Gamify Fish Behaviour using IoT & Cloud : Finding patterns in randomness

Submitted by
Pallavit Aggarwal
Msc Intelligent Systems – 22333721
Trinity College Dublin
aggarwap@tcd.ie

***Abstract*— Group-10 IoTrinity utilizes IoT components, including ESP-EYE v2.1, ESP-32, and cloud resources such as AWS queue and S3 storage, to monitor the behavior of a fish in an aquarium. Our goal is to model its behavior and explore the possibility of using it to play a game. Our project employs machine learning techniques to identify patterns in the fish's movements, which may have practical applications in pet or animal monitoring. Finally, we use lambda functions on AWS to process the fish's position and map it to a corresponding action in the game which is done using a python script to overlay the incoming messages received from function onto a Kafka receiver and execute game actions. Prior experiments have shown that this is possible and that fish can indeed play a game but the fish is completely unaware of this and no reward structure to influence the fish's behavior is used, but simply the mapping evolves as per fish's movements to play the game better.**

Keywords—IoT, fish behavior monitoring, machine learning, ESP-EYE, ESP-32, AWS queue, S3 storage, lambda functions, game playing

I. INTRODUCTION

The Internet of Things (IoT) and machine learning techniques have emerged as highly effective tools for studying animal behavior, with numerous applications in scientific research, wildlife conservation, and pet monitoring. In this paper, we present a case study of Group-10 IoTrinity's IoT project, which utilizes various components to monitor a fish's behavior in an aquarium and explore the possibility of using it to play a game.

Our project incorporates a YOLO (You Only Look Once) model for object detection and localization to identify fish and track their position in real-time. YOLO is a state-of-the-art convolutional neural network that performs real-time object detection and has shown impressive results in various applications, including animal behavior monitoring. The YOLO model is integrated with ESP-EYE and ESP-32 devices, which collect data from the aquarium and process it in real-time. We also employ image recognition techniques using open-cv and edge detection to analyze the collected data and extract meaningful insights from it.

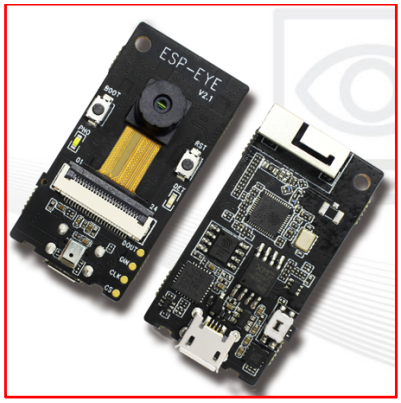
The collected data is stored in AWS S3 storage, which provides a scalable and secure cloud-based solution for data storage. We utilize AWS queue to manage the flow of data between different components of our system. Lambda functions are used for processing the data and mapping the fish's position to a corresponding action in the game. This project highlights the potential of IoT and machine learning techniques in animal behavior monitoring and provides valuable insights into the behavior of fish in an aquarium setting.

In the following sections I first briefly describe each component and its usage, followed by the architecture of our project and how each component is interconnected and further describe our approach and the inspiration we took from other implementations.

II. COMPONENTS

A. ESP-EYE v2.1

As per the main Espressif's own description, the ESP-EYE is a development board for image recognition and audio processing, which can be used in various IoT applications. It features an ESP32 chip, a 2-Megapixel camera and a microphone. ESP-EYE offers plenty of storage, with an 8 Mbyte PSRAM and a 4 Mbyte flash. It also supports image transmission via Wi-Fi and debugging through a Micro-USB port.



The ESP-EYE can be initialized using the esp-who examples for face and image recognition using esp-idf libraries and flashing these examples onto its ESP32 dual core Tensilica LX6 processor chip.

B. ESP - WROOM 32E

ESP32 is a low-cost System on Chip (SoC) Microcontroller from Espressif Systems. It is a successor to ESP8266 SoC and comes in both single-core and dual-core variations of the Tensilica's 32-bit Xtensa LX6 Microprocessor with integrated Wi-Fi and Bluetooth.

