a different specimen. At 20FPS this would take **5 minutes and 15 seconds**.

Table 10 provides a summary of the quantifiable results obtained by this experiment:

Table 10: Summary of Quantifiable results

Metric	Value	Means of Estimation
24h Surveillance Backlog	31.2GB	Predicted
Framerate	18.78FPS	Measured

2D Track reliability	92.5%	Measured
3D Track ID Lifespan	$5\ \frac{1}{4}$ minutes @ 50% confidence or 47 seconds @ 90% confidence	Predicted

3.4 Specification Validation

3.4.1 Back-Trackable Video Surveillance

This section regards point 1 of the Specification Summary (Section 3.1)

Because vision of DC was promised though cameras, its application was pursued foremost of all other objectives. Iterations saw photos of resolutions between 320x240px and 4056x3040px arrive at the hosting laptop, however, the final product settled for 640x640px. OpenCV was used to present the view in the development environment and very little detail is visible, but their presence can be determined by eye. A 2048x2048px was scaled to 640x640px to demonstrate the loss of quality by zooming into the same subject at each resolution and comparing the results, there is a large quality difference, but it is not detrimental to the project's proceedings as the detection model maintained similar accuracy. The comparison is visible in section 1 of the appendix.

Unpredictable framerate in what is written by the H.264 encoders in the Python program meant that less than the expected 20 frames were being appended to a single second of the recorded video. If one second of recorded video takes $\frac{20}{18.78} \approx 1.06$ seconds to produce, 100 seconds of playback would entail 106 seconds worth of events, inducing a 6 second advance on the real-time events and an unnatural feel.

When introducing delays using the trackbar on the VideoSyncPlayer element means that the user cannot predict the real-world time which the event occurred because the supposed delay will be relative to the video file which supposes 20FPS (from section 3.2 point 1).

GStreamer provides a ! videorate ! element, which buffers frames so that they can be appended to the video while the next frame is unavailable. This would mean that even if just 18 frames are available in a one second timeframe, two of these will be the last frame in the sequence, before the next 20 begins. This would match the video's timestamps with the raw video feed.

Although the issue hasn't been ruled out here, it would be easily addressed as above, and only hinders user experience, as the importance of the surveillance is primarily to capture, at all, the events that one might want to rewatch. The fundamental objective has been **met**.

3.4.2 Audio Surveillance

This section regards points 2 & 5 of the Specification Summary (Section 3.1)

Fundamental to the project's success, entailing the extraction of real-world information into software through electronic sensors **must** have been achieved. At hand were cameras: guaranteed to acquire video footage, and piezoelectric devices: designed to detect/create vibrations; their premise specialising in hydrophones for underwater sound acquisition, but not necessarily their lone material. Because the outcome of this objective was not promised, it was heavily sidetracked, which left very little time for its pursuit.

It was required that an experiment was to be launched parallel with the application of the cameras to prove whether raw piezoelectric sensors are sufficient to digitise the vocalisations produced by DC. An independent and unique submodule of the project was developed to waterproof and submerge piezoelectric sensors in water, in a manner which was deemed safe by the University of Nottingham's EEE workshop supervision so that measurement equipment could be accessed.

Encased sensors had longer cables so that if eventually full immersed in the real fish tank, no extensions would be required, maintaining the waterproofing, however, because the signals that were being read were low amplitude, the signal was dominated by electrical noise picked up by the long cabling and as much as this was combatted, the in-water prototype failed.

Migrating the solution to the outside of the tank proved to effectively detect vibrations that occurred within the tank: flicking the water's surface by hand would register a signal of a few millivolts, without the excessive noise picked up by the extended cabling for the in-water solution. This would require the acoustic coupling of a much larger, piezoelectric buzzer to the fish tank's side using epoxy resin, which would permanently bond them, making the fish tank part of the final product as opposed to the subject of the solution. Sound also travels faster through solid objects than water, meaning that DC's signals would arrive at the sensor at unpredictable times, depending on if the sound met the wall of the tank before scattering around the tank's perimeter or arrived at the sensor's immediate location through water, hindering the triangulation of TDOA measurements.

Because DC were never in the premise of a container with a piezoelectric buzzer bonded to its side, it was never proven whether piezoelectric material could detect DC's communication, therefore, this criterion was **not met**. The uncertainty of the outcome of this experiment placed its precedence behind information that could be guaranteed to come from video footage, which left too little time for its success when returning to it.

Objective 5 of the specification was contingent on objective 2, meaning that it was **not met** either, however, the premise of this objective was made clear through section 2.1.4. The abstract class, PublisherThread allows information that the RPi may eventually acquire to be easily made available on the recipient PC, which can run an audio polling class, easily creatable by inheriting the abstract Poller class which is also implemented in the 0MQ video capture solution (section 2.1.7.1). Where TDOA is not available, analysis methods are available through NumPy, creating results which can then be made available to the frontend through the WebSocketServer class (section 2.2.3.3).

3.4.3 Tracking

This section regards points 3 & 6 of the Specification Summary (Section 3.1)

Point 3 of section 3.1 solely required the plotting of bounding boxes around detected fish, enabling the viewer to immediately see where DC are in the video, which can be difficult for humans to spot, particularly through the loss of detail when viewing a photograph.

