Utrecht University

Faculty of Humanities

Department of Philosophy

Bachelor Thesis
Artificial Intelligence
15 ECTS

Natural Optical Flow on Graphics Hardware

Supervisors:
dr. G.A.W. Vreeswijk
dr. ir. R.W. Poppe

Student:
J.C. Goosen

October 26, 2014

Abstract

This paper aims to describe an optical flow detection method to be used for navigational tactics similar to those found in natural agents. These tactics are computationally not complex yet lead to useful behaviour, they make use of information generated by a correlation-based optical flow detection and a study of both is made. To be able to gauge the performance of the detection two proxy-tasks are used, one where the agent judges distance of surfaces and one where it tracks its own movement. The experiments are run in a simulated environment to have full control over relevant parameters such as retinal structure, textural properties of the world and others. An important factor is that the algorithm should run on a graphics processor as these are common in contemporary mobile low processing power platforms, such as smartphones, paving the way for a more autonomous agent.

1 Introduction

One sometimes lies awake of a summer's morning watching the flies as they dance under the ceiling. It is a very remarkable dance. The dancers do not whirl or gyrate, either in company or alone; but they advance and retire; they seem to jostle and rebound; between the rebounds they dart hither and thither in short straight snatches of hurried flight and turn again sharply in a new rebound at the end of each little rush. Their motions are erratic, independent of one another, and devoid of common purpose.

D'Arcy Wentworth Thompson [26]

Autonomous agents, flying or otherwise, represent an interesting field of study where Artificial Intelligence intersects with the complex reality we as animals find ourselves in every day. And like us, any agent that needs to operate in or act on an environment needs to have some representation of it. In the case of a virtual world, one could simply have the agent access the data-structure that represents the environment directly and this simplification allows one to focus on navigation or some other task. Yet if one then tries to use the navigational behaviour from the simulation in a physical agent, a large problem rears its head because the real world does not have any such accessible data-structures. Clearly an essential part of an artificially intelligent autonomous system, that is expected to operate in the real world at some point, is the ability to construct a representation through perception.

The aim of this paper is to implement an insect-inspired rudimentary perceptual system, specifically for the detection of motion. Insects are taken up as a guide because they posses fully formed perceptual systems that allow them to navigate efficiently through the complexity of reality with very limited neural processing systems. The properties of these neural systems are easily studied because between members of the same species they tend to not even vary on the level of individual neurons. Taken in combination with the fact that a lot is known about the navigational tactics insects use means it is possible to some extent to construct a relationship between behaviour and neural structure. Thus to show the usefulness of such a perceptual system a variety of these tactics is described in the next section.

These tactics themselves have in some cases been implemented in virtual or physical agents and the results from these experiments can be compared

to behavioural studies of the inspiring insects themselves. The motion detection systems used in the agent studies are almost all implemented on typical workstations using the Central Processing Unit or CPU in combination with low-resolution cameras. A few cases used an analogue implementation that was more viable then a digital one because the limited capacity of the computers at the time of construction.[6]

Here a different approach will be taken that combines the highly parallel nature of the modular analogue system with the versatility of a digital approach. This is done by implementing the motion detection scheme as a program running on a Graphics Processing Unit. These units or GPU's are ideally suited for the kind of parallel processing required in vision tasks. Currently large popular demand for small form factor low processing power platforms such as smartphones have lead to a proliferation of devices that contain efficient GPU's and all the required sensors needed for an insect-inspired autonomous agent. The mentioned sensors consist of one or more cameras and gyroscopic sensors, although these will not be studied here they could be coupled with a functioning motion detection system for more robust orientation capabilities in an agent or insect, as [22] suggests.

To be able to judge to performance of a GPU-based motion detecting scheme, also with regard to the constraints derived from the properties of the mentioned low processing power platforms, a simulation is implemented in which virtual agents can move while receiving perspective-correct imagery to their retinas. This visual sense-data is then processed by so-called shader programs that run on a typical GPU and the resulting retinal velocity data is used for two proxy-tasks.

Proxy-tasks are used instead of the to-be-discussed navigational tactics because they allow for a clearer interpretation or the experimental results. It is expected that for instance the structure of the eye as well as the resolution of the retina or the amount of samples per time-unit will make a difference to the approximation of retinal velocities. Using navigational tactics, themselves dependent on these velocities, at the same time as finding out what the constraints for a correct detection are, would make interpretation of the results rather opaque.

The first of the proxy-tasks consists of a surface-distance estimation task where the known speed of an agent is combined with the found velocities to calculate, through trigonometry, a perceived distance. This then leads to a simple metric that indicates how well retinal velocities have been extracted through the measurement of the difference between the actual and the estimated surface distance. The other task is an odometric task where the movement of the ground is tracked to see if an agent could use the motion detection scheme to estimate its position over time. The estimated position can easily be compared to the actual position to give a drift over time that indicates rather succinctly how well the motion estimation works on average.

The next section will review the work of others on insect-inspired motion detection and the navigational tactics engendered by these. The studies on the actual or theoretical properties of these systems and algorithms will be used to find the boundaries of the search-space in which relevant parameters such as eye resolution and others can be found.

2 Related Work

2.1 Optical Flow in Navigation

Before some studies on the use of motion perception by insects in navigation are discussed it is helpful to first give a definition of optical flow and the optical array. These two concepts together can be seen as representing an abstract context in which can be worked with the notion of motion perception more clearly.

Optical Array

The concept of optical flow is highly related to the idea of a projection, such as in an image as produced by a camera. Such an image is typically the projection of the colours and shapes of a single static moment in a three-dimensional environment onto a two-dimensional surface of some form. Optical flow on the other hand can typically be seen as a projection of the movements of a three-dimensional environment onto a two-dimensional surface of some kind. This surface could simply be a rectangular plane as is typical in computer science or photography, it could also have a different shape and it is when one uses a closed surface of some kind that we approach the concept of the optical array. This array then is an abstract sort of space on the surface of which optical flow can be placed in relation to the movement of an agent. The shape of the array that will be used here is spherical and is similar to the one defined by Clocksin.[3, 10]

The basic structure of the optical array here will be a unit sphere in Euclidean space as seen in figure 1, this will allow for the angles of the location and the direction of motions to be readily decoded from it. Let the agent's vector of movement V define an axis through the aforementioned sphere so that where it leaves the sphere it also defines the two optical vanishing points on the surface.

As an agent moves through a static scene, optical flow will always move over the sphere from the anterior to the posterior vanishing point because it is always aligned with the agent's movement. The visual sense of the agent then samples values from its specific optical array to obtain an image. In the special case of a completely spherical eye located exactly on the origin of the optical array the result would be a discretized version of the hypothetical continuous array.