ESP32 has a lot more features than ESP8266 and it is difficult to include all the specifications here but mainly it provides us with Support 802.11 b/g/n Wi-Fi connectivity with speeds up to 150 Mbps, 34 Programmable GPIO, Support for both Classic Bluetooth v4.2 and BLE, and Dual-Core 32-bit LX6 Microprocessor with clock frequency up to 240 MHz.

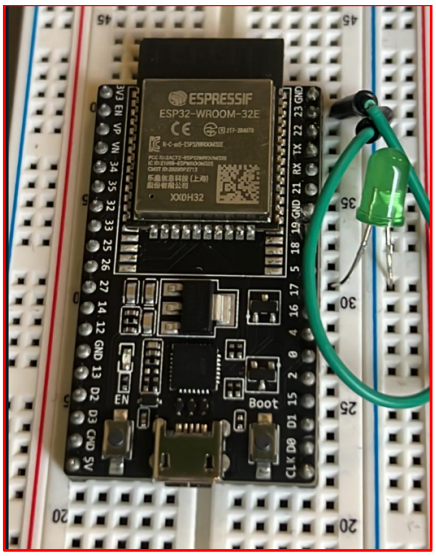


Fig: ESP-32 connected to breadboard

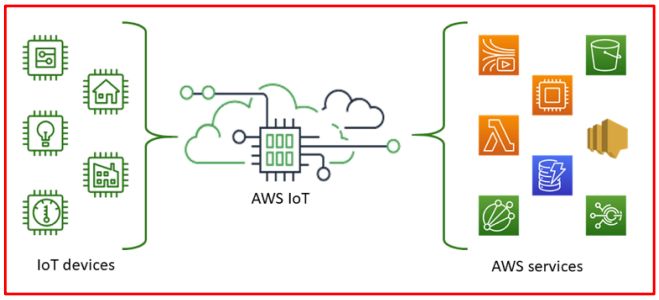
Though we don't completely utilize the power of this device, we have however run multiple out of the box examples on it to completely understand it's capabilities.

C. AWS IoTCore and S3 Services

AWS IoT-Core is a cloud-based service provided by Amazon Web Services that enables the secure connection, management, and integration of IoT devices with the AWS cloud. This service allows IoT devices to communicate with the cloud and exchange data, providing a scalable, reliable, and secure platform for IoT solutions.

With IoT-Core, developers can connect, configure, and manage a large number of IoT devices from a single dashboard. It provides support for various protocols, including MQTT, HTTP, and WebSocket, making it easy to integrate with a wide range of IoT devices and applications.

We define a "Thing" on the AWS-IoT platform first which is an abstraction of the internet of "thing" device that we initialize on their portal. Once this thing is defined, we have to generate certificates and policies to attach this virtual thing to our IoT device so that both are in sync and able to communicate with each other efficiently. AWS IoT-Core allows us to create message brokers which provides pub/sub capabilities so that message from the device can be uploaded to AWS queue and interpreted and processed as per our requirements then.

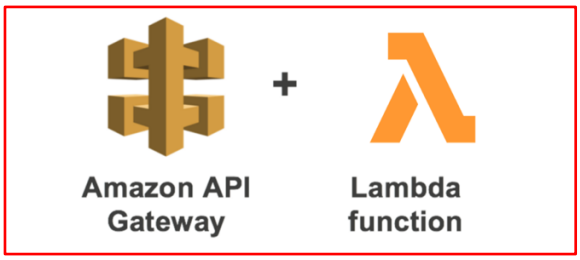


The S3 service is a storage service which is used to save the images captured by the ESP-EYE every certain interval. These images are used by the Machine Learning Algorithms implemented to interpret the fish and its position in the fish tank and also interpret the water level in the fish tank to run the actuator.

D. AWS Lambda Functions and Machine Learning Algorithms

Lambda functions offer the ability to use any programming language like Python, C++, Java and run the code without worrying about the underlying infrastructure. Lambda functions can be triggered by events such as HTTP requests or database changes. In the case of running YOLO on Lambda, the function will be triggered by an S3 bucket upload event, where the Lambda function will process the uploaded image using YOLO and process the detected fish in the fish tank and return the keystroke mapping associated with its position.

