# Dragonfly Neural Simulation
## — Report Two —

Alex Carver, Chris Snowden, Desy Kristianti,
Georgios Kontogiannis, Luka Milic, Zoe Landgraf
{anc15, cps15, dk2015, gk513, lm1015, zl4215}@ic.ac.uk

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

11th March, 2016

## 1 Introduction

The project has progressed and we now have prototypes of the core components completed as well as a containing framework structure (the dragonfly-brain class) for the entire model, which each component can be integrated into.

To ensure efficiency, utility and functionality of our code we have conducted unit and coverage tests for the functions of each component. Alongside ensuring functionality with unit tests, we have also conducted time profiling upon our code to determine how performance bottlenecks that would inhibit near real time execution could be removed or improved. Although functionality has been achieved for the core modules (see Figure 1), these tests revealed performance may inhibit real time execution.

In the process of developing these components we have come across several challenges that we have had to respond to. In addition, certain tasks included in our original specification no longer appear to be either feasible because of encountered limitations, or are no longer relevant as they have been found to be unnecessary to achieve the core goals of the project. Because of our use of the Agile development method, we have been able to adapt our development strategy to respond to these issues, and have recomposed our schedule and specification as a result.

We will first discuss progress made for each individual component, followed by the revised task schedule and finally the testing methodology for each component accompanied by test results.

## 2 Progress Update

### 2.1 Environment

The environment builds upon the *Target Animation* module developed by the previous group [1]. The code has been refactored and extended to abstract the animation process away from the environment state (e.g the dragonfly velocity). However it still enables the animation of the dragonfly's prey (targets, represented by black dots) that either move randomly around a fixed point or with a constant velocity.

Many of the functions have been rewritten to increase efficiency and to suit the holistic integrated design. This includes the addition of a step function that updates the dragonfly's and preys' positions

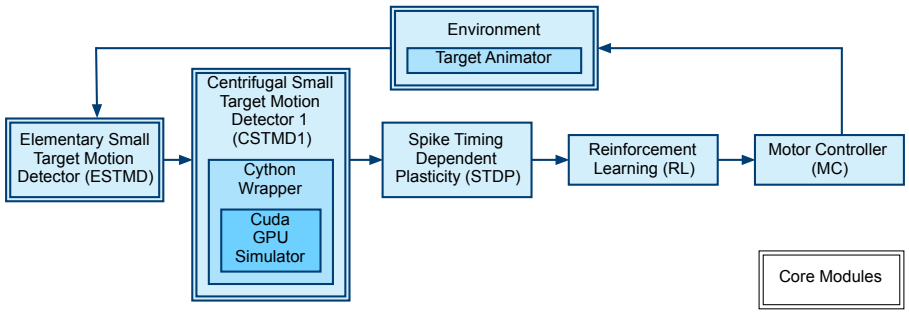


Figure 1: Project Structure

given a specified velocity (from the Motor module) and outputs an array at each time-step. This will be a more efficient input to the ESTMD module compared with the encoding and subsequent decoding of a video file, which was done before.

Due to the need for graphics dependencies we have wrapped a working version of *Target Animation* as a Docker image, allowing it to be run with all dependencies pre-installed.

## 2.2 ESTMD

The ESTMD has been developed to integrate into the planned continuous flow model with the creation of a step method. This will allow it to process a time step given a frame matrix input from the Environment module, detect movement in the frame matrix and pass that detected movement to the CSTMD1. As a result the ESTMD is ready to be integrated into the model framework alongside the Environment and CSTMD1. We also developed the option for the ESTMD to output its detected movement as videos to allow for quick manual comparison and verifcation alongside the source videos.

However, whilst testing the ESTMD, we found that whilst fully functional its performance was below the desired speed to achieve near real time results. After investigation (see Appendix C: ESTMD Profiling) we found this was due to the high resolution of the input frames. Downsizing the resolution of the frames prior to processing significantly reduced processing time, but at the cost of detection accuracy. We will experiment to find an acceptable compromise between speed and accuracy.