A reliable model is less likely to incorrectly determine that an instance of DC is somewhere in the frame that it isn't and is more likely to correctly identify the fish that are present. The rate of recall has a very strong influence on tracking reliability in two dimensions, which also influences the foundation that the 3D tracker is built on. The success of this criterion depends solely upon whether its premise enables reliable 3D tracking; while a stable model would provide a better user experience on the frontend, it is satisfactory if it occasionally helps the end user: with a mean average precision of at least 0.939, the model will help users a vast majority of the time.

Once 2D tracking was available, the algorithm for combining two set of 2D coordinates to form a single 3D position could be developed at any time because it was software based, while access to the lab was required to continue to test progress the audio acquisition solution. As a result, the project somewhat specialised in the 3D tracking of objects, particularly an LED, an object close to DC's size with existing detection datasets attainable from the internet.

Tracking in 3D was never demonstrated on DC, meaning that the nature of their movement, proximity to each other and quantity of specimen couldn't be accurately modelled using the LED. On the other hand, the YOLO model's path and the threshold where objects are deemed as being on the same epipolar line are the only variables that would change in the real application, which makes the current circumstance very close to a finalised product. Section 3.3 predicts that as it stands, the tracker has just a half chance of a tracked object having the same ID after 5 minutes and 15 seconds for a tank of four specimen. This means that for casual viewers of the fish through the digital observation system, the tracks might persist throughout their sitting. On the other hand, to build a profile of each specimen's behaviour, IDs assigned to the fish must persist as long as the study does, which could easily be months, otherwise the behaviour of one fish might be associated with its companion's identifier after a period, invalidating results.

It isn't only human observers who rely on consistent identifiers for each fish over prolonged periods, because the neural network which was to be responsible for finding correlations between DC's behaviours equally

requires this information for its results to converge. There is a chance that the behaviour is not unique to individuals within a fish tank, however, developing the neural network's infrastructure around this assumption would have a significant time cost, particularly if it were eventually adapted to use amended tracking results.

Objective 6 was **not met** due to very brief persistence of its tracks, however, its progress between modelling and implementation saw a reliability growth of $\frac{1}{7.56\%} = 1322\%$ and a solution presented in section 4.1 stands to provide similar or better improvement, increasing the confidence in the foundation that was developed over the course of this project, while equally meaning that the reliability of the detection model is not a fundamental bottleneck of the solution. Objective 3 is therefore considered **met**.

3.4.4 User Interface

This section regards point 4 of the Specification Summary (Section 3.1)

Functionality that **should** be achieved as an extension of the project's core requirements transfer the raw data that was acquired into a tangible observation system. A frontend in the form of a web application was to be available to stakeholders and provide a real-time view of a fish tank's situation, with the help of image detection to enhance quality of life and make steps towards features that could be attainable in future.

A fundamental premise for the webpage was established in the development environment and is available to pull from a public GitHub repository detailed in the appendix at section 3 at any time, establishing the fundamental premise:

- Router setup allowing weblinks between pages, with a root layout that maintains a sidebar and easily facilitates a main navigation bar and other desirable features at the addition of basic HTML elements.
- WebSocket to maintain a continuous open connection to the server, allowing real-time updates to arrive where they can be comprised as JSON.
- Redux for central state management, encapsulating data arriving from WebSocket. A Redux Slice was developed, currently only accepting the 3D coordinate information of DC, demonstrating how it can scale in future development.
- A component for video livestreaming using HLS that can plug into React HTML just like a native <video> element with the enhanced compatibility.
- A page specialised in DC's surveillance, providing synchronised control of both views of DC, including backtracking prior video stored to the server within a safe duration, fast-forward up to 15x speed and slow-motion as low as 0.125x speed.
- 3D view of coordinates on a scatter graph component, interfaced to its parent component with a variety of properties, particularly "data", which a map of IDs to their coordinate, which are drawn onto the points on the plot, which colour codes them. A Boolean member also exists per ID specifying whether said coordinate should exist as a sole point or append to a trace including historical points.

A frontend to view the results was successfully demonstrated in the development environment and will be able to display surveillance footage and tracking data once the RPi's server program is launched within the lab. The objective was **met**.

3.4.5 Neural Network Behavioural Predictions

This section regards point 7 & 8 of the Specification Summary (Section 3.1)

No data was available to train the predictive neural network that was to find patterns in the fish's behaviour, which leaves nothing to show for this objective. Libraries expose functions for neural networks and their training processes, meaning that the outline in section 4.1 and foundation provided by all that has been undertaken thus far would see rapid development of this objective given more time.

These objectives turned out to be contingent on the tracker's success, which was not achieved in the timeframe. Objectives 7 and 8 were **not met**.

