Abstract—The paper describes EnVision, a vision system created for the Environet at Ecole Polytechnique Federale de Lausanne (EPFL)’s BioRobotics laboratory. The project consists of two main tasks. The first is to create some form of visual control, linking input from the cameras to the actuators of the Environet. We will primarily be using a biologically inspired feed forward spiking neural network, detailed in the work of Iman Kamali Sarvestani[11]. We will see that a simple network is capable of exhibiting a variety of behaviors depending on how we tune the parameters of the network and its neurons. The second task is to create a functional vision system, defining and prototyping mechanical, electrical and other hardware aspects of the system so that we have a platform upon which we can test our visual model.

I. INTRODUCTION

Envirobot is a biologically inspired lamprey like robot. The purpose of Envirobot is to provide search and surveying services in smaller to medium sized bodies of water such as lakes or ponds. In order to do so effectively we would like to add a vision system to Envirobot. Currently Envirobot has no means of sensing any visual input from its surroundings. Thus the goal of this project is to provide Envirobot with means of visual input as well as demonstrate its functionality via a biologically inspired visual guidance model. In particular the model from computational model of visually guided locomotion in lamprey[11] was chosen. It provides a spiking neural network model that enables a simulated lamprey to respond to varying stimuli such as preys, predators and obstacles from visual input by generating the appropriate motor responses to either approach, escape from or avoid their targets respectively. It also includes mechanisms for prioritizing between multiple targets of either the same or different type. We start with some background behind Envirobot itself and a brief review of relevant literature. Afterwards we will explain model provided in [11], how we implement it in a Webots 8 Simulation and the experiments we perform on the simulation to observe the model’s behavior. Finally we will discuss the hardware aspect of the project including design choices, hardware selection and practical issues.

II. BACKGROUND

A. Envirobot

Envirobot is a swimming robot based off of the lamprey as seen in Fig. 1. Details of the mechanical construction of Envirobot can be found in [6]. In brief, it consists of multiple segments attached end to end, each with a motor that can
move the segment from side to side. There is also a head unit that contains most of the sensors and processing capabilities of Envirobot, which is what the hardware aspect of the project will focus on. Envirobot’s current locomotion model is based off of the Central Pattern Generator (CPG) method explained in [10]. The essentially generates a propagating sine wave along the length of Envirobot. The method differentiates between a signal generated on the left and right sides of the spine with coupling weights between both horizontal and vertical segment pairings along the spine as seen in figure 2. The purpose is of this dual chain method is to create smooth transitions between motion changes in the robot via sinusoidal parameters such as amplitude and phase. In a single chain CPG you could just simply generate the sine wave along the body directly but you would not have the smooth transition functionality. More details can be found in [10].

Currently direction of movement in Envirobot is guided either via user or GPS control. For user control, the user presses arrow keys on the keyboard to control the speed and turning rate of the robot. This is done by sending various amplitude values (denoted R in the literature), to the dual chain CPG model. These amplitude values are what we will seek to control automatically via the aid of local visual input instead of human or GPS guidance. As will be explained in section III the model we choose to experiment with maps very naturally into controlling the R parameters which makes it a good choice for our demonstration of visual guidance.

B. Braitenberg Vehicles

In Vehicles: Experiments in Synthetic Psychology[3] Braitenberg speaks of robots of simple construction and logic that are able to perform complex actions. He gives an example of a simple two wheeled robot with two light sensors on either side. By creating different wiring and mapping configurations between the sensors and wheels he creates minimalistic robots that exhibit various behaviors that could be considered as varying personalities or even free will for each robot. Fig. 3 shows some examples of some of Braitenberg’s creations. Since what is being created are direct mappings between sensors and motors, these behaviors can be seen as instinctual responses in an organism. This makes Braitenberg vehicles a tempting model to follow for emulating such basic tasks such as avoidance, escape and approach which pertain to the basic survival needs of a living organism.

Braitenberg also talks about a simple vehicle that employs the use of thresholds as a simple building block for controller logic. If the threshold value of a threshold block is exceeded, its activa-
tion is triggered and it sends some activation to the next node or block of the model. This mechanism in itself is very simple, but various nodes with various thresholds and activation values can be combined to create a structure that exhibits much more complex behavior such as counting or even some basic form of memory. We will see in the next section that spiking neural networks share similarities with this model. A concise summary of Braitenberg’s book can be found at [2].

C. Spiking Neural Networks

A spiking neural network should not be confused with the much more widely known artificial or deep learning neural network model. While at a superficial level the two may look similar in terms of nodes and connections, their underlying mechanisms are very different. A spiking neural network is meant to be a biologically accurate model of what is happening in the nervous system when an organism responds to some sort of stimuli. A stimulus is represented as a signal which is inputted to a layer of neurons. The neurons respond to said stimuli by firing new signals to the next layer of neurons, and so and so forth until a final set of neurons is reached where the rate of firing can be mapped to a strength of a response in, for example, a muscle. Graphically the spiking neural network looks much like an artificial neural network though each node is usually represented by a cluster of neurons and the implementation of each neuron itself is biologically inspired.

The Leaky Integrate and Fire Neuron is what will be used for our neuron representation. This is one of the most basic implementations of a neuron. Each neuron has stores a value and has a firing threshold. Every time it receives a signal, either from a previous neuron or directly from a sensor input, it adds to this value. The value is also constantly decaying via a standard exponential decay, which is simply done by decreasing the value by an amount proportional to the current value of the neuron. In our implementation this amount is simply a decay constant $\tau \times V$ where $V$ is the current value of the neuron. Thus if the neuron does not receive a signal for a while it will eventually converge to zero. If the value crosses a threshold, the neuron is activated and fires a signal function with some previously defined strength and shape (we will use a dirac pulse for simplicity) to the next neuron. A depiction of how the neuron works can be seen in Fig. 4 and a more detailed explanation is given in [8]. This can be seen as a more neurologically accurate version of the basic thresholding mechanisms seen in Braitenberg machines. While some amount of complexity is added by the fact that there is stored value for each neuron that is being modified over time, the basic unit is still relatively simple which gives the Spiking Neural network a resemblance to the Braitenberg machines. The end result is still a direct mapping between sensor input and motor control via a simple network with predetermined connections.

![Fig. 4. Example of a leaky integrate and fire neuron. The neuron fires whenever it reaches a certain threshold at t(1-4)](image)

Spiking neural networks have been successfully used in other robotics applications such as in [1] and [4]. In these studies the main application was to navigate a robot through a maze or around obstacles. These papers had a somewhat different approach as they incorporated methods of training the network based on either evolutionary algorithms or training runs which is not something that we will be doing. However, they do attest to the potential for Spiking neural networks to perform robotic control tasks that are similar in nature to what we eventually may wish to achieve with Envirobot.

The model that we will be using is simplified version of the spiking neural network where single neurons instead of neuronal populations are used at each node to reduce complexity. To justify this decision, if we have a group of $N$ neurons
per single neuron our network would run at least N times slower, perhaps \( N^2 \) times slower if all neurons between one group and the next are fully connected. This will be more a concern when moving forwards with hardware tests since processing power is already limited. Using single neurons also makes the model much more understandable and easier to visualize.