One can find a few invariant transformations of such an array, following Koenderink there are some particularly useful examples of it: isotropic expansion or contraction, rigid rotation and pure shear.[13] These invariant transformations can be used in conjunction with knowledge of the array's movement in space to calculate the structure of the space. However it carries too far to go into full detail on how these notions could be used for that and usage of such transforms is probably better suited for more sophisticated perceptual systems, like those in humans or other "higher" animals, in any regard.

These next sections will describe how specific and more limited information from an optical flow array can be used to navigate without collision or getting lost.

Height and Flight Control

An example of height control is the expansion-avoidance model found in free-flying Drosophila where they keep themselves from crashing by monitoring ex-

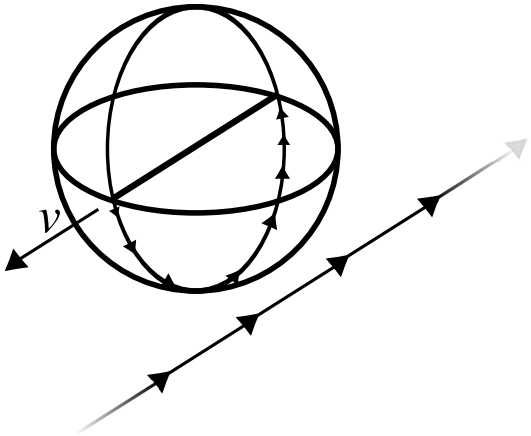


Figure 1: An optical array with movement vector V as well as a simplified projection on it of moving points underneath.

pansion in the bottom halve of their optical array.[25] When the expansion-velocity becomes greater than a certain set value the flies respond by rapidly increasing their altitude. The retinal velocity of the expansion is a function of dropping speed multiplied by proximity to the ground, so that falling very fast from high or slowly from low can lead to the same response.

In a comparable fashion another ground avoidance model for flying insects uses the integrated ventral anterior to posterior optical flow. Francheschini *et al.* describe an algorithm that allows a flying agent to keep a safe distance from the ground through a modulation of lift.[8] The lift is modulated as a function of the ventral optical flow speed ω, where ω is equal to the ratio of ground speed S to ground height H as follows: $\omega = S/H$.

The agent can be modelled as a winged insect or a helicopter and has a pitch, which determines where the vector denoting lift is pointed, and increases the amplitude of the lift vector when ω increases and vice versa.

An agent running this algorithm manages to fly at a rather constant height following the contour of the landscape without ever calculating its actual height. It also allows for the agent to manage taking of and landing very gracefully simply by changing the pitch, this changing can be done by another process such as an operator or some relevant higher level goals that a fly might harbour.

When close to the ground, for instance if the agent just took off, it will steadily increase its lift until it reaches a safe cruising speed. Come time to land the operator simply turns the pitch straight up and the lift no longer generates forward motion making sure that ω, and with it lift, decreases, slowly setting the agent on the ground.

Such responsive behaviour has some side-effects, good and bad. A positive one is that the agent will automatically slow down in heavy headwind and under certain conditions will land, something that could stop drones from crashing during storms, just as them climbing higher and higher during tailwind would. A detrimental one would be that when flying over non-contrasting surfaces such as mirrors and still water ventral optical flow speed would become zero. With zero speed the agent would try to decrease its lift until it crashed or drowned, a

behaviour that has been observed in actual bees. Luckily the prevalence of large clean mirrors laying around or windless aquatic expanses is rather minuscule and can safely be disregarded or, when extra safety measures are deemed required, taken into account.

Hallcentering

A likewise technique can be found in the behaviour of bees as shown in studies undertaken by Srinivasan *et al.*, as well as in robots and flies by others.[23, 24, 12, 5] The so-called hall-centring response is in essence very similar to the algorithm for keeping the ratio of ground speed and height equal, but now in addition to having a set ω value the agent tries to balance the integrated optical speeds on the left and right sides of the optical array as well. In this balancing of the two optical hemispheres an agent, when flying through a tunnel or hallway, not only manages to fly in the center of such a space but will also decrease its speed when the corridor narrows and accelerates up to a certain speed in the event of spatial widening. Valuable behaviour, in other words, when manoeuvring through a building or cave as it tends to increase the lifespan of fragile (flying) machines without the need for intricate behavioural patterns.

Straight Segments

Integrating retinal velocities from large segments of the optical array can be used in a variety of ways as hinted at in the preceding sections. The relatively simple algorithms can form a good basis for a navigation tactic that enables an agent to manoeuvre in predictable ways through environments while avoiding collisions with large objects such as walls or the ground. This however is not enough to avoid smaller obstacles that do not cover large parts of the optical array such as trees or chairs and the like.

One route to solve that problem lies in finding the orientation and edges of static surfaces in the environment, as Clocksin shows that to find these it is sufficient to know that the optical array has been constructed in such a way as to result solely from a translation along the movement axis. This fly-dance, as mentioned in the opening quote, has very useful characteristics as it simplifies the otherwise nigh intractable problem of structure from motion. It does so because an optical array representing the result of a pure translation along the movement axis would have the flow velocity coded as an angular movement, thus allowing elementary trigonometry to be used.

Clocksin's algorithm shares this dance but calculates through derivatives of speeds and Laplacian operators the orientation and segmentation of surfaces as well as their relative distances without needing to know even the agent's exact speed $|V|$. In his article the entire problem of calculating these optical flows from visual information is left as an exercise to the reader though and this hints at a small problem with his approach, in that it relies on an optimal representation of surfaces in the optical flow but such clear flow data is not easily obtained.

On the other hand, there is a technique that shares with Clocksin's algorithm the translation along straight segments and is described by Franceschini *et al.*, it is computationally much lighter.[6, 7] The dance allows for the simple calculation of distances to objects through the angle of the object in the optical array and the instantaneous velocity which together give the distance

to the object. Even with a rather imperfect calculation and detection one can use this notion to avoid collision by avoiding fast moving retinal velocities the closer they get on the optical array to the movement axis and thus to the frontal optical vanishing point. Francheschini has built an analogue robot with only 118 receptors that moves through a field of pole-obstacles at quite impressive speeds. The machine does this by using the trigonometric relation inherent in the optical array together with a behavioural pattern of moving straight ahead and turning on occasion. As the robot only has motion detectors this bursting is necessary for it to perceive anything and it stops either at intervals or at least when it approaches an obstacle closely. When it stops the robot uses its latest representation of the environment to choose a free direction and turns its heading there saccade-like and resumes its straight bursting. The experimental results of the robot are very encouraging and make the strange dance of the flies much more comprehensible.