4 Conclusion

4.1 Future Developments

Less than half of the project's specification was met which appears a failure, until the next steps that are now available to the project because of what was achieved are considered. One more level of immunity to losing DC's tracks needs granting to the tracker first. This province could then be extended using the high-level plan below:

1. A point made in section 2.2.1.3 stated that when two fish are simultaneously lost by the tracker, there is no telling between them once they are re-registered. Introducing a direct approach of 3D tracking (currently the direct-approach tracking is performed on two separate 2D detection models). When mappings are truly lost, their last known 3D coordinates could be stored, meaning that on the next iteration, if they are detected again, prior mappings would be maintained and those that were lost could be assigned their "new" ID based on the lowest distance between the coordinate formed by merging / triangulating the sets of 2D tracks and the previous ID's position. Now, the fish would have to swap positions in between the loss of detection, which at 20FPS would likely be just 50 milliseconds, which is extremely unlikely to happen.
2. CAD design and printing of the product's frame would then see the RPi and its cameras positioned upon and around a fish tank in a manner that can be made permanent, entirely removing the need for further lab access until audio acquisition hardware is produced. Access to a room within the Wi-Fi range of the RPi should then be arranged alongside the placement of a powerful PC that can run the analysis code, which is predicted to track reliably with the previously discussed addition.
3. The playlist length of the HLS stream should be extended to at least span the interval between visits to the host's room, and a maximum length that occupies all the remaining space on the PC's hard drive. If eggs are ever found to be present in the monitored tank, the footage should be reviewed in case the eggs are visible to the cameras at any point. If/when they are, screenshots of the frames should be saved into a directory.
4. A new thread should be introduced on the host's backend software, dedicated to the analysis of tracking data. Information such as the average distance that separates each fish, from each of its companions, placing timestamps against instances where the fish come into contact within a set threshold, their speed of approach and speed of separation are all examples of quantifiable analyses that can be performed using dictionaries and embedded iterative loops in Python code.
5. Further investigation of audio acquisition should then ensue, attempting to digitise the vocalisations made by DC. Where a digital signal can be produced, if the sensors are submerged with DC, it would be reasonable to gather TDOA information using another three identical circuits, otherwise the raw signal can be transmitted to the host PC, particularly if this ends up coming from a dedicated, purchased hydrophone where all raw piezoelectric sensors fail.
6. Where applicable, triangulation of TDOA measurements should be performed by the host, on another separate thread which can then perform any additional processing like the Fast Fourier Transform.
7. All analyses should be made available to the web application. Handlers for simple HTTP Get requests will be acceptable in this instance, as the measurements will be made over prolonged periods. Where data must be livestreamed, the "socket" instance already exposes a method for sending JSON – a refactor should be performed to parcel the data in the JSON packages with a flag to indicate its destination. This could use enumerate values common at both the client and server-side, interfaced to the Typescript as in figure 54. This topic attribute will allow strong typing of incoming data.
8. Metrics that can be served to the webpage using HTTP Get are likely to be high-level metrics that can be tokenised as part of the output of a neural network responsible for pattern finding in the fishes' behaviour. The size and shape of the data in this format must be carefully planned such that the

metrics can be concatenated as part of a single matrix, which will be flattened before backpropagation and recovered when viewing the result.

9. A model should be created with the dimensions that fit the input and output matrices. A loss function and optimiser from Torch's neural network library should be created and take the neural network's parameters as a constructor argument. Numerous layers could be created at this stage to handle discrete and continuous metrics separately if necessary.
10. The model could be tested by looking for any correlation in DCs behaviour, for example, events that lead contact between fish. In this example, every available metric could be gathered over a set duration such as 10 seconds, then, a Boolean flag could be set to true / false based on whether fish made contact in the 10 second snippet. A prediction can then be made by passing the flag into the model before this prediction is passed as an argument, alongside the other metrics (excluding contact data) gathered over the snippet, to the loss function and backpropagated using the optimiser. The accumulated metrics can then be reset (never the trackers, these must maintain specimen IDs across iterations), and the process continually repeated as many times as seen fit.
11. The model's weights should be stored to the hard drive and updated on every iteration so that they can be saved across sessions and loaded elsewhere in the program, particularly where the server responds to "get" requests regarding the model's progress. Here, the weights should be loaded on each request so that they are up to date, before the chosen Boolean flag that indicates the chosen example behaviour "did occur", then "did not occur" can be passed at the model's immediate input, remembering the output for each. The difference in metrics could then be quantified by performing elementwise subtraction and getting absolute values, before decoding the data (perform the exact inverse of the process that tokenised, concatenated and flattened each analysis snippet into one matrix) and parsing this information as JSON to be sent to the webpage. These predictions may converge and provide insight into what causes the chosen event to occur.
12. If images of the eggs can be acquired of the course of described proceedings, they could be coupled with the images that produced the dataset which trained the image detection model that currently only detects DC. Annotations that distinguish the eggs from DC could then be drawn up to formulate a dataset for the detection of both classes. A new model can be trained in place of the detection model that was used until this point. The example input metric described between steps 8 and 9 can then be replaced with a Boolean flag that becomes true the instant eggs are detected in the tank.
13. The duration of the snippets of metrics can be refined, alongside the predictive model's hyperparameters, to try and predict what causes DC to procreate. Additional neural networks could also be introduced to the system for a variety of input features of interest, following the same structure as the current model.

