THE BLIMP GOES VIRTUAL

Using Biological Models and Computer Modelling to Give Insight in the Functionality of the Brain

THE BLIMP GOES VIRTUAL

A research done by Melle Hofman

‹Student number U59073›

Student of the Interdisciplinary Master in Cognitive Systems and Interactive Media at the University Pompeu Fabra, Barcelona, Catalonia

Published 23rd of June 2009

Commissioned by SPECS, Barcelona, Catalonia

Supervised by
Dr. Sergi Bermúdez i Badia

Using Biological Models and Computer Modelling to Give Insight in the Functionality of the Brain

ABSTRACT

The traditional approach of building autonomous robotic systems is using sensors that are dedicated to a specific task. These sensors are very good at a specific task, but cannot adapt that well to new environments. Biological systems, specifically insects, have proven to be very good at adapting to new environments. This thesis is therefore aimed at the development of systems that can attract multiple features from a single camera input for navigation. Using a blimp gives us the flexibility we need to test insect like behaviour. We developed several different models estimating speed, looming and detecting direction. These systems are inspired by insect behaviour and biological systems. In order to speed up the development and testing we built a flight simulator controlling the conditions of a virtual blimp.

We used the flight simulator to test each of the developed systems, and adapted them to use the sensor readings for autonomous feedback systems. Our results show a reliable reading for direction, looming and speed estimation. Further developments should integrate these parts in order to reach the next level of cognitive processing.

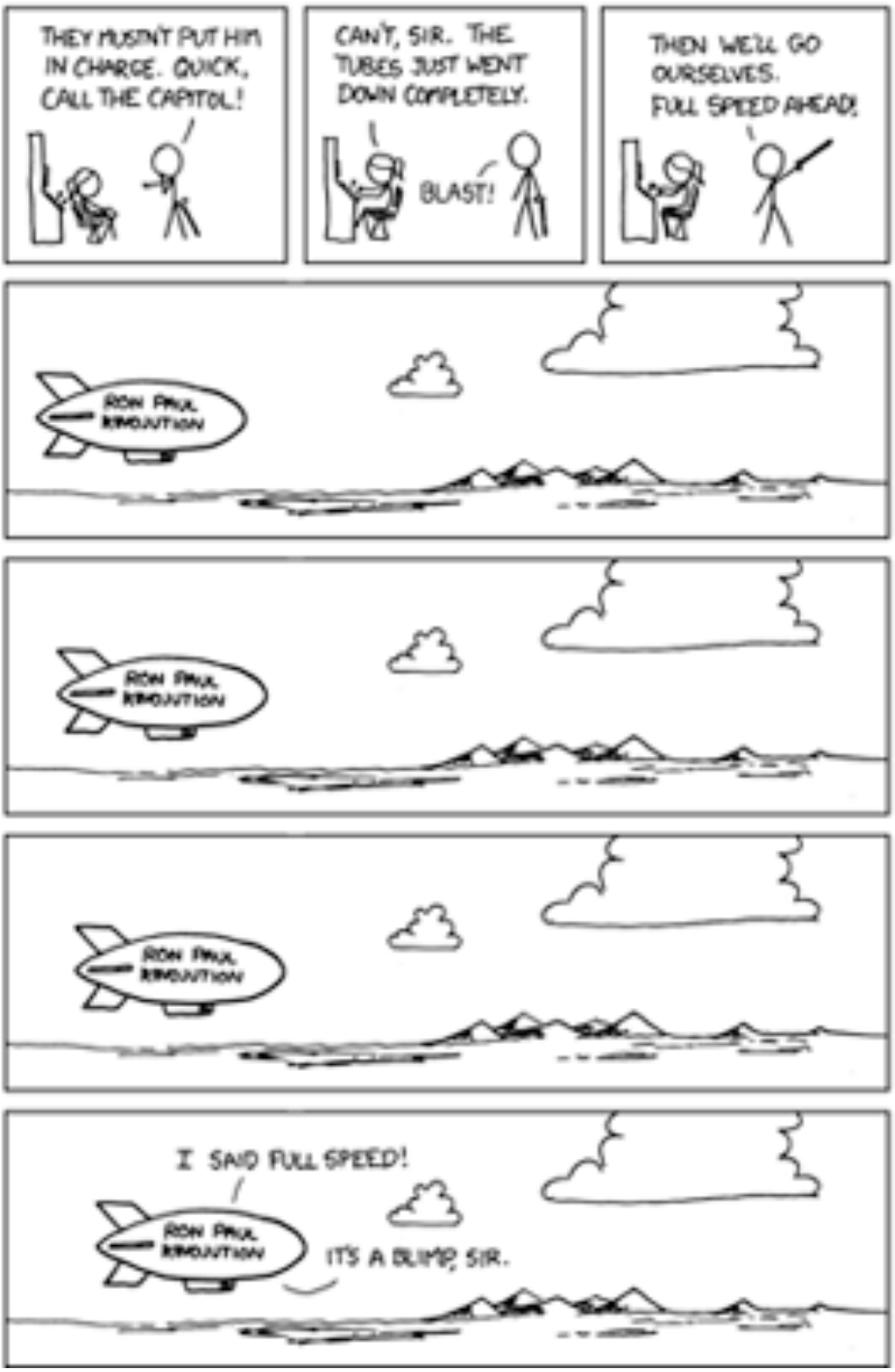

TABLE OF CONTENTS

1 INTRODUCTION

1.1 THE WIRING OF THE BRAIN

> Behaviour is a product of the wiring of the brain. We can have a look at intelligence once you know how this wiring works. Organisms can change their sensors and mapping through an evolutionary process, or can adapt the mapping of these sensors in the brain. Elaborated sensors breed the possibility or even necessity to develop cognitive skills such as memory. The hard- and soft-wiring can be used as good models for artificial systems. Creating many of these systems will provide the building blocks for cognitive systems or artificial intelligence.

In our study, "The Blimp goes Virtual", we look at how an insect brain uses a compound eye to compute and extract features from its environment. This study is interesting because it can give enlightenment in the philosophical study of the brain and consciousness. We will show how the process of neuroethology is important for understanding the computational approach and we will link this to the importance of adaptive sensing. Is the wiring of the insect brain the same as the wiring of the human cerebellum? Perhaps adaptive sensing is nothing but the rewiring of the brain, which can happen in the form of hard wiring and soft wiring.
Scientists and philosophers like Minsky, Fodor and Pinker support a computational approach towards the mind.

> The mind is what the brain does; specifically, the brain possesses information, and thinking is a form of computation (Pinker, 1997)[1].

Thus if thinking is a form of computation, we can take an organism and measure they way its brain computes. In addition, if the brain is set up out of a collection of computational modules, then it is not important how *intelligent* the organism is, but it becomes more important how well it performs its task. Gabbianni recently unravelled why it is so difficult to swat a fly, by studying the neurology of a *locust migratorio*.

> Recent work on an identified neuron in the locust visual system (the LGMD neuron) that responds well to objects looming on a collision course towards the animal suggests that this cell represents a good model to investigate the biophysical basis of multiplication and invariance at the single neuron level. Experimental and theoretical results are consistent with multiplication being implemented by subtraction of two logarithmic terms followed by exponentiation via active membrane conductances, [...] (Gabbianni, Krapp, Hatsopoulos, Mo, Christof, & Gilles, 2004).

Thus, Gabbianni's experiment shows that there is one neuron responsible for the computation of looming, and this neuron is very important for the insect's perception. Based on this study you can see that the insect is wired up in such a way that it identifies looming objects as predators and, in flight, some forms of looming as obstacles. The insect does not need to learn that looming is a predator; this is hard-wired in the brain of the insect. This works fine for the *locust migratorio* being an herbivore, but for the *tenodera aridofolia* (the praying mantis) a looming object can be a predator, but it can also be its meal. This dilemma was addressed by Yamawaki and Toh, who did a similar experiment on the mantis as Gabbianni did on the locust. They found the possibility that the looming-sensitive unit plays a role in the triggering of escape, but also trigger an attack reflex on smaller moving stimuli. (Yamawaki & Toh, 2008).
The locust and the mantis both have a very similar anatomy. They both have a compound eye. They both have more or less the same nervous system. Even the brain contains a very similar group of neurons to calculate looming, in both animals classified as the LGMD neurons with its helper neurons. Despite that, both insects have a different behavioural response. That gives me the support to use the model of the LGMD neuron for a different purpose. Using the same sensory input, and same basic effectors, for slightly different purposes is a form of adaptive sensing.

[1] Citation taken from Murphie and Pots. Murphie, A., & Potts, J. (2003). *Culture & Technology*. New York.

> The process of adjusting sensing functions, finding new feature primitives, or changing observables is another form of learning, the learning of new categories rather than learning with existing categories (Cariani, 1998).

Of course this adjustment of wiring is a relatively small deviation from the original design, but fact is that the difference in response to the looming is not a result of habituation, it is a really is a different wiring of the brain. Additionally this rewiring of the brain is a product of evolutionary process. New organisms appear with a slightly different wiring, and natural selection decides if the adaptation (learning) is appropriate for the environment the organisms live in. Peter Cariani continues his previously cited paragraph with:

> In order to realize this kind of learning, a system must be able to adaptively alter its sensing functions, thereby changing the external semantics of its feature primitives [...]. This can be accomplished in two basic ways: by redeploying existing internal degrees of freedom, or by creating new ones (Cariani, 1998).

This alteration of degrees-of-freedom occurs in higher organisms. These higher organisms can adapt their behaviour with the help of previous experiences. We would like to define this as the soft wiring of the brain. One of the first experiments on animal learning shows a behavioural pattern that supports this soft wiring stance. Edward Thorndike is famous for doing animal learning experiments. As a PhD student he studied the behaviour of cats in a box, and measured how long it took them to get out of the box. He used this experiments to set up his laws of exercise and effect.

> The Law of Exercise is that: Any response to a situation will, other things being equal, be more strongly connected with the situation in proportion to the number of times it has been connected with that situation and to the average vigour and duration of the connections (Thorndike, 1911).

Cariani's observations are in line with Thorndike's foundations. This shows that there is a process in which organisms can adapt to their surroundings, using the same sensory input. Actively adapting the way sensors are mapped in the brain. Of course creating a new mapping in the brain is not easy, higher organisms have to go through a process of learning and habituation that adapts their sensor mappings. A nice example where there is a range of sensor development is when learning to drive a car. Firstly, the driving student would have to learn how to deal with sensor prostheses. The speedometer, for example, is a form of adaptive prostheses. It makes sensory data available to the driver that they would not have gotten without this sensory extension. The learning process looks as how Hubert Dreyfus describes it. First the driver learns a set of basic rules for driving. Shift gear at a certain amount of revolutions per minute, have a distance of two seconds etc. After a while the driver starts to recognize when cars are more or less on a collision course and when not. The driver learns to shift gear without looking at the revolutions meter, and in the end after becoming an expert driver; the majority of actions go without thinking. The brain has rewired itself to facilitate driving. This is a sensory adaptation of the brain so it can execute all the basic features of driving without have to consciously think about it.

> Once a network has encountered a particular situation from a particular perspective and has performed an appropriate action, the same or a similar situation, seen in the same way, will tend to produce the same or similar appropriate behaviour (Dreyfus, 1996).

Insects do not have a process of learning (other than habituation and classic conditioning (Wikipedia, 2009)) and rewiring of the brain to adapt their sensor mappings to a new situation. Vertebrates on the other hand do have ways to make more advanced use of their sensors. The development of sensors provides the basis to planning and cognition. McIver calls this the "Buena Vista Sensing Club" hypothesis.

> The Buena Vista Sensing Club hypothesis suggests that the expansion of the range with which animals can monitor external space, relative to their usual velocity, has been one – perhaps the dominant – driving force for the ability to plan (MacIver, 2006).

In my opinion the soft wiring of the brain to adapt the sensors to a new mapping is the first step towards planning and cognition. If your sensor processing and motor reflexes are automated in such a way that the most basic processing and output become a part of reflexes, then the rest of the brain can be occupied by planning and conscious handling. Communication between organisms with cognition can even further enhance this, because organisms are not dependent on their own sensors, but have –

through language and symbols – access to other organisms' sensors. Finally McIver concludes that neuroethology provides

> an ideal set of tools for pushing forward a rigorous understanding of the factors influencing when reactive control has to give way to a planning mechanism in those special creatures with exceptional margins of sensation over movement (MacIver, 2006).

I have showed now that there is a form of learning and automation of sensory motor processes going on in the brain, but in order to make the distinction between hard and soft wiring I would have to prove that there is such a thing as wiring in the brain.

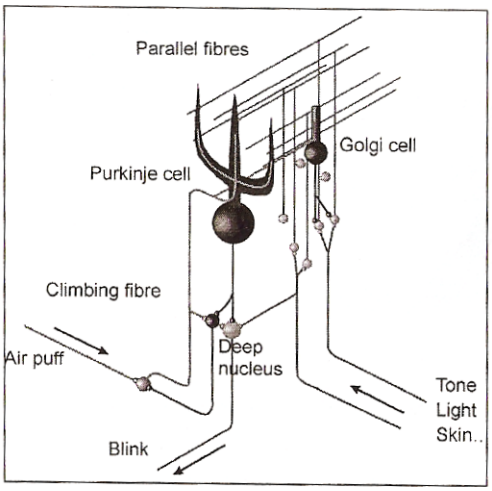

Figure 1.1 – A simplified diagram of the basic cerebellar circuit. The cerebellar Cortical Conditioning Model assumes that the Unconditioned Stimulus, via climbing fibres, induces changes in synapses between parallel fibres and Purkinje cells and/or various interneurons in the cerebellar cortex. Adapted from Verschure (Verschure, 2008).

Verschure showed in his lecture studies Rescorla & Wagner and Thompson in order to explain how blocking these Purkinje fibres triggers learning behaviour. Learning happens if an unconditioned stimulus accompanies a conditioned stimulus, and after a while the trained stimulus can trigger the conditioned response. As you can see in figure 1, a rewiring of the Purkinje fibres does this. The student driving the car will rewire its part of the cerebellum that is responsible for motor learning, and therefore you can say that learning and intelligence is a part of the soft wiring of the brain.

Many other recent studies show that learning, and associative memory is a product of the rewiring of Purkinje fibres in the cerebellum. Koekoek et al. published an article in science where they study learning abilities between normal and transgenic mice. These transgenic mice were given impairment in their Purkinje fibres in the cerebellum.

> Our study provides evidence for a solution to a long-standing issue in the field of classical eyelid conditioning, i.e., whether LTD contributes to the temporal shaping of conditioned responses (Koekoek, et al., 2003).

Thus, Koekoek et al. gives us a neurological explanation of how Dreyfus' learning works. This therefore also gives proof that learning and behaviour is a part of physical wiring in the brain. In order to understand and built better systems for the future I experimented with creating small insect like robots that have simple neurological wiring. Essential for this study is that the amount of neurons is reduced to a minimum. Brooks gives other important guidelines for creating intelligent machines.

> ·We must incrementally build up the capabilities of intelligent systems, having complete systems that each step of the way and thus automatically ensure that the pieces and their interfaces are valid. ·At each step we should build complete intelligent systems that we let loose in the real world with real sensing and real action. Anything less provides a candidate with which we can delude ourselves (Brooks, 1991).

Therefore, we am using these principles and guidelines for doing my thesis research and creating a flying blimp that works with hard-wired brain, inspired by an insect brain is one of the many steps we have to take towards building intelligent systems.

In this thesis we tried to give a rationale for my type of studies. It shows that sensor development is one of the main drives for the development of cognition and that similar sensor can be wired up differently in order to create an enhancement. Moreover, I claim that cognition and planning is a product of the enhancement of these sensors and that different wiring is an important motor for this. Whilst lower organisms such as insects have an evolutionary approach towards learning – hard wiring –, higher organisms can condition parts of their sensors by learning skills through classical conditioning. I would like to define this approach towards sensor adaptation soft wiring. Because the wiring of the brain is the main motor behind the development of cognition we will develop with my thesis an autonomous working system using this model. We will do this by using the guidelines Brooks developed after a long

investigation of Artificial Intelligence and his experience he obtained whilst making these kinds of systems.

1.2 PROBLEM DEFINITION

The traditional approach of building autonomous robot systems is using sensors that are very much dedicated to one task. Proximity sensors, for example, can detect if a robot is about to collide with a wall. A speed measure on the wheels, can give a system a speedometer, and an altimeter can give a robot a sense of altitude. But, these are all dedicated sensors that can be used for their designated purpose only. To be able to test the hypothesis that intelligence is a product of the wiring of the brain a universal type of sensor needs to be used. Also Cariani's approach to active sensing and MacIver's Buena Vista Sensing Club requires that a lot of features should be extracted from one universal sensor. The last requirement for proofing the wired brain hypothesis is to use a modelling system that resembles the building blocks of the brain. Therefore, this thesis aims for extracting as many complex features as possible from one simple universal sensor, using the real world as a model to work in. Additionally, working with standardised models to process sensory input through a camera enhances the way sensors are currently used. Providing a lot bigger bandwidth of features to extract from one single sensor, and that would lead to an explosion in applications using a camera as the main sensor[2].

In order to facilitate the adaptive sensing process, the models should be as simplified as possible, reducing the amount of synapses and neurons. A linear increase in amount of neurons means an exponential increase in combinations that could lead to an increase in failure of the adaptations of the sensors.

For this study three models have been developed using a camera as the only sensor. Each of the three models extracts an important feature from the environment and the goal is to get a reliable reading after visual image processing. The model should provide a stable velocity reading, a stable looming reading for height estimation and reliable direction estimation.

1.3 THE INSECT VISUAL SYSTEM

The insect brain and the insect visual system is often used for making artificial models, this because the insect visual system is, compared with other animals, a lot less complex (Rajesh, David, & Derek, 2002). The insect visual system consists out of four layers, the *retina*, *lamina*, *medulla* and the *lobula* (Figure 1.2).

[2] http://news.bbc.co.uk/2/hi/technology/8077369.stm

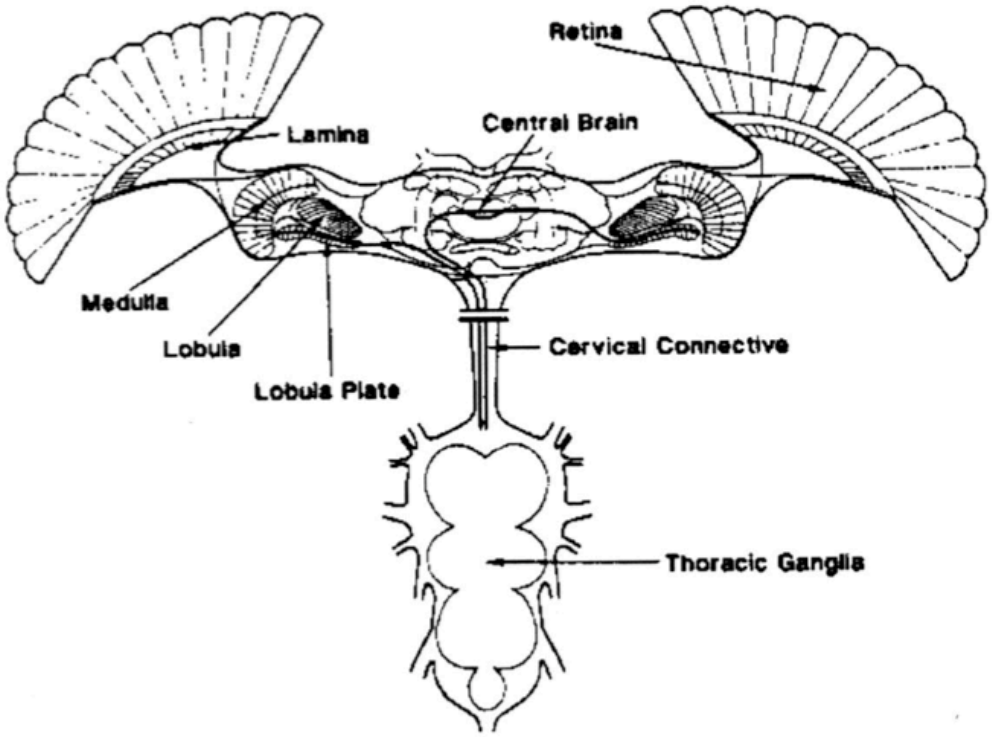

Figure 1.2 Cross section of a fly brain with compound eyes. Adapted from Rajesh et al., (Rajesh, David, & Derek, 2002)

1.3.1 Retina

The retina of an insect eye is not like the retina in the human visual system. Insect eyes are large compound eyes that consist of a collection of facets. The amount of facets varies per insect. Larger insects such as the honeybee have three to four thousand facets, whilst the number of facets of a fruit fly is considerably lower, approximately 750 facets (Deckert, 2001). These facets are placed on the outside of the insect and are sensitive to light, each of these facets have one ommatidium, which contains lenses and rods refracting the light to one of eight photoreceptors. This process gives a blurred image, and thus accounts for spatial low pass filtering.