Odometry and Vision

An overview has been given of some simple behaviours from the animal kingdom that can and in most cases have been successfully implemented in robots and can be employed to make them more autonomous. These behaviours are however not very useful for things like map building without a means of determining the distance travelled or at least the current position of the agent. For this one could use a system like GPS, built-in even in the smartphones proposed to be used as computational platforms, but such an external dependency would dramatically decrease their autonomy. To track the position autonomously would imply using some form of odometry, which means nothing more than the measurement of each step as the agent makes it, and as the main modality here is visual this would mean the agent tracks the movements of the environment through the resulting retinal velocities and integrating them to guess its own position over time.

One example of odometry in biologically inspired robotics can be seen in a study by Lambrinos *et al.* where the home-seeking behaviour of a desert ant is studied and modelled in a robot.[15] They note that these ants can move in a random walk through the rather featureless expanse of desert seeking food which the insects, when found, return to the nest in a very direct path. This indicates that the ants accumulate the distances and directions they moved in, a common tactic in robotics as well.

Yet it is a tactic with some problems, as small measurement errors quickly accumulate and for instance a slightly under- or overvalued estimation of heading would get an agent lost rather fast. The desert ant solves this problem as far as heading is concerned by have a polarized-light-sensitive compass in its visual system which allows it to judge its turns exactly because the sun projects an polarization-orientated pattern in the sky, and thus allowing it to find its nest quite well.

Although bees for instance also have access to such a compass, indeed many insects do, the necessity of using it for path finding is less because of the fact that most non-desert environments tend to contain much more varied texture. Bees have been shown to use the accumulated sum of ventral velocities as a measure of distance in experiments where the apparent motion of the floor was greater or smaller in the learning phase as opposed to the trial one, leading the

insects to misjudge the distance.[23, 24]

The visual measurement tactic can be used in more than one way, shown for instance by Fumiya Ida who has build a robot using a panoramic eye to enact both the earlier-mentioned hall-centring response and integrate speeds over time on both lateral sides for a visual odometry.[12] This works to some extent but of course is a tactic limited to indoor environments. An agent in the outside world would measure no distance travelled as soon as visible surfaces recede beyond a certain distance, that is, when the sampling base is larger than the projected movement on the retina. Ventral flow then seems a better choice as a measure of distance, as long as it has a constant reference value for ω this flow could turn out to be a reliable metric.

2.2 Calculating Optical Flow

Until now a variety of strategies have been discussed, all coming from nature and all involving optical flow, yet no mention of how to actually calculate that. This subsection can be considered a rectification, for here a look at how conventional computer vision approaches work out will be taken, followed by the nature inspired path.

Feature Based Motion Detection

Optical flow calculation in computer science has many uses and when applied real-time tends to work at relatively low framerates. This results from the fact that most algorithms only look at two or a few more frames to calculate the displacement of a image-segment between one frame and the next.

One finds such a displacement by, for instance, segmenting the frames into a coarse grid of cells and calculating some features, like gradients or something else, within each cell and trying to find the most likely position of it in the next frame. A comparison of some such algorithms with the hall-centring response and visual odometry tasks as benchmarks shows latencies occurring in real-time experiments, despite the system only running at 12fps.[16] High latencies are undesirable in an autonomous agent because only noticing a wall after crashing into it does not constitute obstacle avoidance. This latency results directly from the many (sub-)calculations done over the few entire frames available at each single step. The near instantaneous nature of the correlation type detectors that will be described in the following sections on the other hand is much more suited for fast visual processing and is limited only by its minimum delay interval ϵ. It also scales well to larger resolutions due to its modular nature. For a more detailed look at the more conventional techniques of optical flow detection and comparisons one is referred to Barron *et al.*[1]

Biologically Inspired Motion Detection

The mentioned techniques also bear no relation to how nature seems to have taken up the task of motion detection where, besides the already abundantly mentioned insects, motion detection is present as basic percept of the world in the vertebrate visual system as well. It could very well be responsible for such strange phenomena as blind sight in humans and perhaps, when missing, also implicated in the condition of cerebral akinetopsia. A person with akinetopsia

no longer perceives any motion, only still images and with this condition it becomes daunting to even fill a cup without causing overflowing, not to mention performing something as basic as crossing the street. In blind sight on the other hand patients report being blind, meaning they see absolutely nothing, yet strangely enough are able to point to moving objects and even avoid obstacles to some extent. These conditions seem to imply that conscious form and motion perception and the usage of it in locomotion do not necessarily pass through the same subsystems of the brain.

Although some perceptual systems use features of objects in the visual world to find motion, segments of the visual field moving in unison are themselves very salient features that enable the visual segmentation into objects. Readily apparent as it is to any non-visually-impaired subject in watching a static picture of random noise of which a part suddenly moves, because it will be visible as such. From insects it is hard to get their subjective look on the matter but fortunately for us studying them has an advantage, in that the neuronal structure in all members of a given species tends to always be more or less equal. This allows verifiable and objective theories on the computations done by neuronal structures to be formed and checked by intracellular recording with electrodes.

Structure of the Compound Eye

An insect has two kinds of visual sense, one of which consists of so-called ocelli, which are essentially light sensitive spots distributed around the body. These ocelli can be used for many tasks but as they are not well understood as of yet and seem to operate in a rather different fashion they fall out of the current scope.

Instead I will focus here on the compound eye of insects, which consists of a number of so-called ommatidia. To give an example a description of the structure of a housefly's compound eye as found in [6] will be given, this might or might not be fully extrapolated to the compound eye in other species of insect but here it will suffice as a general model. A single eye of the fly consist of roughly 3000 ommatidia, arranged on an almost perfectly hexagonal two dimensional grid. Each ommatidium is a mini-eye with its own fixed focal lens, which means that it cannot accommodate, that lets the gathered light fall on at least 8 receptors. 2 of these (R7 and R8) receptors are primarily used for colour discrimination while the other standard 6 are implicated in motion detection. Besides these receptors different regions of the eye feature extra or different receptors such as for the perception of the polarized-light sun compass earlier mentioned. For a closer look at the theoretical and actual properties of insect and other arthropods eyes the reader is referred to Warrant *et al.* [27]

The spacing of these ommatidia however is not equal in all parts of the eye, a sine gradient can be noted in the density of the retina where the frontal part of a compound eye has a much higher resolution than the regions perpendicular and thus anterior to it. Why this should be so can be explained when the concept of the optical array is taken into account, where the angular retinal velocity of a stimulus close to the frontal vanishing point is much smaller than of similar stimuli situated at the same distance yet more perpendicular to the movement axis. This structuring of ommatidia allows a fly to use identical modular detectors for the perception of motion. The larger speeds are compensated for by

Figure 2: A close up of the common housefly *Musca Domestica*.

larger spacing and vice versa, thus the same timing can be used for each.

By stimulating them directly in a live animal, while performing intracellular recording from known motion-sensitive processing cells further down the visual pathway, the R1 to R6 cells have been shown to be implicated in the process of motion detection. As of yet the exact neuronal structure of the elementary motion detectors themselves is unknown, but the way in which integrated motion detection behaves indicates a computation akin to that in the Reichardt-detector is performed.