## 2.3 CSTMD1

A C++/Cuda simulator, which has been wrapped in Cython, has been developed that can model several mutually inhibiting multi-compartmental neurons using the Hodgkin-Huxley mathematical. The simulator successfully models spike propagation from the dendritic tree [2] to the soma although it can overflow due to the model's sensitivity to its time step.

Simultaneously modelling several thousand differential equations has a significant computation overhead. Therefore if the simulation pipeline is to run near real-time (requirement B5 Table 2) the performance of the CSTMD1 simulator is critical. Therefore NVIDIA libraries were used including cuBLAS which contains highly optimised linear algebra subroutines. Table 1 shows the current average computation of the simulator for varying numbers of compartments and electrodes. There is small increase in GPU computation time (a primary reason for using the CUDA platform) and a large increase in overall execution time attributable to the relatively slow process of copying data to and from GPU memory. As such only the minimal amount of data will be copied back onto host memory.

In order to better understand the module's characteristics (see requirement B1, Table 2), a "Neuron Visualiser" (Figure 2) has been developed which generates a 3D model of a collection of neurons and animates the propagating spikes. This will help in observing the effect of the morphology and inhibition on the performance of the simulator.

| Compartments | Electrodes | Avg GPU Computation /ms | Avg Total Time /ms |
|---|---|---|---|
| 10 | 4 | 1.17 | 2.87 |
| 2000 | 35 | 1.25 | 26.2 |
| 7500 | 50 | 1.30 | 41.2 |

Table 1: Time to simulate 1ms. *Avg GPU Computation* is the time to simulate 1ms in pure GPU computation, *Avg Total-Time* is the time to execute one step from the Python wrapper and includes the overhead of retrieving the electrode data from the GPU memory and transforming it into a Python data structure (GPU time step = 0.05ms)

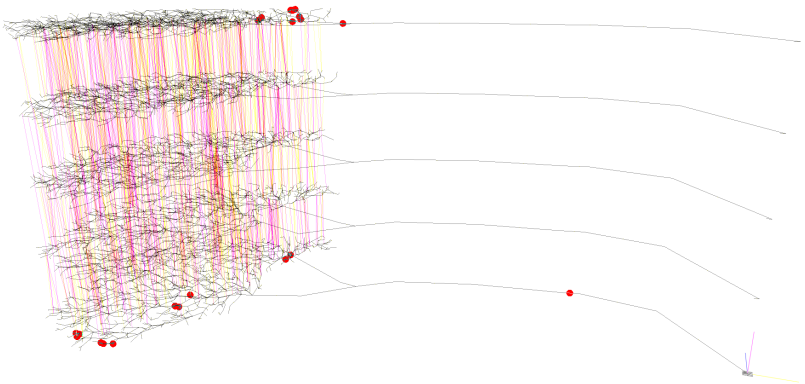

Figure 2: CSTMD1 network, 5 neurons, 7500 compartments viewable using the Neuron Visualiser

We will conduct model verification in order to tune the parameters to observe the bi-stability required for successful target selection. It is also possible to enrich the neuron model with individual compartment parameters and a topological mapping of the ESTMD stimulus, however the group's focus will remain on integration and model verification.

### 2.4 STDP

The pattern recognition module using Spike Time Dependent Plasticity has been adjusted to allow for integration into the sequential process of the dragonfly brain simulation. The module now accepts spike information of one time step from the CSTMD1 and delivers output in the form of a spike array corresponding to the spiking of one of the module's four recognition neurons. The module can be run in a training state, during which it performs STDP on its synapses or be used for pattern recognition, after it has been trained.

The module has been observed to learn and adjust to a trial pattern, however, performance decays over time. In order to adjust this, parameters will have to be tuned and further testing will be required.