1.3.2 Lamina

The lamina does the first part of the image processing, and pre-processes the images for the input of the correlators. The lamina is also responsible for contrast enhancement, and some evidence is found that spatial low pass filtering takes place in the lamina as well (Rajesh, David, & Derek, 2002).

1.3.3 Medulla

Rajesh et al. reports that columnar neurons in the medulla are responsible for key stages of local motion detection. Other than that, it contains a collection of classes of local columnar neurons and tangential cells. Because of its complex composition, little is know of this part of the insect visual system. Rajesh refers to a research of Douglass and Strausfeld (Douglass & Strausfeld, 1995) that recorded a number of motion sensitive cells that could serve as components or an array of EMDs.

1.3.4 Lobula

> The lobula receives input from the medulla to perform higher level functions (Rajesh, David, & Derek, 2002).

Other than this, Rajesh et al. conclude that little is known from the lobula and that makes it an interesting field of study for many researchers. Gabbianni et al. (Gabbianni, Krapp, Hatsopoulos, Mo, Christof, & Gilles, 2004) were interested in the computational functions of the visual cortex and have

cross-referenced to researches done on vertebrates. However, their main interest lies in a single computational neuron that can be found in locusts, the Lobula Giant Motion Detector (LGMD). The spiking of this neuron can be measured extracellular by measuring action potentials at the post-synaptic side of the target neuron. Gabbianni et al. developed a formula which can be used to calculate the collision time compensating for the angular size: $\theta(t) = 2\tan^{-1}(l/vt)$
Using different sized objects, the firing rate is measured and the time-to-collision is compared with the firing rate.

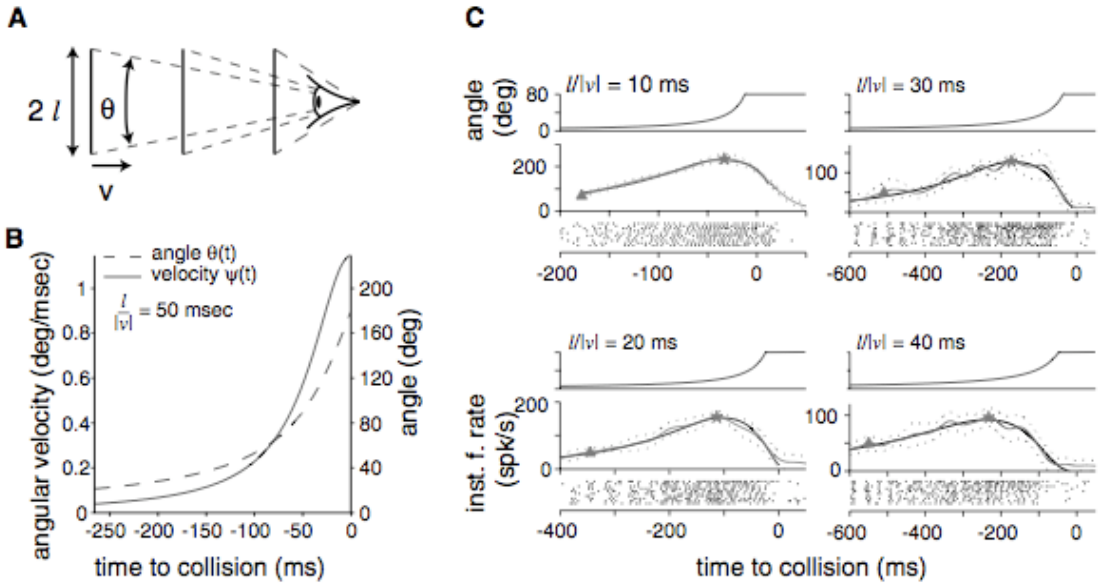

Figure 1.3 LGMD firing rate. Figure A shows how angular size is perceived. Figure B shows the relation between collision time and angular growth velocity. Figure C shows the spiking rate of the Locust when being presented with a looming object. Adapted from Gabianni (Gabbianni, Krapp, Hatsopoulos, Mo, Christof, & Gilles, 2004).

The graphs show that the firing rate is at a peak just before the time to collision is zero, with variances depending on the approach angle used. The dendritic tree arborises in three subfields within the Lobula that receive three specific inputs. Further investigations on the autonomy of the fly's visual system suggest that the LGMD has a retinotopic input that is excitatory, and two inhibitory inputs. In order to confirm this theory a behavioural study has been done to test the linear combinations of angular velocity and size. Additionally, the inhibition in the Lobula has been blocked by using an antagonist, resulting in more peaks at the LGMD neuron. Finally, Gabbianni et al. wanted to find out whether one neuron was responsible for the calculation. Although the LGMD seems to be the main indicator for collision detection, the logarithmic element in the formula suggest that a part of the equation is done pre-synaptically. In order to recreate such a neuron on an artificial level, the focus must be placed on one integrate and fire neuron with three pre-synaptic connections. However, expecting this single neuron to account for the complete computation is a bit bold. Graham (Graham, 2002) wrote in 2002 a review paper on the research done by Gabbianni et al., which at that time still had to be published. This article provides a good summary on what the function is of the Lobular Giant Motion Detector or LGMD Neuron. This neuron is capable of giving a representation of target's angular size, direction and velocity. Therefore, the LGMD Neuron provides a good model on what computational power one neuron can have. There are three main features of the LGMD Neuron that gives its function:

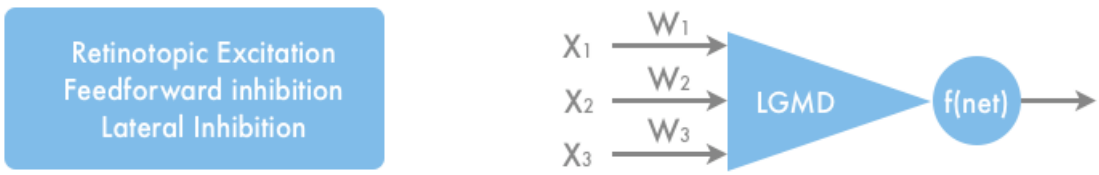

Figure 1.4 schematic representation of the LGMD neuron with relations

O'Shea and Rowell (O'Shea & Rowell, 1975) already showed in previous studies that the two inhibitory pathways can be up regulated and therefore compensate for different moving backgrounds. Gabbianni et al. used this approach to activate the pre-synaptic lateral inhibitory pathway during the presentation of a looming object. Additionally, in order to evaluate the role of the inhibitors several channel blockers were used to get different responses. All the results taken together show consistency on the authors applied

model that inhibition related to object angle acts post-synaptically on the LGMD, suppressing an excitatory input related to object angular velocity; sodium channel spike generation completes the multiplication through an expansive nonlinearity. Gabbianni et al. have therefore created the foundation for building models based on computations based on a single neuron. Moreover, Gabbianni et al. have provided the mathematical function on which these neuron could be based:
Gabbianni et al. fit the LGMD neuron spiking response to looming objects, R(t), to the product of two functions of the visual input's angular velocity, θ(t), and angular size, θ(t): $R(t) = \theta(t) \times \exp(-\theta(t))$
By exploiting the additive properties of logarithms, functions of the inputs were subtracted, instead of multiplied, yielding an equivalent expression that more directly reflected the underlying biophysics: $R(t) = \exp[\log(\theta(t))-\theta(t)]$ a decision on which approach to use.

1.4 STATE OF THE ART

1.4.1 Collision Detection in Complex Dynamic Scenes (Yue & Rind, 2006)

The schematics of the LGMD model can be used to make autonomous systems that do collision detection using camera input. Yue and Rind (2006) got inspired by the LGMD neuron to build a Khepera[3] robot with a mounted camera. They did this in order to find a solution for the sensor problem and they wanted to solve this by developing a collision detection unit based on a universal sensor, in this case a camera. The main principle for their model is, measuring expanding edges with lateral inhibition. Using the full speed range of the Khepera robot they had a success rate of 69% of collision detection, which could be even increased to 90% when using only half of the speed range of the Khepera.
Their neural network is based on eight layers. The photoreceptor layer with contrast enhancement gives each pixel an excitation depending on the luminance. There is an Inhibition and Excitation layer that transports their excitation and inhibition to the next layer. The excitation goes one on one to the Summation layer, whilst the inhibition undergoes a computation, before reaching the summation layer, only inhibiting the insides of a surface so only the edges remain. The Summation layer is linked to the Grouping layer that enhances the visual features defining a colliding object. The LGMD layer is a sigmoid function and has a high membrane potential. The Feed Forward Mediation layer is added to mediate the response of the LGMD by varying its threshold so it can cope with extreme luminance conditions. The Spiking mechanism then finally spikes if the correct threshold is reached.

$$S_f^{Spike} = \begin{cases} 1, \text{ if } k_f \geq T_s \\ 0, \text{ otherwise} \end{cases}$$

The final layer is the Feed Forward Inhibition layer; this layer suppresses the spike from the spiking layer when a sudden change in scenery produces more excited edges.
Yue and Rind have tested this model in several different scenarios. Because of their detailed description and good performance of their model, this article has formed a good source of information for this thesis.

1.4.2 Reichardt correlation model.

The most common approach when creating speed estimation models based on insect vision is the Reichardt correlation model.

> A basic Reichardt correlator is formed by combining two EMDs that are tuned to opposite direction, to indicate bidirectional motion (Rajesh, David, & Derek, 2002).

Summarizes Rajesh et al. in their paper on insect vision. Elementary motion detectors (EMDs) are direction sensitive cells.

[3] http://www.k-team.com

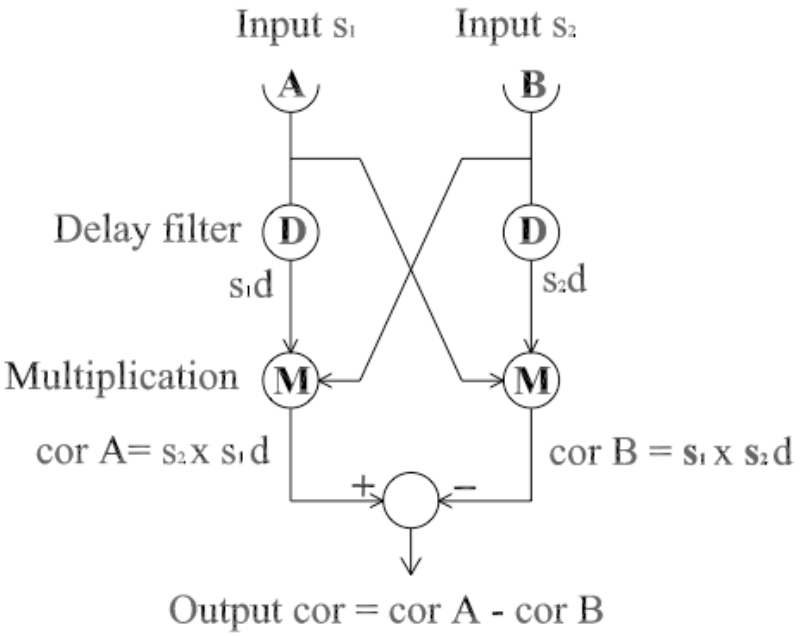

Figure 1.5 Reichardt correlator. Adapted from Rajesh et al. (Rajesh, David, & Derek, 2002).

Previous studies have suggested using this Reichardt correlation model for speed estimation and have come up with good models for this. Bermúdez used an interpolation method of several Reichardt correlators in order to give a good speed range.

> Then, the wide field neuron (HS/VS type, which sum the responses of the corresponding EMD populations) with the closest sensitivity to the actual angular speed shows the highest response. This method can deliver speed estimation, which is contrast and texture independent since all EMD populations are subject to the same conditions (Bermúdez i Badia, 2006).

This provides then a combination of speed estimation and direction estimation. Rajesh in his paper addresses the complexity of the Reichardt correlation model.

> Though insects and humans appear to be capable of estimating image velocities, the basic correlator model does not function as a velocity estimator. It reliably indicates directional motion of sinusoidal gratings, but the response depends on contrast (brightness) and spatial frequency (shape) as well as velocity. Furthermore, since the EMD is a complex spatiotemporal band pass filter, any one pattern produces same response at two unique velocities (Rajesh, David, & Derek, 2002).

That the patterns only show sensitivity at a specific speed is very well illustrated by Borst (Borst, 2007) in his comparison between a correlation-based model and a gradient-based model. This is illustrated in figure 1.6 where results are shown of a correlation-based model with its response. As long as the velocity is close to the pre-defined shift the response is high. This means that the situation in which the model is going to be used needs to be known at for hand. Currently the Reichardt correlation model is assumed to be the model used by insects, but Haag et. al. tested the hypothesis that an increased signal-to-noise ratio would cause a shift in the motion processing scheme from a correlation based model to a gradient based model, but they could not proof this hypothesis. (Haag, Denk, & Borst, 2004)

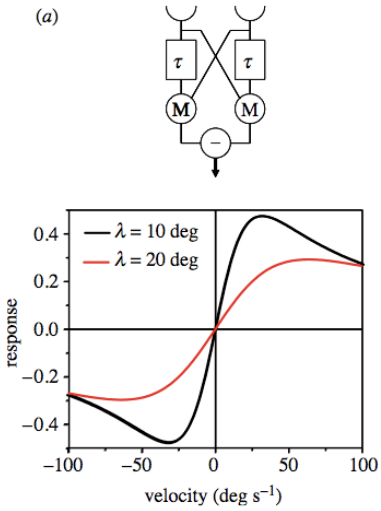

Figure 1.6 Sensitivity of a correlation-based model on velocity, adapted from Borst (Borst, 2007).

Thus, if flies rely so much on a correlation-based model, how can such a model be build serving a wide range of velocities? One method would be creating an array of motion sensitive neuron groups. Each of the motion sensitive units in the array should be sensitive to a different angular velocity. An approach used by Bermúdez i Badia (Bermúdez i Badia, 2006).

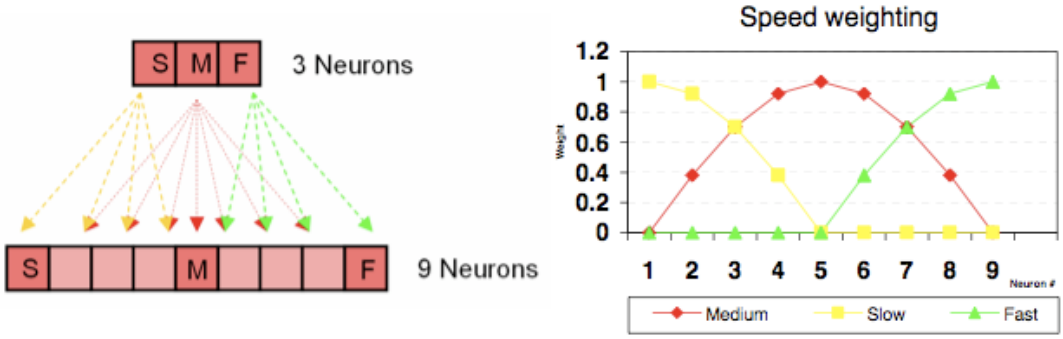

Figure 1.7 How an array of motion detectors can be used to create reliable speed estimation. Adapted from Bermúdez i Badia (Bermúdez i Badia, 2006).

The peaks of sensitivity are placed in a sequential array covering three different speeds; an interpolation method provides a wide range of speed estimation.

1.4.3 Gradient based speed estimation

When a set-up has a downward facing camera, the direction for flight is seldom linear. Therefore, another approach is needed for estimating velocity. Luckily, there have been models developed that are not dependent on correlation model. The most prominent alternative is the gradient detector. Haag et. al. describe on of these models in their experiment to compare it with a correlation model. In comparison with the Reichardt correlation model Borst concludes:

> while the Reichardt detector is ideal for operation under noisy conditions, the gradient detector shows superior and uncorrupted velocity dependence even locally. Therefore, the proposal made by Potters & Bailek (1994) made a lot of sense: the ideal motion detector should by of Reichardt type under low luminance conditions where noise is prominent, and switch over to a gradient detector under high luminance conditions when signal-to-noise levels at the photoreceptor are high (Borst, 2007).

These findings combined with the fact that our set-up has no clear linear flight directions puts the focus for this thesis on developing a gradient based speed estimation model.

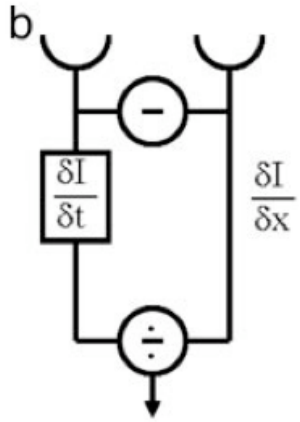

Figure 1.8 (b) In the gradient detector, the temporal luminance gradient as measured after one photoreceptor (Left) is divided by the spatial luminance gradient (the spatial gradient is approximated by the difference between the luminance values in two adjacent image locations (Haag, Denk, & Borst, 2004).

2 MATERIALS AND METHODS

Figure 2.1 shows a basic systems diagram of the model that is built. The set up consists out of a blimp that flies a natural landscape, such as a city. The modelling applies for real blimp flying over a city landscape or any other landscape that contains enough features for the modelled insect brain to extract direction, height and speed information. The laptop computer runs Fedora 10 Linux with the IQR neural network simulator. Although the insect brain is a method of creating small and light systems with dedicated circuitry, the modelling is done using a neural network simulator allowing a high level of abstraction so circuits can be easily updated and adapted, without having to write this straight into a chip-language. That, however, means the software has to run on a modern computer that simulates this micro circuitry. In order for the blimp to be able to operate using the processing of the artificial insect brain. It has to be connected to the system using a Bluetooth connection. When using the simulation on the other had, the simulation of the blimp communicates directly with IQR.

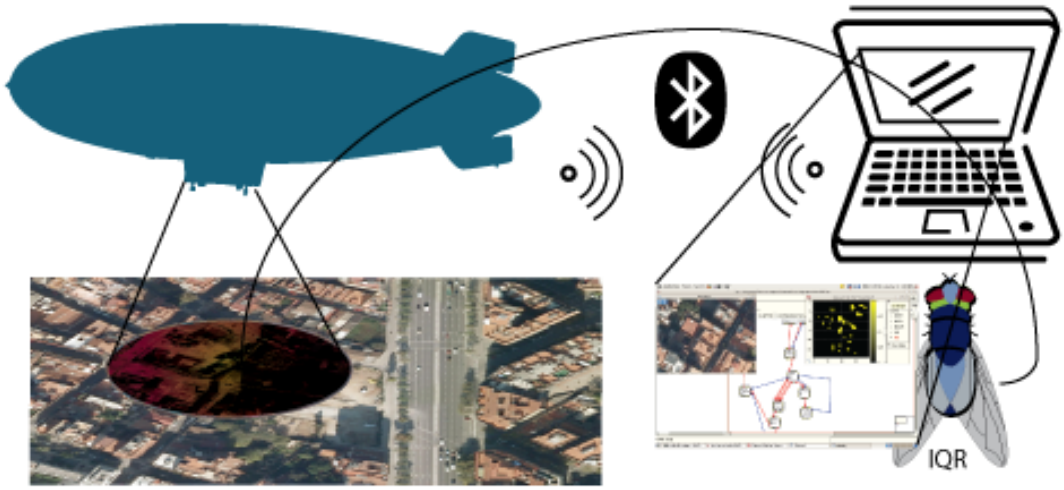

Figure 2.1 Functional Diagram – Real life or virtual blimp, connected with Bluetooth or virtual connection to an artificial model of an insect brain using IQR.

2.1 IQR: NEURAL NETWORK SIMULATOR

For this thesis, IQR[4] is used in order to build systems that are based on the functioning of the brain (Bernardet, Blanchard, & Verschure, 2002). This neural network simulator is based on the basic design paradigm of the brain and therefore contains different type of neurons and synapses. IQR is chosen as a modelling system because it gives the possibilities to work with the principles of real time processing of sensors and effectors according to a biological model, and with good abstraction. There is for example no need to know all the details of real-time programming, but the focus can be put on building a model that resembles a biological model. Additionally, IQR contains a wide range of modules that allow communication with robots, such as Lego Mindstorms and the ePuck. Because it is published open source, I had the opportunity to quickly develop my own modules.