III. VISION SYSTEM

As lampreys are the earliest form of vertebrate still alive it is often used a subject of study in biorobotics and neurobiology. Much research has been done with regards to the locomotion model of the lamprey but less so in the field of visual guidance. As stated before we chose to implement A computational model of visually guided locomotion in lamprey[11] as our visual guidance model for the Envirobot. There are two main areas from this paper that we will address: the response system and the arbitration system. The model is a spiking neural network. The response system is the part of the network that governs how the network responds to a stimuli of a single type. The arbitration system is another network connected to the response system network that allows the model to prioritize between different types of stimuli appearing within the visual field simultaneously. The network we will describe can be seen for prey and predator in Fig. 5.

A. Response System

The first layer of the response layer corresponds directly to the camera input. These neurons accumulate the response strength of the incoming signal which is just measured as the diameter of the detected red, blue or green ball in pixels over some max diameter size, the maximum strength therefore being 1.0. In our setup all neurons have their spiking thresholds set to spike at a value of 1.0. This is not a value that we will be changing since it is equivalent to just changing the strength of the input activations/signals themselves. The number of neurons in the first layer or the network density is a parameter to be tuned. The number of bins chosen spatially quantizes the camera view column-wise and determines the resolution of target detection. There are three sets of these layers, one each for prey, predator and obstacle. We call each of these sets a visual center. Also note that the neuron layers for the left and right camera are logically separated into two separate arrays. This gives us a total of 6 arrays for the input layer. Also of importance is the organization of the arrays. If we place the two left and right arrays side by side and visualize them as a single array the center of the combined array represents the front of the field of view, thus the arrays are spatially consistent with what is being perceived by the cameras.

Next, each of these layers is connected to layers of equal size called the response layers. It is easiest in explain the connections between these two layers in terms of indices. For the visual centers corresponding to prey, a neuron with index \( i \) maps straight to a neuron with the same index \( i \) in the response layer. For obstacles and predator, the connections are crossed so index \( i \) is mapped to index \( N - i \) where \( N \) is the number of neurons in either the left or right array. The response layer has multiple functions, but it primarily serves as the layer that determines the strength of a response to a stimulus hence it’s name. In this layer there is also lateral inhibition, where each neuron inhibits every other neuron in the response layer in both the left and right arrays. Thus the neuron with the highest response (and thus highest) firing rate will inhibit other neurons more and will take priority. This acts

![Fig. 5. A) Response System for prey. B) Response system for predator. Figure taken from [11]](image-url)
as a selection mechanism between the same type of stimulus. Another function of the response system is to determine the speed of forward motion. To achieve this the response neurons are also all connected to the locomotor neurons, as will be explained in more detail shortly. As with the input layer there are 6 total arrays for the response layer.

After the response layer comes the last set of 6 arrays of neurons. This layer is the auxiliary layer and determines the amount that the lamprey should turn by. Let us take the right array as an example. Neurons of the response layer corresponding to the center of the field of view only map directly the neuron of the same index in the auxiliary layer. The next neuron \(i + 1\) moving farther away from the center maps to maps to neurons \(i + 1\) and \(i\). Neuron \(i + 2\), maps to \(i + 2\), \(i + 1\), and \(i\), etc. This is mirrored in the left array. Thus responses in the center of the response array (a prey in front or an obstacle/predator to the side), activates fewer neurons in the auxiliary layer than a response towards the side (a prey to the side or an obstacle/predator in front).

Finally there is a single left and right reticulo-spinal neuron that the left and right auxiliary layers map to. The left auxiliary layers map to the left reticulo spinal for obstacles and predators and to the right for prey. The opposite is true for the right auxiliary layer. Thus the more activations we get from the auxiliary layer, the more activations we get from one of the reticulo spinal neurons, meaning that it fires more often, generating a higher fire rate, and thus more motor activation on that side of the spine. A higher firing rate on the right corresponds to moving to the left, and vice versa. Note that the prey maps to the opposite side as opposed to the other two stimulus types. This is because for a prey, if we see a stimulus on one side we want to activate the motor neurons on the opposite to orient ourselves towards the target, as opposed to for obstacles and predators where we want to turn away from that side.

At this point the predator and obstacle share exactly the same network structure. The difference between the two is that obstacles need to generate a lower firing rate than predators since it is not as important to avoid obstacles as it is to escape predators. This can be done in many ways: decreasing the strength of the initial input, raising the firing thresholds for the obstacle network, reducing the activation between the input and response layers, etc. We will be experiment with the last of these.

We now return for a moment to response layer. The reticulo spinal neurons handle turning but we also need forward locomotion. For example, if we see a predator, we don’t just want to turn but to run away as well. This is where locomotor neurons come into play. As is the case with the reticulo spinal neurons, there are only two locomotor neurons, one for left side and one for the right. The response layer neurons are connected to locomotor neurons which map bilaterally to both the left and right reticulo spinal neurons projecting equally to both, and thus generating forward motion. The more the response layer fires, the faster forward the lamprey moves. The full connectivity between these different systems can be seen in Fig. 6.

Note that this figure also mentions something called the Arbitration system. This is what we will go over in next subsection.

### B. Arbitration System

The arbitration system is sub network that connects to the response layer. It allows the visual system to prioritize one type of target over another so that the lamprey does not get confused with competing signals between different types of stimuli. This structure of the arbitration system is biologically inspired, modeled after the basal ganglia. We will not be looking into the accuracy of the biological correspondence but just describe the structure of the network and its performance.

This is a redundant system. It has multiple paths by which it can inhibit competing signals. It starts with an Subthamallic Nucleus (STN) neuron for each of the 3 visual centers. Every response neuron in a visual center maps to its corresponding STN neuron. The STN neuron maps to both an internal Globus Pallidus (GPi) and external Globus Pallidus (GPe) neurons of which there are also one for each visual center. It maps to each of these differently.

The STN Neuron for a visual center maps to the GPe neuron of the same visual center with a positive activation. Each GPe neuron maps to
Fig. 6. Full connectivity of the visual network taken from [11], dorsal view. Each of the blocks for Escape, Avoidance and Approach represent the full input, response and aux layer structure of each visual center. The thicker black path shows how a predator seen from the right camera can generate pure forwards motion via the locomotor neurons. The red path shows that an asymmetric input to the right reticulo-spinal neuron is also generated, steering the lamprey towards the left away from the predator. One distinction with our implementation of this model is that we will take firing rates directly from the reticulo spinal neurons and won’t be connecting the neuron to any motor neurons.

Fig. 7. Connectivity of the arbitration system. A red arrow is an inhibitory connection, green is excitatory.

The STN Neuron of the other two visual centers with negative activations. Each GPe neuron also maps to its corresponding GPi with a negative activation. Thus this loop serves to increase STN and GPe firing rate for the strongest stimulus while decreasing the activation of the stimulus type’s GPi inhibitory neuron, which as we will see are in charge of inhibiting the their visual centers.

The STN Neuron for a visual center maps to the GPi neurons of the other two GPi neurons with an excitatory pulse. Each GPi neuron in turn maps back to the it’s own corresponding STN neuron (prey to prey) with an inhibitory pulse. This loop excites the GPi neurons of the weaker stimulus types. Finally the GPi maps back to it’s all of its corresponding response layers neurons with a negative activation. The goal of this network is to have a strong firing rate for the GPi neurons that correspond to weaker stimuli and a lower rate for stronger stimuli.