### 2.5 RL

As the working of the Reinforcement Learning module depends upon the output of the fully integrated prototype of the preceding components, integration of RL into the dragonfly brain class, which had been of secondary importance so far, can now be undertaken

### 2.6 Motor

The initial motor module has been completed which takes a discrete action and maps it consistently onto a movement direction which it then passes to the Environment. Whilst dragonfly motor control

could be the subject of further research, it was deemed beyond the scope of the project and the controller currently restricts movement to $2n$ directions in $n$-dimensional space.

# 3 Specification and Task Schedule Update

The project's specification was updated according to current progress on the modules, unit tests and time profiling. The updated requirements and their phase status are displayed in Table 2.

Table 2: Updated specification: Complete(C), Testing(T), Outstanding(O)

| | | **Task** | **Phase** |
|---|---|---|---|
| Minimal | A1 | Develop an efficient implementation of a multi-compartmental Hodgkin-Huxley neuron model | C |
| | A2 | Apply this model to create a CSTMD1 simulation | C |
| | A3 | Integration of ESTMD, STPD and RL with the CSTMD1 neuron(s) into a continuous flow simulation | T |
| | A4 | 2D environment in which the agent will attempt to capture prey | C |
| | A5 | Implement and integrate a motor module which moves the dragonfly within the environment and automatic system | T |
| Full | B1 | Providing a transparent interface in order to understand the characteristics of each module | O |
| | B2 | Create and display a visualisation of agent's neural behaviour | T |
| | B3 | The CSTMD1 module exhibiting all theoretically expected features | O |
| | B4 | Comparison of our own simulation with $3^{rd}$ party simulators including NEURON | O |
| | B5 | The connected modules should be able to run in an approximation of real-time | T |
| Extensions | C1 | Create a user interface for the virtual simulation environment | O |
| | C2 | Implement with real-time input from camera | O |
| | C3 | Extend the environment and motor controller to improve realism | O |
| | C4 | Implement on a simple physical robot or quad-copter | O |

Seven sprints were planned for this project, with tasks having a priority from 1 (highest) to 3 (lowest). By the end of Sprint 3, the Environment, CSTMD1, ESTMD and STDP modules are functional. Some work to integrate the modules has been done although the team has been focusing with the individual performance of the modules. Overall we are still within range of the initial goal of having a fully functional integrated system by the end of the fifth sprint. After reviewing the updated specification and task requirements, an updated task schedule has been generated and is given in Appendix A).

# 4 Testing

## 4.1 Strategy

As our project has a clearly predefined framework with set inputs and outputs for each module, testing is relatively simple. Each module has its own test suite with a set of test data, which will also be created for the fully integrated framework. In addition our use of Object-Orientated design ensures we can and have tested lower-level code such as the Environment's *Target* or the CSTMD1's *Compartment* classes, allowing us increased confidence in the robustness of the system.

To achieve the goal of near real time performance we will identify bottlenecks using profiling tools and where possible remove them through code optimisation. If robust and high coverage tests are frequently used, we can be more confident that future optimisations will not introduce any problems with the component functionality.

However whilst unit and coverage testing will ensure the code is functional, it does not ensure it

accurately performs to our desired model. Now that the core components have been integrated, we have begun the process of developing model verification tests for this purpose. This is necessary as many aspects of the components rely upon various parameters, the values of which need to be determined and verified through experimentation, and their reasonable limits with stress testing.

Once all components have been tested independently and as an integrated framework, and the model has been verified we can then perform rigorous system testing as well as start developing potential extensions.

### 4.2 Implementation

We chose to use the PyUnit framework, which is the standard for Python [3], as nearly all our code is in Python and it allows us to re-run the tests for modules we inherited from the previous group (*neuron.py*, *sample.py* and *simulation.py* Appendix B). The CUDA library we have written has been wrapped using Cython and hence we are also able to test this with PyUnit, hence allowing for a uniform testing methodology. We use the package *Coverage.py* to measure the statement and branch coverage of the code and the package *Mock* to ensure we can test parts of our code which generate graphical output. A generic test suite template has been constructed enabling a consistent testing setup for each module and for independent testing. A top level "Dragonfly Test Suite" can then easily import each module's respective test suites allowing for quick project wide testing.