In practice IQR works with several different levels. These levels resemble the different parts of the brain. The first level is the process. Within IQR the first thing to be done is defining the separate processes, each process can be compared with a part of the brain and usually contain neurons that are dedicated for a specific task. This keeps the whole building of neural networks manageable and well arranged. A process within IQR has to be used in order to communicate with different modules, for example to connect to an external device, such as a camera or an ePuck. The neuron groups within this process are often linked as input/output parameters for the module. The second level involves the neuron groups that can be found in the processes. Each neuron group contains neurons of the same category. The amount of neurons per group depends on the topology; a neuron group can for example consist out of 40 by 30 neurons and therefore represents 1200 neurons. The type of neuron is defined per group and there are five basic neuron types available in IQR. It is also possible to create customized neuron types by programming them as a separate library for IQR. The three neuron-types we used for my models are the integrate-and-fire neuron, the linear threshold neuron and random spike neuron. The integrate-and-fire neuron sums and subtracts the excitatory and inhibitory gain it gets through the synapses from

[4] http://iqr.sourceforge.net/

neighbouring neurons. When the result value reaches a certain threshold the specific neuron that reaches the threshold spikes with predefined amplitude. The linear threshold neuron has the same properties for calculating the activity, but always provides the sum as an output value. Therefore, the output of this type of neuron can be variable. There is also the possibility to add a threshold to a neuron group, the neuron group would only give output when the threshold is reached, the output in that case is a the result of the sum, thus still variable. This principle is used to filter out low amplitudes. The random spike neuron gives a random excitation based on the probability and the pre-defined spike amplitude.
The last level is the synapses. The synapses provide the communication between the neuron groups. There are three main synapse types, excitatory, inhibitory and modulatory, and each of the synapse can have a basic property. These basic properties tell something about the way the excitation or inhibition is passed on to the next neuron. In our models we use only the fixed weight and the shunting inhibition. The shunting inhibition acts as a synapse the folds itself around the neuron group, creating the mathematical effect of a division. Each of these synapses also has other properties, such as arborisation, delay and different patterns to map to another neuron group.
Building neural network models in IQR provides a good level of abstraction for building intelligent systems. These models can be used to understand better how the wiring of the brain works, or as a prototyping tool for low level applications.

2.2 FRAMEWORK: FEDORA, C++, OPENCV AND GNU OCTAVE

IQR is programmed using the programming language C++, and it is very easy creating extra modules using C++ (Wikipedia, 2009). Therefore, we used C++ for building my personal flight simulator. The coding of the C++ code was done using the VIM editor that is by default present in all modern Linux distributions. For the graphical processing inside the module we relied on the OpenCV (Wikipedia, 2009) library. This is a computer vision library that is available under the BSD license, and can therefore be freely used. In order to compile the code we used CMake. Code snippets and examples can be found in appendix 6.1.
The platform we chose for IQR to run on is Fedora 10 64bit installed on a Fujitsu-Siemens Amilo La 1703 especially purchased to serve this purpose only. This laptop computer has an AMD Turion 64 CPU with two cores that run each at 1900Mhz. This laptop computer has 1 GiB of memory.

GNU octave is a computer program designed for performing numerical calculations. It shares the same syntax as MATLAB and is therefore a very good alternative. It contains a wide range of standard statistical functions the combination with gnuplot makes it also a good tool for presenting the data. All the plots presented in this thesis are made using GNU Octave. We used the GNU Octave scripting language in order to automate the drawing of the plots. For this thesis three basic scripts are created, one for the looming estimation, one for the speed estimation and one for the direction estimation. The script for the direction estimation has two main modalities, one analysing the curved movements and one analysing linear direction.

2.3 METHODS OF DATA COLLECTION AND ANALYSIS

In total there are three parts of the designed model that need to be tested independently. For each of these models we collected data separately for analysis in Octave. The samples are taken from IQR using the data sampler. For each trial we created a directory in the file system and used well-coded filenames. When the samples where taken we used a standard regular expression to convert them into an Octave vector.
The regular expression changes all semi colons into commas, and then replaces all commas at the end of the line back into semi colons. The result is a file that contains a vector readable to Octave. In the first samples for the direction estimation we also used output straight from the module. This output was already normalised into data that can be read as a vector in Octave. This data was collected by using the standard output of IQR and store it in a text file. The cycle id is normalised with the cycle id from IQR's data sampler.
Several Octave scripts have been built in order to standardise the plots and the read-outs, eliminating human error in data processing.

2.4 FLIGHT SIMULATOR

2.4.1 Structural Design

The flight simulator is based on a physical blimp. The blimp is a helium filled balloon equipped with four propellers and one simple camera. Weights can be added in order to give it a gradual descent or to

keep it at float. The simulator allows for a setting of gravity in order to choose the behaviour of the blimp. The simulator provides six motor outputs. The horizontal propellers can be both operated individually and can go either forward or backward. An unequal speed in the propellers makes the blimp turn or rotate. Throttling both propellers at the same time creates a forward or backward motion. The vertical propellers are activated both at the same time, thus always inducing an equal upward or downward motion. A good tactic during the modelling of the behaviour of the blimp could be to always have a basic amount of thrust in the vertical propellers. The blimp stays at float if this trust has the right force to balance out the gravity. Main advantages are that if the model were switched of, the blimp would always return to the ground reducing the risk of loosing the blimp. Additionally, most insects that are used as a base for this study also has a density bigger than air and therefore also descent as long as they do not operate any motor output.

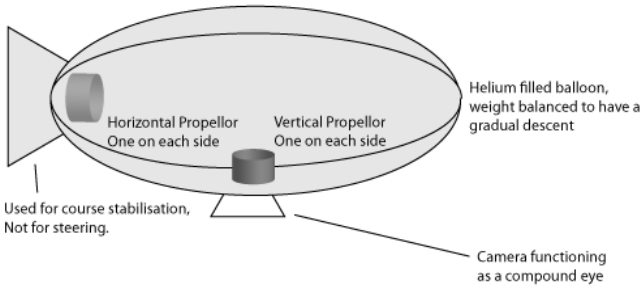

Figure 2.2 Structural design of the blimp, used as a model for the virtual blimp

2.4.2 Specifications of the flight simulator

The flight simulator should be able to create a realistic representation of the world the real blimp is going to fly in. As the citation of Brooks says in the introduction, anything that looks less like the real world model provides a candidate with which we can delude ourselves. Important pointers to look at when creating a realistic flight simulator are acceleration and the gravitational acceleration. Also an important simulation is the behaviour of the two engines with unequal power this should induce a rotation and a forward movement. Environmental variables such as wind are also important. This flight simulator has therefore the possibility to have side wind from either West or East.
The behaviour of the vertical movements is dependent on two factors, the vertical engine propulsion and the gravity. The vertical propulsion compensates for the gravity creating a net gravity that is used for the vertical motion. In order to make the simulation realistic we implemented a gravitational acceleration on the vertical movements using the following formula:

$$A = A \cdot (G - P)$$

Where A is the altitude, G the gravity of the simulation and P the net propulsion; this gives a realistic representation of the vertical movement of the blimp. The altitude can be measured within iqr by using the group that is connected to the altitude meter of the module, this way we can compare the visually calculated altitude with the actual altitude.
The forward propulsion is based on two engines, left and right. The forward acceleration is based on the physics of acceleration to give a realistic change in speed. The speed is calculated using the following formula:

$$V = (V \cdot P)^2$$

Where V is the velocity and P is the propulsion. This speed calculation is done only when the two engine speeds are the same, if the engine speeds are different the rotation speed is calculated, and also the rotation angle is calculated, and the resulting forward propulsion is calculated.

$$V_r = E_1 - E_2$$

Were Vr is rotation velocity and E1 and E2 are engine one and two.
When the two engine velocities are unequal, the flight simulator uses half the velocity for a rotation motion, and half the velocity for a forward motion. The flight simulator does these two steps without

giving feedback, so there error in the representation. The formula for calculating the rotation is the following:

$$\Phi = \Phi + 2 \cdot (10 \cdot E_1 - 10 \cdot E_2)$$

Where Φ is the rotation in degrees and E1 and E2 are engine one and two.

In order to increase the realistic representation of a flying blimp, we also made the blimp fly backwards. The formulas for acceleration and deceleration are the same as the previous programmed functions. Also if you are flying forwards and you give full throttle backwards the virtual blimp reduces speed faster and once it comes to a halt it starts flying in reverse. This is programmed using some exception handling code and this code snippet can be found in the appendix (6.1).

One of the main problems encountered with the flight simulator is the speed in which it operates. As a module it can sometimes reduce the speed of the whole iqr system to two cycles a second. Additionally, when flying at high altitudes, the simulation reaches more cycles per second than flying at low altitudes.
The following three steps show improvement of the overall performance of the simulation, and gives control over the cycles per second reached. Instead of using a resizing method with anti-aliasing filter we used a linear resize method.
If the simulation runs on a multi-core or multi processor system, it can be interesting to change the simulator into an iqr threaded module. If the module is threaded the simulation thread could run on a separate CPU core and therefore increase the overall performance of the system. By having the threaded module, we encountered a very important problem, the speed in which the module runs depends on the load the image processing gives to the system. A higher load could mean that the module can only do five cycles a second, whilst a small load would mean 20 cycles per second. It is important to keep a steady operating speed of the threaded module in order to have no distortions, and to have no errors when testing the system. Therefore, we implemented a control for cycles per second. For example, the simulation can run the threaded module at 10 cycles per second whilst the simulation runs at 20 cycles per second.

2.4.3 Graded realisms of testing

In order to quickly generate large images of landscapes, we programmed a small Perl (Wikipedia, 2009) script that automatically downloads Google Maps (Wikipedia, 2009) tiles and stitches them together. The Perl script generates takes the desired parameters and uses that to generate a bash (Wikipedia, 2009) script. The bash script contains the wget (Wikipedia, 2009) commands downloading all the separate Google Maps tiles and the ImageMagick (Wikipedia, 2009) commands that stitches these tiles together. ImageMagick allows stitching all the separate tiles using a command line tool. Thus, Perl generated a bash script for stitching all the downloaded tiles together and executing this script provided one big landscape of the region of choice. In order to find out which tiles to download, we used Firefox (Wikipedia, 2009) in combination with Firebug (Wikipedia, 2009). This allows seeing the URL of the starting image tile. We called the written script to create maps *MapMaker*. Google Maps has the tiles of their landscape images well numbered and categorized. Having found the number of the starting tile and the basic URL we added these parameters to the *MapMaker* script and automatically download a large part of the Google Maps. The x and y coordinates of the URL, have to be added in the *MapMaker* script. All the individual tiles get downloaded from Google Maps and stitched together. The *MapMaker* script can be found in the appendix (6.2).
There are three different maps we used for testing the models. Two of which are artificial and one is based on images of Google Maps. In all cases we used an automated process to generate the images. Each of these images is 6400 by 6400 pixels. We used two artificial surfaces because they would always guarantee an equal amount of edges, and that gives a reliable comparison to Barcelona our main image for testing. Here we have to emphasize that a model should be validated using the map of Barcelona, any artificial surface will give a delusion of success before success is really there.

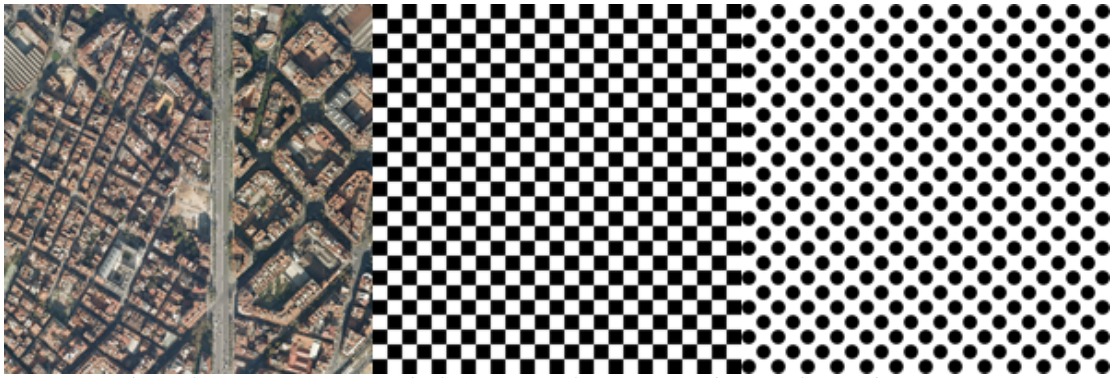

Figure 2.3 Three different surfaces with different gradations of realism and complexity

In order to prove the gradation in realisms we have pre-tested the images using simple edge detection. These pre-tests can be seen in the appendix (6.3).

2.5 DRIVING ROBOT

Because it is easy to lose control over a flying blimp, we chose to do a first real world experiment using a driving robot. This allows testing of all models developed and tested using the flight simulator and see if it works using camera input on a controlled surface. For this set-up we used an exciting robot and provided it with a wireless camera. Only the camera input is used and all other sensors are ignored.

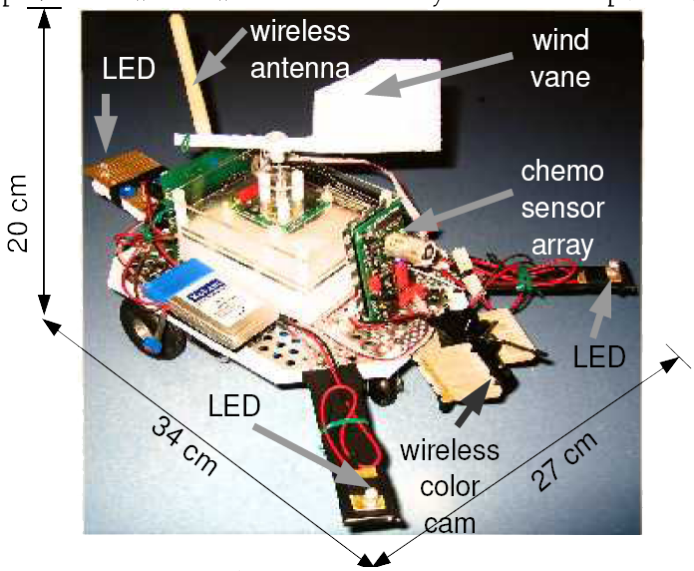

Figure 2.4 Driving Robot

The robot has two motors, left and right, that each van be activated for forward and backward movement. Two separate engines that are not connected with a differential always provide unequal drive to the engine even if the motor control commands give an equal value. The robot uses four neurons for motor mapping, two for each motor and two for each direction (figure x.x).

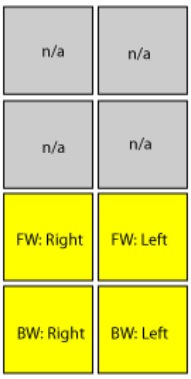

Figure 2.5 Neurons representing the motor-out group, the n/a fields are represented on the logic board, but not used in this robot.

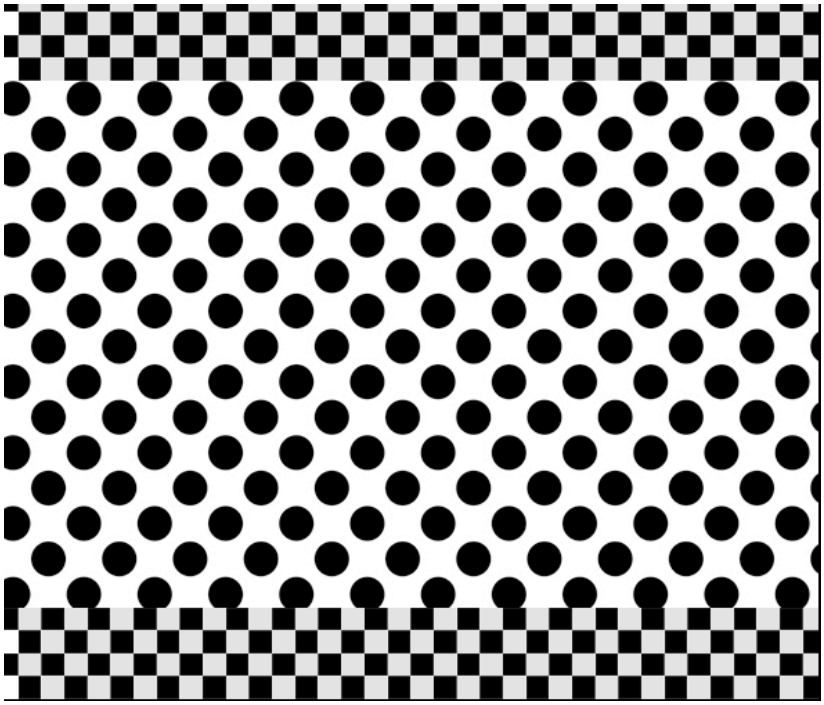

Figure 2.6 Driving Robot Arena

Testing the set-up with the driving robot is done in a test arena that provides the camera with sufficient visual input and allows for measurement of performance. The black circles have a diameter of 1.7 cm and the squares on the chequered strips are each 1 by 1 centimetre. In figure 2.5 a representation of the arena is shown.

2.6 MODELS

The separate models all form a part a big system. Figure 2.4 illustrates the sensory motor loop of the system. From the sensory input there is a divergence into separate sub-systems that each process the images to extract separate features. These features converge again into the behavioural layer and this layer control the motor output that operates the blimp. When the motor output operates the blimp the movement also induces different visual input and that closes the sensory motor loop.

The **visual input system** gets the input from the *virtual* photoreceptors with a resolution of 40 by 30 pixels and standardises the visual input for all the sub-systems. It forms two operations; it detects the edges of the input images and creates a high contrast. The output per pixel is either TRUE or FALSE.

The **looming detection system** gets the edges with high contrast from the *visual input system*. This system uses these edges to compare with edges from a previous cycle. If the amount of edges increase it detects looming, but not all edge increases are looming. Therefore, the system also compensates for lateral movements by filtering out high frequency movements. Moreover, when the blimp is flying forward the amount of edges is constantly changing because the amount of edges in the landscape is never the same. An increase in edges from forward motion is not looming thus these edges are normalised by inducing feed forward inhibition. The end result is one neuron that represents the amount of looming in the form of activity.

The **speed estimation system** uses the high contrast edges to make an overlay of these edges with a delay. There are ten connections with each a bigger delay. The output of the pixels is a TRUE or FALSE value therefore if the velocity is small there is a high overlap and low excitation, and increase in velocity would create a bigger shift in the pixels and would create a higher excitation. Because the amount of edges is not always the same, these have to be normalised by dividing the excitation by the total amount edges. The velocity gets expressed in excitation by one neuron, more excitation means higher velocity.

The **direction estimation system** compares the current image with a previous image having a one-pixel shift. We have developed three different models and each use a slightly different method for estimating the direction. Although there are three different methods for estimating the direction, the output is all

the same. One neuron has an excitation between 0.45 and 0.360, which equals the amount of degrees of the direction.

All these systems can be used to adapt behaviour to the environment of the blimp, such as drift compensation or flying in a specific direction. The **behavioural system** activates the motor output based on the pre-defined drive and the sensory input. Each motor command triggers behaviour that influences the sensory input and therefore closes sensory-motor loop.

The following chapters will give a detailed description of the functioning of each individual system.

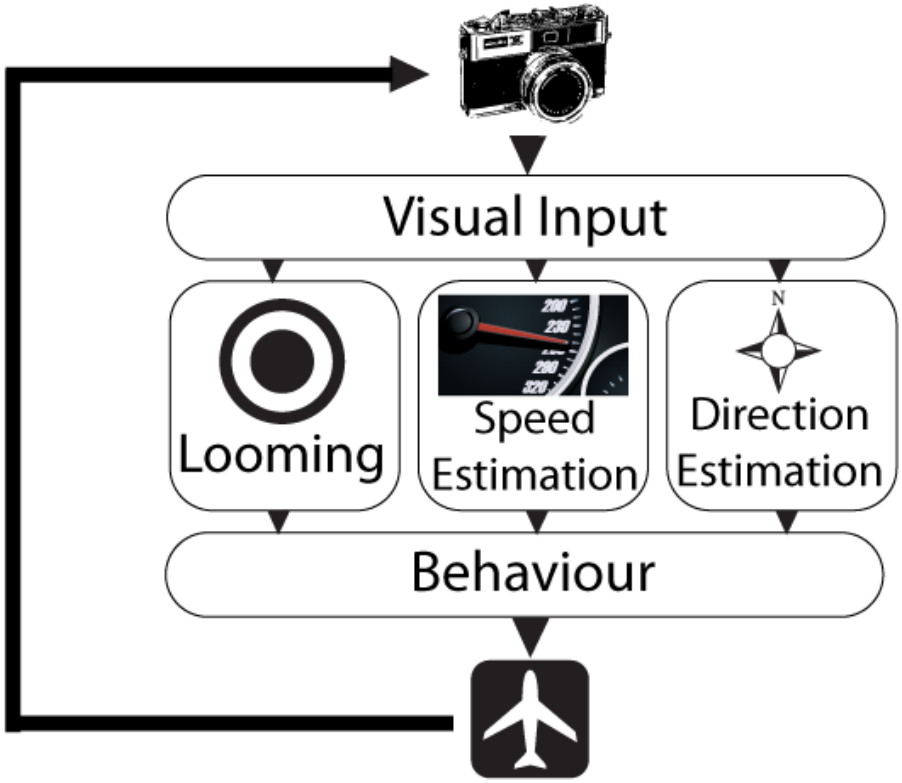

Figure 2.7 Sensory Motor Loop: the five sub-systems that make up a model of a virtual blimp

2.6.1 Visual input processing

2.6.1.1 V-layer

The video input layer captures the video input from the virtual camera. The pictures are captured and translated to a luminance value for each of the pixels. V(x,y) = [0.000 – 1.000]. The total amount of neurons in this group is 40x30.