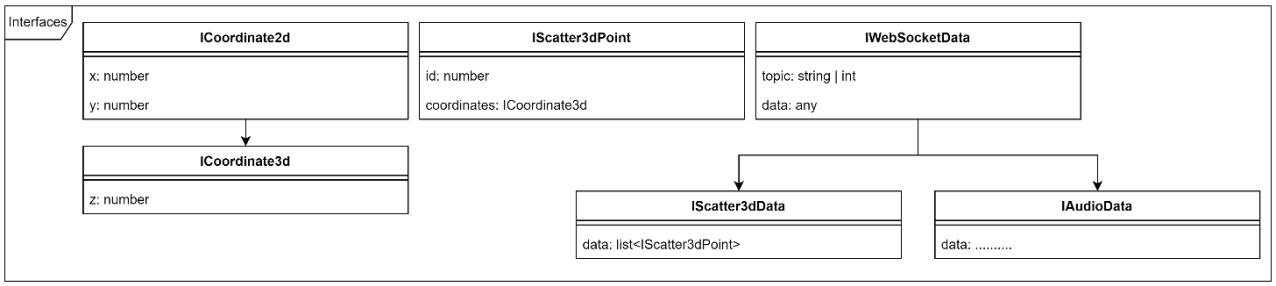


Figure 54: Structure for future WebSocket handling refactor

Predictions made by the model could take a long time to converge due to long intervals between egg laying, meaning that the tracks for each fish could have to persist 100% reliably, which may require an additional orthogonal camera. Each internal ID map in the current tracking solution would then need an additional external ID from an additional 2D tracker, which will be easily applied by the nature of the scalable solution.

Driving this would require an additional SBC, which doesn't have to be an RPi or capture such high-definition cameras; 640x640px is acceptable. It is important that the latest operating system is used, and the device's

specification allows for this overhead. The CM4 used in this project used deprecated features because it had the bare minimum amount of RAM that would run both cameras, and too much of this was consumed by the latest OS.

Time cost of the outlined potential future developments has been predicted and presented using a Gantt chart. The example used in its demonstration assumes that the proceedings would begin at a similar date to when this project commenced except this year instead of last year; what matters is the relative duration of the tasks. In table 11, each row represents a previously discussed numbered list item and each column represents a week worth of time.

Table 11: Gantt chart presenting the estimated duration of the tasks outlined above.

	Week																												
Task	14/10/2025	21/10/2025	28/10/2025	04/11/2025	11/11/2025	18/11/2025	25/11/2025	02/12/2025	09/12/2025	16/12/2025	23/12/2025	30/12/2025	06/01/2026	13/01/2026	20/01/2026	27/01/2026	03/02/2026	10/02/2026	17/02/2026	24/02/2026	03/03/2026	10/03/2026	17/03/2026	24/03/2026	31/03/2026	07/04/2026	14/04/2026	21/04/2026	28/04/2026
1																													
2																													
3																													
4																													
5																													
6																													
7																													
8																													
9																													
10																													
11																													
12																													
13																													

4.2 Significance in Wider Context

If the deduced steps in 4.1 were completed successfully, behavioural patterns discovered by the system could be exploited in DC through the replication of the predicted stimuli. All work undertaken by the University of Nottingham is considered by their Animal Welfare and Ethical Review Body, which expects that such work meets or exceeds the legal requirements and associated guidance issued by the Home Office [20]. In the hands of Dr. Rob Wilkinson, it can be assured that the product would be used ethically. Ethically enhancing reproduction in DC at the University of Nottingham would encourage their upkeep, which is important to their study and wellbeing.

Principles of this product behind the behavioural study of DC also stand in the study of a vast variety of alternative organisms. Specialisations made for DC included a bespoke image detection dataset, a single SBC for driving both cameras, DHCP setup on the SBC due to network limitations and the waterproofing of audio acquisition.

Adapting the solution to monitor birds, for example, could require an additional SBC to allow greater separation of the orthogonal cameras, and a generic microphone for sound acquisition. Aside from this, the project fundamentally stays the same, proving to be adaptable, expanding the potential market for the finished product. If the product were published, it might require disclaimers to ensure its ethical intentions, acknowledge its potential misuse and dissociate the product's service from the end-user's intentions.

4.3 Reflection on Management

While the underlying skeleton of the project is now in place, when marked against the criteria that the project set out to achieve, very little appears to have been accomplished. This does not bode well with stakeholders which means that project management was ineffective. Looking back at the progression, a level of perfectionism was at play. Evidence of the timestamps provided here is available in section 4 of the appendix. Some email conversations play the role of project review proformas albeit having less structure:

- TCP was initially implemented and proved to capture video feed by 05/12/2024. At this stage, it was time to continue to the next step. Instead, a wide variety of different video streaming architectures were consulted, under the impression that 4056x3040px @ 10FPS was achievable over the network. This saw a progress review more than a month later (27/01/2025) proceed to drag this solution out.
- When the 3D tracker was in development, beginning around 03/02/2025, the library Matplotlib was used for plotting the 3D coordinate of tracked objects, however, it would only run on the main thread and ran extremely slowly. At this stage, plotting coordinates at all was an achievement and attention should have returned to the audio acquisition system, however, with the development of the web app awaiting, creating this for the plotting of the points in place of Matplotlib appeared to solve two objectives at once.
- The web app successfully begun to portray 3D tracking results satisfactorily and truly parallel to the Python on the backend. A user interface available through the browser would have met its objective, but bug fixes and user experience were pursued instead of turning attention back to the poor tracking and audio acquisition.
- When the submerged piezoelectric sensors appeared to fail during the week beginning 03/03/2025, ruling the opportunity to triangulate TDOA measurements out of the project brought reluctance in the pursuit of sensors on the tank walls instead. A professional decision was made to continue with the tracker instead, however, this left too little time to finalise a system for audio acquisition.

