Implementation of attentional bistability of the dragonfly visual neurons in an intelligent biomimetic agent
— Report Two —

Juan Carlos Farah, Panagiotis Almpouras, Ioannis Kasidakis, Erik Grabljevec, Christos Kaplanis
{jcf214, pa512, ik311, eg1114, ck2714}@doc.ic.ac.uk

Supervisors: Professor Murray Shanahan, Zafeirios Fountas, Pedro Mediano
Course: CO530/533, Imperial College London

12th March, 2015

1 Challenges and Motivation for New Specification

In this section, we motivate our change in specification by highlighting the biggest challenges that we have encountered in our project.

Our most significant challenge has been to generate the output we expect from the CSTMD1 neurons, given an input from the initial visual processing (the ESTMDs). In part (iii) of Stage 1A as specified in Report 1, we expected to observe evidence that the CSTMD1 selects one target between two that are presented in the visual receptive field. The CSTMD1 model is not currently displaying this selectivity as shown in Figure 1. What we expected was that the firing rate graph for both targets would emulate that of one of the two targets, but instead it usually just fire more when both targets are presented.

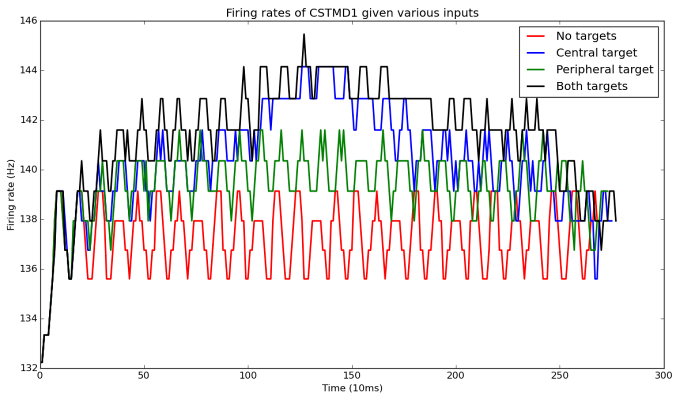


Figure 1: Firing rates of CSTMD1

Given the complicate nature of the morphologically modelled CSTMD1 (which is largely third party code) and the fact that this target selection has never been shown before in a CSTMD1 model, we are concerned that we may not be able to reproduce this phenomenon shown by biological CSTMD1 neurons within the time constraints of this project. While it is still our goal to do so, we are moving this part of the specification out of minimum requirements and into possible extensions. In light of this, we thought that what might be useful instead is to create a webtool that enables the user to:

1. Modify key parameters in each of the components of our dragonfly visual system.

2. Run each component individually and display key metrics demonstrating the functionality of each component.
3. Run the components in unison and display key metrics demonstrating the functionality of the system as a whole.

This would enable us, or an external user, to efficiently investigate the properties of our dragonfly visual system and better tune the parameters in order to achieve target selection and prey capture in our model.

2 New Specification

Below we layout our new requirements and the stage of completion for each part:

Minimum Requirements (Stage 1)	**Completion**
(A)(i) Create an animation tool to create inputs for the visual processing	Full
(A)(ii) Build a model of the visual processing (ESTMD) that occurs between the retina and the actual CSTMD1 neurons of a real dragonfly	Full
(A)(iii) Decide how many CSTMD1 neurons we will use and how exactly to connect them to the output of our visual processing	Full
(B) Build a layer of pattern recognition neurons that can learn to recognise spike patterns within a noisy input	Full
(C) Integrate the visual processing and pattern recognition system to detect patterns within the CSTMD output and add a simple action-selection mechanism.	Partial

Expected Implementation (Stage 2)	**Completion**
(A) Develop webtool to analyse metrics of each component of the dragonfly visual system	Partial
(B) Create a virtual 3D environment for the dragonfly agent	None
(C) Enhance the action selection mechanism to control the agent within the environment	None

Possible Extensions (Stage 3)	**Completion**
(A) Succeed in getting the CSTMD1 to exhibit target selection by changing parameters and the connection with the ESTMD	Partial
(B) Improve usability and features of webtool.	None
(C) Implement the agent in a quadcopter drone	None