You can see that this system is a redundant one. If we take away one of the sets of connections, either between STN and GPi, or STN and GPe or GPi and GPe, the target selection mechanism will still work. If you remove the GPe altogether the system would still work to some degree and would be closer to just the simple lateral inhibition that was used for selection between stimuli of the same type. We will see in our experiments that by decreasing or removing one of these connection groups altogether we can still observe inhibition between different target types. Fig. 7 shows the connectivity between the different parts of the
The paper also proposes an extension system which can be attached to the arbitration system that can be used as a framework that can modify the response to certain stimuli though it is not well motivated and indeed the examples they provide for modifying such responses rather arbitrary, such as negating prey stimuli if you happen to see some yellow stimulus. It is basically a structure to add inhibitions or excitations onto the network in response to some new type of stimulus that can be used to create different rules of behavior if the existing prey, predator, obstacle interactions aren’t sufficient. We will ignore the extension system and focus on the response and arbitration systems.

C. Limitations/Assumptions

There are a number of limitations and assumptions made with this model which we will describe here.

- The model assumes that we have full 360 degree information of the surroundings of the robot. The strength of certain inputs are determined by distance to and size of targets but since tests are only run in simulation it assumed in the paper that they are just known before-hand. In our simulation we do not have a full 360 degree view (only about 240) and will have access to the only information we obtain from our cameras. Since we only have monocular view depth and size are ambiguous thus we must rely on the size of the detected blob in the images. We also cannot respond to stimuli that we cannot see in the Envirobot’s blind spots.
- The complexity of this model is also an issue. The model assumes that the network is simulated which large populations of neurons in both the visual and locomotor centers and all testing is done via simulation. It may be an issue to have such a computationally intensive model on our hardware so to begin we will stick with a simplified model. For each of the neurons in the model (the ones given in Fig. 5 and Fig. 7), we will use a single neuron instead of a population. This will also make the network simpler to visualize which will aid us in understanding the effect of changing various parameters of the network.
- The locomotion model also assumes that the spine is divided up into almost 100 segments, where as Envirobot typically uses less than 10. The laboratory from which the paper was written also has it’s own lamprey robot as described in [13] though even it lacks the number of segments assumed by the paper. We replace their motion model entirely with our dual chain CPG model.
- In addition, the experiments from the paper are only via simulation and the frequency at which they receive input from the environment is not limited by camera frame rate as it is in our case. This allows for faster more accurate response to stimuli. We will see why this is an issue in the next section during which we will discuss the pipeline of our implementation.

IV. PIPELINE

In this section we will go over our implementation pipeline which we mainly will describe in terms of our simulation but also applies to hardware. We use Webots 8 for simulation. Note that Webots 7 or higher is required as we will need to use the IMU module. In our simulation the prey, predator and obstacles are represented as green, red and blue balls respectively. We use an Envirobot with 7 segments and an oscillation frequency if 1.5 Hz throughout our simulation tests. Forces of water and buoyancy are also simulated via a physics plug-in.

A. Obtaining images

We start by simply obtaining the images from two wide angle spherical lenses positioned on the head of the simulated lamprey. Due to issues with distortion and calibration specific to the Webots software we cannot extend the angle of said cameras much beyond 120 degrees, which means that with 2 cameras we will have a 120 degree dead zone behind the robot. This is not ideal for avoidance and escape tests as the target will quickly leave visual range and no longer be detected which is not ideal for escaping from a predator that is directly behind it. In the actual robot there will be a blind spot behind as well though it will be quite a bit less than 120 degrees as we will be using 185 degree fish eye lenses,
though angled forward by 20 degrees. The angling towards the front creates an area of overlap that provides Envirobot with binocular vision capabilities in front and monocular for most of the sides. This is the case for the actual lamprey and indeed most fish, as seen in Fig. 8. For our purposes we only considered monocular vision and will crop the left and right images appropriately such that our Envirobot has a 240 degree monocular view. The actual Envirobot will have about a 320 degree field of view.

![Field of View Diagram](image)

Fig. 8. Visual field of view for a fish, similar to what we will have for envirobot.

B. Rotating Images

As mentioned in section III-C one problem we have to deal with is that the head where the cameras are mounted is not stable. Turbulence in the (simulated) water will cause the horizon line in the image to rotate as a result of both pitch and roll of the head module. This will certainly be the case for actual hardware tests as well. Since in the model it is assumed that the horizon line is completely horizontal and that targets are mapped vertically downwards into input arrays we have to compensate for this in our video input by unrotating the images.

We can do this by using a 9 axis IMU module on the Envirobot. We can use the roll and pitch angles along with the yaw angle corresponding to the mount angle of the camera to determine a rotation transformation matrix using the Rodrigues’ rotation formula [16]. Note that we do not need the yaw from the IMU as we don’t need to account for the sway of the head and it does not contribute to alignment of the horizon. We can apply this transformation matrix to the negative Z axis which represents the initial forward facing direction of each camera to get the post-rotation view axis. Like-wise we can apply the transformation to the X axis and then determine the angle between the resulting X axis and XZ ground plane via rotation around our rotated Z axis. This gives us the angle by which we have to rotate our images. You can see the result of this rotation in Fig. IV-B. This has been implemented in both simulation and hardware though we expect some inaccuracies on hardware as even a 9 axis IMU will still have some issues with measurement drift over time. We do not need it to perfect however, as long as the horizon is roughly horizontal.

![Image of Rotated Images](image)

(a) Left image before rotation (b) Right image before rotation
(c) Left image after rotation (d) Right image after rotation

C. Camera Calibration

We are using fish-eye lenses with increasing distortion as we get farther from the center of the image. These need to be properly rectified before we can map the input as well. This is done...
via OpenCV’s calibration methods. In OpenCV’s model the distortion is modeled as a 4th order polynomial for which the coefficients need to be determined. To do so we feed the method many images of a black and white checkerboard taken from the camera in question. We then find the internal corner points of the checkerboard either via OpenCV or manually. Once the distortion coefficients are determined just once they can be used to undistort all future images from the camera. In Webots 8 the spherical lens is not really a fish-eye lens and at its core is a cube mapped sphere, meaning the distortion can’t be properly modeled by OpenCV’s methods. We use a standard tangent mapping model such as one described in [15]. This provides an undistortion that is good enough for visualization though is not perfect. Note how near the edges the grid on the ground floor near the edges of the image start to curve slightly. You can see left and right images side by side after undistortion in Fig. 9.

Fig. 9. Our final left and right combined view after image rotation and calibration.

D. Colored blob detection

Next we filter the image into three separate binary maps, for red, green and blue objects. We then use OpenCV’s simple blob detection libraries to find corresponding blobs that correspond to targets of interest. Note that overlapping targets of the same color will end up being merged into a single blob but without any sort of depth information there is not much that we can do about that. The result of blob detection can be seen in Fig. 10.