Current objectives always absorbed full focus, which led to the disregard of the time plan and a combination of incomplete and over-performing specification points instead of meeting every requirement's mark as it was proposed.

Objectives that were achieved also took up the entirety of the project's duration, which might indicate that when the project specification was drawn up, it was overly optimistic. Producing an analysis solution for either video **or** audio feed may have been a more realistic prospect.

If the project were undertaken again, more realistic expectations would have been laid out from the start, through deeper analysis of what each point might entail through setbacks and surprises. Time would be better shared amongst tasks that had made less progression by "checking off" specification points as and when they are complete; ensuring that on completion, they are set aside until every objective is met as a minimum.

Alternative perspectives of the progression should have been regarded more highly too. Supervisor meetings strongly advised that drawing of attention to the audio acquisition section of the project, but personal desire to see results from the tasks that were at hand at the time was compelling in the moment. A benefit of this is that although progress may have been misplaced from the specification's point of view, the hours of work were there which provides a significantly stronger foundation in the tracking, webpage and their scalability than what may have otherwise been available.

5 References:

[1] Personal conversation with Dr. Rob Wilkinson

[2] Verity *et al.*, "Ultrafast sound production mechanism in one of the smallest vertebrates," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 121, no. 10, Feb. 2024, doi: https://doi.org/10.1073/pnas.2314017121.

[3] "Autograd: Automatic Differentiation — PyTorch Tutorials 1.5.0 documentation," *pytorch.org*. https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

[4] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, Dec. 2023, doi: https://doi.org/10.3390/make5040083.

[5]"YOLOv8: A New State-of-the-Art Computer Vision Model," *roboflow.com*. https://yolov8.com/

[6] E. Muñoz, J. Buenaposada, L. Baumela "Efficient Tracking of 3D Objects Using Multiple Orthogonal Cameras," [Online]. Available: https://www.researchgate.net/publication/221259825_Efficient_Tracking_of_3D_Objects_Using_Multiple_Orthogonal_Cameras

[7] J. F. Tressler, S. Alkoy, and R. E. Newnham, "Piezoelectric Sensors and Sensor Materials," *Journal of Electroceramics*, vol. 2, no. 4, pp. 257–272, 1998, doi: https://doi.org/10.1023/a:1009926623551.

[8] B. Xu, G. Sun, R. Yu, and Z. Yang, "High-Accuracy TDOA-Based Localization without Time Synchronization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1567–1576, Aug. 2013, doi: https://doi.org/10.1109/tpds.2012.248.

[9] "Soundspeed data for pipe materials and liquids, chemicals and water," *Rshydro.co.uk*, 2019. https://www.rshydro.co.uk/sound-speeds/.

[10] A. Shundo, S. Yamamoto, and K. Tanaka, "Network Formation and Physical Properties of Epoxy Resins for Future Practical Applications," *JACS Au*, vol. 2, no. 7, pp. 1522–1542, Jun. 2022, doi: https://doi.org/10.1021/jacsau.2c00120.

[11] D. Fallows "Library file to simplify OpenCV functions Session 6/7 – Computer Vision," The University of Nottingham Moodle. Available to the Department of Electrical and Electronic Engineering.

[12] Raspberry Pi Ltd, "RPi Compute Module 4 – A RPi for Deeply Embedded Connections," 2023. Available: http://datasheets.raspberrypi.com/cm4/cm4-datasheet.pdf.

[13] Raspberry Pi Ltd, "Raspberry Pi 5," 2025. Available: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

[14] J.-W. Chen, C.-Y. Kao, and Y.-L. Lin, "Introduction to H.264 advanced video coding," Jan. 2006, doi: https://doi.org/10.1145/1118299.1118471.

[15] O. P. Orhagen, "Memory requirements for your Raspberry Pi with SD and eMMC | Mender," *Mender.io*, Jun. 29, 2021. https://mender.io/blog/memory-requirements-for-your-raspberry-pi (accessed April 15, 2025).

[16] Roboflow, "Roboflow: Go from Raw Images to a Trained Computer Vision Model in Minutes.," *roboflow.ai*. https://roboflow.com/

[17] Ultralytics, "Hyperparameter Tuning," *Ultralytics.com*, 2023. https://docs.ultralytics.com/guides/hyperparameter-tuning/#preparing-for-hyperparameter-tuning (accessed April 25, 2025).

[18] OpenCV Documentation "Camera Calibration" [Online]. Available: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

[19] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[20] “University of Nottingham Policy on the Use of Animals in Research - The University of Nottingham,” *www.nottingham.ac.uk*. https://www.nottingham.ac.uk/animalresearch/policy/policy.aspx

6 Appendix

6.1

Figure 55: DC Captured at 640x640px, scaled to the resolution captured in 2048x2048px photo once cropped.