3 Progress

1. Animation tool. The animation tool, created using Pyglet, gives the user the option to create a video of black targets moving across a custom background that is either stationary or moving. The size and velocity vector of each target is adjustable.
2. Visual processing. We successfully implemented a model for ESTMD (elementary small-target-motion detectors) based on [1]. The model can detect small-target motion across a moving, complicated background. This stage required a lot of research and understanding of spatio-temporal filters that approximate the function of real ESTMD neurons. The input of the ESTMD can either be a full video or frame by frame as it is produced by the animation tool. The output of the ESTMD model is a time series of matrices of processed pixels, which can be viewed in a video. See screenshots of videos in Appendix A. This output is connected to the CSTMD neurons by converting each pixel into a firing rate for a simple integrate-and-fire neuron and connecting the output of each of these neurons to the CSTMDs, biasing the weights of the centre of the visual input with a Gaussian distribution.

3. Pattern recognition. To model the pattern-recognition neurons needed for our project, we initially replicated experiments conducted by T. Masquelier et al. (Citation Needed). The resulting code was a Python module for Spike Response Model (SRM) Leaky Integrate-and-Fire (LIF) neurons that successfully recognised input patterns based on the samples described in the respective papers. A single of these neurons is able to successfully recognise a recurring pattern within background noise and a network of them is able to do so for multiple patterns. We then extended the module so that the neurons can be easily adapted to recognise input with varying properties such as average firing rate, number of afferents, frequency with which the pattern appears, amongst others. This implementation is able to recognise patterns output from our CSTMD1 neurons and measures the effectiveness of the pattern-recognition neurons by tracking key information such as true-positive, false-positive and true negative spike incidences.

4. Web client. The web client is designed to be a simple interface through which simulations for each of the modules can be run and automated, both separately and jointly. We have currently implemented a prototype using Bottle and MongoDB of the interface for the pattern-recognition module. This graphical user interface provides the minimal functionality needed to create sample spike trains, test pattern-recognition neurons against them and save the results of each experiment, providing key insight to the effect of each parameter on the output.

4 Updated Task Scheduling

Below is a condensed version of our updated schedule.

4.1 Stage 1C: Connecting CSTMD to pattern recognition

Task	**Priority**	**Sprint**
Get pattern recognition to recognize small patterns from CSTMD output	1	3/4
Get second layer of pattern recognition neurons to recognize longer patterns	1	3/4
Create simple action selection mechanism.	1	4

4.2 Stage 2: Webtool and Enhanced Action selection

Task	**Priority**	**Sprint**
Define range of actions available to biomimetic agent	2	4
Design and implement basic environment for simulated agent	2	4/5
Implement adapted Braitenberg vehicle as action selection mechanism	2	4/5
Parametrize components of visual system for integration into webtool	2	4
Integrate classes of components into webtool	2	4/5
Implement useful metrics for each component in webtool	2	4/5

4.3 Stage 3: Target selection and improve webtool

Task	**Priority**	**Sprint**
Get CSTMD to select between targets	3	3-6
Implement agent in a quad-copter drone	4	6/7

5 Testing

5.1 Methodology

Our testing strategy has evolved from simple "on the run manual testing" to full systematic unit testing of our entire codebase. We followed our project's modular and class-based architecture while designing our testing structure. Each class has a corresponding test class, whose test methods aim to mirror the methods of the class it is testing. However, it is possible that several test methods cover different parts of the same method, or that in turn one test method covers several methods. Following the white-box testing strategy, each test class is designed by the team members who wrote the original code and who therefore have the best most insight regarding the expected behaviour of the original class. Currently we have focused our testing on covering as many branches as possible, each of which is analysed for the different possible behaviours it may have. We are measuring our code coverage in percentages of the lines covered.

As we progress with our development we aim to augment our testing strategy to include component, integration and system tests. Component tests will cover end-to-end cases for each of our modules, while integration tests will cover the connections between each of these components. Finally, system tests will cover the complete project, when all modules have been connected between themselves and the web client.

5.2 Implementation

Given that all of our code is written in Python, we use Python's unittest framework. To measure code coverage we are using Python's coverage library, which allows us to run tests on each file separately and combine results to create comprehensive reports.

5.3 Results

Results from our latest test run covering all of our modules can be found in Appendix A. Average test coverage is COVERAGE.