E. Conversion into input array

The blob detection method also provides us with information about the size of each blob. We can use find the diameter at each camera frame which are then mapped proportionally into an input strength and added to the bin of the response topographically corresponding to the column position of the blob. The strength value ranges from 0 to 1 as we divide the diameter by a preset maximum obstacle diameter constant (see network_config.h). Note that in our simulation we have two arrays, one for the left and right cameras. The end of the left array and beginning of the right array corresponding to the front of view. At the bottom of Fig. 10 you can see the mapping where the saturation of the color in the array represents the strength of the stimulus.

F. Running the visual network

We then run our inputs through the visual network described in section III-C. We do this in two passes. In the first pass we deal with activations and decays but we don’t yet add the value the activation pulse to connected neurons until the second pass over the network. Otherwise we’d be resolving the neurons at some layers with the a value that is the sum of both the value at the current and next time step. At the end of our network there are the left and right reticulo spinal neurons whose fire rates we need to map into our motion model for the lamprey.

It is difficult to actually see what is going on in the network without some form of visualization. We’ve written a visualization of the network using OpenCV as seen in Fig. 11. It shows each of the visual centers for prey, predator and obstacle, corresponding GPi, GPe and STN neurons as well as the locomotor and reticulo neurons at the bottom. It also shows the current neuron value for every neuron. With simple networks such as these a visualization of what the network is doing at each step is key to both debugging the network and also to understanding the dynamics of the network given different parameter sets.

G. Mapping to R values

Finally we have to find a way to take map the reticulo spinal neurons into our left and right amplitude values that we feed to the dual chain CPG. We do this by looking some predefined number of frames in our stream, and counting for which frames the reticulo spinal neuron fired. The resulting percentage of activations is our firing
Fig. 10. An experimental setup with left and right view points visualized. Detected blobs are circled in black and the strength of the blob is mapped into an array which is visualized at the bottom of the image. The saturation of the color in the array is proportional to the strength of the stimulus.

Fig. 11. A visualization of the network used to debug and understand the effects of parameter tuning on the visual system. A video recording of the visualizer in action along side corresponding left and right camera views can be seen at https://www.youtube.com/watch?v=I9OsfSZbMp0.

rate which we map proportionally to our CPG amplitude parameters. Note that the left fire rate needs to be mapped to the right amplitude and the right fire rate to the left. There are time delay limitations to this method. Our cameras will only allow us 30 frames per second at maximum so if our history is 90 frames long it will take 3 seconds to have an accurate reading of the firing rate of the response from the Envirobot’s position 3 seconds ago. This results in behavior like turning later than expected which causes frequent overshooting past a target. In most of our simulations we use 30 frames for a one second delay to lessen this problem, though we sacrifice resolution in our firing rate. Another solution would be to duplicate frames so that we are not limited by frame rate. This wouldn’t be completely accurate as in the ideal case we would just receive a much higher frame rate. The changes between frames at such high frame rates would be so small though that this perhaps would be an acceptable solution. This was not tested in this project.

V. EXPERIMENTS

Due to time and hardware constraints all experiments were performed via simulation in Webots 8. Our goal of the simulation is to observe the various behaviors of the robot in response to tuning network parameters. There are methods to automatically tune the many parameters of a spiking neural network such as in [5] but they require a specific fitness function and evolve the parameters via evolutionary algorithms. There is no such specific function in this project since our goal is closer to the characterization done by Braitenberg in [3] where he assigns personality traits to each of his simple robot variations. As is such we focused more on performing a wider breadth of simulations with different parameters. In this section we will show the results of each of
our experiments, for each one plotting the ground (XZ) plane position of Envirobot and obstacles throughout the course of the simulation. Simulations are not run for a constant amount of time but rather until the Envirobot reaches the target or a state where it no longer sees any targets, though they are at maximum run for 60 seconds. In all experiments Envirobot starts facing in the negative Z axis direction. In Z axis of the simulation lies on the X axis of our plots and the simulation X axis is on the plot Y axis. This is to make the view consistent with the default view in the Webots simulation. The legend used for these results is given in Fig. 12.

A. Network Size

The network size corresponds to the number of equal sized bins the visual field is split up into in topographical order. For example, with an image width of 320 and a network size of 10 we would get 10 bins each covering a column of width 32 in the image. For this test we have a single target, the prey and we adjust the network size over values from 20 to 100. Note that this value corresponds to the combined size of the left and right arrays. We plot the paths of the simulated Envirobot below for the first 15 seconds as every simulation in this experiment was run for at least that much time.

First we observe that in all cases the model correctly responds to the prey stimulus by turning towards it (remember that in every experiment the Envirobot starts off facing in the negative Z axis direction). Generally, with a higher network density Envirobot responds to stimuli much more weakly than at lower densities. This is due to the fact that with at lower densities, the same input array accumulates value from a stimulus more frequently as opposed to the stimulus being spread out over multiple bins. Therefore it activates more often eventually resulting in a higher firing rate at the end of the network. One might say that by changing the network density we are changing how lazy or aggressive a lamprey instance is in engaging in the task of catching its prey.

Note that the results do not reflect this perfectly since the 100 neuron test gets slightly farther than the 80. This is perhaps due to variations between trials in the water simulation. Admittedly more trials should have been run and averaged to get a more accurate result. However the overall trend is still visible. We choose 40 as our network size for all subsequent tests, partially because it makes the Envirobot faster and partially because it’s less computationally taxing which lets the simulation itself run faster.

We lastly also rerun this test using a stable head locomotion model. Namely given an amplitude value R, we map different R values to each motor on Envirobot such that the average amplitude remains the same but the head has the least motion while the tail has the most. There are many ways to map the change in amplitude from motor to motor along the length of Envirobot. We just use a constant difference. The results are in Fig. 14. The result is that the visual system responds too strongly, turns too fast and moves forward too quickly, resulting in a lot of wild turns, overshooting and then overcompensation of the
overshoot (much like a poorly tuned proportional controller). There are certainly ways to make the model more resistant to this erratic behavior as we will see in the next test. For the rest of our tests however we chose the original unstable head model as our current parameter set works better with it. This does not mean that the stable head model is not worth trying. In fact, if the network is tuned properly it would likely have more predictable behavior since we are removing the dynamic element of the oscillating head/cameras.