Figure 56: DC captured at 2048x2048px, cropped to fit.

6.2

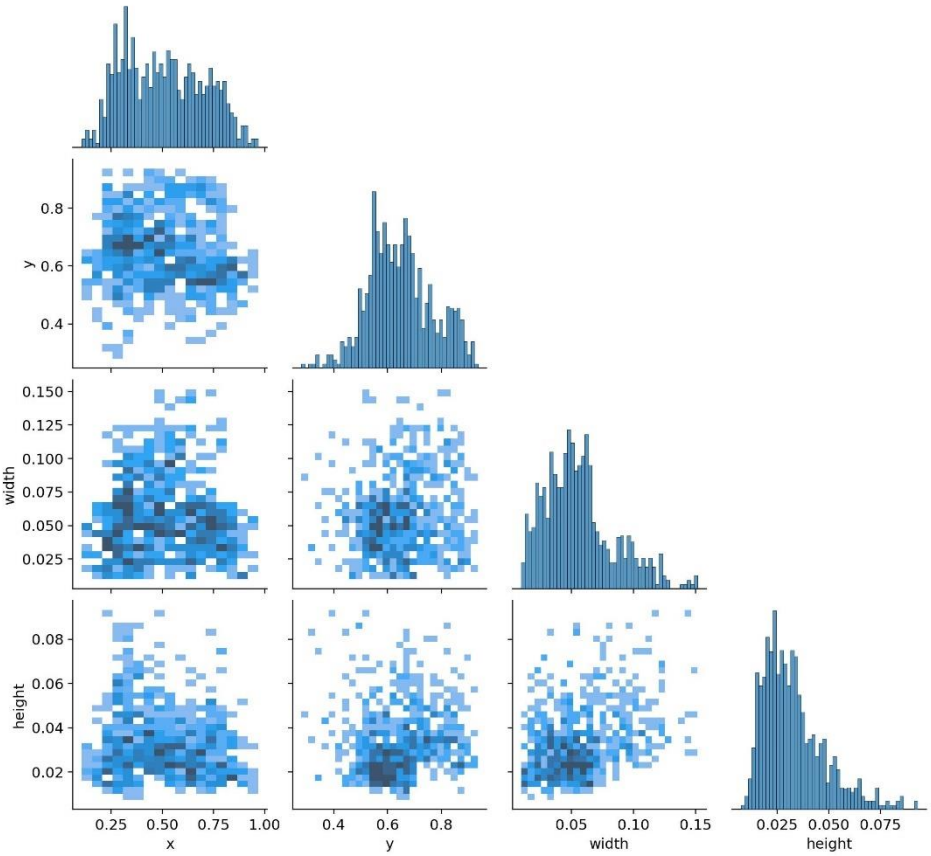


Figure 57: Correlogram presenting the average size and position of DC throughout the dataset formulated from their photos.

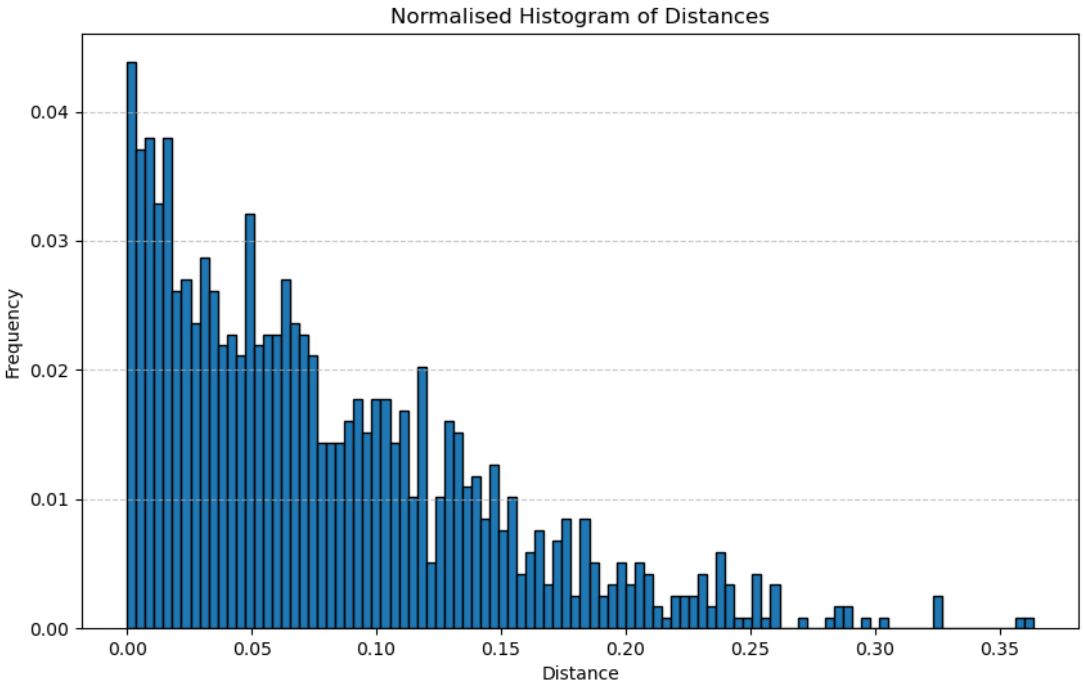
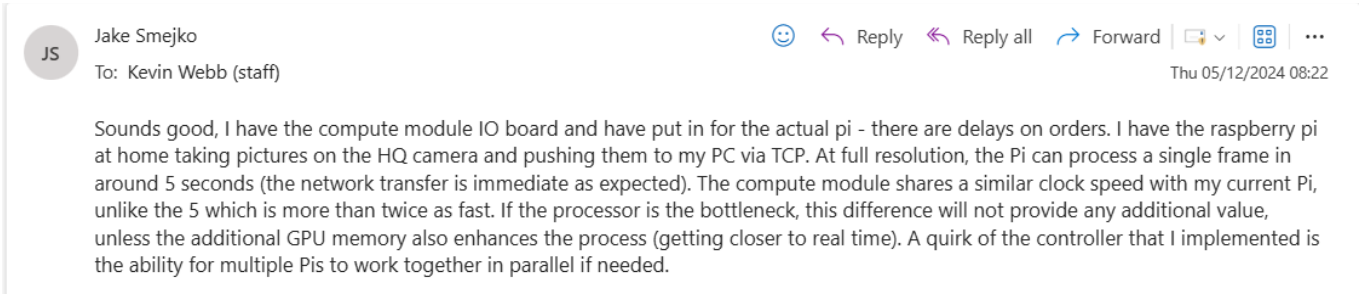