Despite the fact that not a single identifiable motion detector has been found, the integrated result of them can be studied in, for instance, the activity of the HSE-cell (horizontal system equatorial cell) that responds when a fly rotates around its vertical body axis. When the activity of that cell, which can be located in every fly on the same location, is studied in detail one finds that depending on the experimental paradigm taken the performance can seem lacking, but when an ecological approach to the study is taken, such as using "natural" stimuli, the fly's motion detection scheme is very effective, being able to recognize and adequately code angular turning velocities of up to 2000° per second.[5, 21]

Reichardt Detectors

The matter of the singular motion detectors remains however and if they cannot be located physically, another way to study them shall have to be found. As their neuronal structure is not known a secondary path to knowledge is found in studying their hypothetical computational structure. Such a detector is discussed in much detail in [18, 2] and the reader is referred there for an exhausting description, now follows an overview of the basic structure and constraints as they found them.

Since the 1960's many studies have been undertaken on how humans and other animals see the world. Different animals and different researchers led to different models of motion detectors, yet when analysed and put together most of functional models, at least those that were confirmed by behavioural or psychophysical research, seem to be equivalent to the so-called correlation model. The correlation model's core consists of the idea that a movement of light-flux over the retina passes receptors in sequence. Taken together with a function that computes how much alike two light intensities in two neighbouring receptors are this leads to an assessment of motion. Such a detector can be constructed hypothetically and here the same rough structure as in [2] will be used to do so.

A consideration is that it has to convert light intensity into a vector describing motion, so it will need to have at least two neighbouring photoreceptors separated in space such as in figure 3, simply because less would leave the detector incapable of discriminating between a moving stimulus or a flash of light. The distance between the two receptors is equal to the sampling base $\Delta\varphi$.

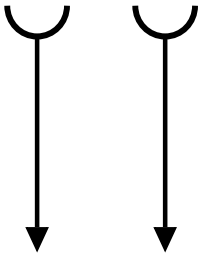


Figure 3: Two non-interacting receptors.

The interaction of these two inputs should be non-linear like in figure 4, otherwise calculating on the time-averaged inputs would lead to the same result as time-averaging the outputs of the original input. The averaging of the signal removes any temporal ordering and this means that found responses do not represent motion as for that such ordering is required. To avoid the loss of order the lowest order non-linear function of multiplication can be used, as it is not only trivial to implement but also seems to correlate well with what insect velocity detectors use.[4]

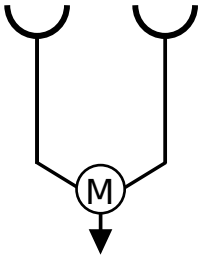


Figure 4: A correlation detector.

There should not be a symmetric treatment of the inputs because with such treatment a motion with the detectors optimum speed and one with the opposite would lead to identical responses. A delay in one of the inputs would suffice to introduce a preferred direction for the detector as shown in figure 5.

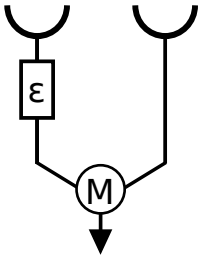


Figure 5: A uni-directional velocity detector.

With the currently described detector a movement in its preferred direction can be detected but the opposite gives no response, or worse yet, a positive response with only background luminance. To augment this we can combine two of these detectors as subunits in a more complicated one that will give a positive response in its preferred direction and a comparable but negative response in the opposite.

This then is a fully fledged Elementary Motion Detector (EMD) as Reichardt himself calls them and the full schematic for one can be seen in figure 6.

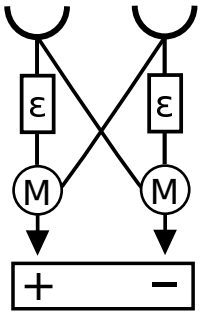


Figure 6: a bi-directional velocity detector.

Some observations can be made on what velocities can be detected and how to use the found range optimally. The delay ϵ is directly related to the speeds that can be found, as a smaller ϵ will lead to higher detected speeds and vice versa. One can change $\Delta\varphi$ without changing ϵ for more laterally placed receptors to be able to detect the higher expected velocities there without the need to change the computational hardware. This change in spacing can be done according to the earlier mentioned sine-gradient as found in actual compound eyes and allows for a more efficient modular system.

Spatial aliasing should be prevented and to do so the input is processed with a band-pass filter. High frequency components should be removed to prevent spatial aliasing, where high frequency means that these components have smaller wavelengths than the sampling base divided by 2 as per Nyquist.[17] These high frequency components can be removed through the application of a Gaussian convolution. A high-pass filter added afterwards completes the band-pass filter and it increases the clarity of the response of the detector by removing background luminance and also functions as an edge or contrast enhancer.[2]

Using such pre-filtering allows the difference in luminance of a stimulus moving to be the feature that is detected instead of only the luminance. Finding a dark stimulus moving should not be less likely than the opposite but without the filter would be the result. Not only this, the theoretical work has mostly been done in relation to sine-gratings and having an appropriately blurred and then high-pass-filtered image approximates such a structure, thus allowing for a cleaner application of the theoretical knowledge.

Integration of Detector Outputs

There remains the fact that when a detector gives a strong response it means a movement at that point in time occurred, yet such a pulse can only be interpreted as a movement of fixed distance $\Delta\varphi$ in fixed time ϵ. Integrating these pulses over times larger than ϵ makes it possible to have a less binary approach but nature's manner of integration has proved hard to study. In flies for example, this integration happens somewhere between a photoreceptor's firing over time and the output of the HSE-cell giving an averaged integrated response of the entire visual field's horizontal speed, but the intermediate stages are hard to study in vivo and remain elusive.

Not only the temporal dimension is one were information can be integrated to give a more graded output, one could also integrate the output of more than a single one-dimensional detector to find two-dimensional flow. Yet there a problem occurs in the guise of the aperture problem, where not only the movement but the structure of a stimuli determines the response. For instance take diagonal gratings that move perpendicular to their orientation and when seen through a slit of some sort they would seem to move in the orientation of the slit instead of their actual orientation.

This occurs in many movement detection systems, including those of man, to resolve the ambiguity Reichardt has studied the mathematical properties of such an integration and suggests using Gestalt theory as a means of structuring the input in such a way as to resolve ambiguities in a similar way to how natural systems manage.[19] For the purposes of this paper where there is no other inherent need for a complicated perceptual categorizing or even large scale field integration of optical flow in the full array, this use of Gestalt would mean a massive complication and is thus unwarranted. The aperture problem should be kept in mind though and depending on the implementation could have implications for visual odometry.