**B. Locomotion to Reticulo Response**

It at first seems intuitive how changing the activation pulse strength between the locomotion and reticulo neurons would also affect aggressive/lazy chasing since a lower activation from the locomotor neurons simply results in a lower firing rate in the reticulo neurons which means less speed. We can this experiment in Fig 15, depicting a constant run time for each simulation. However, the results here are bit less clear as the activation has also seems to have a strong effect on the resultant trajectory. For example, for the 0.25 activation actually travels the longest path in the same amount of time, meaning it is the fastest. This is a surprising result. One possible explanation is that if we look at the path we see that the Envirobot initially turns away more than the the 0.5 and 0.75 activations because the lower locomotor neuron activity means relatively more activation from the auxiliary layer and thus takes a sharper initial turn. The obstacle would then appear more to the side, resulting in a higher response layer activation and thus higher locomotor firing rate. We observe the firing rate for the 0.25 and 0.5 activation cases in Fig. 16, in which the former actually exceeds the latter. The 1.0 activation is also an interesting case as this results in saturation of the reticulo spinal neurons. In the parameter set for this experiment the activations values of the rest of the network
are also either 1.0 or relatively high. This means that usually the locomotor and auxiliary layers will send an activation pulse to the reticulo spinal neurons at the same time since they both originate from the response layer are the same number of layers away from the reticulo spinal neuron layer. If the locomotor neurons are always firing at maximum however, even if there is some induced asymmetric input, it would have no effect and trajectory would simply be straight as we observe. We can see this via the fire rate graph in Fig. 17. If the position of a stimulus stayed constant relative to the Envirobot, the speed certainly be higher as a higher firing rate in the reticulo spinal neurons would be generated. From these results we can see that there is another interesting behavior that arises out of adjusting this parameter which is that it affects how strongly the Envirobot turns to react to a stimulus. Given the same asymmetric signal, a higher locomotion activation will reduce the ratio between the larger and smaller firing rates, resulting in a straighter path. We can see that the Envirobot with 0.25 activation has a tendency to turn too much, risking missing the prey altogether. This behavior doesn’t help too much when chasing prey but one could reason that it is useful for escaping predators, making it more unpredictable by zigzagging away. It is difficult to make this test with our setup however since due to our limited field of view the predator would quickly leave the field of view of the lamprey altogether.

Based on these results we choose an activation of 0.75 to use for the rest of our experiments, which allows the Envirobot to not be too erratic in its turning while also not saturating the reticulo spinal neuron output (0.5 may have been a reasonable choice as well though it does not seem to approach as quickly).

C. Decay Strength

We experiment with the decay parameter $\tau$ of the network’s neurons. In our experiment we have one prey in front of the Envirobot. We set all neurons to have the same value of $\tau$ and test over the values 0.1, 0.05 and 0.0 (no decay). The decay parameter is another direct way to limit the fire rate of the network and thus the aggressiveness behavior. See Fig. 18 below.

![Fig. 18. Decay tests with a single prey over values 0.1, 0.05 and 0.0 applied to all neurons in the network.](image)

For decay rates of 0.05 and 0.0 the results are similar though expectedly the 0.0 result reaches the prey slightly faster. However, at a decay rate of 0.1 then network cannot keep up. The value decays so quickly that the reticulo neurons barely fire at all. During the test the reticulo neurons only fired once or twice every few seconds, just enough to create a tiny amount of motion but not enough to allow Envirobot to exhibit the periodic motion necessary to move it forward. In fact, tiny amount of motion served only to paddle the lamprey body slowly backwards. It can be surmised that the network density and decay rate are closely linked as the
0.1 may not have been too large of a decay rate if the network had a lower density and thus each neuron were activated more frequently. We choose 0.05 as our decay value for the rest of our tests as we wanted to have at least some amount of decay, otherwise we would no longer be using *Leaky Integrate and Fire* neurons as defined by the model. You can also compare the network activations via the visualizer for decay rates of 0.0 and 0.1 in the Youtube videos listed here:

- Decay Rate 0.0:  
  https://www.youtube.com/watch?v=u_t7gz0b89s
- Decay Rate 0.1:  
  https://www.youtube.com/watch?v=I90sfSZbMp0

**D. Predator Test**

With the set of parameters currently decided upon we also run the same test on a single predator to confirm that Envirobot attempts to escape as expected. In these tests we manually move the predator during the simulation to the left of the Envirobot’s head module. We can see this in figure 19.

![Envirobot trajectories on ground (XZ) plane: Predator Escape Test](image)

Fig. 19. Test predator escape given parameters set in tests from sections V-A, V-B and V-C.

As expected and the Envirobot continually tries to steer away from the predator. The initial firing rate response is higher and decreases as the predator moves away from the center of Envirobot’s vision, see Fig. 20.

![Firing Rates vs Time: Predator Escape Test](image)

Fig. 20. Fire rates from predator escape test in Fig. 19. Peaks in the graph correspond to when the predator was placed in a position that was more threatening to the Envirobot (left and more in front).

**E. Obstacle Strength tests**

In this section we run tests on obstacles. As in the predator test we use the same set of parameters as before but we also have another variable to adjust which is the strength scaling of obstacle stimuli. Obstacles need to generate lower firing rates than predators since escaping is a much stronger response than avoiding. We do this by setting the strength of the activation between the input and response layers of the obstacle visual center to values of 0.5, 0.75 and 1.0. The result can be seen in Fig. 21. To get the Envirobot to approach these obstacles we give the Envirobot an initial forward speed (settings the amplitude $R=0.25$ for both left and right sides of the CPG).

![Envirobot trajectories on ground (XZ) plane: Obstacle Avoidance Test (Input-Response Scaling)](image)

Fig. 21. Testing response to obstacles with different input to response layer scalings (0.5, 0.75, 1.0).
The result is as expected. With a smaller scaling the Envirobot turns away less than with a larger activation value. We choose the middle value, 0.75, as the activation value for future obstacle tests. Additionally for this experiment and all others involving obstacles we set the activation strength from the obstacle response layer to locomotor neurons to 0. This is because obstacles should not be producing any forward locomotion, just turning. Forward motion should only come from prey, predators or user guidance.

**F. Lateral Inhibition**

1) **Multiple Prey:** We now move to the testing the visual systems ability to prioritize between targets of the same type via lateral inhibition in the response layer. In the first test we place two prey at either side of the Envirobot with one being be slightly farther away than the other, as seen in Fig. 23.

![Fig. 23. Effect of different lateral inhibition activation values (0.5, 0.75, 1.0) on prioritizing between multiple prey.](image)

We can see that with a lower inhibition Envirobot takes a longer more arching path to get to its prey. From the point of view of the network this is simply because with less lateral inhibition there is another stimulus pulling Envirobot slightly more to the other side. By just looking at the behavior alone one might say that Envirobot is exhibiting hesitation or deliberation, that is is trying to decide between which of two the prey to chase after. With higher inhibition it takes a greedier approach, immediately going after the closest one and with lower inhibition it takes some time to decide which is a better choice. For a lamprey it seems like having a higher inhibition is more beneficial for catching dinner.

2) **Multiple Predators:** We now run a similar test but with two predators. In this experiment a first closer predator drives Envirobot towards a farther away predator. Both start well within the view of Envirobot, as seen in Fig. 23.

![Fig. 23. Effect of different lateral inhibition activation values (0.5, 0.75, 1.0) on prioritizing between multiple predators.](image)

In this experiment see that each different inhibition value generates a very different path. For a value of 0.5, Envirobot weaves in between the two predators, moving away from the first predator but also making sure to not get to close to the second. For a value of 0.75, it initially heads away from the first predator but needs to get much closer to the second predator before it decides to escape from it. For a value of 1.0, Envirobot responds very strongly to the closer predator turning so sharply away from it that the second predator ends up in the left eye’s view and it turns to the right to avoid it. Note however that it may as well have ended up in the same or worse situation as the 0.75 inhibition Envirobot and run straight into another predator if it were in its path. The only reason it managed to respond to the second predator properly was because it initially turned so sharply that it the first predator left its field of view, turning the experiment into a single predator setup. From this test we see that a lower inhibition is more helpful for the task of avoiding predator as lamprey takes
into account more threats simultaneously to decide on a safer path to avoid multiple threats.