Figure 58: Normalised histogram of the portion of the frame's height that separates DC in the water depth they are depicted swimming at.

6.3

All code is available online here: https://github.com/js-smejko/FYP/tree/main

6.4



Jake Smejko

To: Kevin Webb (staff)

Reply | Reply all | Forward

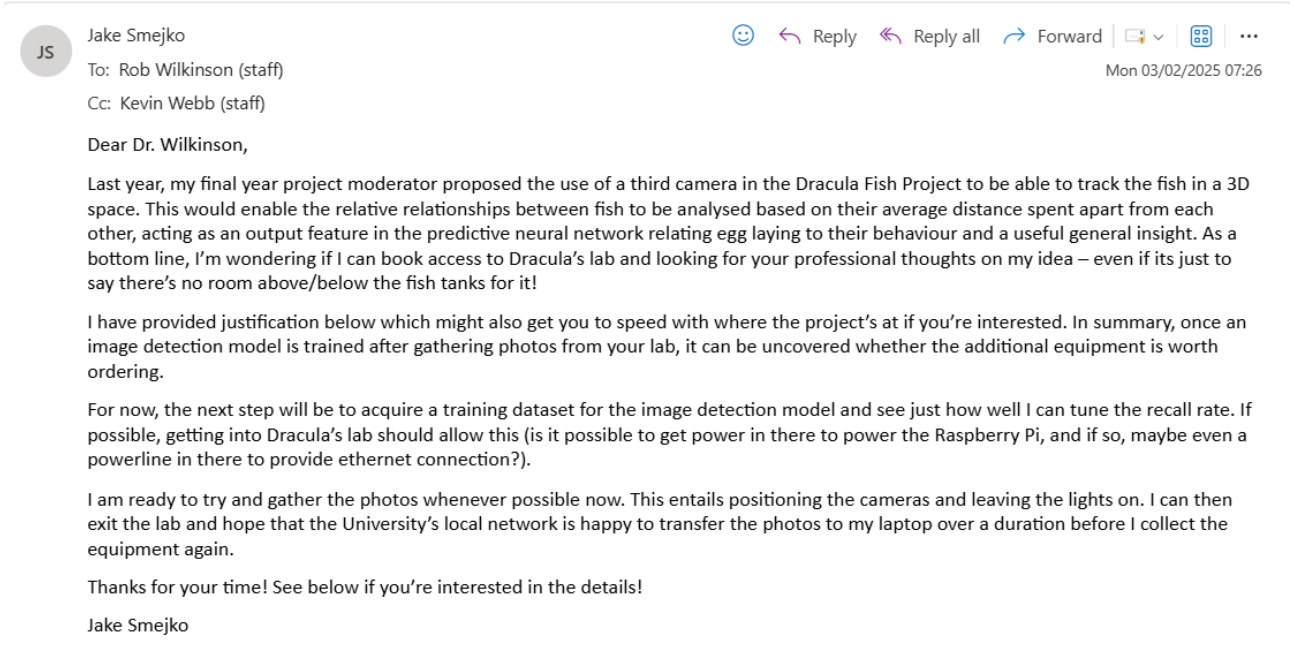
Thu 05/12/2024 08:22

Sounds good, I have the compute module IO board and have put in for the actual pi - there are delays on orders. I have the raspberry pi at home taking pictures on the HQ camera and pushing them to my PC via TCP. At full resolution, the Pi can process a single frame in around 5 seconds (the network transfer is immediate as expected). The compute module shares a similar clock speed with my current Pi, unlike the 5 which is more than twice as fast. If the processor is the bottleneck, this difference will not provide any additional value, unless the additional GPU memory also enhances the process (getting closer to real time). A quirk of the controller that I implemented is the ability for multiple Pis to work together in parallel if needed.