### 4.3 Results

We have tested all modules apart from the Reinforcement Learning module (see Section 2.5) to achieve a total coverage average of 90%. Note that the average is brought down by the coverage of last year's code that was used. We will attempt to increase this coverage in future testing. A full exposition is available in Appendix B.

## References

[1] J.C. Farah, P. Almpouras, I. Kasidakis, E. Grabljevec, and C. Kaplanis. Software from previous group project. `https://github.com/juancarlosfarah/anisopter`, May 2015.

[2] Byron Galbraith. Neural modelling with python. `http://www.neurdon.com/2011/02/11/neural-modeling-with-python-part-4/`, February 2011.

[3] Python Software Foundation. unittest unit testing framework. `https://docs.python.org/2/library/unittest.html`, January 2016.

# Appendix

## A Updated Task Schedule

Table 3: Revised task prioritisation and scheduling

| | Task | Priority | Sprint | Finished | Revised |
|---|---|---|---|---|---|
| CSTMD1 | Generate CSTMD1 morphology using Python and MATLAB script | 1 | 1 | ✓ | |
| | Implement multicompartmental Hodgkin Huxley neuron on GPU | 1 | 1 | ✓ | |
| | Implement morphology as a GPU neuron model | 2 | 1 | ✓ | |
| | Develop 3D visualisation tool | 2 | 1 | ✓ | |
| | Generate synapses from individual neuron morphologies | 1 | 2 | ✓ | |
| | Integrate ESTMD1 output stimulus | 1 | 3 | | 4 |
| | Model verification and parameter search | 1 | 3 | | 5 |
| | Re-implement for step simulator for continuous flow model | 2 | 3 | ✓ | |
| | Integrate with STDP | 1 | 4 | | 5 |
| ESTMD | Analyse and test current ESTMD module for usage and performance | 1 | 2 | ✓ | |
| | Make necessary changes for continuous flow model | 1 | 3 | ✓ | |
| | Integrate with CSTMD1 | 1 | 3 | ✓ | |
| | Link with Environment | 1 | 3 | ✓ | |
| | Investigate possible implementation in CUDA and OpenCV | 2 | 4 | | 5 |
| STDP | Analyse and test current STDP module for usage and performance | 2 | 2 | ✓ | |
| | Implement into continuous flow model | 1 | 4 | ✓ | |
| | Integrate with CSTMD1 | 2 | 4 | | 5 |
| | Integrate with RL | 2 | 4 | | 5 |
| RL | Analyse and test current RL module for usage and performance | 2 | 2 | | 4 |
| | Implement into continuous flow model | 1 | 4 | | 5 |
| | Integrate with CSTMD1 and STDP | 2 | 4 | | 5 |
| MC | Implement motor controller | 1 | 3 | ✓ | |
| | Integrate with Environment | 2 | 3 | ✓ | |
| | Integrate with RL | 1 | 4 | | 5 |
| Environment | Analyse and test current animation module for usage and performance | 1 | 1 | ✓ | |
| | Implement 2D environment model including agent and prey location | 1 | 2 | ✓ | |
| | Extend animation visualisation onto 2D model | 1 | 3 | ✓ | |
| | Integrate with ESTMD | 2 | 3 | ✓ | |
| | Build and integrate 3D environment | 2 | 5 | | 6 |
| Integration | Integrate current dependencies into a Docker module | 2 | 2 | ✓ | |
| | Brain Studio investigation for module integration | 2 | 3 | * | |
| | Investigate Shared memory use for module integration | 2 | 3 | * | |
| | Testing, quality control and dependency checking | 1 | 5 | | 6 |
| | Quad-copter implementation | 3 | 6 | | 6 |