AWS Lambda and S3 both require an API-Gateway to securely communicate with public traffic. While this is handled internally by AWS IoT-Core, for Lambda and S3 we also implement an API-Gateway service of AWS that enables us to send and receive traffic directly from devices or any services outside of AWS.



Apart from YOLO, we also implement the Image Recognition algorithm using open-cv and edge detection to identify the level of water in the aquarium to use our actuator to fill water in the aquarium. The algorithm checks the height of the water from the top edge of the aquarium using image recognition and returns a message to ESP-32 board using MQTT protocol to turn on the motor for the actuator.

E. Message Broker and Game on Localhost

We use Kafka services that are deployed online for a free subscription per certain number of messages exchanged to act a broker between the lambda functions and the output from machine learning algorithms and the final game that has to be played using this output.

The message is published from the lambda function onto a Kafka queue and the Python script subscribes to this queue to poll messages from it every certain interval. In an event based setup, whenever there's a message on the queue, the script processes it to play the game.

This is an individual essay assignment for IoT subject
For any feedback or queries please contact me at aggarwpa@tcd.ie

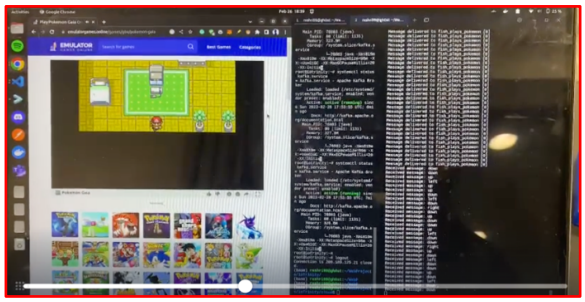


Fig: Pokemon running on Localhost and taking input from python script

The idea is that since the fish continues to move in the aquarium, it can continuously send new data to the models to be processed and it can be overlayed to different buttons which can be sent as an output to the script to press them in the game.

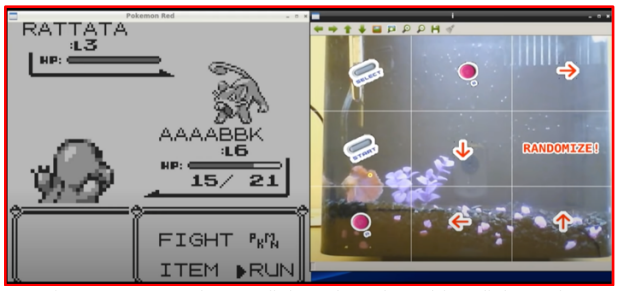


Fig: Button overlay on fish tank and tracking fish to play Pokemon

F. The Actuator

The actuator is a moving part in the project which reacts to a certain particular input and takes some motion based action based on that. For the actuator we have used a mini water pump 3-6V. When the lambda function which is separately built for the actuator does edge detection on the level of water it sends a trigger to turn the level on one of its selected pins to HIGH. This completes the circuit for the motor and switches it on, pumping water into the tank. Another pump will be used to pump water out of the tank in case it overflows or there's more water than required.