5.4 Challenges

At this stage, some of the functionalities we are developing have not been factored out optimally, which causes some methods to be very tightly interlinked. This in turn requires more complex tests, some of which will need extensive modification as we restructure the code. We have thus decided to omit some of these methods from our current tests. As we gain more experience with unit testing and refine our design, these methods will be covered.

We also found it difficult to test the graphical output that our modules produce. While testing it manually is very straightforward, we encountered problems while writing tests to do so systematically. One solution we employed was to store images of the graphics and check the values of specific pixels against the expected ones, which we defined beforehand. This approach becomes more complicated when random elements are involved and when the result can vary. We currently handle this by setting specific seeds for the pseudorandom number generators and allowing some error in the test calculations. However, we are still working to make these tests more resilient.

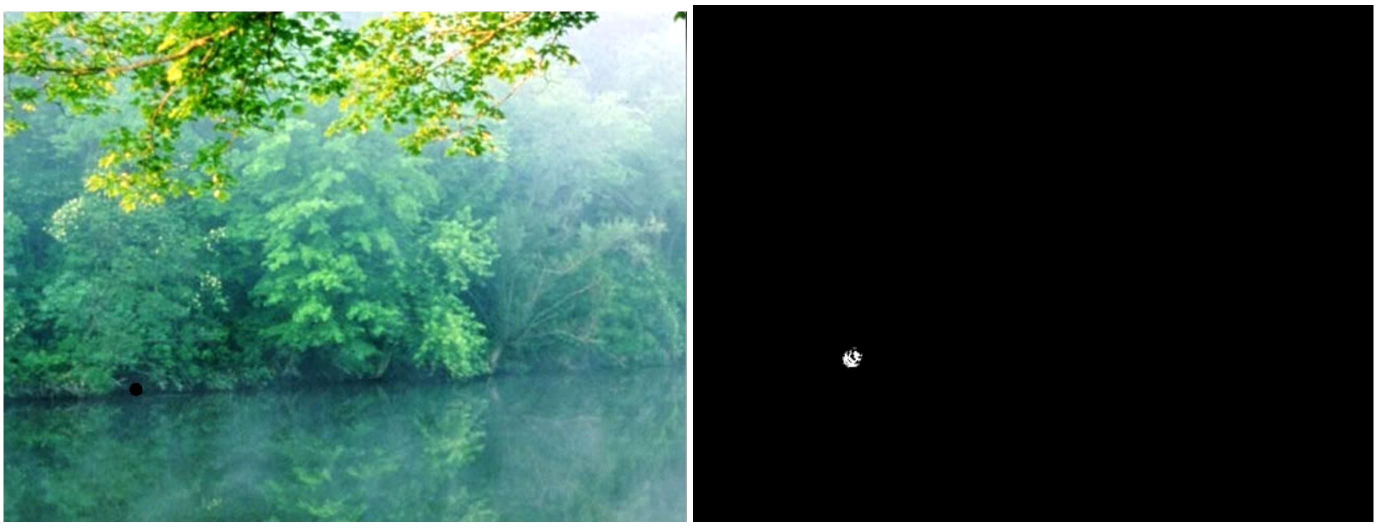
References

[1] K.J. Halupka, S.D. Wiederman, B.S. Cazzolato, and D.C. O'Carroll. Discrete implementation of biologically inspired image processing for target detection . In *ISSNIP*, 2011.

Appendices

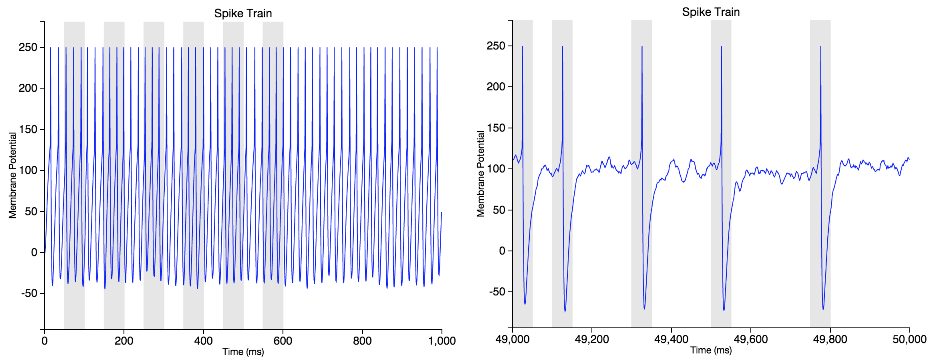
Appendix A Animation / ESTMD screenshots

Below are screenshots of an animation of a small target moving diagonally across a moving complicated background (left) and a screenshot of the corresponding output of the ESTMD. Despite the moving background, the ESTMD highlights the small moving target.