3 Methodology & Implementation

An overview has been given of what has been described in the literature on insect's navigation and their system of optical flow detection. The constraints following from the overview will be the focus of the first part of this section and will be followed by a more in-depth look at the actual implementation that results. These constraints will be varied in the experiments and used to examine the viability of the motion detection scheme under study here by using it in the proxy-tasks of finding the distance to surfaces in the environment and optically tracking the movements of the ground in relation to the agent.

3.1 Influences on Reliability of Optical Flow Recognition

It is important to note that the motion detection algorithms in general use assume that characteristics of a scene such as global illumination, location of objects or other features are known. Many computer vision tasks assume for instance that global illumination does not or only slowly changes as this not only makes for a much lighter computational load but is also warranted due to the fact that in reality this is in fact tends to be the case. Only if one (inadvertently) subjects a motion detector to a stroboscope-like illumination, where the visual systems frequency is roughly equal to the frequency of an artificial light source, motion detection will be disrupted.

So to avoid all that the simulation will keep light intensity and the like constant, other factors could still be of influence and a closer look at them is warranted. It is expected that the choice of textural pattern will have an influence on the performance of the EMD's, as will the relation between the textural and retinal resolution. Such texture cannot be visible without some surface to map it on and thus both object-shape and the properties of the texture will be discussed in more detail below. Of course to be able to know which retinal resolution to take into account the shape of the sensor or eye needs to be defined as well as how further processing of the resulting visual data occurs. This all is in combination with the necessity of having some assurances on the behaviour of the agent to be able to use the optical array as a unit sphere with angular encoding of the environment's optical movements.

World Texture

There are multiple factors in a scene that together determine the performance of a correlation based motion detector and therefore of the representation of optical flow. First and foremost the environment will need to contain sufficiently discernible texture, which means there should be enough visible contrast. Ideally this means some sort of fractal structure so that, no matter how close an agent gets, detail to be resolved remains. In the real world resolving is not usually an issue as it is nigh-infinitely detailed and only bounded by the sampling resolution engendered by the size of each single receptor, within which all incoming light is averaged together.

This possibility of discerning holds as long as the agent is not operating in a hall of mirrors or similar environment where there is no texture to resolve, although according to personal unpublished research it can pose a formidable problem to any visually navigating agent. At a first glance this problem might

seem to exist with other seemingly evenly textured surfaces as well, such as concrete slabs or white office walls. Yet on closer inspection of such a surface one can resolve many details such as relief, spots and other smaller yet detectable irregularities. So while an agent might not be able to resolve such a surface from a larger distance, its EMD's will start to respond as soon as the resolving capacity of its retina matches that of the details on these surfaces. One could make a likewise case for a perfectly clear blue sky as it would indeed be imperceivable to such an agent yet to collide with the atmosphere would also be rather hard, not to mention the fact that other parts of its optical array would still contain motion from the other surfaces around such as the ground.

In a virtual world texture size is usually fixed but this need not be a problem if the displayed resolution has enough detail to be resolved by the virtual retina. A sufficiently high resolution can be found by calculating the projection of the texture on the retina at its expected maximum proximity to a surface. Then secondly the texture will need to be properly filtered when projected or mapped to avoid problems with aliasing, such as moire patterns and the like.

Then a pattern to fill this grid is needed and there are many possibilities but this can be limited to some generic prototypes. Here cases such as gratings, white noise or chequered will be used. The aperture problem occurs mainly when the structure of the texture and its movement over the retina can lead single detectors to multiple in-congruent interpretations, and is to be expected with a grating of some sort. A chequered pattern would not avoid the ambiguity yet due to its alternating nature motion detectors might respond differently and the comparison to a grating could be worthwhile. A noise texture on the other hand, with its relative absence of order, should be more apt to avoid the aperture problem. It would have as a disadvantage that, through filtering, on a larger distance would appear smooth not unlike the mentioned concrete slabs. Of course this also depends on the algorithm with which the noise is generated and could be avoided by using, for instance, Perlin noise.[20] This specialized noise requires a rather involved algorithm and would carry far beyond the need to avoid the aperture problem and thus a simple pseudo-random white noise generator should suffice here as long as the limitations are kept in mind.

Object Geometry

As far as object shape is concerned there is not much reason to generate complicated geometries as most of the optical information will need to come from contrasting parts of the visual field. An object, regardless of its texture, will naturally have its contours as a visible contrast and if it contains many edges these could perhaps add to that. At a distance this contrast can be clearly detected and silhouette forms the basis of Franceschini's analogue optical flow robot. Yet a robust algorithm should be able to navigate in an environment that is more varied than simple black poles in a further empty white-walled space. In an experimental condition where an agent is to navigate through a hallway for instance an outline will be hard to find yet the texture will fill almost the entire optical array. With these thoughts then the obstacle complexity can be limited reasonably to rather simple shapes, as long as textural complexity is assured, without any loss of detection possibilities for the system.

However if one were to implement the surface detection algorithm Clocksin proposed then more complicated objects could lead to differing results over

simple ones but it would also require a an optical flow recognition method that is much more sophisticated than the one under study here.

Panoramic Eye Structure

Eyes come in many shapes and sizes and a choice is to be made to constrain this somewhat. Concerning natural optics a simple yet effective structure, geometrically, computationally and optically, is a round eye. In humans and other organisms with a single lens eye there is the advantage that this shape allows most of the retina to receive focused light through having the same focal distance from the lens. In an insect this sphericity of the entire sense organ is not precisely necessary as each facet has its own fixed lens, but having a roughly round eye still has advantages in that it allows for a nearly full field of vision without redundant capacity.

In robotics there are numerous possibilities of at least emulating such structures, despite silicon photosensors for the most part still being flat, by having parabolic mirrors and the like to enlarge, bend and distort the visual field perceived by a camera.[12] In a simulation, while theoretically being able to emulate any kind of structure, be it photosensor or otherwise, this is not traditionally the case. To be able to compare the performance of the motion detecting scheme under study in a virtual world, to that in the real world it is important that at least perspective is emulated correctly.

There curves do not come gracefully, as normally a world-space is constructed in a Euclidean grid and transformed by the viewing frustum to emulate perspective. A frustum is essentially a truncated pyramid and in computer graphics defines the viewable volume of the simulation. Due to the nature of the transform this volume represents it cannot generate smooth curves from geometrically straight lines, such as occur when one takes a photograph through a lens. This is because the perspective-transform consists of shortening vectors as a function of their perpendicularity to the viewing direction and distance which, together with the vertex-based nature of surface graphics, means that the result of the transform can only turn out straight as well. The same transform also limits the possible field of view in such images to a bit less than 180° and the closer to that the heavier the distortion. This is not a problem with a limited field of view, but for a navigating agent a large field of view is desirable as it means more available information.