Behaviorally, for both prey and predator we see that varying the lateral inhibition seems to change the level of deliberation that Envirobot exhibits when choosing to chase or escape multiple prey or predators. They do so however in opposite ways, where a higher inhibition means more deliberation in the prey visual center but less in the predator center and vice versa.

G. Arbitration system inhibition tests

The arbitration allows the visual system to prioritize between different types of stimulus as opposed to different stimulus of the same type. As explained in section III-B the arbitration system is structured so that there is redundancy in the network connections. In these tests we will test this redundancy. Specifically we will have two different experimental setups, one where a prey is behind a predator and another where predator is behind prey. In both we will disable parts of the arbitration system that correspond to prey inhibiting predator. We will do so by disabling the appropriate connections between STN and GPi, between STN and GPe, or both. A connection can be disabled simply by setting the activation strength to 0. Fig. 24 shows which connections need to be disabled. We show our results for the prey behind predator setup in Fig. 25 and the predator behind prey setup in Fig. 27.

In Fig. 25 the paths of Envirobot in the ‘full network’, ‘STN-GPi disabled’ and ‘STN-GPe disabled’ tests are fairly similar. The ‘full inhibition’ and ‘STN-GPi’ paths overlap each other. In the ‘STN-GPe disabled’ test, Envirobot does seem to get a bit closer to the prey though not by much. In the case where both STN-GPe and STN-GPi connections are disabled Envirobot steers clearly away from the predator as it is not being inhibited at all.

In Fig. 26 the path of Envirobot in the full net-
work, STN-GPi disabled and STN-GPe disabled tests are fairly similar as well. Again, the full inhibition and STN-GPi paths overlap each other. And again the STN-GPe disabled test does seem to differ somewhat and as it steers to the prey in a slightly larger arc. Since the predator is initially in its right camera view, if inhibition of the predator were not as strong it would make sense that the path would stray farther to the left. In the case where both STN-GPe and STN-GPi connections are disabled Envirobot is very clearly affected by the predator. It does start off by approaching the closer prey but never gets too close due to the presence of the predator driving it strongly to the left.

And again the STN-GPe disabled test does seem to differ somewhat and as it steers to the prey in a slightly larger arc. Since the predator is initially in its right camera view, if inhibition of the predator were not as strong it would make sense that the path would stray farther to the left. In the case where both STN-GPe and STN-GPi connections are disabled Envirobot is very clearly affected by the predator. It does start off by approaching the closer prey but never gets too close due to the presence of the predator driving it strongly to the left.

Fig. 27. In this setup the predator is behind the prey. We test the effect of disabling prey to predator inhibition in the STN-GPi network, the STN-GPe network and both at the same time.

From these results we can firstly confirm that the arbitration system is functional. We can also confirm that it has redundant connections. The STN-GPe connection though seems to play slightly more of a role than the STN-GPi connection since its removal results in less inhibition of the predator in both experimental setups. This may be because the STN-GPe subnetwork has two paths in which inhibition can occur. Firstly there is the STN-GPe connection in which a higher prey response means inhibition of the predator STN neurons which adds to the inhibition effect of the STN-GPi network. Additionally there are the connections between the GPe and GPi. If a GPe prey neuron does not inhibit the STN predator neuron then the STN predator neuron is free to in turn excite the GPe predator neuron more. The GPe predator neuron then also can inhibit the GPi predator neuron more which means the GPi predator neuron will not inhibit the predator response neurons, meaning a stronger response from the predator. This additional inhibition from GPe to GPi is likely what gives the STN-GPe subnetwork more influence over the arbitration system than the STN-GPi subnetwork.

H. Input Function Mapping

Another interesting area to experiment with is not a parameter of the network itself but rather how we map input signals to a strength value in the input array. Throughout our tests we have simply used a linear mapping, the diameter of a detected blob being directly proportional to its strength. However we can create many other interesting behaviors by changing this mapping.

1) Obstacles: By default obstacles strengths just increase linearly with the size but really we only want to pay attention to an obstacle if we get very close to it and ignore it otherwise. To do we calculate the new strength as a gaussian centered around the maximum strength value of 1.0, thus $N(1.0, \sigma^2)$. We can vary $\sigma$ to vary the tightness of the gaussian curve. In our experiment we set Envirobot to be moving at a forward speed of $R=0.15$ with an obstacle to the front and slightly to the right, Fig. 28.

As one might expect a higher $\sigma$ leads to the prey responding to the obstacle farther away and low $\sigma$ means Envirobot responds only when it gets closer, in a sense being more or less cautious around obstacles. This would be useful behavior for actions such as approaching a prey near a large number of obstacles directly behind it. If the prey is unobstructed we would not want Envirobot to avoid approaching it but once it neared and caught the prey we also don’t want it hitting nearby obstacles. It could also be useful in a scenario where we want Envirobot to more closely follow a wall of obstacles instead just staying far from it, as in Fig. 29.

2) Prey: We make a very simple modification to the prey’s input mapping function; we simply reverse it, setting the value to $1.0 - V$. We run a single prey test with the new inversely proportional mapping function in Fig. 30.
In this test, to really see the influence of changing the mapping we also move the prey while test is running. What we see is that as Envirobot approaches the ‘prey’ it slows down. As soon as we move the prey away, it speeds up, (faster if it is farther away) and then slows down again as it draws near. This can be seen in the graph in Fig 31 where Envirobot initially starts at rest, speeds up because the prey is farther away, slows down as it approaches and then speeds up again after we manually move the prey away.

This mapping can be understood in terms of behaviors as a form of attachment where it follows the green stimulus around and cannot stand to be too far separated from its target. Thus just by inverting the mapping function we have transformed the prey into a friend/ally.

3) Predator: As in the case of the prey we also test setting the mapping function to \(1.0 - V\). We run a single predator test with the new inversely proportional mapping in Fig. 32.

In this test the predator initially responds very strongly to a predator far away. During the
course of the test we move the predator to stay within the left camera view of Envirobot while also decreasing the distance between the two. As the predator draws near it starts to slow down and turns less violently. The corresponding speed graph can be seen in Fig. 33.

Behaviorally there are multiple ways we can understand this behavior. One could say that it is somewhat of a defeatist, giving up as soon as it sees its chances of escape dwindling. Alternatively you could say that the predator actually becomes an undesirable but also harmless acquaintance, which Envirobot initially tries hard to avoid when it sees it coming from far away, but reluctantly gives in and stops to converse when it sees it can’t avoid the encounter.

I. Complex Scenario Tests

We also ran few more complex scenario tests with many objects in the environment. They can be viewed as videos.