2.6.1.2 E-Layer

Another group of 30x40 neurons that only illuminates the edges of objects. It does that by using an IQR implementation of a convolution multiplication. Using two synapses, one excitatory and one inhibitory, connecting the C-layer with the E-layer with a rectangular arborisation. Rectangular arborisation makes the connections of the synapses in squares with a window on the inside. The inhibitory synapse has a bigger window than the excitatory synapse so the excitation gets inhibited until the image reaches the edges, the excitation excites the edges, but because there is no inhibition at that stage the edges stay, see figure 2.5.

2.6.1.3 C-Layer

After having extracted the edges, the edges should get an absolute value. This is done by creating a neuron group of 40x30 with a threshold of 0.9 and spike amplitude of 1.0. This makes sure that only real edges appear in the space plot and filters out small blimps or not clear edges.

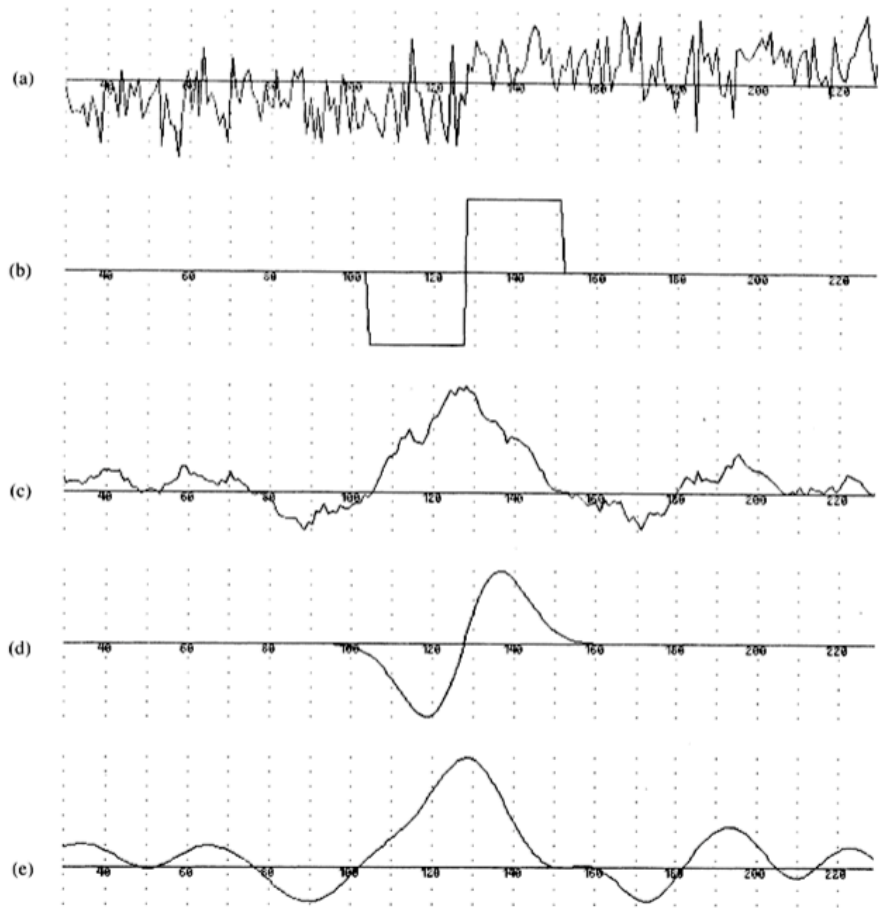

Fig. 1. (a) A noisy step edge. (b) Difference of boxes operator. (c) Difference of boxes operator applied to the edge. (d) First derivative of Gaussian operator. (e) First derivative of Gaussian applied to the edge.

Figure 2.8 The detection problem is formulated as follows: We begin with an edge of known cross-section bathed in white Gaussian noise as in Fig. l(a), which shows a step edge. We convolve this with a filter whose impulse response could be illustrated by either Fig. 1 (b) or (d). The outputs of the convolutions are shown, respectively, in Fig. l(c) and (e). We will mark the center of an edge at a local maximum in the output of the convolution. The design problem then becomes one of finding the filter which gives the best performance with respect to the criteria given below. For example, the filter in Fig. l(d) performs much better than Fig. l(b) on this example, because the response of the latter exhibits several local maxima in the region of the edge (Canny, 1986).

2.6.2 Model For Looming Estimation

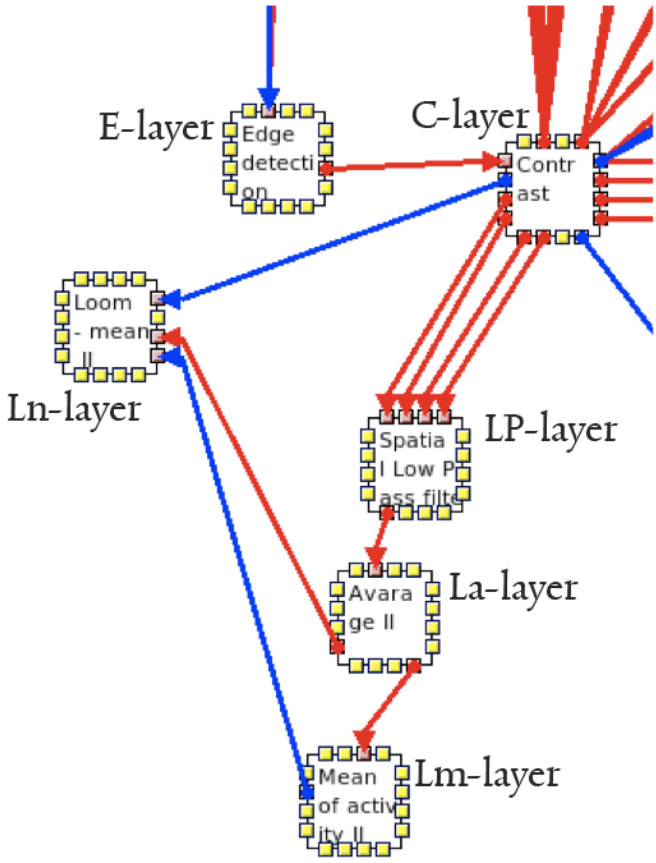

Figure 2.9 IQR model for looming estimation – The edge detection neuron group is a neuron group of 40 by 30 pixels and detects the edges in the images. The contrast filter excites neurons with an absolute value, either 0 or 1. The spatial low pass filter inhibits all lateral movements. The Average neuron group combines all activity from the spatial low pass filter and presents them as one single value. The mean of the activity is calculated by having high membrane persistence. This compensates for moving edges on the background. The loom – mean is the final calculating neuron that represents the amount of looming measured. This neuron also compensates for the amount of edges by having a shunting inhibition.

The LGMD model of the locust migratorio largely inspires the model for looming estimation. The system is broken down in different layers and their functions, all carefully coded in order to keep overview over the references. The model is broken up into seven separate layers. The Video Input layer (V-Layer), the Edges layer (E-layer), the Contrast Layer (C-Layer), the Low Pass filter (LP layer), the average Looming layer (La-Layer), the Loom mean layer (Lm-Layer) and the net loom layer (Ln-layer). The V-, E–, and C-layer are taken from the visual input system. The working of the other layers is described here.

2.6.2.1 LP-Layer

The spatial low-pass filter should filter out all fast movements in order to ignore fast lateral movements. It is important to filter the fast movements, but to keep enough movement to detect the looming. The building principle is similar to the high pass filter, but here delayed images have less value than the current image, and there is no inhibition.

$$LP_{(x,y)} = \left(C_{(x,y)} \cdot 0.6\right) + \left(C_{(x,y)}^{1\partial} \cdot 0.4\right) + \left(C_{(x,y)}^{2\partial} \cdot 0.3\right) + \left(C_{(x,y)}^{3\partial} \cdot 0.1\right)$$

2.6.2.2 La-Layer

We wanted one neuron to measure the looming on a 0.00 – 1.00 scale, therefore, we created a Looming average layer.

$$La = \sum Lp_{(x,y)} \cdot 0.0025$$

2.6.2.3 Lm-Layer

In order to get only looming spikes, we created a group to calculate the mean of the activity. We created a neuron with a high membrane persistence and low excitatory gain.

$$Lm = (La \cdot 0.1) + (Lm^{\partial 1} \cdot 0.9)$$

2.6.2.4 Ln-Layer

Finally the Ln-Layer gives the net loom by adding subtracting the mean from the average and compensating for the amount of edges. This compensation for the amount of edges is a form of gain control. When there are a lot of edges, excitation is higher and gives thus extra excitation to the looming sensitive unit. In order to compensate for this, a shunting inhibition is used. This is a synapse that connects to the dendritic tree rather than the neuron itself. In the biological model the synapse releases Cl^- ions decreasing the resistance of the neurons connection (Prescott & De Koninck, 2003). This creates an effect that can be best compared with a division.

$$Ln = \frac{10La - 9.8Lm}{C}$$

2.6.3 Model for Speed Estimation

The most famous model for visual speed estimation model is the Reichardt correlation model. This model is based on the comparison of an image with a delayed image. If the right correlation is found between the two images, the direction and speed can be extrapolated from this. In figure 2.7 shows an ePuck with a side-mounted camera measuring forward speed by comparing the two images. The model in this figure would only measure a forward motion. A backward motion would require the shift in the other direction. Additionally, we would need a different delay for each measured speed, resulting in the creation of multiple synapses and neurons. The result of the Reichardt correlation model would give both a reliable speed estimation and direction estimation. We have practiced using this model using a camera with a resolution of 40x30, but we very promptly ran into some methodological objections. If for example, you want to create reliable speed estimation, where you combine the results of three different delays, and you want to extrapolate the angular velocity from that, you would have three times 40x30 neurons including synapses. We would have already 3600 neurons just for the estimation. You would have to multiply that for each direction you want to be able to measure speed in, so on a two dimensional plane that would be easily eight directions, and that makes 228800 neurons, and then you would still have to extrapolate the correct velocity from that.

The second objection is that the Reichardt correlation model assumes that you have a logical shift, meaning to say that you have a predictable direction of the movement. However, if you have a movement in any direction other than the ones predefined, this correlation model does not pick up anything.

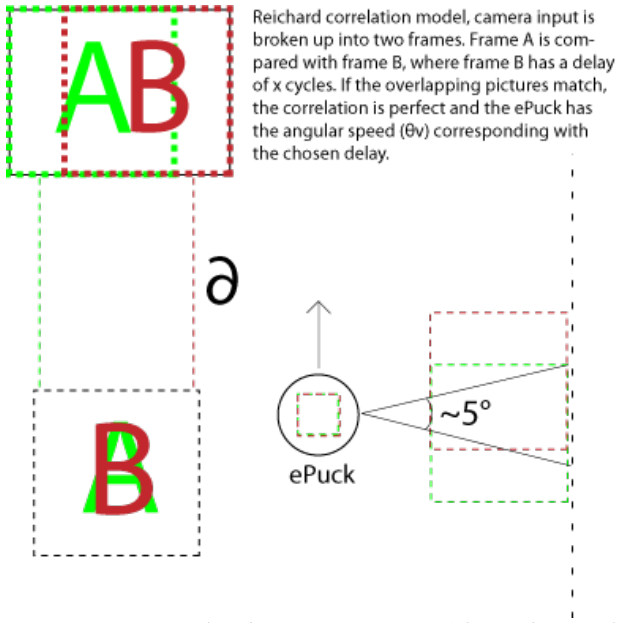

Figure 2.10 a graphical representation of how the Reichardt correlation model would function in a real world experiment, using an ePuck as carrier for the camera.

After some tests with the Reichardt correlation model we decided to take a completely different (and minimalistic) course. This resulted in a model that estimates speeds in pixels shifted using no more than 1201 neurons and 12 synapses. The rationale behind this model is a series of delays in each synapse, each one cycle more. At slow speeds that will mean that a lot of the shifts are in fact overlapping. The pre-synaptic neuron group gives only edges in a Boolean value. The postsynaptic neuron is also an integrate-and-fire neuron group, resulting in that overlapping excitations give one value. When the shifts grow bigger, what you get at higher velocities the excitations will not occur overlapping but cascading, increasing the amount of excitation of the postsynaptic neuron group. Additionally the current frame gets inhibited from the postsynaptic neuron group not exciting anything when there is no shift. A calculating neuron then receives the excitation of the postsynaptic neuron and a shunting inhibition from the pre-synaptic neuron normalising the amount of edges. The result is an accurate speed measure using only 1201 neurons.
This model has two layers only, the motion sensitive array (M-layer) and the normalisation (N-layer) computation. This model takes the visual input from the system dedicated for visual input processing.

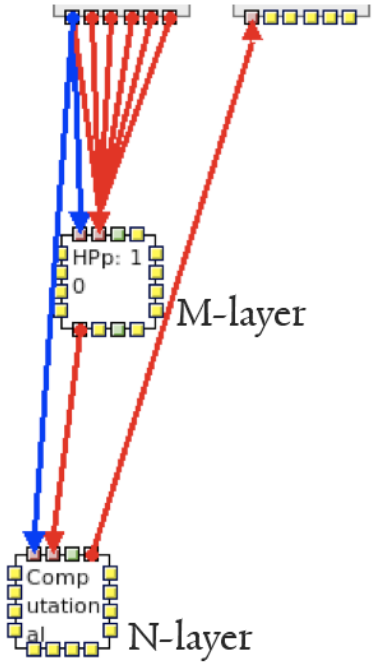

Figure 2.11 The IQR set-up for speed estimation, containing 12 excitatory connections one inhibitory connection and one shunting inhibition. The HPp:10 neuron group takes all excitations with their delay and puts them in an array. This array gives a fixed amount of excitation for each active neuron. The computational neuron group consists out of one single neuron getting the sum of the excitation and dividing it by the average amount of edges using a shunting inhibition connection.

2.6.3.1 M-layer

The motion sensitive array makes a map of the movement by exciting the neurons and have them excited having a delay. The excitation stays in total ten cycles. If the visual input is gradually moving, the array has more excitation because the edges will excite a neighbouring neuron in the next cycle (see figure 2.10). This motion sensitive array has a size of 40 by 30 neurons, and the excitation follows the following formula:

$$M_{(x,y)} = 0.1 \cdot C^{\partial 1}_{(x,y)} + 0.1 \cdot C^{\partial 2}_{(x,y)} + 0.1 \cdot C^{\partial 3}_{(x,y)} + 0.1 \cdot C^{\partial 4}_{(x,y)} + 0.1 \cdot C^{\partial 5}_{(x,y)} + 0.1 \cdot C^{\partial 6}_{(x,y)} + 0.1 \cdot C^{\partial 7}_{(x,y)}$$
$$+0.1 \cdot C^{\partial 8}_{(x,y)} + 0.1 \cdot C^{\partial 9}_{(x,y)} + 0.1 \cdot C^{\partial 10}_{(x,y)} - 0.1 \cdot C_{(x,y)}$$

The output of the neurons is normalised as either 0 or 1, meaning that neuron with an excitation of 0.2 gets normalised to give a reading of 1.0.

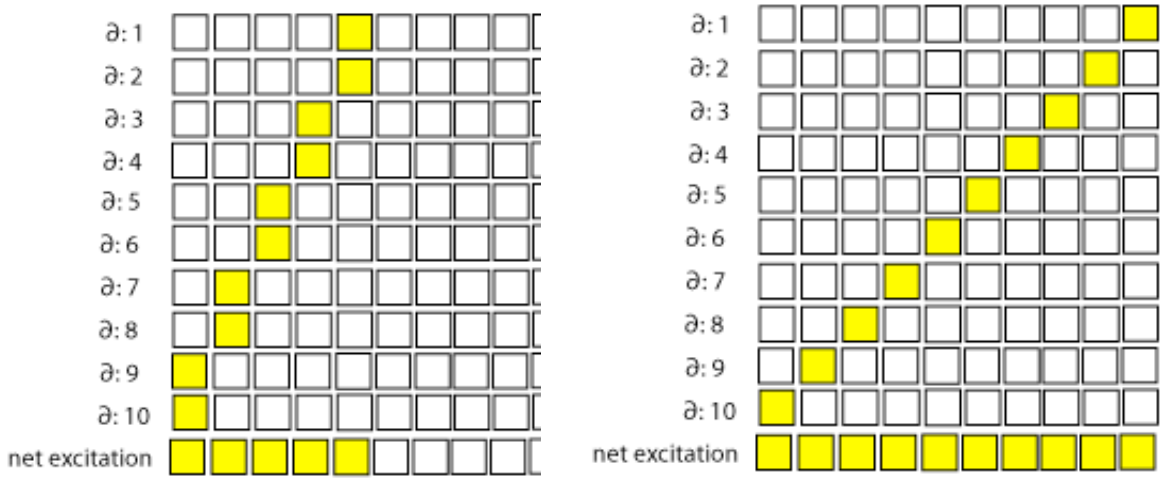

Figure 2.12 Graphical representation of motion sensitive array

2.6.3.2 N-layer

The amount of edges during flight varies a lot depending on the surface, and this gives a different read-out. If we would not compensate for these edges the read out of the motion sensitive array would not make sense as an indication for speed. Therefore, we compensate for this by dividing the amount of excitation from the M-layer with the total amount of edges. The N-layer consists out of one single computational neuron that calculates the sum of the pixels from the M layer and divides them with the sum of the pixels from the contrast layer.

$$N = \frac{\sum M_{(x,y)}}{\sum C_{(x,y)}}$$

2.6.4 Models for direction estimation

Because the speed estimation model used does not provide any information about the direction, a separate direction estimation model has to be implemented. In an ideal situation eight directionalities should be able to be measured (referred to as, north, northeast, east, southeast, south, southwest, west and northwest). This however is a very complicated problem. Horizontal and vertical linear shifts are very easy to detect. Diagonal motion is already a little bit harder to detect, and rotational movements are really head braking. In the process we have developed three direction estimation models and compared their results. The first to models look very much alike and are shown in a graphical representation on figure 2.10 and it is based on comparing one pixel shift. The winner of the matrix will indicate the direction (as described in the paragraph of the interpolation method). We have implemented two slightly different approaches for the diagonal movements, one linear and one curved. The third model is completely different and works with an interpolation model.

2.6.4.1 Winner-takes-all matrix

Each of the used models uses a standardised winner-takes-all matrix. This matrix takes a vector and calculates the maximum and minimum value as illustrated in figure 2.10. The effect of which is a function that reads out multiple signals giving the maximum and minimum value.

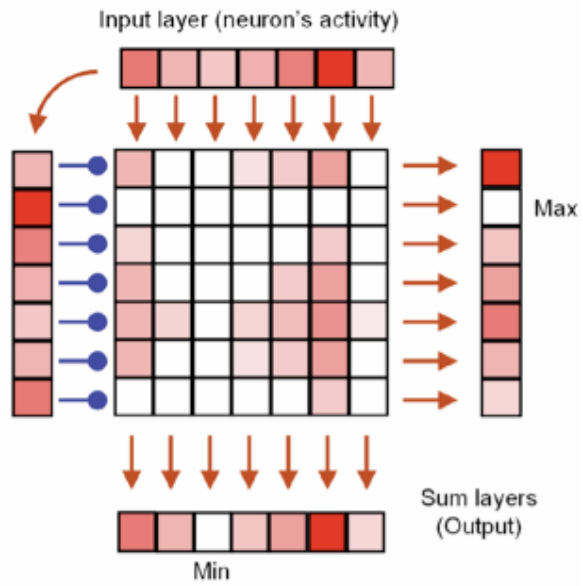

Figure 2.13 Structure of the proposed Winner-Takes-All network. The input layer of size n is used to excite and inhibit a n x n population of neurons. All the neurons in a column are excited by a corresponding excitatory neuron, at the same time that all neurons in a row are inhibited by a corresponding inhibitory neuron. If the readout is done by summing the neural activity of a row, the position of the row that has zero activity corresponds to the winner, since all neural activity has been suppressed by the activity of the winner. Instead, if the readout is the sum of columns, the column with zero activity corresponds to the neuron with the lowest response, since all neurons are able to inhibit its activity (Bermúdez i Badia, 2006).