To avoid such issues a visual system can be built where many different perspectives from the same origin are taken, all only representing a small segment of the total field of view. This is to enable a full-round view of the environment with appropriately naturally curving lines instead of the usual rectilinear imagery of computer graphics, this way coming closer to fish-eye-lenses or other panoramic lenses. Besides that, it will also allow a sine-gradient to be implemented by having segments closer to the optical vanishing points have, for the same field of view, a relatively larger resolution than those perpendicular to them.

Odometric Eye Structure

Of course when all that is needed is a rather limited field of view, such as for instance the ventral view of the ground, then such a complicated system is not

needed and one can simply use a single standard view.

To have a visual odometry one needs to be able to track the movement of the ground as one moves and to make this possible there will need to be some sort of two-dimensional integration of detector responses. There are myriad manners in which that could be done, such as a grid of detectors linked to each other to find for instance the centre of rotation or expansion and using those to find the movement.

To be able to find the centre of rotation a much more complicated perceptual system than is possible here is necessary. That is to say, if one cannot be sure where such a rotational point might be, the entire bottom halve of the optical array or even the full array needs to be covered with two-dimensional motion detectors and all this information somehow integrated. Besides the perceptual problems, like the aperture problem, arising from such integration this would go well beyond the scope of finding parameters influencing the functioning of digital motion detectors and well into building an operating system for a flying robot.

Here however the orientation of the camera is known, as it is aimed downward and always aligned with the worlds frame of reference. This is done because of the movement update cycle of the agent, where translational steps are done incrementally allowing motion detectors with a fixed $\Delta\varphi$ and ϵ to arrive at more graded velocities than just $\Delta\varphi/\epsilon$. The rotational phase on the other hand is instantaneous and when one would make it incremental, like the translation phase, it would no longer be a quick saccade-like turn. If on the other hand the rotation is stretched out over the entire translational step then this step stops being straight and can no longer be used as a simplification for the integration of speeds over the entire series or the estimation of distance with trigonometry.

Because of these issues the camera is aligned with the worlds frame of reference and the centre of its view is guaranteed to be the agents ground position which makes a simpler heuristic possible. The method that can then be used is rather similar perhaps to that in an optical mouse and boils down to finding the most likely direction away from the centre the ground seems to move in.

With this the aperture problem cannot be avoided as the direction of the true movement is not guaranteed and some two-dimensional integration is required. It is expected that the results will show this when grated textures of some sort are used.

Motion Detection Pre-filtering

After input has been collected it could be ready for processing by a motion detector immediately, as it could simply operate on luminance. But, as has been noted in the literature section, insect eyes are not sharp in any sense comparable to a human eye or camera. This has consequences for the visual data collected and the main one is that the fixed lens of an ommatidium blurs the incoming image. This can be emulated by applying a Gaussian convolution of some sort.[11] This then implements the low-pass part of the band-pass filter described as optimal by Borst and Egelhaaf and it is likely that it is the reason that an ommatidium does not focus “adequately”. The completion of the band-pass filter then only needs a high-pass filter of some sort to remove background luminance and enhance the contrast. A stage similar to that occurs in the first stages of vertebrate visual perception in the so-called centre & surround

ganglions. Which, essentially, are no more than a sharpening convolution and although it is not yet known if insects do something similar, the computational analysis shows that it is a highly desirable stage and therefore safe to both implement here and perhaps to assume in insects. [2]

Besides that there is the fact that all images are generated in RGB colour-space and the motion detectors so far described respond only to monochrome luminance values. To augment this, one can simply take the normal motion detector, apply it to each colour-channel separately and add the resulting values together.

Behaviour

As seen in the sections on insect navigation, the behaviour of flies and other insects consists in part of short bouts of straight flight interspersed with saccade-like quick rotations, so that they might better detect the environment from their own motion. It makes sense then to implement this in a similar way in a digital agent, I shall design the agent in the same fashion as Francheschini's 1992 robot.

To have this work in a three-dimensional environment the agents will be restricted in their movements, only rotations around the y or height-axis will be allowed, effectively constraining them to a single plane of the world. By confining the movement of the agent to a single plane one can also avoid the aperture problem to some extent. That is to say, in the distance estimation task, as there the agent can be limited to using information of the intersection of this plane with its optical array and thus be assured that any detected movement found would be horizontal. The influence of the aperture problem would be expected to have a much larger influence in the visual odometry task, as there still some two-dimensional integration is necessary. This is also likely to be the task where the choice of texture will have the most influence on the results.

Many behaviour possibilities lie in the navigation tactics that one finds in nature and some of which have been described in preceding sections. To be able to implement them properly, one would first need to find proper values for the parameters under discussion here. Evaluating the performance of the navigation tactics at the same time as the parameters of the optical flow detection system would make the evaluation phase needlessly complicated and perhaps impossible and thus is to be avoided. The experiments run here should then not use any navigation tactics yet and will simply random-walk through the simulated world. When these parameters are filled a system such as described here could easily incorporate such behaviours.

3.2 Implementation

In this section the software for the experiments is described which takes into account the mentioned constraints. To give a better overview of the algorithm its rough structure is outlined first.

Initialization
: First of the environment is generated, which includes texture generation, obstacle placement and agent set-up.

Update Cycle
: Then the update cycle is started which continues to run until the user

terminates the program or the maximum amount of cycles is reached. During each cycle the agent rotates if it collided with an obstacle last cycle or has a certain rotational factor set. Besides the movement update one or both of the following two subroutines is run:

Panoramic Subroutine

A panoramic view is generated from P views and a equatorial slice of the view is taken. This slice is added to a texture for processing. At the end of the cycle the resulting TxR texture is then convoluted with a Gaussian kernel and the result is again convoluted but with a different contrast enhancing kernel. The result is a band-passed texture suited for velocity detection and this detection is done, like the kernels mentioned before, through a fragment shader on the GPU. Together with a R-wide one-dimensional texture representing angular properties of the retina this velocity-representing is used to find the distance as a function of the agents speed and the retinal velocity. The distances are set negative if a guess falls behind a surface or positive when it does not.

Odometric Subroutine

Each step N slices, each rotated from the next by $\frac{\pi}{N}$, are taken and thus ending up at the end of each cycle with N textures. Each of these represents a view over the course of the cycle similar to the single TxR view texture from the panoramic subroutine, they are also processed alike up to the distance calculation. Instead of distance calculation the slices are evaluated to find which one represents the most likely direction the agent moved in and how fast. Its found movement is then added onto its internal position tracking value.

Exit

When the program finishes its cycles the found distance-estimations are averaged, as well as its number of hits and misses with regard to surfaces. This or the difference between the internal tracked position of the agent and the agents real position is output.

Environment

The world consists in essence of a ground-plane on which boxes and cylinders are placed as obstacles. Obstacles can be randomly sized and placed anywhere on the world map and where they overlap the borders the obstacles clone themselves to be placed on the opposite side. This allows for easier surface guess estimation and collision detection through the enabling of a torus-like virtual environment.