* No longer required

# B Testing Coverage Detailed Reports

## B.1 Testing Coverage

Project_Coverage.dat

| Name | Stmts | Miss | Branch | BrPart | Cover |
|---|---|---|---|---|---|
| CSTMD1/Cstmd1.py | 88 | 0 | 22 | 2 | 98% |
| CSTMD1/Morphology/Compartment.py | 95 | 0 | 20 | 0 | 100% |
| CSTMD1/Morphology/Coordinate.py | 30 | 0 | 0 | 0 | 100% |
| CSTMD1/Morphology/MultiCompartmentalNeuron.py | 117 | 5 | 42 | 5 | 94% |
| CSTMD1/Morphology/NeuronCollection.py | 163 | 3 | 60 | 1 | 98% |
| CSTMD1/Morphology/NeuronGenerator.py | 92 | 0 | 24 | 1 | 99% |
| ESTMD/Estmd.py | 111 | 0 | 30 | 0 | 100% |
| Environment/Background.py | 12 | 0 | 0 | 0 | 100% |
| Environment/Dragonfly.py | 13 | 0 | 0 | 0 | 100% |
| Environment/Environment.py | 60 | 4 | 22 | 9 | 84% |
| Environment/Target.py | 28 | 0 | 4 | 1 | 97% |
| Environment/TargetAnimation/AnimationWindow.py | 24 | 3 | 6 | 2 | 83% |
| Motor/Motor.py | 16 | 0 | 2 | 0 | 100% |
| STDP/neuron.py | 261 | 41 | 78 | 20 | 79% |
| STDP/sample.py | 130 | 0 | 34 | 1 | 99% |
| STDP/simulation.py | 155 | 50 | 50 | 13 | 61% |
| STDP/stdp.py | 70 | 4 | 28 | 2 | 92% |
| TOTAL | 1465 | 110 | 422 | 57 | 90% |