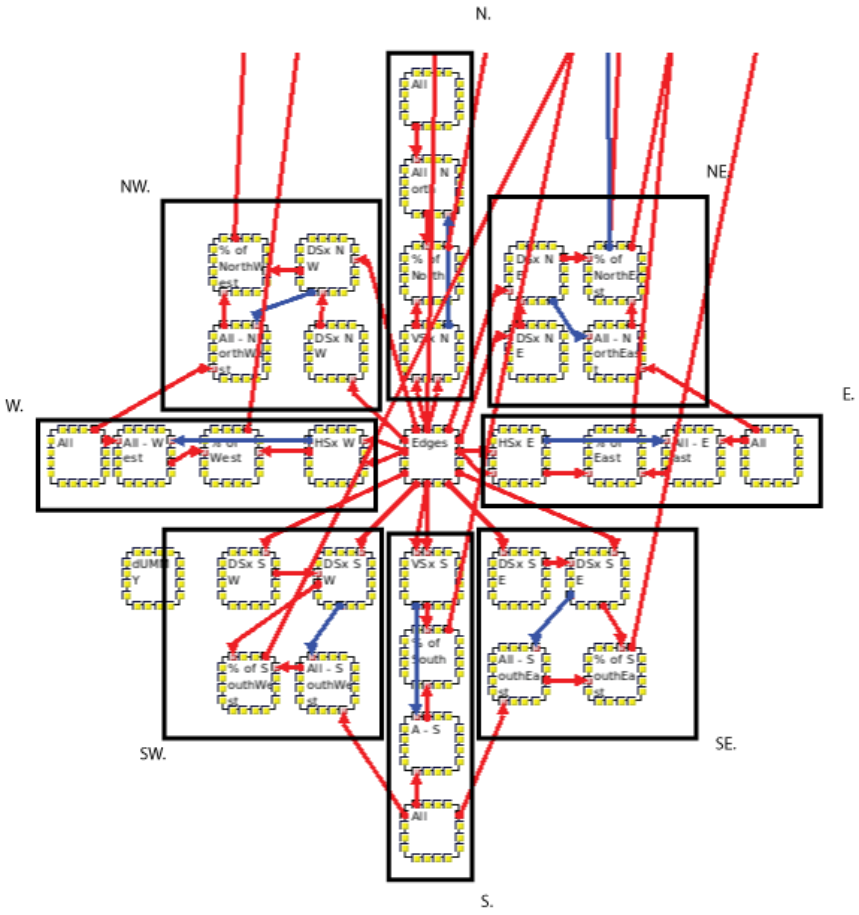

Figure 2.14 IQR set-up of direction estimation using direction sensitive neurons for each direction. Each of the outlined groups has its own sensitivity with a correspondent direction. Box NE works for example by comparing a linear shift and a rotation in case of Model I.

2.6.4.2 *Model I Curved*

When the blimp turns, the result is not a diagonal or linear movement, but it has a curve. Therefore, the diagonal motion detector does not trigger this motion.

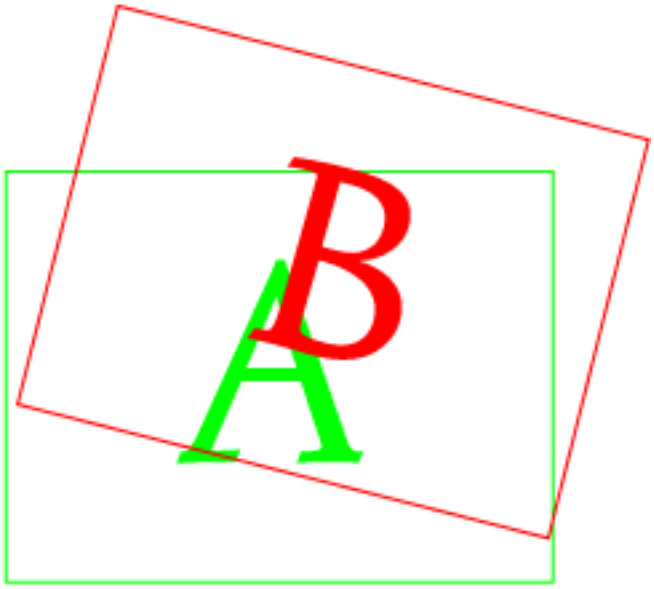

Figure 2.15 Visual representation of a curved shift

This model tries to detect this motion by creating a shift in two parts. One part is a linear forward motion, and the other part is a circular motion clockwise or counter clockwise, depending on the direction. The clockwise shift is created using a complicated tuple model, where the pixels are shifted in four different ways. Right hemisphere was shifted one pixel up, the top hemisphere was shift one pixel to the left, the left hemisphere was shifted one pixel down and the bottom hemisphere was shifted on pixel to the right. As with the linear detection model the correct pixels are summed up to a group of neurons representing the percentage of correct pixels and these are summed into the winner take all matrix.

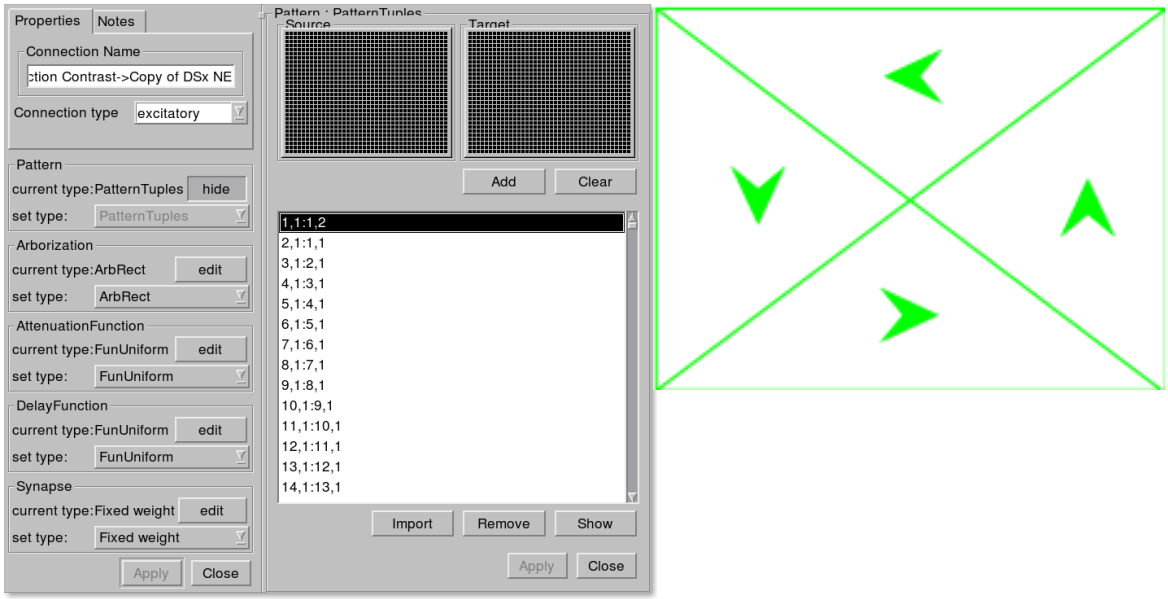

Figure 2.16 On the left the tuple configuration of the synapse inducing rotational sensitivity, on the right the graphical representation of this tuple model.

2.6.4.3 Model II Linear

Figure 2.10 shows a very detailed calculation of the number of correct pixels in a certain direction. A one-pixel shift is used for each of the directions. The linear direction is represented using a diagonal shift. The direction sensitive neurons receive two synapses from the edge detection. One synapse has the current image. The other synapse has one pixel shift including a delay of one cycle. This should give an excitation for all of two for all the correct pixels and an excitation of 1 for incorrect pixels. The pixels that have an excitation of zero are pixels that give the correct blank value. This blank value gets inverted and reaches a summation neuron group that calculates the percentage of correct pixels. The result of this visual compass shows eight neuron groups with excitation, the sum of each of the neuron groups is placed in a Winner Take All matrix, and the winner is considered to indicate the correct direction.

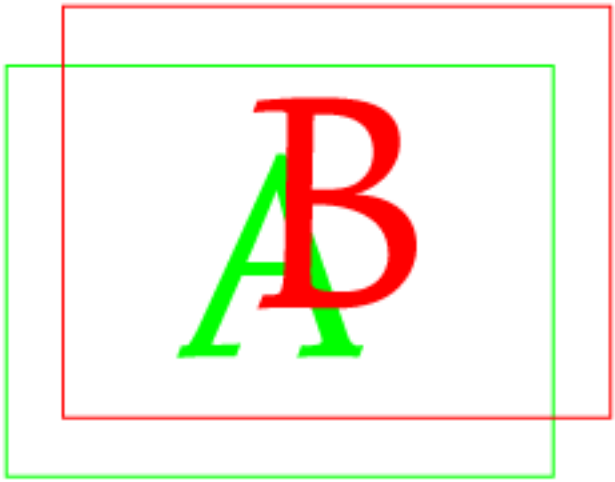

Figure 2.17 visual representation of linear shift.

2.6.4.4 Model III Interpolation

This last model is the cheapest model concerning the amount of neurons used. As we can see in figure 2.15, there are four neuron groups that measure direction; these neuron groups are each 1200 neurons big. These neuron groups estimate the direction by comparing the current image with the previous cycle with one pixel shift in a direction. The four directions measured are north, south, west and east. The neuron groups use the synapses to overlay the two images, the neuron groups have an excitatory gain of 0.5 and a threshold of 0.50001, so it only excites if the overlap is correct. This model assumes that when flying in a diagonal direction, for example, north-west, both neuron groups for north and west excite more than the other two. We used this principle to make a very simple and straightforward interpolation method that takes in this case half the excitation from north and half the excitation from west with a slightly higher gain than four directions measured. This should result in a reliable winner for diagonal movements. The winner take all matrix calculates the winner and gives a representation of one neuron per direction excited, a computational neuron translates this in a linear threshold providing a value between 0.045 and 0.36 for indication in degrees which direction the blimp is flying. This measure can be directly used as a value in proportional[5] error correction. The model is divided in four calculating parts.

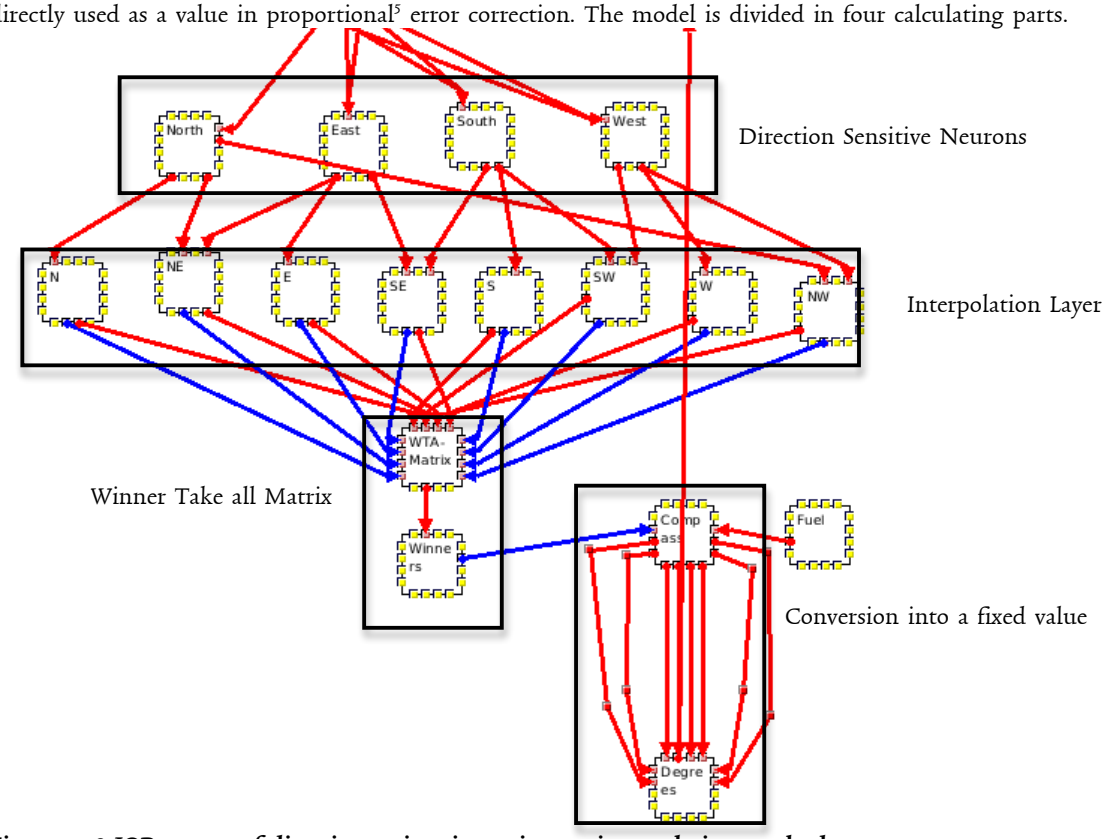

Figure 2.18 IQR set-up of direction estimation using an interpolation method.

Calculations Direction Sensitive Neurons:

$$North = Thresh_{0.50001}\left(.5 \cdot C_{(x,y)} + .5 \cdot C^{\text{old}}_{(x,y+1)}\right)$$

$$South = Thresh_{0.50001}\left(.5 \cdot C_{(x,y)} + .5 \cdot C^{\text{old}}_{(x,y-1)}\right)$$

$$West = Thresh_{0.50001}\left(.5 \cdot C_{(x,y)} + .5 \cdot C^{\text{old}}_{(x+1,y)}\right)$$

$$East = Thresh_{0.50001}\left(.5 \cdot C_{(x,y)} + .5 \cdot C^{\text{old}}_{(x-1,y)}\right)$$

Calculations Interpolation Layer:

$$N = North$$
$$NE = .5 \cdot North + .5 \cdot East$$
$$E = East$$

[5] http://en.wikipedia.org/wiki/Proportional_control

$$SE = .5 \cdot South + .5 \cdot East$$
$$S = South$$
$$SW = .5 \cdot South + .5 \cdot West$$
$$W = West$$
$$NW = .5 \cdot North + .5 \cdot West$$

Conversion Into a Fixed Value:
The winner column gets inhibited from a direction matrix mapped to the correct direction. This mapping is done in one neuron group in such a way that opening the space-plot would give a readable preview of the direction. The matrix that gives this representation looks as follows:

$$\begin{matrix} Thresh_1(1-Winner_1) & Thresh_1(1-Winne_2) & Thresh_1(1-Winner_3) \\ Thresh_1(1-Winner_4) & & Thresh_1(1-Winner_5) \\ Thresh_1(1-Winner_6) & Thresh_1(1-Winner_7) & Thresh_1(1-Winner_8) \end{matrix}$$

In order to get one neuron that can be used for a read out of direction estimation, this matrix gets normalised to a single value. Creating eight synapses that map the source neuron to one single target neuron each having their own correspondent weight gives the correct output. We chose a value mapping that can be easily understood as a direction in degrees. Therefore, the values are between 0.045 (northeast) and 0.360 (north).

$$Degrees = (0.315 \cdot Compass_{(1,1)}) + (0.36 \cdot Compass_{(1,2)}) + (0.045 \cdot Compass_{(1,3)}) +$$
$$(0.27 \cdot Compass_{(1,2)}) + (0.09 \cdot Compass_{(3,2)}) + (0.225 \cdot Compass_{(1,3)}) +$$
$$(0.18 \cdot Compass_{(2,3)}) + (0.135 \cdot Compass_{(3,3)})$$

2.6.5 Model for course stabilisation

As seen in figure x.x all sensory processing comes together in a behavioural layer that controls the motors. This layer combines all information it gets from the outside world trough its sensors and measures this against its desires. If, for example, the robot wants to move in a straight direction with a fixed speed it sets its direction parameter to the desired direction with a value for speed (figure x.x).

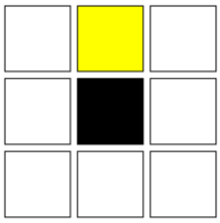

Figure 2.19 Set point representation. The excitation gives the desired velocity, and the coordination of the neuron excited the desired direction.

It is important to have standardised the representation of speed and direction for calculating the error, figure x.x shows how the sensory reading of direction and speed are combined into a single representation of 8 neurons.
The speed measure and direction estimation are combined both in a 9x9 matrix where the speed excites the neuron that is equal to the estimated direction.

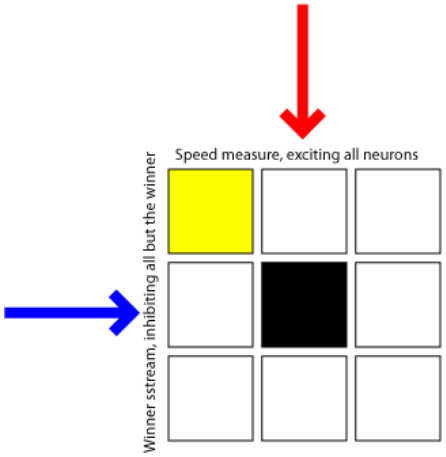

Figure 2.20 Combining Direction Estimation with Speed measure, in this case going NW

Since we have now the representation of the sensory processing and the set point normalised we can build up a model that calculates the error in terms of speed and direction between the measured direction and speed and the desired direction. If taking zero as a baseline value, a subtraction of the estimated value of the desired value gives the negative error (e^-), and a subtraction of the desired value from the estimated value gives the positive error (e^+). With this calculation we have a proportion of the error that we can integrate over time to balance out noise. This is done by creating an inverse subtraction layer to another set of neurons with a high membrane potential and therefore integrates the error over time (figure x.x).

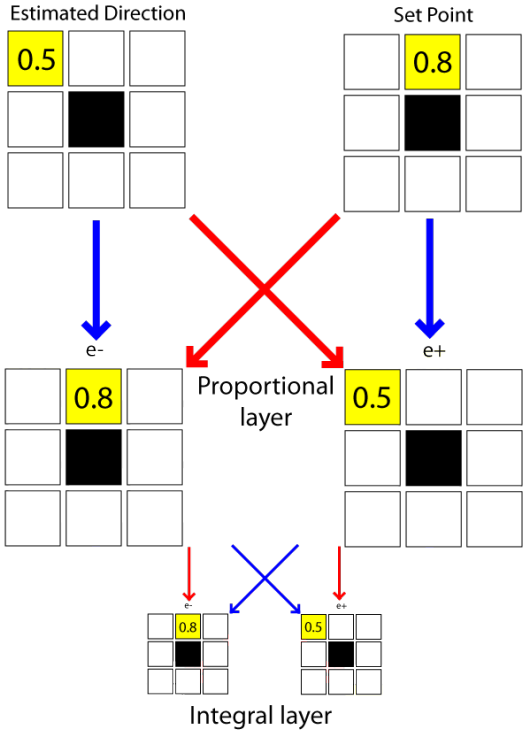

Figure 2.21 System to calculate motor compensation

These calculations have to be translated into motor actions. Therefore, a tuple mapping has to be made between the Integral layer and pre-motor output. In the case of the driving robot there are only four motor neurons used. Figure x.x show the mapping between the Integral layer and the pre-motor layer.

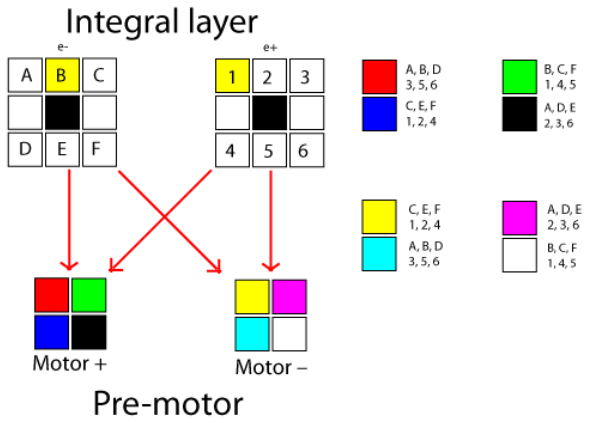

Figure 2.22 Mapping between the integrated error and the pre-motor compensation

The motor+ and the motor- have a gain to regulate the strength in the before they come together in the motor calculation layer, here other pre-motor streams come together and calculate the net motor output. The gain is a way to tune input streams. Mappings between the integral layer and the pre-motor layer used here only serves compensation for forward and backward motion. Thus, is perfect for testing the integration of different models in a driving robot, if a flying blimp is going to be used a wider range of the sensor processing should be used.