Of course, obstacles can also be set in size and location and this could be used to allow a more rigorous comparison between different runs of specific set parameters. Such environmental planning would only be useful if the navigational behaviour of the agent was what is under study, because in that case the flightpath would depend on the exact placement and size of obstacles. Here no navigational tactic is attempted and the exact placement of any objects or surfaces should not make any difference in the results.

Besides that, each obstacle can also have a colour, to allow for better discrimination at larger distances. A function of collision detection is implemented

where each obstacle can be asked if a certain world-coordinate is within or without them. The world as a whole integrates all responses in a similar function and returns whether or not a collision happened at the desired location and the agent asks for one or two further ahead to keep a safe distance.

The ground plane and obstacles feature repetitions of a single texture, the full texture is mapped to each 1.0 by 1.0 unit of surface and has a resolution of 1024 by 1024 pixels. Due to the fact that the agents are not allowed to get closer than 1.0 unit to any obstacle, through collision detection, this ensures that the retinal resolution will not be higher than that of the scaled and mapped texture. It also means that the measure of grain or detail is fully under control of the texture generation algorithm so that more or less smooth textures can be generated as desired. Mipmapping and anisotropic filtering are used to make sure Moire patterning and other aliasing does not occur when the texture is viewed from a distance.

The texture in use is generated by a fragment shader and any number of functions can be used for the generation of different patterns. The four different patterns used can be seen with a short description of their generation in figure 7.

The entire world is initially generated and compiled into a displaylist and is, when called, drawn 9 times on a grid so that the agents can keep flying and seeing when they cross over the edge of the world.

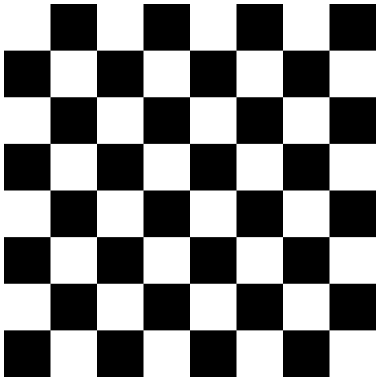
Agents

The agents have their own class and keep track of their position, speed and orientation. Their position is a vector representing the displacement from the world origin, the speed is a constant and the orientation is an angle. Each update cycle a movemement vector V is calculated by taking a unit vector that represents the orientation, multiplied by the agents speed.

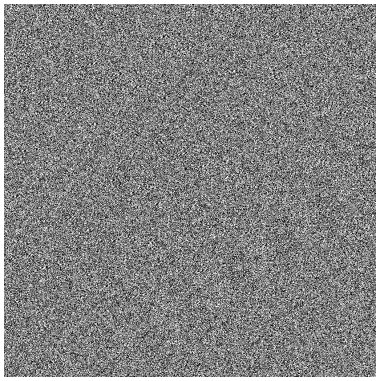
All agents move along their V in a series of T translations each update cycle. Thus they move by $V \cdot \delta$ each step of the series, where $\delta = 1.0/T$ and T represents the time resolution. Because of the way in which the motion detector compares results from the different steps with each other, any ϵ is clearly related to some multiple of δ.

Besides the series of translations, each update cycle also updates the orientation's angle. It is done through the addition of the differential γ to the angle of orientation and occasionally adds to γ a small positive or negative value. Having its heading updated with only a single angular scalar is possible despite the agents three-dimensional frame of reference because its only degree of rotational freedom is around the vertical axis. Although these additions happen only on occasional update cycles, the angular movement decreases each cycle through the relation $\gamma_{time} = \gamma_{time-1} \cdot 0.7$, thus leading to the smooth tapering off of turns.

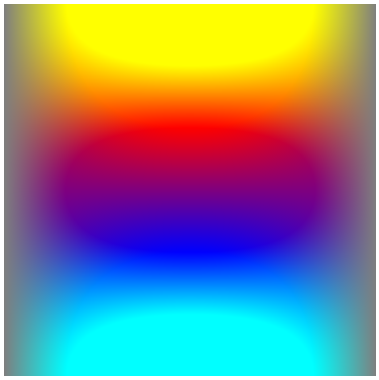
During each part of the translation the agent checks if the next position is free through the world's collision detection and if so moves there. If not free it considers itself bumped, stops moving that cycle and will not contribute to any surface gauging, as this would pollute the results. When it bumped the last update cycle it turns at large non-smooth intervals in the addition to γ, until its preferred next position is a free one. For an example of what the sort of path such an agent would follow looks like, figure 8 shows one.



(a) Chessboard world texture.



(b) Noise world texture.



(c) Low frequency sine world texture.

(d) Higher frequency sine world texture.

Figure 7: The world textures as used in the implementation. (a) the chequered texture is generated by having the coordinates multiplied by 8 and then modulo 2 taken for the colour value. (b) the noise texture is generated with pseudo-random numbers. (c) is a sine or grating texture that consists of per-colour-channel shifted sines where the texture coordinates are multiplied with π or 2π and offsets added. (d) is also a sine or grating texture and consists of multiple gratings where each colour-channel is the sum of several sines of different powers of 2 multiplied by π.

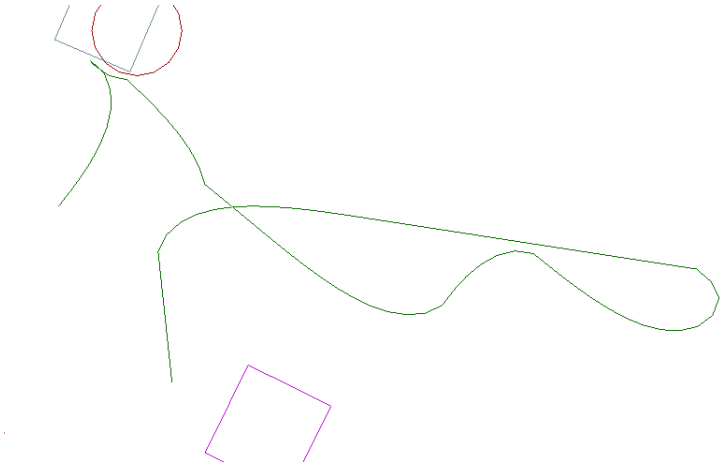


Figure 8: An agents random walk to, collision with and short path away from two obstacles.

Framebuffers

Framebuffers can be set to receive normal rendering of three dimensional scenes with any desired resolution and field of view, from these the corresponding depth and colour textures can be read that are the result of such a render. Orthogonal rendering can also be used to enable image processing and generation through fragment shaders. All textures used here consist of a number of 32bit floats, arranged in RGBA tupels, and can store any float value, although when rendered to screen each channel will always be displayed as normalised to $[0, 1]$.