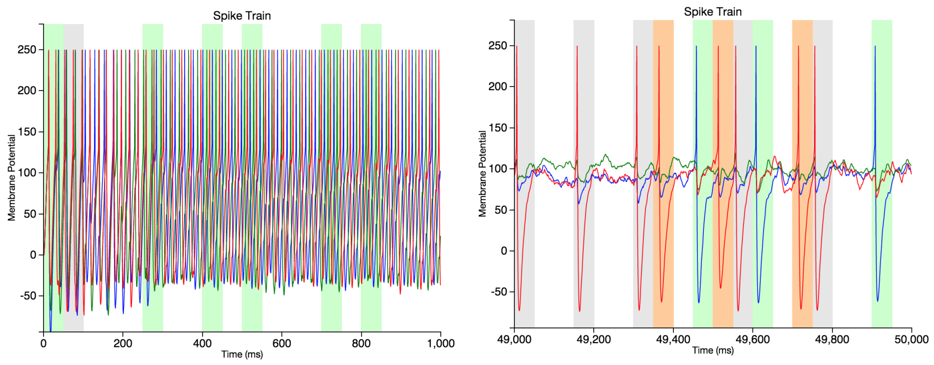


Appendix B Pattern recognition screenshots

Single neuron



Multiple neurons



Appendix C Test Results

C.1 Overview

```
$ coverage report -m

Name                 Stmts   Miss   Branch   BrMiss   Cover   Missing
-------------------------------------------------------------------------------------
estmd                 114    0       16       1        99%
cstmd                 380    84      130      44       75%    213, 218, 317-401,
                                                              487-488, 513-514,
                                                              600-603, 638,
                                                              642, 647-648, 662
target_animation      125    0       20       1        99%
neuron                248    59      72       32       72%    170-179, 197, 209-212,
                                                              216, 230-233, 237, 261,
                                                              300, 331, 357, 368-381,
                                                              388-393, 417-420,
                                                              447-450, 488, 562-563,
                                                              570-579
sample                127    0       32       0       100%
simulation            109    35      30       18       62%    88-103, 133, 144-146,
                                                              159-162, 175, 202-232
sample_dao            52     8       10       2        84%    78, 113-120
server                80     19      6        3        74%    44-51, 59, 79, 93-100,
                                                              146, 160-161
simulation_dao        154    18      46       14       84%    34-35, 88, 118, 160-161,
                                                              239-247, 259-260, 285
-------------------------------------------------------------------------------------
TOTAL                 1389   223     362      115      81%
```

C.2 Individual Tests

C.2.1 Visual Pre-processing Module

ESTMD

```
$ coverage run --branch test_estmd.py
....
----------------------------------------------------------------------
Ran 4 tests in 3.845s

OK
```

Target Animation

```
coverage run --branch test_target_animation.py
...........
----------------------------------------------------------------------
Ran 11 tests in 2.615s

OK
```

C.2.2 Dragonfly Neuron Module

CSTMD

```
coverage run --branch test_cstmd.py
..
----------------------------------------------------------------------
Ran 3 tests in 64.135s

OK
```

C.2.3 Pattern Recognition Module

Neuron

```
coverage run --branch test_neuron.py
................
----------------------------------------------------------------------
Ran 16 tests in 0.197s

OK
```

Sample

```
coverage run --branch test_sample.py
.
----------------------------------------------------------------------
Ran 6 tests in 39.788s

OK
```

Simulation

```
coverage run --branch test_simulation.py
..
----------------------------------------------------------------------
Ran 2 tests in 1.080s

OK
```

C.2.4 Web Client Module

Sample Data Access Object

```
coverage run --branch test_sample_dao.py
.
----------------------------------------------------------------------
Ran 1 test in 7.838s

OK
```

Server

```
coverage run --branch test_server.py
...
----------------------------------------------------------------------
Ran 8 tests in 5.172s
```

```
OK
```

Simulation Data Access Object

```
coverage run --branch test_simulation_dao.py
----------------------------------------------------------------------
Ran 1 test in 5.211s

OK
```