The only way for the system to compensate for errors is if it provides a bias to the motor calculations, the bias is taken directly from the set-point and has a gain high enough to make it drive, and low enough to have good trajectory compensation. For our first experiment we used a 0.25 gain on the integral stream and a 0.5 gain on the bias stream.

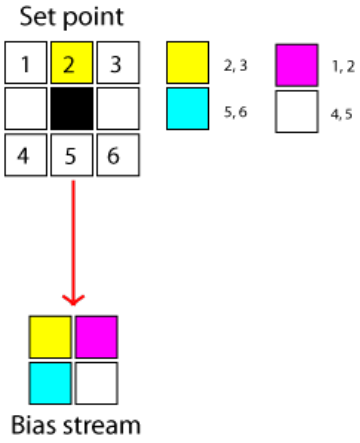

Figure 2.23 Motor Bias mapping

Figure x.x shows the motor calculation of the complete stream following the example as in figure x.x.

2.7 PRE-DEFINED FLIGHT TRAJECTORIES

Besides the standard blimp simulator, we also implemented specific flight simulators for testing separate subsystem. In order to measure the accuracy of the sub-systems each of the specific flight simulators have a solid flight plan. The independent variables are hard coded in the flight simulator so they provide a reliable comparison.

The first subsystem is the flight simulator and is specially designed to test the direction estimation model. This flight simulator does not listen to any motor commands and only flights according to the predefined variables. The directions are hard coded and there is a small detail in direction when choosing the linear flight plan versus the curved flight plan. The only difference between the curved and the linear flight plan is the way it turns. When the simulation is flying 45° for example the linear flight plan will show a diagonal movement, this is very useful to test if it can detect drift during flight. In the curved flight plan on the other hand flying in the direction of 45° will show a circular motion depending on the curve radius. This test is necessary because it gives a realistic view of a blimp flying in a certain direction.

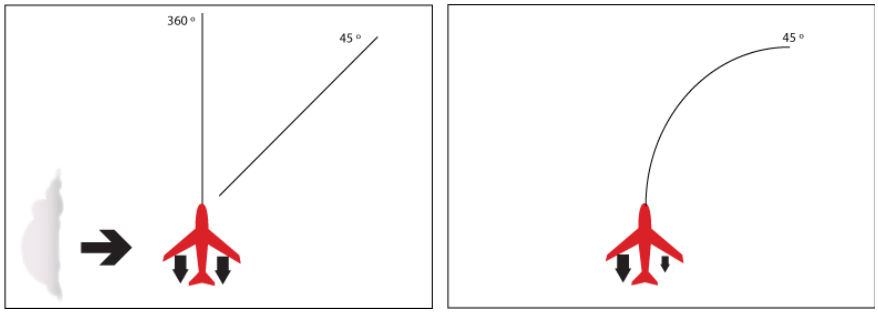

Figure 2.24 Linear Flight versus Curved flight.

Figure 2.16 shows how real life situations can constitute linear and curved movements. A forward motion induced by having the two engines powering forward flight, together with side wind will give a result direction of 45° linear. Having one engine with more power than the other will induce a curved motion, which are both a radial turn and a forward motion. This is how the blimp would fly in real life, in wind still conditions, when one engine gets more power than the other.

The independent variables that can be controlled are the starting altitude, forward and backwards speed and the side speed. The flight simulator can be also used to measure the accuracy of the compass at different speeds. The direction follows a fixed sequence. For linear movement: 270°, 45°, 180°, 315°, 135°, 225°, 90° and for curved movement: 270°, 45°, 180°, 315°, 135°, 225°, 270°, 90°. This slight difference is to reduce the risk of the simulation to fly off the field.

The speed simulation is a flight simulator that flies at designated speeds in predefined patterns and does not listen to any motor commands. When using the speed test simulator, a value for the velocity can be chosen, this value is then used as the multiplier for the simulated speed. If for example a multiplier of two is used the simulated velocity is 16, 14, 12, 10, 8, 6, 4, and 2. This velocity is expressed in shifted pixels. The flight simulator also follows a fixed flight plan, flying in a square, 270°, 360°, 90°, 180°, 270°, 360°, 90° and 180°.

The altitude test simulator is the least complicated of all. It flies the virtual blimp to the middle of the field, and goes respectively to seven different altitudes. The blimp flies to the centre on altitude 1.0, which is the ground, and from there it increases the height with a factor eight to zero (the ceiling). The simulation relies on the model to extract the right features to estimate the height. Because the model has to extract the features, the simulator listens to motor commands. A model can be built, for example, that flies in a sinusoidal pattern on the same height. The looming sensitive unit can be tested in this way, if it can be used as an altitude estimation model.

2.8 PILOT: DRIFT COMPENSATION

A first pilot model shows the effectiveness of the sensory processing. This pilot aims at using the direction estimation tool as a measure for course correction. For this pilot we build a very simple reflex model for course stabilisation and tested this to see if it can compensate for drift. The blimp is supposed to fly straight, whilst it gets side wind induced error. The model compensates by first calculating the error, and then use that error input to compensate for the drift. When the drift gets compensated, the error gradually disappears hence the drift is coming back. Therefore, we developed a ten-cycle routine that allows the blimp fly straight uncompensated for ten cycles, and then compensates for the drift for another ten cycles. The calculated error is used in a proportional control unit compensating for the error by giving one of the engines more power.

$$P_{out} = K_p e(t)$$

Where P_{out} is the output of the proportional controller, Kp is the proportional gain and e(t) is the instantaneous process error at the given time. This is calculated using the desired flight direction minus the estimated flight direction.

2.9 FIRST REAL WORLD EXPERIMENT

Before having to chase the blimp around we decided to do a real world experiment with a driving robot in the driving robot arena. The objective of this experiment is to see in the separate models combined allows the robot to compensate for drift induced by a motor output differential between the left motor and the right motor. The arena is designed to show the offset between the start and the end point after driving over the track. Variables used in this experiment are the starting offset 40cm, the set point 1 North, the model (with or without error correction) as an independent variable and the finish offset as a dependent variable. The experiment is run ten times as a repeated measure. The control experiment consists out of the same set-up, and executes the same model, except it does not have the integral stream connected to the motor calculation group and therefore only drives the engine with the bias.

3 RESULTS

3.1 DIRECTION ESTIMATION

In a real world situation, a flying blimp is not only propelled by its own effectors, i.e. its own propellers, but there are several factors that influence the direction of the blimp. The blimp robot can therefore not rely on its own states activating the propellers as a direction indication. The three proposed models have been tested using a simulation. The simulator provided the real directions, whilst the different models estimated the direction. In order for the blimp to have a successful course correction, the mean of the error should not be bigger than 22.5°, the system is based on direction estimation in steps of 45° degrees, thus the error processing model could be designed to snap to these values. The error variability should not be bigger than 45°. Model I has rotation sensitivity and direction estimation looking at eight different directions. Therefore, it should perform better on the curved flight path and should be able to indicate each direction. Model II is equal to model I, with as only difference that the neuron groups that are sensitive for 315°, 225°, 135° and 45° are working with a diagonal shift rather than a rotational shift. This should increase the performance using the linear flight simulation and decrease the performance using the curved flight simulation. In the experiments the measure is taken from the neuron in the model that represents the direction in degrees.

3.1.1 Experiment one: Curved Flight plan

The three proposed models have each their own rationale behind the development of it. In order to find out which of these three models perform best on a curved flight plan, our first experiment is designed to test each of the models on the same flight plan. Model I is a curved-sensitive model and should therefore outperform Model II in this experiment. Model III is an interpolation method and should therefore outperform both Model I and Model II. The data collected is plotted in table 3.1.

Table 3.1 Data Experiment one, on a curved flight plan. Comparing the mean and standard deviation of each model, in each direction.

Model	Error	W (270°)	NE (45°)	S (180°)	NW (315°)	N (360°)	SE (135°)	SW(225°)	E (90°)
I	Mean	2.2959	136.84	3.2143	33.980	-0.45918	50.510	-39.490	-30.170
	Std.	32.975	12.791	22.616	33.614	4.5457	31.167	26.849	62.055
II	Mean	3.2143	131.79	1.3776	35.357	-1.8367	48.214	-45	-25.568
	Std.	17.018	43.466	13.637	29.077	8.9497	22.616	0	85.978
III	Mean	-2.2959	1.3776	1.3776	0.45918	-2.2959	0.91837	0.45918	0
	Std.	0	13.637	13.637	4.5457	22.728	9.0914	4.5457	0

This table quickly gives an overview of the performance per model for each of the direction, and it shows that the third model performs best in all the estimated directions. The mean of the error is in all of the cases the lowest and the standard deviation in most of the cases. When plotting these values into bar plots, the results become a lot clearer.

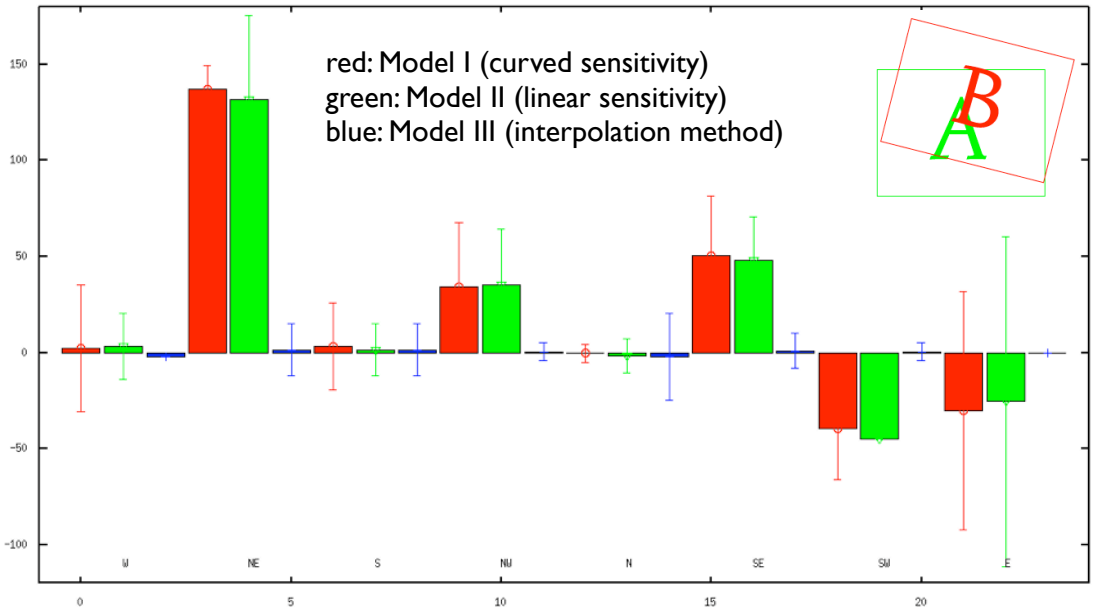

Figure 3.1 Comparing the performance of the three models in a curved flight plan. The vertical axis shows the mean of the error in degrees, the horizontal axis show the actual flight direction.

This bar plot makes it possible to see an a-symmetry in the reading. This indicates that the model is more prone to error depending on the direction to estimate. The difference between the curve-sensitive model and the linear-sensitive model is insignificant, which is not inline with the hypothesis that Model I should outperform Model II.

3.1.2 Experiment two: Linear Fight Plan

The second experiment is to test the performance of the three models on a linear flight plan. Needless to say, this experiment should show a better performance for Model II. Nevertheless, still Model III should outperform the previous two. Table 3.2 shows the detailed figures of the experiment.

Table 3.2 Data Experiment two, on a linear flight plan. Comparing the mean and standard deviation of each model, in each direction.

Model	Error	W (270°)	NE (45°)	S (180°)	NW (315°)	N (360°)	SE (135°)	SW(225°)	E (90°)
I	Mean	2.2959	101.48	3.2143	-26.173	-3.6735	-2.2959	-21.582	8.6932
	Std.	32.975	48.489	39.437	64.267	16.692	68.497	52.187	37.901
II	Mean	3.2143	59.235	-1.8367	-15.612	-6.4286	34.898	-18.827	10.227
	Std.	17.018	62.178	28.105	52.136	17.096	96.174	50.847	37.821
III	Mean	-2.2959	1.3776	1.3776	0.45918	-2.2959	0.91837	-1.3776	0
	Std.	0	13.637	13.637	4.5457	22.728	9.0914	13.637	0

Looking at the means and standard deviations of Model III in table 3.2 and comparing that with the data of table 3.1, data shows that the interpolation model has exactly the same performance of each of the directions. This reading is completely normal for linear movements, but for the curved movements that is rather remarkable. The reading on the direction east shows a discrepancy ruling out the an error in data analysis. Plotting this data in a bar plot gives use a better overview of the performance (figure 3.2).

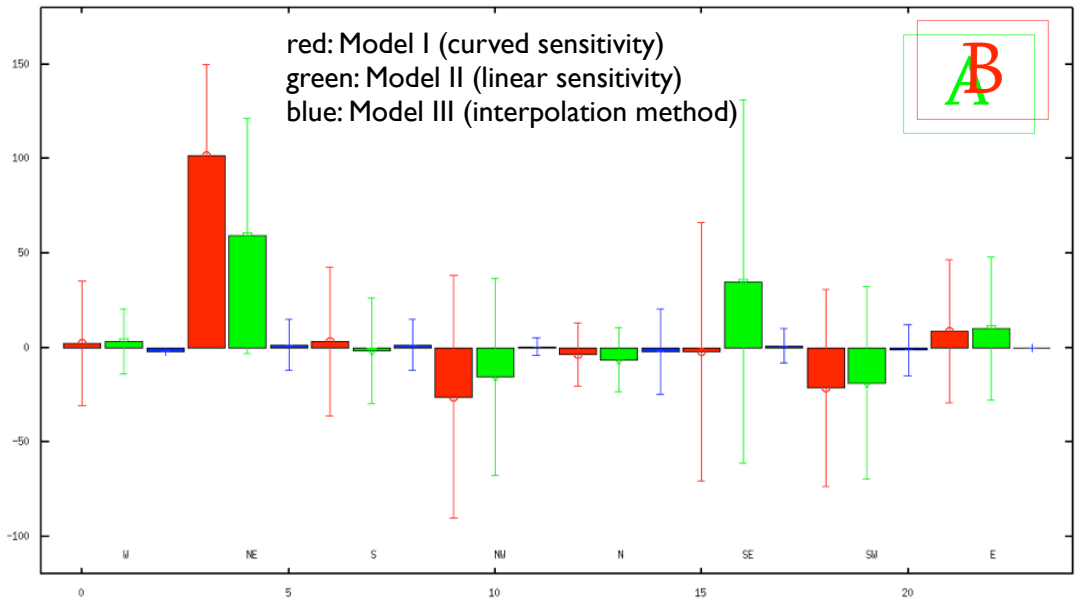

Figure 3.2 Comparing the performance of the three models in a linear flight plan. The vertical axis shows the mean of the error in degrees, the horizontal axis show the actual flight direction.

It shows that Model II outperforms Model I and Model III outperforms Model II. As expected it also shows that Model III gives almost the same reading as in the previous experiment. Also this experiment shows an a-symmetry in the reading, showing that the performance varies depending on the flight direction.

3.2 LOOMING ESTIMATION

The rationale behind using the looming detection as a model for extracting altitude properties from the environment is based on the fact that if objects are close by they appear bigger. Objects close by have a bigger angular size. Therefore, if the blimp would fly in sinusoidal wave, the features on the ground would induce a looming response, the amount of looming should account for the altitude. More looming equals lower altitude.

The ideal result for an experiment based on this principle is shown as a yellow line in figure 3.4. This data is the control data for the other plots and shows the real altitude taken from a module to groups neuron group. The figure shows that the set up flies first close to the ground and gradually ascends into higher altitudes. The read out of the looming should give a similar pattern, but as figure 3.4 shows, the opposite is measured.

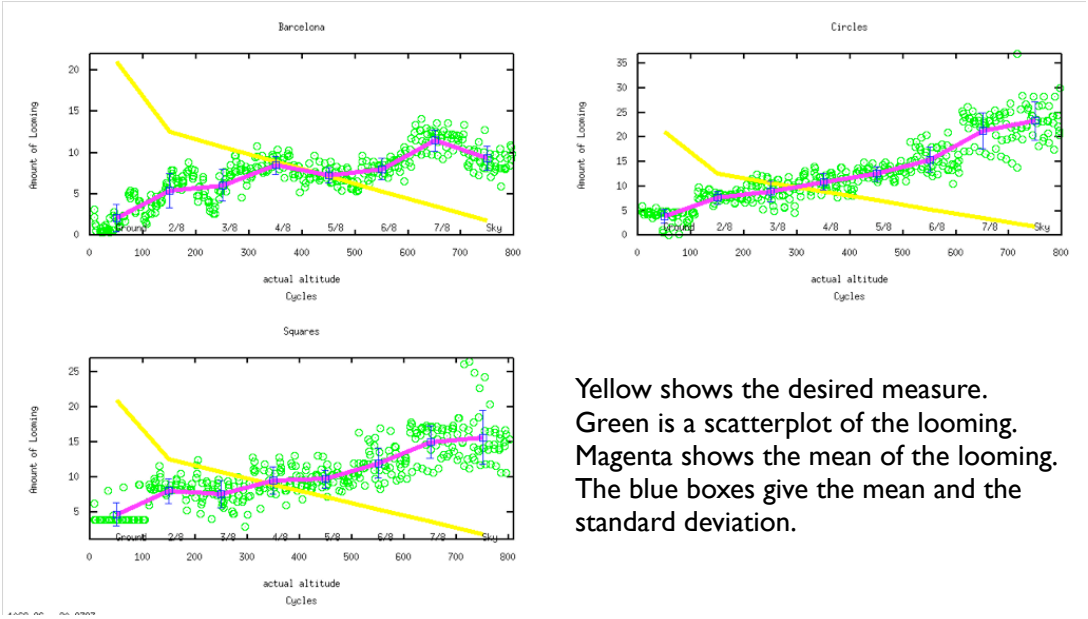

Figure 3.3 Detected looming using when having a sinusoidal flight above the patterns

The measure shows a gradual increase in loomed edges. A logical explanation for this could be that when the blimp flies at a higher altitude there are more objects that appear to be smaller, and therefore there are more edges. Doing two more experiments, excluding the possibility of fallacy of the looming model, can proof this hypothesis.

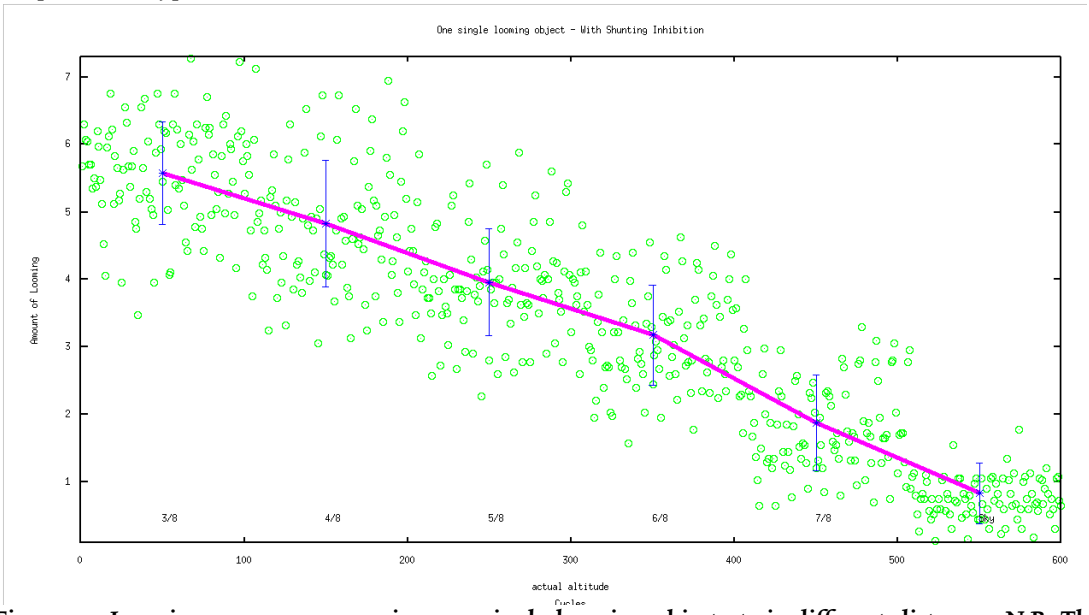

Figure 3.4 Looming measure, measuring one single looming object at six different distances. N.B. The first 200 cycles were used to bring the virtual blimp into position and are therefore not a part of the looming experiment.