## B.2 Testing Output

Dragonfly_Test_Suite_Output.dat

```
test_make_video (ESTMD.tests.test_ESTMD.TestESTMD) ... ok
test_no_video (ESTMD.tests.test_ESTMD.TestESTMD) ... ok
test_run_frames (ESTMD.tests.test_ESTMD.TestESTMD) ... ok
test_run_no_frames (ESTMD.tests.test_ESTMD.TestESTMD) ... ok
test_with_parameters (ESTMD.tests.test_ESTMD.TestESTMD) ... ok
test_without_preprocess (ESTMD.tests.test_ESTMD.TestESTMD) ... ok
test_adding_parent (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_can_join (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_join (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_length (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_midpoint (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_removing_non_existent_parent (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_removing_parent (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_split (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_string (CSTMD1.tests.test_Compartment.TestCompartment) ... ok
test_add (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_array (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_distance (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_div (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_equals (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_mid (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_midpoint (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_mult (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_string (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
```

```
test_sub (CSTMD1.tests.test_Coordinate.TestCoordinate) ... ok
test_compartment_data (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_connection_matrix (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_correctly_loaded (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_homoginise (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_homoginise_median_stdev (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_length_median (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_length_stDev (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_plot_compartments (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_radii (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_rebase (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_steps_to_root (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_string (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_wrong_SWC_path (CSTMD1.tests.test_MCN.TestMCN) ... ok
test_NoSynapses (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_NumberOfNeurons (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_Synapses (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_SynapsesFail (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_compartment_data (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_compartment_offsets (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_cstmd1_electrical_connections (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_cstmd_get_synapses (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_generate_electrodes (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_get_compartment (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_get_wrong_compartment (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_import_electrodes (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_import_estmd_mapping (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_import_synapses_from_file (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_incremental_load_spikes (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_load_spikes (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_load_voltages (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_map_electrodes_to_compartments (CSTMD1.tests.test_NeuronCollection.TestNeuronCollection) ... ok
test_GenerateNeurons (CSTMD1.tests.test_NeuronGenerator.TestNeuronGenerator) ... ok
test_Points (CSTMD1.tests.test_NeuronGenerator.TestNeuronGenerator) ... ok
test_constructors (CSTMD1.tests.test_Cstmd1.TestCstmd1) ... ok
test_run_estmd (CSTMD1.tests.test_Cstmd1.TestCstmd1) ... ok
test_run_no_estmd (CSTMD1.tests.test_Cstmd1.TestCstmd1) ... ok
test_save_spikes (CSTMD1.tests.test_Cstmd1.TestCstmd1) ... ok
test_x (CSTMD1.tests.test_Cstmd1.TestCstmd1) ... ok
test_incorrect_direction (Motor.tests.test_motor.TestMotor) ... ok
test_step (Motor.tests.test_motor.TestMotor) ... ok
test_draw_PNG (Environment.tests.test_animationwindow.TestAnimationWindow) ... ok
test_update (Environment.tests.test_dragonfly.TestDragonfly) ... ok
test_update (Environment.tests.test_target.TestTarget) ... ok
test_update (Environment.tests.test_background.TestBackground) ... ok
test_step (Environment.tests.test_environment.TestEnvironment) ... ok
test_heavy_side (STDP.tests.test_neuron.NeuronTests) ... ok
test_ltp (STDP.tests.test_neuron.NeuronTests) ... ok
test_plot_ltd (STDP.tests.test_neuron.NeuronTests) ... ok
test_plot_ltp (STDP.tests.test_neuron.NeuronTests) ... ok
test_plot_weight_distribution (STDP.tests.test_neuron.NeuronTests) ... ok
test_plotting (STDP.tests.test_neuron.NeuronTests) ... ok
test_sum_epsps (STDP.tests.test_neuron.NeuronTests) ... ok
test_time_delta (STDP.tests.test_neuron.NeuronTests) ... ok
test_update_epsps (STDP.tests.test_neuron.NeuronTests) ... ok
test_update_weights_time_delta_0 (STDP.tests.test_neuron.NeuronTests) ... ok
test_update_weights_time_delta_1 (STDP.tests.test_neuron.NeuronTests) ... ok
test_update_weights_time_delta_None (STDP.tests.test_neuron.NeuronTests) ... ok
test_general_test (STDP.tests.test_simulation.SimulationTests) ... ok
test_load (STDP.tests.test_simulation.SimulationTests) ... ok
test_load_file (STDP.tests.test_simulation.SimulationTests) ... ok
test_loading_pattern (STDP.tests.test_stdp.StdpTests) ... ok
test_produce_spike_array (STDP.tests.test_stdp.StdpTests) ... ok
test_step_with_more_afferents (STDP.tests.test_stdp.StdpTests) ... ok
test_step_with_time_delta (STDP.tests.test_stdp.StdpTests) ... ok
test_step_with_training (STDP.tests.test_stdp.StdpTests) ... ok
```

```
test_step_without_training (STDP.tests.test_stdp.StdpTests) ... ok
test_step_without_training_withHistoricWeights (STDP.tests.test_stdp.StdpTests) ... ok
test_avg_hz (STDP.tests.test_sample.SampleGeneratorTests) ... ok
test_avg_hz_with_noise (STDP.tests.test_sample.SampleGeneratorTests) ... ok
test_custom_pattern (STDP.tests.test_sample.SampleGeneratorTests) ... ok
test_empty_pattern (STDP.tests.test_sample.SampleGeneratorTests) ... ok
test_pattern_duration (STDP.tests.test_sample.SampleGeneratorTests) ... ok
test_save (STDP.tests.test_sample.SampleGeneratorTests) ... ok

----------------------------------------------------------------------
Ran 98 tests in 25.677s

OK
```

# C ESTMD Profiling

ESTMD_Profile.dat

```
9 Second video with 100 frames a second

Without preprocess resizing

         252205 function calls (251265 primitive calls) in 407.986 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1998  342.208    0.171  342.208    0.171 {scipy.signal.sigtools._linear_filter}
     2997   20.800    0.007   20.800    0.007 {cv2.filter2D}

With preprocess resizing

         252205 function calls (251265 primitive calls) in 8.495 seconds

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1998    4.239    0.002    4.239    0.002 {scipy.signal.sigtools._linear_filter}
     1000    1.078    0.001    2.289    0.002 vidproc.py:31(get_frame)
```