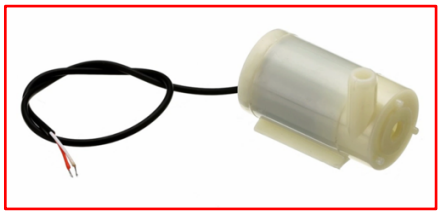


Fig: mini water pump

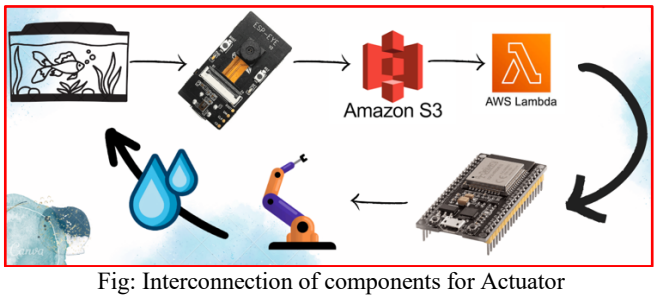
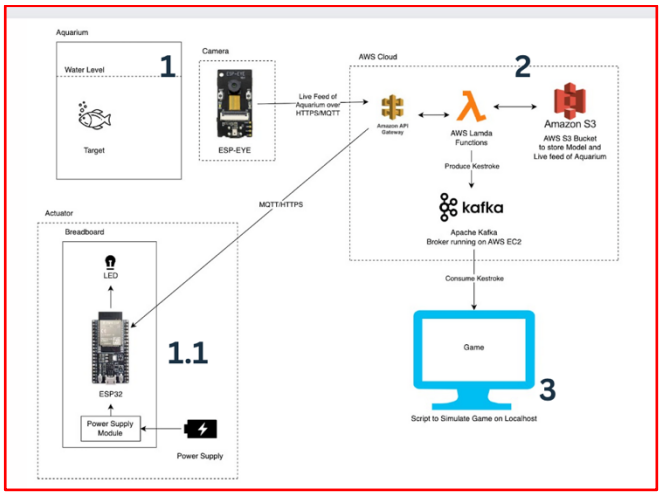


Fig: Interconnection of components for Actuator

III. ARCHITECTURE

These individual components are connected together in the following way. This may not be the final architecture, as we are yet to integrate everything together, and integration might reveal newer challenges.

In the next image **1)** shows the aquarium and the esp-eye in-front of it. The ESP-EYE will take images every certain interval (5 seconds) and send it to AWS. **2)** shows AWS cloud components, it has an API-Gateway so that we can communicate with any public device or service and the components inside AWS securely. It also has AWS Lambda which would run the .h5 file generated from the machine learning model. It has S3 for image storage, and a Kafka instance which could or could not be inside AWS.



3) In this section we see that the output from the lambda function in 2) is sent to Kafka which is read by the python script running on the localhost machine. This script also triggers the game as per the input. Ultimately section **1.1)** of the architecture shows the actuator, and the connection of ESP-32 to AWS. The ESP-32 is currently able to power the bulb based on an input from the lambda function and we would replace the bulb with the motor so as to power it on and off based on the decision made by the lambda function iteratively using a constant feed from the ESP-EYE.

IV. IN-DEPTH DISCUSSION ON CAMERA AND CLOUD

My main contribution so far has been on getting the ESP-EYE working first with esp-who examples and boot loading and flashing it onto the device and secondly integrating it with cloud using Arduino ide for better management of libraries and easier debugging. However, I have not limited to myself to camera and actively participate in defining the architecture, cloud, choice of algorithms, dataset for the training and script for game-play as well.

I describe the approach and provide the code snippets necessary for getting the camera started.

A. ESP-IDF and ESP-WHO

For working with the esp-who examples and setting up the esp-who environment, esp-idf libraries are required.

We have to follow the following steps in order to get started with esp-who:

Go to the following Espressif website and install all the necessary prerequisites.
https://docs.espressif.com/projects/esp-idf/en/release-v4.3/esp32/get-started/index.html#step-1-install-prerequisites

Note: The GitHub repo for esp-idf https://github.com/espressif/esp-idf which has all the necessary libraries has to be downloaded recursively as there are many inter-dependent libraries implemented by others separately. These usually cause errors while running esp-who and the missing dependencies have to be downloaded and put in the particular namespace manually.

Now running the following commands should install and setup the esp-idf properly

1. Execute the install.sh file to make and install the esp-idf project.
   cd ~/esp/esp-idf
   ./install.sh esp32

2. Check if the device is connected to the laptop.
   /dev/cu.*

3. Run the export.sh to get the idf.py command available in the environment
   . export.sh

4. Now, run the cmake command inside the esp-who examples folder
   cmake

5. make install will further install all the libraries and dependencies required by the project
   make install

6. Run the following command to open the bootloader and configure WIFI and other properties.
   idf.py menuconfig

7. Finally, flash the program onto the device
   idf.py flash monitor

Using the above steps and with some patience and perseverance I was able to run the esp-who example and load it onto the device. The cat-recognition, face detection and motion detection example ran successfully.