Figure 3.4 shows that the model for detecting looming objects works correct. The measure is taken by letting the blimp fly in a sinus over one single circle shape object. The height of the blimp was each increased every 100 cycles. The reading shows conformity shows the ideal reading for altitude similar to the control value in figure 3.3, thus looming decreases with altitude. Another experiment is done in order to reproduce the trend in figure 3.3.

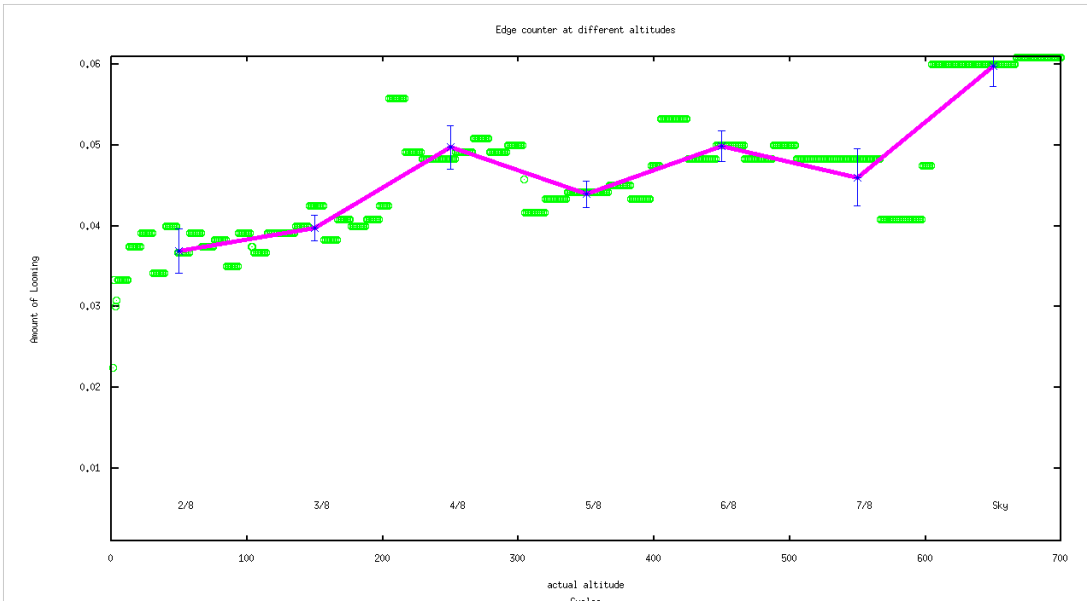

Figure 3.5 An edge counter, counting the amount of edges over the map Barcelona at seven different altitudes, N.B. the first 100 cycles were used to bring the blimp into position and are therefore not a part of the counted edges

Figure 3.6 shows the counted edges at seven different altitudes and this shows a steady increase in edges with altitude. Thus, the looming estimation could work perfect for collision avoidance, but not for height estimation.

3.3 SPEED ESTIMATION

When moving over a natural surface, the amount of edges are always different, and the shape of the objects cannot be predicted. Therefore, a gradient-based speed estimation model should account for those edges. This is done using the shunting inhibition. Shunting inhibition is an important method for normalising the calculations with the amount of edges. In order to show the importance of the shunting inhibition, we did an experiment having the shunting inhibition disabled (Figure 3.6).

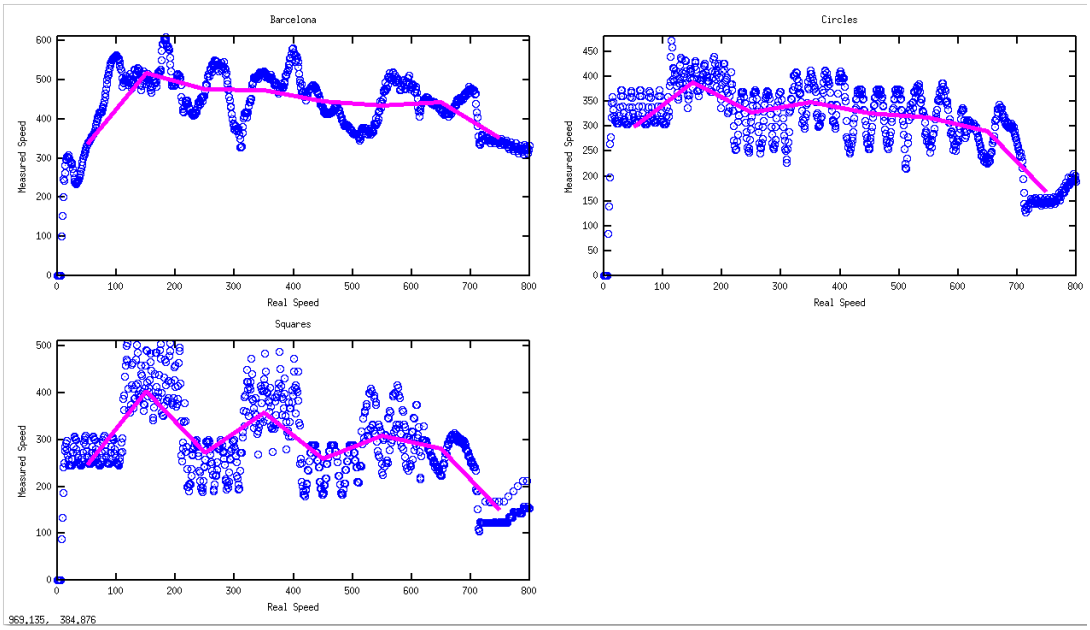

Figure 3.6 Speed measure on three different textures without using shunting inhibition, top-left: flying over Barcelona, top-right: flying over circles, bottom-left: flying over squares. The vertical axis shows the excitation of the speed measuring neuron. The horizontal axis shows the cycle, the velocity decreases every 100 cycles.

The model for speed estimation assumes that faster movement would create more excitation because the edges will cascade and spike in a fixed amplitude. Figure 3.6 show that this approach works, but also that

the reading is not reliable. When the blimp is flying over a specific type of surface the amount of edges is never the same. Therefore, the reading does not provide any data as long as no form of normalisation is used. Even the type of surface can be recognised when looking at figure 3.6. Flying over circles give a more or less sinus shaped scatter plot whilst flying over squares shows a very rigid shift in the amount of edges. Moreover, the results of figure 3.6 show that although the speed is reduced, the amount of excitation stays the same. Thus, shunting inhibition is necessary in order to create a speed estimation model that is reliable.

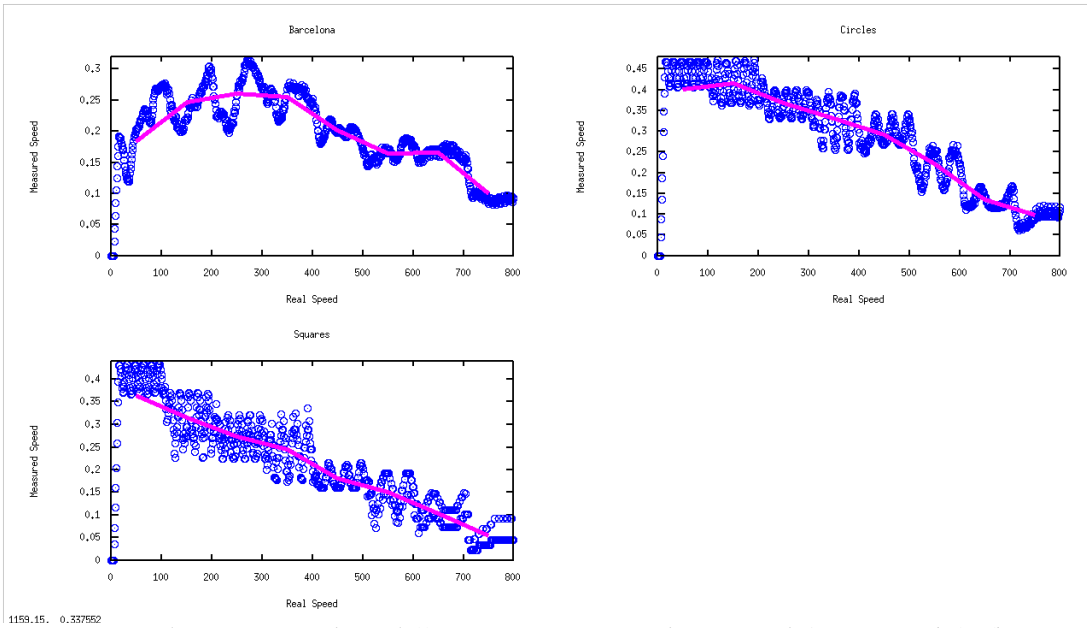

Figure 3.7 Speed measure on three different textures using shunting inhibition, top-left: flying over Barcelona, top-right: flying over circles, bottom-left: flying over squares. The vertical axis shows the excitation of the speed measuring neuron. The horizontal axis shows the cycle, the velocity decreases every 100 cycles.

Figure 3.7 shows a read out with shunting inhibition and therefore has a normalisation on the edges. This shows that the approach works. The figure also shows that there is a ceiling in the reliability of the speed estimation. The relatively low resolution of the image processing allows for a saturation of the neuron group, and when the neuron group is saturated the maximum speed measure is reached. Measuring speed is always relative to the amount of edges, but having a good speed measure means that the values should always stay in the same range. Therefore, another experiment has been done letting the blimp fly over three different altitudes, figure 3.8 gives the results of this experiment.

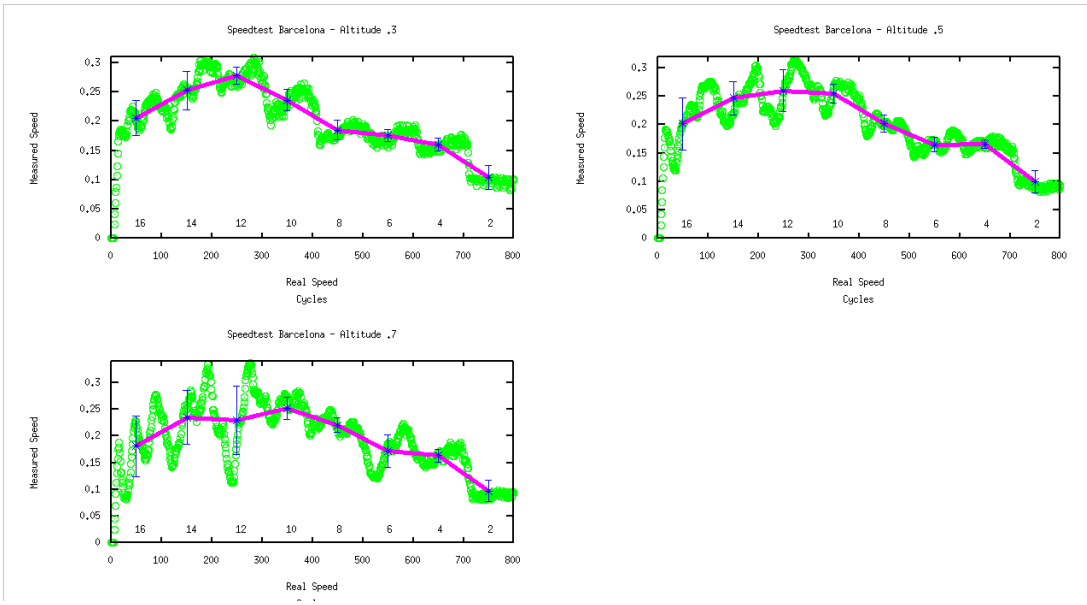

Figure 3.8 Speed measure over Barcelona using three different altitudes, top-left: flying over with an altitude of 0.3, top-right: flying over Barcelona with an altitude of 0.5, bottom-left: flying over

Barcelona with an altitude of 0.7. The vertical axis shows the excitation of the speed measuring neuron. The horizontal axis shows the cycle, the velocity decreases every 100 cycles.

When putting these values in a table it becomes visible that the vales for the same speeds on different surfaces are more or less the same. Table 3.3 shows a comparison of different values and they are more or less the same for each of the velocities, with some discrepancies.

Velocity	Barcelona	Barcelona (altitude 0.3)	Circles	Squares
16	0.20160	0.20515	0.44000	0.40051
14	0.24714	0.25213	0.415000	0.31581
12	0.26014	0.27764	0.36932	0.27356
10	0.25522	0.23572	0.33094	0.24516
8	0.20224	0.18485	0.29210	0.18045
6	0.16529	0.17552	0.22290	0.15028
4	0.16553	0.16011	0.13663	0.10375
2	0.10005	0.10459	0.098234	0.054221

An important comparison is looking at the difference in measure for Barcelona between reading 2 and 4, and between 8 and 10, the step is almost of the same size. However, these steps cannot be found between all the velocities, but the pattern is regular enough to have reliable speed estimation.

3.4 DRIFT COMPENSATION

Having the capability of sensing is of course only useful if it can be used to do something interesting. Figure 3.14 shows therefore the first attempt to use direction estimation for drift compensation. The data shows that the first half of the journey goes flawlessly, but in the end the blimp looses its heading direction and overcompensates. The challenge is to find a good balance between reading out data and compensating flight. In this case, the blimp flies 10 cycles without compensation, followed by 10 cycles of compensated flight. This trend can be found back when looking at the first two sinusoidal shaped readings.

This pilot test shows that the direction estimation can form a basis for more feedback systems leading to the integration of the sensory processing, and the development of sensor reflexes.

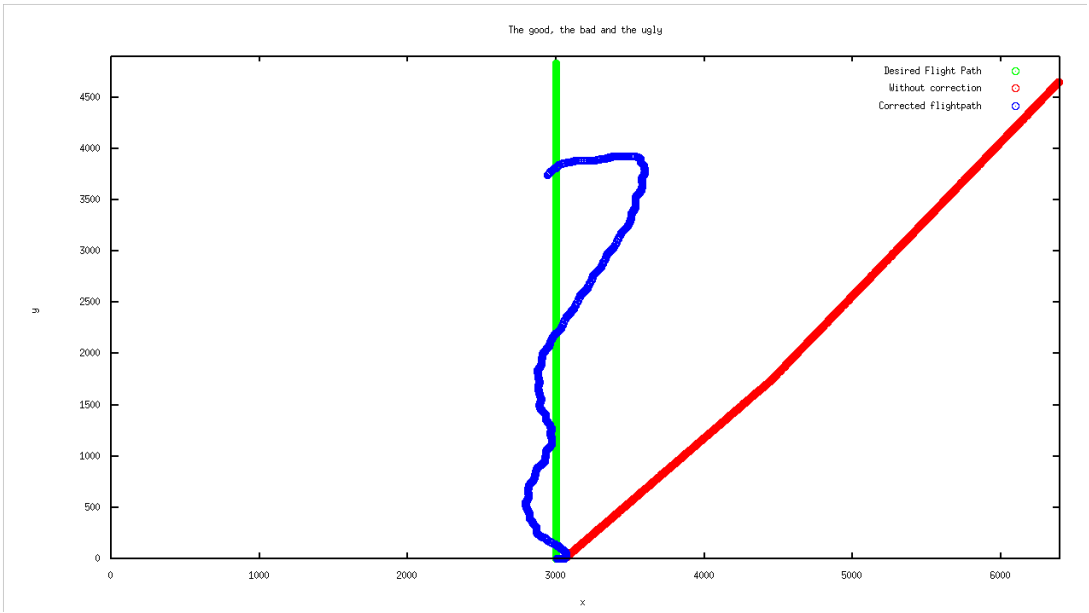

Figure 3.9 the direction estimation is used to compensate for drift. Green is the desired direction, red is the direction without compensation and blue is the corrected direction.

3.5 RESULTS OF THE FIRST REAL WORLD EXPERIMENT

After running the experiments we clearly see that the drift compensation helps the robot to go into the right direction. Obviously, the robot has a bias towards the error-induced direction as seen in the majority of the results, but this should be compensated later by adding a path integration layer. Table x.x compares the dependent values between the test and the control group.

Table 3 Data of real world experiment with a driving robot

Trial number	Test (error correction)	Control (fixed motor output)
1	+14 cm	Not finished
2	-10 cm	Not finished
3	-3 cm	Not finished
4	-7 cm	Not finished
5	-2 cm	Not finished
6	-2 cm	Not finished
7	-23 cm	Not finished
8	-9 cm	Not finished
9	-10 cm	Not finished
10	-7 cm	Not finished

4 CONCLUSION AND DISCUSION

Simplifying sensors and extending sensors has been one of the main drives of research for many years. The complex environments robots have to operate in require multipurpose and flexible sensing, preferably standardised and simplified. Many authors have been inspired using insect based models for visual image processing. Besides that the development of these kinds of sensors can play a big role in the development of advanced sensors, they can also help us understand how the brain works. Inspired by the McIvers Buena Vista Sensing Club hypothesis, building more advanced sensors and better sensory processing leads by itself to the discovery of cognitive systems. Wit this thesis we put a first step in the direction of building sensory processing and autonomous behaviour, and we managed to develop several essential base systems.

In order to reach our goal, we decided to develop a system to operate a flying blimp robot. This robot uses one main sensor, a camera and four main effectors, the propellers. Because using a real blimp for developing the models we decided first to make a detailed flight simulator. This allowed us to develop a subset of a model and test this in a controlled environment. With the help of a flight simulator we were able to control all the independent variables of the separate experiments.

During the development of this thesis several separate models have been created to process visual input. We have developed one model for speed estimation, one model for looming estimation and three different models for direction estimation. Each of these models is developed following being inspired by biological systems, and looking at the state of the art. All forms of redundancy are removed and the levels of complexity reduced to a minimum, this is shown in the development of the direction estimation and speed estimation systems, where expensive alternatives have been traded for less complex solutions. This reduces the chance of failure and increases the flexibility of each of the sub systems. A good example of this is the separation of the speed estimation model from the direction estimation. Traditional models based on the Reichardt correlation model compares a delay of large frames that have a shift with the current frame. This does should give a reliable estimation of speed and direction, but this also assumes that the brain is wired to make these complex comparisons. Additionally, it reduces the flexibility of the sensors used, if for example a forward mounted camera suddenly gets mounted faced downwards, a whole new mapping of the Reichardt correlation model is necessary. The speed estimation model used in this thesis is not depended on direction because it does not compare complete frames. Instead it looks at how often a pixel appears in a unique spot of the compound eye in a period of ten cycles. The result gives one measure that is independent from direction and previous calibration, which would make it a universal speed measure. The result section teaches us that it is very important to normalise the amount of edges. A realistic surface such as a map of Barcelona never has the same amount of features on the ground for speed estimation. As long as there is a steady amount of features large enough, the speed estimation is always equal and reliable. Only exceptional cases, such as no edges or a quick change in the amount of edges would require a recalibration of ten cycles that the Reichardt correlation would not need, but the advantages way up against this disadvantage.

Of the three models for direction estimation one clearly stood out as being more efficient than the others. The third model, the interpolation model is a combination of linear sensitivity with an interpolation method. The four linear direction work with a comparison of on pixel shift in each direction and the resultant direction can be extrapolated from that. The results of this model show a very small error and the mean of the error is in all of the cases smaller than 3°. The results did show however an a-symmetry in the reading. This a-symmetry is the result of how the model calculates the direction in degrees from the winner-takes-all matrix. It sums up the winner direction multiplied by the degrees it represents, but there are moments when there is more than one winner. For example, the matrix both excites 45° and 135° when the simulation is flying 45°, then the calculating neuron calculates 180° thus an error of 135°. If the simulation is flying in the de direction of 180° and the matrix excites both 180° and 90° it will sum up to 270° which gets compensated by the normalisation never to give an error of more than 180° and is thus represented as an error of 90°. This explains the a-symmetry in the results. If the model is more accurate the amount of errors decrease, and therefore also the a-symmetry and this a-symmetry is therefore no problem in the assessment of our models using this data. Furthermore, we could argue that the direction estimation uses a form of Reichardt correlation introducing the same disadvantages as the combined model. But this is not the case. Instead of having several shifts with several delays, the direction estimation has only a one-pixel shift with one cycle delay. The difference between the other directional read-outs is big enough to create a winner take-all-matrix that gives the correct reading. The direction estimation gives a reliable feedback of direction and drift. The principle of the direction estimation is an egocentric compass, and the world moves around this point. It should therefore not be confused with a normal compass. The blimp could very well fly west whilst the direction estimation indicates 360°. In this

case the blimp is flying forwards in the direction west. This is very important to know when the direction estimation is used for drift compensation. When relying only on the direction estimation, the flight direction can be only compensated into a resultant direction. If the blimp has been flying in a wrong direction for a while, the direction estimation has no power to compensate for such an error. This means that the development of the sensor with direction estimation opens the door for a higher cognitive model. If the blimp should be able to compensate for large shifts in direction it should develop a path integration tool using the direction estimation. Using the direction estimation as a first step of higher cognitive systems such as path integration follows McIver's *Buena Vista Sensing Club* hypothesis. Therefore, having developed this direction estimation tool is a first step into the direction of higher cognitive systems. If the process of course correction would be combined with a process of path integration, the system would acquire the possibility or error computation of the correction. It would then have the ability to calculate that correction x gives error y on the flight path. The system could be developed in order to try a couple of different flight corrections and would be able to link this with the help of the error calculation to a form of learning. After this process of learning, the modelled brain would be able to execute the right course correction without having to calculate the error. This would be learning cognition in the way Dreyfus describes it. A network has encountered a particular situation similar to a previous one and will reproduce the same appropriate behaviour.