- **Dense Obstacle Field:** In this test we set the Envirobot to move forward with R values of 0.15. We see that it gets awfully close to the obstacles and even hits once of the obstacles at the end. [https://www.youtube.com/watch?v=OfUFW0QbzVY](https://www.youtube.com/watch?v=OfUFW0QbzVY)

- **Less Dense Obstacle Field:** The previous test was too difficult for the given parameters and initial speed of the Envirobot so we spread the obstacles out more. It does much better. One behavior that this example makes more noticeable however is that even when the obstacle is to the side of the Envirobot it keeps turning which results in the unfortunate behavior of Envirobot just bouncing around the obstacle field rather than swimming through while steering away from obstacles. We could solve this by removing the response to obstacles that are at a 90 degree angle or more from the center of the field of view. [https://www.youtube.com/watch?v=cieLAXYvYvU](https://www.youtube.com/watch?v=cieLAXYvYvU)

- **Surrounded by Predators:** In this test a ring of predators surround the Envirobot. The asymmetric signals cancel out and we are left with only forward locomotion. As Envirobot moves forwards towards the edge of the it speeds up and tries to find a gap between two predators to swim through. There isn’t anything particularly smart Envirobot can do here to survive. Accelerating straight at the edge of the ring and trying to break through it may in fact be its best option. [https://www.youtube.com/watch?v=D-hTSSKZl4](https://www.youtube.com/watch?v=D-hTSSKZl4)
- **All Target Types:** The last test is a test that uses all target types. In this test two predators drive Envirobot through a gap in wall of obstacles, on the other side of which there is a prey which Envirobot then begins to chase.

  https://www.youtube.com/watch?v=WYgg3fiN0hw

### VI. HARDWARE

We will next discuss the hardware aspect of the project. This will be an overview of the selected parts and interfaces. More detailed information and setup information can be found in the appendix and README of the *Envirovision* NanoPi project (the link to which will also be provided in the appendix).

#### A. The Main Board

We chose to use the NanoPi Neo Plus 2 primarily for its compact size (40mm x 52mm) and acceptable performance. A link to the specs of the NanoPi Neo Plus 2 can be found in the appendix. We will be using 2 USB and 1 UART for communication which corresponds to the cameras and IMU respectively. The operating system, Ubuntu16 Core, is flashed directly onto the eMMC (8GB on board storage) using the eFlasher tool. Details are provided with the NanoPi installation instructions.

#### B. The Camera

We used the FA-CAM 202 which was built to be interfaced with the NanoPi. It is a cheap light weight camera with a 1/4 inch sensor. The default lens is rectilinear with a very limited field of view. We will use a 1/4 inch 3 MP fish-eye lens with a 185 degree field of view instead. Separate lens bases need to be purchased as well to properly focus the lens on the sensor. The camera with fish eye lens attachment can be seen below in Fig. 34. Communication happens via the USB ports.

We use OpenCV to read in data from the cameras and the CSCore library to stream the data to a local http server for debugging purposes. The camera can output images at a resolution of 640x480 at 30 frames per second. Settings for the camera are also adjusted via the CSCore library. The CSCore library works well for streaming frames and setting the camera settings but has trouble capturing images from the FA CAM202 which is why we don’t just use it for the all of our camera interfacing tasks. However if we initially connect to the camera to set parameters OpenCV cannot read the camera output unless we forcibly disconnect the camera in software from CSCore. The original CSCore library has been modified to expose API to do this forced disconnection thus it is necessary to use our version of the library is provided in the CSCore_Envirovision project. Other available parameters besides frame rate and resolution are listed in the appendix.

It is important to turn off automatic adjustments (auto exposure and auto white balance) so that the image remains consistent in the same environment. To detect the red, green and blue target it helps to bring the saturation level up somewhat, a value of 40 works well. While this results in red and green appearing well, the camera has a tough time with blues which appear to have a lot of extra green, making them appear closer to turquoise. We were unable to tune the parameters to get lessen this issue. White balance adjustments only make the image more red or more blue but cannot decrease the green component. This may be a limitation of the sensor itself or some lower level post processing that we have no control over. We could decrease the amount of green with hue settings but this negatively affected detection of red and blue colors too much.

Lastly, the camera needs to be calibrated to
adjust for fish eye distortion, this can be done using the CameraCalibrate project in the Environment_NanoPi repository. Details on usage are given in the README. Unlike the simulation, our images from are not entirely centered so we need to some realignment and cropping of the frames post calibration to get the desired image. In Fig. 35 you can see raw camera views that need to be calibrated and cropped.

Fig. 35. Uncalibrated from the two FA CAM-202 with fish-eye lens mounted on the head module.

C. The IMU

The IMU used is the XSens MTi-1s-Dev. It is a 9-Axis IMU which can output absolute orientation values. The IMU needs to be configured over MT Manager (I did this on Windows) to output the correct set of values, in our case just the rotation angles. The pin configuration on the dev board should be configured as in Fig. 36 for UART communication. The baud rate should be configured to run at 115200 kbps. The wiring between the NanoPi and IMU is quite simple, ground to ground, VDD to 3.3V, RX to TX, TX to RX. Note that we use the UART1 ports on the NanoPi. The UART ports on the IMU are clearly labeled on the board itself. A diagram of the wiring between the IMU and NanoPi can be seen in Fig. 37. Communication is done via the wiringNP library which is an API provided by FriendlyArm to simplify communication with devices from the NanoPi.

On the firmware side we need to process the output of the IMU. The measurement output follows a specific format given in the MT Low Level Communication Protocol documents linked to in the appendix. This need to be properly parsed from hexadecimal to determine the correct floating point angle values. The implementation is done in the main controller provided in envirovision.cpp

in the Environment_NanoPi project. To retrieve the IMU values associated with each camera frame, we let the main run loop be controlled by OpenCV’s frame capture loop. The IMU data capture happens on a separate thread which modifies angle angle variables shared with the main camera thread. A mutex is used to lock reading and writing of these variables so that they are set all at once and so that we cannot run into situations where we read the roll from one measurement but the pitch from another.
D. New Head Module

A new head module was designed by Mehmet Mutlu to house the above hardware. The module can easily be attached and detached from the main body of the envirobot. It has opening for 3 cameras, two of which are mounted on the side but facing inwards by 20 degrees and the other which faces straight down. The design can be seen in Fig. 38 and a prototype with cameras attached can be seen in Fig. 39. Fig. 40 shows the full hardware setup.

E. Frame Rate Tests

While we use a resolution of 640x480 from the camera we only need to process the visual network on the central 640x240 of each image, assuming the horizon line on the water always appears somewhat centrally in the image. From a max of 30 fps, camera rotation only decreases FPS by about 2 but undistortion by about 8. As a rough estimate based on simulation performance, the visual network processing brings the frame rate down another 5 seconds to a final value of 15 FPS, which is acceptable. If we want more performance we could decrease the resolution to 320x240 (though the camera will still only output frames at 30 fps).

VII. CONCLUSIONS

To summarize what we have achieved with this project is to create a vision system for Envirobot using two wide angles cameras mounted on a replaceable head module for Envirobot. The system will allow for live experimentation with the computational visual models such as the model we have chosen. It will also pave the way for any other visual guidance system that will surely benefit from the calibration and image stabilization systems as well as the Wifi streaming setup which will be useful for debugging. Indeed gaining visual input is a major enhancement to Envirobot’s ability to carry out search and surveying tasks.