B. ESP-EYE with Arduino IDE and AWS SDK

While running esp-who examples was done using command line and using idf.py command prompt, for setting up ESP-EYE with AWS, AWS-SDK had to be downloaded.

While downloading the SDK was straightforward, setting up it's paths as per the requirements of the examples and defining the cmake.txt file took some time and did not yield positive results. In the interest of time, I resolved to Arduino IDE to get the camera working.

For Arduino IDE, the first step is to add the esp-dev board for development purposes to the IDE as it doesn't come pre-installed. In setting, preferences, additional boards manager the following url can be added: https://dl.espressif.com/dl/package_esp32_index.json to install the board onto the IDE.

Now using the "AI Thinker ESP32-CAM" board, the device should be connected and detected by the IDE. The following image shows the pins that need to activated in the case of ESP-EYE v2.1

```
#define CAMERA_MODEL_ESP_EYE
#define PWDN_GPIO_NUM -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 4
#define SIOD_GPIO_NUM 18
#define SIOC_GPIO_NUM 23
#define Y9_GPIO_NUM 36
#define Y8_GPIO_NUM 37
#define Y7_GPIO_NUM 38
#define Y6_GPIO_NUM 39
#define Y5_GPIO_NUM 35
#define Y4_GPIO_NUM 14
#define Y3_GPIO_NUM 13
#define Y2_GPIO_NUM 34
#define VSYNC_GPIO_NUM 5
#define HREF_GPIO_NUM 27
#define PCLK_GPIO_NUM 25
```

Even though the cam is set to AI Thinker board, we have to initialize ESP-EYE cam specific pins.

```
void cameraInit() {
  gpio_install_isr_service(ESP_INTR_FLAG_LEVEL2);
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;  //YUV422,GRAYSCALE,RGB565,JPEG
  config.frame_size = FRAMESIZE_HVGA;
  config.fb_count = 2;
  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    ESP.restart();
    return;
  }
}
```

Next the camera init in the above image shows that we use the HVGA frame size, JPEG format for the pictures, and set the freq hz to 2GHz.

```
void grabImage() {
  camera_fb_t* fb = esp_camera_fb_get();
  if (fb != NULL && fb->format == PIXFORMAT_JPEG && fb->len < bufferSize) {
    Serial.print("Image Length: ");
    Serial.print(fb->len);
    Serial.print("\t Publish Image: ");
    bool result = client.publish(ESP32CAM_PUBLISH_TOPIC, (const char*)fb->buf, fb->len);
    Serial.println(result);

    if (!result) {
      Serial.print("Something went wrong....");
      ESP.restart();
    }
  }else{
    Serial.print("Coming here: \n");
    delay(1000);
  }
  esp_camera_fb_return(fb);
  delay(1);
}
```

The grab image function uses the esp_camera_fb_get() function to get the image, this can be called in every 5 minutes and the image uploaded to S3. Currently in the above example it sends the image to the queue by first converting it to char format and then writing to the queue buffer.

```
esp_http_client_handle_t http_client;

esp_http_client_config_t config_client = {0};

 WiFiUDP ntpUDP;
 NTPClient timeClient(ntpUDP);
 timeClient.begin();
 timeClient.update();
 String Time =  String(timeClient.getEpochTime());
 String MAC = String(WiFi.macAddress());
 Serial.print("Time:" );  Serial.print(Time);
 Serial.print("MAC: ");  Serial.print(MAC);

 String post_url2 = "https://f22kokvbo6.execute-api.us-east-1.amazonaws.com/dev/" + MAC + "/" + Time;
 char post_url3[post_url2.length() + 1];
 post_url2.toCharArray(post_url3, sizeof(post_url3));

config_client.url = post_url3;
config_client.event_handler = _http_event_handler;
config_client.method = HTTP_METHOD_POST;

http_client = esp_http_client_init(&config_client);

 esp_http_client_set_post_field(http_client, (const char *)fb->buf, fb->len);
```

Sending the picture to S3 is not very different, as can be seen in the above image, it is just a matter of calling the POST HTTP method which is defined in the API-Gateway that sends the image in char format to S3 which decodes it and saves it as a jpeg.