The looming estimation unit shows a good reading for detecting looming. The results section shows a linear response. However, the looming estimation model was initially designed to estimate the altitude of the blimp. The rationale behind this was that objects at higher altitude appear smaller, and therefore produce less looming. Figure 3.3 in the results section shows that this principle works both in theory and in practice, but in a real world model there are more than just one object. When flying over Barcelona an increase in altitude means an increase in edges, and more edges mean more looming. This the looming estimation does not suffice for height estimation on a 2D plane. On a 3D plane, objects closer to the blimp would induce more looming, thus it would be possible to use this model for collision avoidance. Also if an object would move towards the blimp independent from its background it would detect looming and could initiate a collision manoeuvre to avoid the object. Looking back at the biological model, this is what looming estimation is used for. Thus the model for looming could be implemented in the blimp for collision avoidance, but for height estimation another biological model is necessary.

One of the most important deliverables of this thesis is the blimp flight simulator. The flight simulator gives a realistic virtual environment for a robot blimp to operate in. Creating new models, and finding new ways for the robot blimp to operate can sometimes take days from design to working model. If then, a model has some problems, it can take another couple of days before the model is updated so it would work in real life. If a blimp has to be filled with helium and calibrated for each step in this process, the speed in which to develop these models is reduced significantly. Additionally, having the blimp to make controlled movements in order to test a subset of a system is quite complicated in a during a real-world test. The blimp flight simulator can be easily adapted to facilitate all of these needs. However, does this mean that one of the important criteria to make artificial systems is jeopardized? Looking back at one of Brooks requirements: *"At each step we should build complete intelligent systems that we let loose in the real world with real sensing and real action."* raises concern to which extend this form of modelling is a fallacy. That falls or stands with the quality of the flight simulator, and the way it is used. This flight simulator is built in order to mimic all characteristics of the real world. It includes the physical laws of acceleration, it uses real life satellite images as a ground surface and it has real gravity. It also allows the air density to be controlled and to induce side wind. If the flight simulator is used correctly, i.e. use real satellite images and make the blimp perform a real life action, Brooks criterion is met. Hence, the models developed for the flight simulator should work in real life models as well. The drift compensation experiment shows that it is possible for the blimp to perform a real world task in the flight simulator.

4.1 DRIVING ROBOT

The results of the driving robot experiment shows that the systems are robust enough to implement them in the flying blimp. However, the models used to build the robot do not fully cover the functionality of the blimp. Therefore, these models should be updated before testing them on the real world blimp. Additionally, the experiment with the driving robot only tells us something about the behaviour on the horizontal plane, the vertical plane is not included. Thus, these are boundaries to overcome to make the models work on the blimp.

4.2 FURTHER DEVELOPMENTS

4.2.1 The Flight Simulator

If the flight simulator model should be further developed, it will be interesting to look at some very important features. Currently the flight simulator has only side wind from the linear directions, and this side wind only induces a sideways drift in the simulation. However, a real life implementation of the blimp, the world model would have several forms of wind interference. Wind can move diagonally on the blimp inducing a diagonal drift or maybe even a rotation. Additionally, vertical wind can also take the blimp of course, to a different altitude. Therefore, the first step in the further development of the simulator would be implementing more realistic wind interference.

4.2.2 Visual Altitude Estimation

The looming has been used to experiment whether it can be used for altitude estimation. This, however, proofed not to be the case. The looming sensitive unit developed would work fine for collision avoidance, but we need to develop an altitude estimation tool. Therefore, one of the main challenges for future development is to find a way to estimate altitude through visual input.

4.2.3 Moth like behaviour

In order to fully validate the developed models and proof that this can form a basis for an autonomous flying robot, more autonomous reflexes need to be developed. Integration of the speed estimation, looming estimation and direction estimation should constitute for a solid robot that can fly according to bio-inspired behaviour. The next step would be developing a system that detects the wind direction and fly lateral to this wind direction. Building such a system can help mimic moth like behaviour for detecting odour plumes (Li, Farrell, & Cardé, 2001).
This system could then be used as the final validation showing that all related sub-systems work. Additionally, the blimp flying this system could then form a base for other insect like behaviours of robots, perhaps mimicking the behaviour of the moth.

5 BIBLIOGRAPHY

Bermúdez i Badia, S. (2006). *The Principles of Insect Navigation Applied to Flying and Roving Robots: from Vision to Olfaction.* Zurich: ETH ZURICH.

Bernardet, U., Blanchard, M., & Verschure, P. F. (2002). IQR: a distributed system for real-time real-world neuronal simulation. *Neurocomputing* , 1043-1048.

Borst, A. (2007). Correlations versus gradient type motion detectors: the pros and cons. *Philosophical Transactions of the Royal Society* , 369-374.

Brooks, R. A. (1991). Intelligence without representation. *Elsevier* , 139-159.

Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Trans on Patern Analysis and Machine Intelligence* , 697-698.

Cariani, P. (1998). *Epistemic Autonomy through Adaptive Sensing.* Boston: Eaton Peabody Laboratory for Auditory Physiology.

Deckert, C. (2001, November 1). *Eye Design Book Chapter 3*. Retrieved September 3, 2009 from Eye Design Book: http://www.eyedesignbook.com/ch3/eyech3-c.html

Douglass, J. K., & Strausfeld, N. J. (1995). Visual motion detection circuits in flies: peripheral motion computation by identified small-field retinotopic neurons. *Journal of Neuroscience* , 5596-5611.

Dreyfus, H. L. (1996). *The Current Relevance of Merleau-Ponty's Phenomenology of Embodiment.* Berkeley: University of California.

Gabbianni, F., Krapp, G. H., Hatsopoulos, N., Mo, C.-H., Christof, K., & Gilles, L. (2004). Multiplication and stimulus invariance in a looming-sensitive neuron. *Journal of Physiology* , 19-34.

Graham, L. (2002). How not to get caught. *Nature* .

Haag, J., Denk, W., & Borst, A. (2004). Fly motion vision is based on Reichardt detectors regardless of the signal-to-noise ratio. *PNAS* .

Koekoek, S. K., Huslcher, H. C., Dortland, B. R., Hensbroek, R. A., Elgersma, Y., Ruigrok, T. J., et al. (2003). CErebellar LTD and Learning-Dependent Timing of Conditioned Eyelid Responses. *Science* , 301.

Li, W., Farrell, A. J., & Cardé, R. T. (2001). Tracking of Fluid-Advected Odor Plumes:Strategies Inspired by Insect Orientation to Pheromone. *Adaptive Behavior , 9* (3-4), 143-170.

MacIver, M. A. (2006). *Neuroethology: From Morphological Computation to Planning.* Evanston: Northwestern University.

O'Shea, M., & Rowell, C. H. (1975). *Nature* , 53-55.

Pinker, S. (1997). *How the Mind Works.* New York: WW Norton.

Prescott, S. A., & De Koninck, Y. (2003). Gain control of firing rate by shunting inhibition: Roles of synaptic noise and dendritic. *PNAS* , 2076-2081.

Rajesh, S., David, O., & Derek, A. (2002). Elaborated Reichardt correlator for velocity estimation tasks. *SPIE* .

Thorndike, E. L. (1911). Retrieved *6 1*, 2009 from Classics in the History of Psychology: Thorndike (1911) Chapter 5

Verschure, P. (2008). Cognitive Science & Psychology: Mind, Brain and Behaviour. *Universitat Pompeu Fabra.* Barcelona.

Wikipedia. (2009). *Bash*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Bash

Wikipedia. (2009). *C++*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/C++

Wikipedia. (2009). *Classical conditioning*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Classical_conditioning

Wikipedia. (2009). *Firebug*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Firebug

Wikipedia. (2009). *Firefox*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Firefox

Wikipedia. (2009, September 3). *Google Maps*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Google_Maps

Wikipedia. (2009). *ImageMagick*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/ImageMagick

Wikipedia. (2009). *OpenCV*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/OpenCV

Wikipedia. (2009). *Perl*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Perl

Wikipedia. (2009). *Wget*. Retrieved September 3, 2009 from Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Wget

Yamawaki, Y., & Toh, Y. (2008). *Responses of descending neurons to looming stimuli in the praying mantis Tenodera aridifolia.* Springer-Verlag.

Yue, S., & Rind, F. C. (2006). Collision Detection in Complex Dynamic Scenes Using an LGMD-Based Visual Neural Network With Feature Enhancement. *IEEE TRANSACTIONS ON NEURAL NETWORKS , 17* (3).

6 APPENDICES

6.1 ACCELERATION/DECELERATION CODE

C++:

```
if (deceleration== 0 && acceleration == 0) {
      if (HnetMotor > 0 && speed >= 0) {
            netProp = AIRDENSE + HnetMotor;
      }
      else {
            netProp = dAIRDENSE + HnetMotor;
            netProp *= -1;
      }
   }
else {
      if (HnetMotor < 0) {
            netProp = deceleration;
      }
      else {
            netProp = acceleration;
      }
}
```

6.2 MAPMAKER TOOL

```
#!/usr/bin/perl
#
#NL: http://mt3.google.com/mt/v=w2.92&hl=en&x=525&y=331&z=10&s=Ga
#Brussel: http://khm0.google.com/kh/v=37&hl=en&x=8386&y=5491&z=14&s=G
#
#Starting tile of the GMaps x,y:
#
$x = 8386;#$x = 1035;
$y = 5491;#$y = 764;
#Size of your downloaded images each tile = 256 pixels 25t X 25t -->
6400px X 6400px
#
$sizeX = 25;
$sizeY = 25;
$End_x = $x + $sizeX;
$End_y = $y + $sizeY;
#Base URL:
#The base URL you copy paseted from the Google Image tile...
#Make sure you strip the GET variables x and y and add a double
escape character ("\\") before the &
#If you don't know how to get the URL of your ImageTile get Firebug
(getfirebug.com) and click on inspect
#click on the image tile where you want to start, and voila you've
the image URL in the bottom.
#e.g.
#1.  http://mt3.google.com/mt/v=w2.92&hl=en&x=1035&y=764&z=11&s=Galil
<-- Copy pasted image URL;
#2. http://mt3.google.com/mt/v=w2.92&hl=en&z=11&s=Galil  <-- Removed
x, y variables (Remember to put the values in the $x and $y variable
of this script)
#3.  http://mt3.google.com/mt/v=w2.92\\&hl=en\\&z=11\\&s=Galil    <--
Added escape characters in front of the &
#
```

```
@GImageURL   =   "http://khm0.google.com/kh/v=37\\&hl=en\\&z=14\\&s=G";
#@GImageURL                                                          =
"http://mt3.google.com/mt/v=w2.92\\&hl=en\\&z=11\\&s=Galil";

$c = 0;
$gridX = 0;
for ($i = $x; $i < $End_x; $i++) {
      #print "x: ", $y, "\n";
      $gridX++;
      $gridY = 0;
      for ($j = $y; $j < $End_y; $j++) {
            $gridY++;
            #print "y: ", $y, "\n";
            $c++;
            print "wget -O GridX", $gridX, "Y", $gridY, ".jpg " ,
@GImageURL, "\\&x=", $i, "\\&y=", $j, "\n";
      }
}

print "echo ", $c, " Images retrieved\n";
print "echo Start stitchin\n";

for ($j = 1; $j <= $sizeY; $j++) {

        print "convert ";
        for ($i = 1; $i<=$sizeX;$i++) {
                print "GridX", $i, "Y", $j,  ".jpg ";
        }
        print "+append row", $j, ".jpg \n";
}

print "echo remove the Grid Files...\n";
print "rm GridX*\n";

print "convert ";
for ($i = 1; $i <= $sizeY; $i++) {
        print "row", $i, ".jpg ";
}
print " -append GMapsImage.jpg\n";
print "echo remove Row files\n";
print "rm row*\n";
```

6.3 PRE-TESTS EDGES

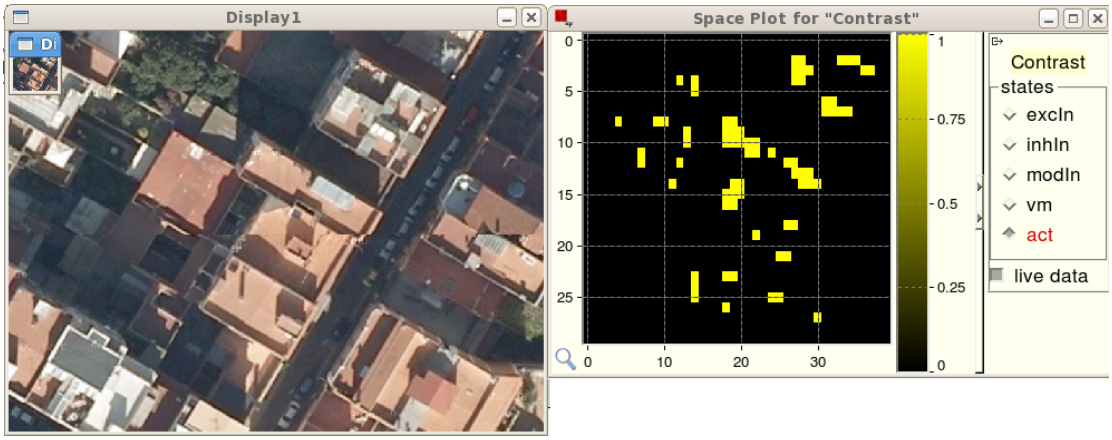

Figure 6.1 Snapshot of Barcelona with the generated edges

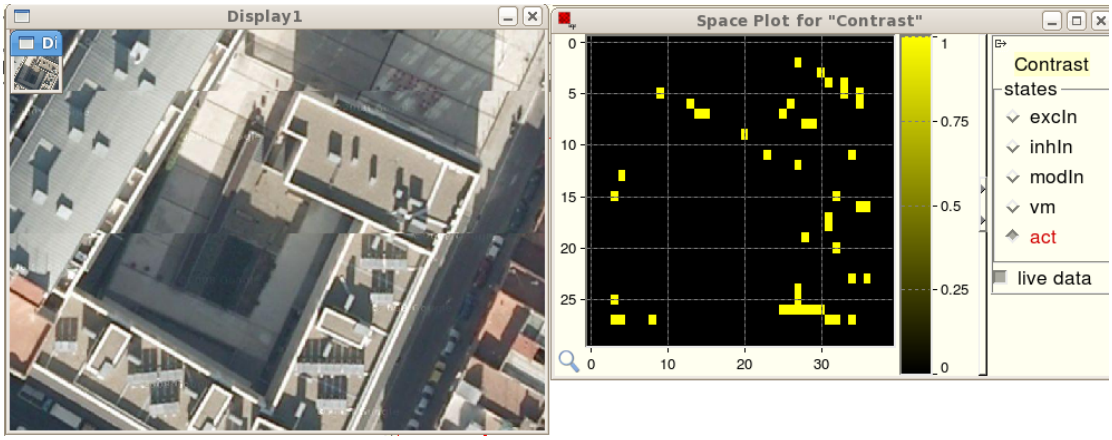

Figure 6.2 Snapshot of Barcelona with the generated edges

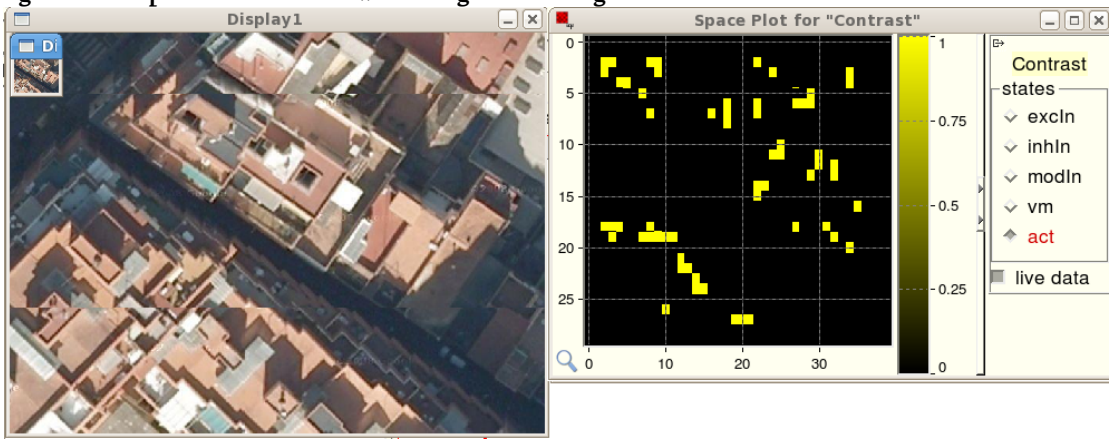

Figure 6.3 snapshot of Barcelona with the generated edges

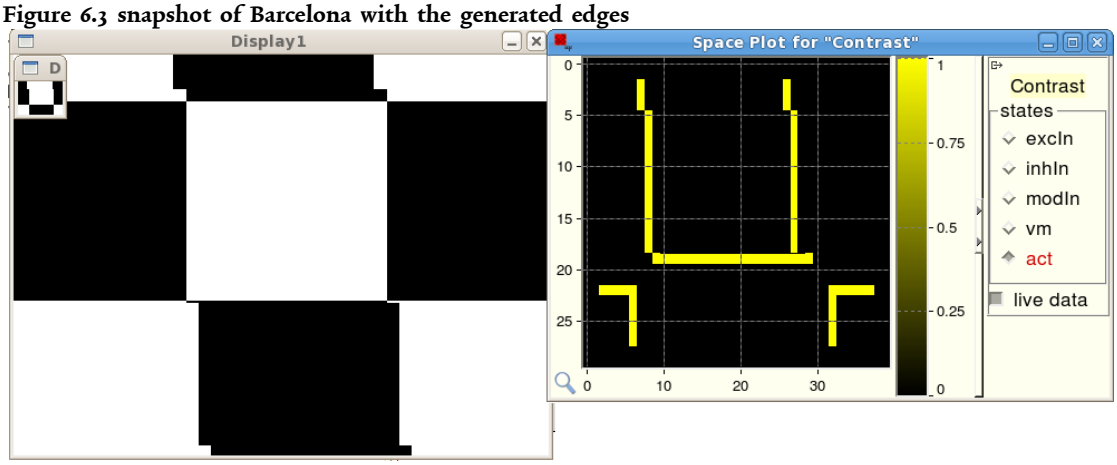

Figure 6.4 Snapshot of an artificial surface with generated edges

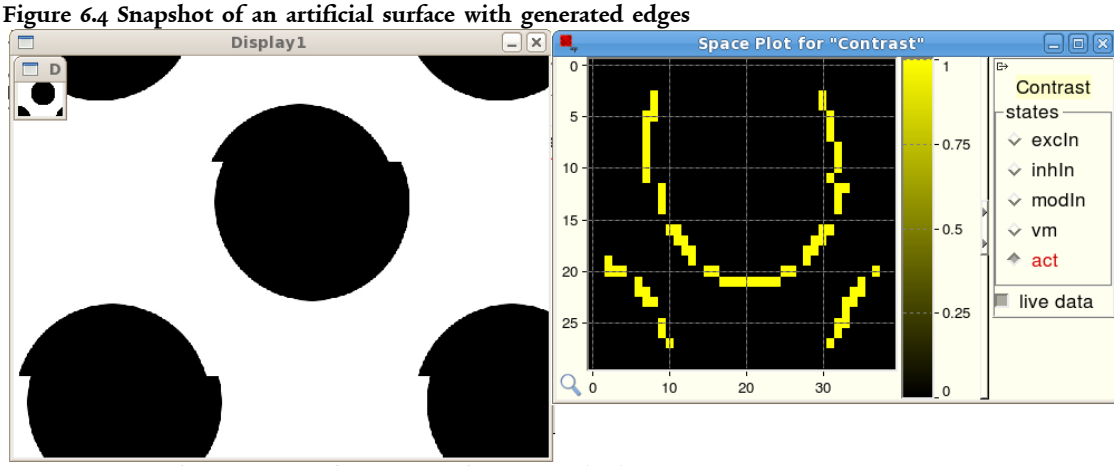

Figure 6.5 Snapshot of artificial surface with generated edges