05/12/2024

Summarised Planned State of Project: *[This section should summarise where you expected to be with the project at this review stage according to your Time Plan.]* Samples were to be acquired from the piezoelectric sensors and a camera was to be used to gather footage of Danionella Cerebrum.	**Actual Progress Since Last Review** *[Summarise the actual progress you have made on the project since previous Progress Review (or start of project). You should consider relevant deliverables, milestones, and Gannt chart tasks.]* Hardware limitations of the Raspberry Pi were addressed using TCP in a demonstration using an existing Pi, which further enabled insight into the ideal specification for the Compute Module 4, which was ordered before it was found that the camera sockets on its breakout board used a different connector. A PR has been filed for these. A skeleton of image tracking software, time-distance-of-arrival triangulation and the classifier is in place for when these arrive, while software for gathering the initial training images for the image detection model has already proven to work.
Next Steps and Supervisor Feedback *[In this section you should consider the differences between where you planned to be and where you are. How will this affect the project? What are you going to do to get back on track? Are there on-going problems that will have a knock-on effect to future tasks? Is there additional support or resource that is required? You should include feedback from your supervisor as discussed in the meeting and their approval of changes.]* Although the order of achievements has varied, the project is on schedule because the most challenging setbacks have been uncovered and dealt with.	

27/01/2025



JS

Jake Smejko

Reply Reply all Forward

To: Rob Wilkinson (staff)

Mon 03/02/2025 07:26

Cc: Kevin Webb (staff)

Dear Dr. Wilkinson,

Last year, my final year project moderator proposed the use of a third camera in the Dracula Fish Project to be able to track the fish in a 3D space. This would enable the relative relationships between fish to be analysed based on their average distance spent apart from each other, acting as an output feature in the predictive neural network relating egg laying to their behaviour and a useful general insight. As a bottom line, I'm wondering if I can book access to Dracula's lab and looking for your professional thoughts on my idea – even if its just to say there's no room above/below the fish tanks for it!

I have provided justification below which might also get you to speed with where the project's at if you're interested. In summary, once an image detection model is trained after gathering photos from your lab, it can be uncovered whether the additional equipment is worth ordering.

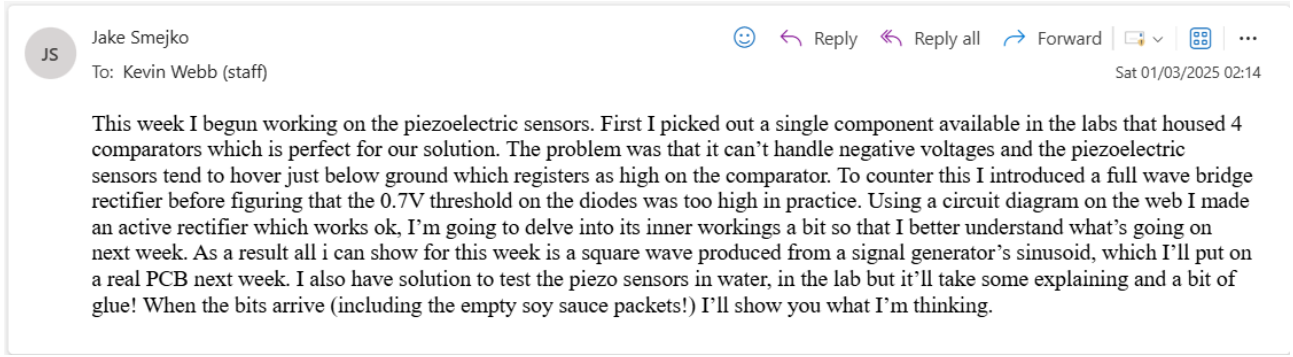
For now, the next step will be to acquire a training dataset for the image detection model and see just how well I can tune the recall rate. If possible, getting into Dracula's lab should allow this (is it possible to get power in there to power the Raspberry Pi, and if so, maybe even a powerline in there to provide ethernet connection?).

I am ready to try and gather the photos whenever possible now. This entails positioning the cameras and leaving the lights on. I can then exit the lab and hope that the University's local network is happy to transfer the photos to my laptop over a duration before I collect the equipment again.

Thanks for your time! See below if you're interested in the details!

Jake Smejko

03/02/2025



JS

Jake Smejko

Reply Reply all Forward

To: Kevin Webb (staff)

Sat 01/03/2025 02:14

This week I begun working on the piezoelectric sensors. First I picked out a single component available in the labs that housed 4 comparators which is perfect for our solution. The problem was that it can't handle negative voltages and the piezoelectric sensors tend to hover just below ground which registers as high on the comparator. To counter this I introduced a full wave bridge rectifier before figuring that the 0.7V threshold on the diodes was too high in practice. Using a circuit diagram on the web I made an active rectifier which works ok, I'm going to delve into its inner workings a bit so that I better understand what's going on next week. As a result all i can show for this week is a square wave produced from a signal generator's sinusoid, which I'll put on a real PCB next week. I also have solution to test the piezo sensors in water, in the lab but it'll take some explaining and a bit of glue! When the bits arrive (including the empty soy sauce packets!) I'll show you what I'm thinking.

01/03/2025