The other major part of this project was the implementation the computational visual system of [11] in simulation. We ran a multitude
of experiments with said simulation to ascertain the potential of the spiking neural network based model to produce interesting behaviors. What we found that there are a plethora of parameters to tune and adjust and that small changes in these parameters can lead to very different behaviors. Just as in Valentino Braitenberg’s text[3] we can consider these different behaviors to be different personalities or inclinations such as, to name a few, deliberation, aggressiveness, attachment or defeatism. An interesting parallel can be drawn to an actual population of lamprey, where different lamprey may have different responses to the same stimulus as a result of learning from different experience or genetic mutations. From the viewpoint of our visual model we can understand the variety of behaviors as a result of the parameters of each lamprey’s neural system being perturbed in some way. It would be interesting to observe a population of simulated Envirobots each with different spiking neural network parameters all responding to the same stimuli.

To conclude, I believe that this model would certainly be interesting for demo purposes of the visual capabilities of Envirobot. If an object detection system is added on top that could classify targets as either prey, predator or obstacle it could also be applied to real life situations. However, in terms of use in an actual outdoor environment, given the large number of parameters and their sensitivity it would be difficult to create a generalized model that could apply to many different situations unless the parameters themselves could be adapted and changed in real time based on the environment. This would be a very complex task, perhaps requiring the aid of a separate deep neural network to learn the proper mapping between visual input and parameter values. For example, it could learn to use parameters that result in less aggressive behavior while catching prey if it sees that it is surrounded by a dense field of obstacles to lower the probability of collisions. A neural network however requires a GPU to be run in real time, which the NanoPi does not have.

VIII. FUTURE WORK

A. Simulation

There are also a number of items that would be interesting to look into in the future. With regards to our simulation there are two items which were mentioned previously that were not experimented with. The first is to use neuronal populations instead of single neurons in the spiking network. While this might be too computationally intensive for the real robot, in simulation it would be interesting to see what the effect would be of having even a small group of say five neurons versus just a single one.

In our experiments we modified parameters by hand just to observe the effect that each parameter had on Envirobot’s behavior. In the future, if we have a more goal oriented task it would beneficial to employ some sort of automated network parameter tuning system such as using training examples as in [4] or evolutionary algorithms as in [1] and [5].

There are also two previously discussed modifications to the current simulation that would be worth trying. The first would be to use the duplicating frames method described in section IV-G to fix the issue with the time delay in response to new stimuli. The second is to try tuning the network parameters to work with a stable head model as in the experiment in section V-A.

B. Hardware Tests

In terms of hardware, the first thing to do would be to actually run the visual network on a completed Envirobot as we did not have time to do so during the project. As of now our hardware tests only output the left and right firing rates and are not at all connected to the actual Envirobot’s CPG. Doing hardware tests may reveal other practical issues that may interfere with the performance of the visual system.

For example, while we have most corrected the rotational turbulence thanks to the IMU, we have not addressed any form of translational jittering which also may result from turbulence in the water or in the motors themselves. It may suffice to use techniques such as stabilization via optical flow as in [12] to help reduce this, especially if we know that we only have to focus on translational jitter.
The performance cost of running these stabilization techniques along with all of the other visual processing we do on the nano pi however remains to be seen though the method in [12] claims real time performance.

C. Alternate Localization Techniques

With our vision system there is an overlap in the field of view at the front of the robot of about 45 degrees. This can be used to create some form of stereo depth estimate from the rectified images. One concern is that when creating dense depth estimates from fish-eye lenses, the accuracy of the depth map decreases as we approach the edges of the frame due to distortion issues [14]. In our case we’d only be using the areas near the edges for depth reconstruction so this could be an issue. However, depending on the application even a fuzzy estimate of depth may be helpful to have.

Lastly the construction of the new head module for a downward facing camera which allows for alternate solutions for localization. For experimental purposes where testing is done in a pool, one suggestion is to implement the approach used in [9], employing an anoto dot pattern on the bottom of the pool which the bottom facing camera can process and determine a pair of absolute coordinates from. While not in any way a biologically inspired approach it could be useful as a replacement for the LED detection method that the current Envirobot pool setup uses for measuring position.

IX. ACKNOWLEDGEMENTS

Thanks to Mehmet Mutlu and Behzad Bayat for supervising me during this project and for helping me with all of my hardware issues. Also thanks to Alessandro Crespi for fixing my wifi access on the nanopi.

APPENDIX

A. Git Repositories

- Webots 8 Simulation: https://c4science.ch/source/envirovision_simulation.git
  Note: The README file contains a lot of useful information on the installation and usage of these libraries. It is somewhat lengthy and much of it pertains to specifics of the and terminal commands so I will not include all of its contents in the appendix here.
- CSCore Libs needed for NanoPI implementation: https://c4science.ch/diffusion/5709/cscorelibs_envirovision.git
- WiringPI: git@github.com:friendlyarm/WiringNP.git

B. Available Camera Parameters for the FA CAM-202

- raw_brightness (int):
  value=-50 min=-127 max=127 step=1 default=0
- brightness (int):
  value=30 min=0 max=100 step=1 default=50
- raw_contrast (int):
  value=64 min=0 max=127 step=1 default=64
- contrast (int):
  value=50 min=0 max=100 step=1 default=50
- raw_saturation (int):
  value=64 min=0 max=255 step=1 default=64
- saturation (int):
  value=25 min=0 max=100 step=1 default=25
- raw_hue (int):
  value=0 min=-16000 max=16000 step=1 default=0
- hue (int):
  value=50 min=0 max=100 step=1 default=50
- white_balance_temperature_auto (bool):
  value=1 default=1
- gamma (int):
• value=100 min=16 max=500 step=1 default=100
  • power_line_frequency (enum):
    value=1
    0: Disabled
    1: 50 Hz
    2: 60 Hz
  • white_balance_temperature (int):
    value=5200 min=2800 max=6500 step=1 default=5200
  • raw_sharpness (int):
    value=4 min=0 max=31 step=1 default=4
  • sharpness (int):
    value=12 min=0 max=100 step=1 default=12
  • exposure_auto (enum):
    value=3
    1: Manual Mode
    3: Aperture Priority Mode
  • raw_exposure_absolute (int):
    value=625 min=1 max=5000 step=1 default=625
  • exposure_absolute (int):
    value=12 min=0 max=100 step=1 default=12
  • exposure_auto_priority (bool):
    value=0 default=0

C. Hardware Datasheets/Purchase Sites

• MTi-1s-Dev datasheet:
• MT Low level communication protocol:
• MT-Manager install:
  https://www.xsens.com/mt-software-suite/?locale=en
• NanoPi Neo Plus 2 Schematic:
• NanoPi guide and instructions:
• FA CAM-202:
• 185 degree Fisheye lens:
• M12 Lens Base:

REFERENCES


[16] Rodrigues’ rotation formula. (2017, December 27). In Wikipedia, The Free Encyclopedia. Retrieved 11:06, January 12, 2018, from https://en.wikipedia.org/w/index.php?title=unhbox\voidb@x\bgrouplet\unhbox\voidb@x\setbox\@tempboxa\hbox{R\global\mathchardef\accent@spacefactor\spacefactor22R}\egroup\spacefactor\accent@spacefactorodrigues\%27_rotation_formula&oldid=unhbox\voidb@x\bgrouplet\unhbox\voidb@x\setbox\@tempboxa\hbox{8\global\mathchardef\accent@spacefactor\spacefactor17254383}