The libraries that need to be installed in Arduino IDE to support these functionalities are NTPClient, MQTT and AWS-SDK support library.

C. *AWS-IoT Core, AWS S3, API Gateway Setup*

The setup for AWS-IoT Core, requires the creation of a json file for policy setup which is the key setup criteria that not many online resources focus on and which is key in connecting the IoT device with the cloud. The setup information can be found online in a step-by-step format for connecting and setting up AWS-IoT thing, S3 and API-Gateway and generating certificates, that will be included in the secrets.h file for the above codebase.

V. Discussion on Other Modules

A. *Fish Detection Algorithm*

There is no single "best" algorithm for object detection, as the choice of algorithm depends on several factors such as the specific use case, the size and quality of the dataset, and the computational resources available. YOLO emerged as the choice of algorithm for our project because it was relatively faster than Faster R-CNN, easier to implement, and open-source.

B. *Edge Detection Algorithm*

Using open-cv, we can use different edge detection techniques like Canny edge detection or Laplacian edge detection as out of the box implementations in open-cv.

C. *Message Broker*

Kafka emerged as the popular choice to implement a message broker with pub/sub functionality due to its widespread availability and a huge community with a lot of examples to integrate it with different technologies.

D. *Gameplay*

There are multiple examples available that help us achieve a customised behaviour in the game by loading it into a python script and using the input to press buttons or take actions in the game.

E. *Actuator*

The actuator is a simple mini pump motor which gets the voltage LOW and HIGH commands from the ESP-32 based on the input received to it from Cloud. We can do processing on the ESP-32 but that would be as a next step if we first setup the entire architecture on cloud which is a faster to implement due to limitations of the esp-32 in terms of memory and libraries to be used.

VI. Existing Research

The idea for implementing this project was inspired from [1] which is a video of fish playing Pokemon on Twitch which is a social platform by randomly mapping buttons to its movements to play the game. In the video, the fish can also be seen making a credit card payment on App Store accidently. While this sounds like a good experiment, it has deep rooted research aspects associated with it.

In [2] and [3] researchers create a system that analyses mouse behaviour by monitoring them through a video feed and based on the repetitiveness of their actions, find a suitable action to help the mouse perform that action better. E.g. if it comes to a section to eat every time, then this classification model appropriately identifies when the mouse has come to eat and gives it food. In [4] researchers use IoT to monitor a live basketball game and analyse every action made on the field. [5] presents a system that uses thermogenetic to control the behaviour of fruit flies in real-time. The system can be used to study the neural circuits underlying behaviour and decision-making.

There are numerous efforts put to understand and model animal behaviour and find meaningful patterns [6] in their behaviour. Our efforts are only to scratch the surface and open up this field of animal behaviour analysis using IoT in a novel way. As a next step, the prediction model could study the movement patterns of the fish and in turn identify the best game suited for the fish. Could we identify other movements that indirectly could benefit from the randomness of the fish's movements? Moreover, we don't currently reward the fish, but a pavlovian experiment could also be an extension to our implementation.

References

[1] Fish Plays Pokemon: Pallet Town Syndrome ; https://www.youtube.com/watch?v=48-qOC4fCdk&list=PLFYL2jYKtv0dT2LKOiu1reh7NrrtxMo2m

[2] "Automatic Visual Tracking and Social Behaviour Analysis with Multiple Mice" https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0074557

[3] The Mouse Action Recognition System (MARS) software pipeline for automated analysis of social behaviors in mice , https://elifesciences.org/articles/63720

[4] "Video Analysis and System Construction of Basketball Game by Lightweight Deep Learning under the Internet of Things", https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8940549/

[5] "FlyMAD: rapid thermogenetic control of neuronal activity in freely walking Drosophila", https://pubmed.ncbi.nlm.nih.gov/24859752/

[6] "Seeing Patterns in Randomness: A Computational Model of Surprise", https://onlinelibrary.wiley.com/doi/full/10.1111/tops.12345