To process or generate a texture through a framebuffer object one simply binds it as the current recipient of image data and enables the relevant shader program, consisting of a vertex and fragment shader. The vertex shader in practice does nothing else than pass through the vertexes, for image processing this would be to encode two triangles that cover the entire framebuffer exactly. The fragment shader is then queried for each pixel for a value, to calculate it this shader can (but need not) have access to a number of textures and other values passed from the main program. When the shader stage is finished one can bind the resulting texture to access the output values or use them for another calculation.

Panoramic Visual System

As done by Franceschini and to allow for a surface gauging of a large swath of the environment, the distance estimation proxy task uses a panoramic view. This panoramic view is defined with a certain horizontal field of view θ, here at all times 360°, with a certain resolution R. To render this properly a single render cannot work due to the nature of the viewing frustum and thus must be done in separate passes. This is done by subdividing the desired view into P views, each camera having its own orientation but sharing the location. They can be generated most simply by dividing R and θ by P. With a high enough P this leads to a very realistic looking full-round view, but the noted problems of slow velocity close to the vanishing point and high velocity at perpendicular points

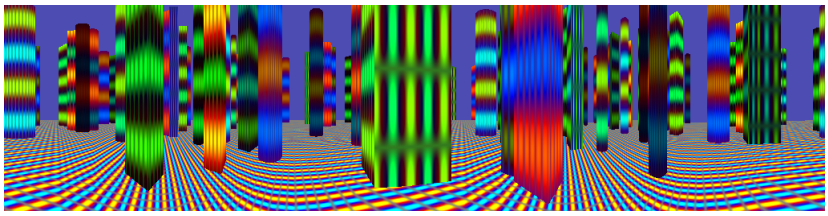


Figure 9: The sine-gradiented view of a simulated world as it appears to an agent at a single V/T step.

are not diminished. An example of a view like those that an agent receives for retinal input can be seen in figure 9.

To diminish them another pattern of division is devised and it is generated as follows: θ is divided by P and for each segment the central angle φ in the horizontal plane of the optical array is taken. With the following functions and the additive constant C the sine-gradiented resolution for each segment can be found:

$$W(\varphi) = 1.0/(|\sin(\varphi)| + C)$$
$$\Gamma = R/(\sum_{i=0}^{P-1} W(\varphi_i))$$
$$H(\varphi) = W(\varphi) \cdot \Gamma$$

In the function to determine the relative width of each segment $W(\varphi)$, a constant C has been added. C has a two-fold purpose where it avoids a division by zero when $sin(0) = 0$ and it can be tweaked to have the resolutions of each segment of the view be less divergent. figure 10 shows the differences in such views having a smaller or larger C in comparison with a regularly divided view. It is perhaps helpful to note that a regular view can be generated using almost the same formulas but with a constant $W(\varphi) = 1.0$.

Each segment then uses $H(\varphi)$ to find its horizontal resolution while the last segment fills up exactly the last remaining part of R as rounding errors in floating point arithmetic are not to be avoided.

For both approaches there is also a simple one dimensional texture generated with width R, each pixel containing the angle (minus 180°) that the pixel has within θ and thus immediately also within the optical array on the horizontal plane. The other value stored is the angle between it and the next pixel, which corresponds to the sampling base $\Delta\varphi$. At each of the agent's translation steps each of the panoramic sub-buffers is bound and the camera translated to the agent's location. Each view is then rotated appropriately and the world rendered into it. When all views have been rendered all framebuffers are combined by copying them into the large full view panoramic buffer and from its equator a horizontal slice is copied to the relevant location in the first texture of the processing stage.

Band-Pass Filter

For both tasks a slice of the visual world is copied to its place in the first stage of this processing stage. The first texture has the time resolution T as its vertical

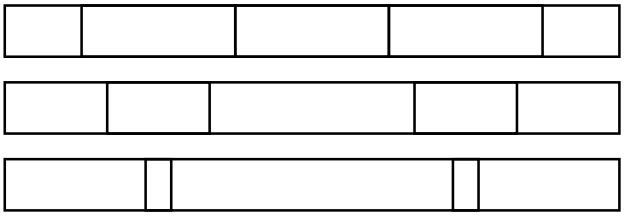


Figure 10: Diagram of the differences between views with 4 segments, where the middle segment is the frontal view and the backward view is seen split. Each view can be read as follows from top to bottom: a view without a sine-gradient, a sine-gradient with a large C and finally one with a small C.

resolution, T also defines the amount of steps the translation series done by the agent has. At the end of each movement stage the first texture is filled up completely and the processing stage is invoked. These next two preprocessing stages operate only horizontally, as each line represents a single time-step and here it is not useful to contaminate them with previous or future ones. In other implementations, perhaps to take into account motion blurring or slow photoreceptor's after-images and the like, this could be otherwise.

The first step consists of a rather straightforward Gaussian-like blur or convolution, this way any high-frequency components of only 1 pixel wide are smoothed out over 5 and this also allows for a closer likeness to the much-tested sine gratings. Many 1 pixel wide details occur and is the expected result when proper filtering is used and also highly desirable when one wants to project a clear static image as is usual in computer graphics.

The resulting texture is then copied to the next with a high-pass filter that for the 5 pixels in its scope, ordered as: $[ll, l, m, r, rr]$, calculates its output M. The output is to be placed at the same coordinates as m in the next texture and the following formula is used to find M:

$$M = ((l + m + r) \cdot 2.0) - ((ll + rr) \cdot 3.0)$$

It is a slight variation on a standard high-pass filter, which in this case could be something along the lines of $M = 2m - (l + r)$. The difference here is that the size of the details that are enhanced can be three pixels wide instead of only one while still removing non- or only slow-changing background luminance. A width of $3\Delta\varphi$ also comes closer to the noted best performance of a motion detector of signals at wavelength $4\Delta\varphi$ while still retaining symmetric processing around m.[2]

With these two convolutions performed the end-result is a band-passed version of the visual input of the entire translation series V and the next step would be to extract motion information from this result.

Velocity detection

This is the stage where the Reichardt detectors are implemented and an integration over time is done, as perhaps by an intermediate stage of the fly's visual system before reaching the HSE-cell would. At all times the detector only looks at two adjacent receptors or pixels, thus the sampling base $\Delta\varphi$ cannot be varied,

meaning the only way to allow different velocities to be detected is varying its ϵ. This is done by rendering the filtered image to a single line through the velocity detection shader. Inside this shader for each pixel on the line two entire vertical slices of the previous stage are compared to each other in the following fashion: Let the left vertical slice consist of $(l0, l1, ..., ln)$, the right vertical slice of $(r0, r1..., rn)$. $l0$ and $r0$ would represent the beginning of the translation series, ln and rn the end. Let the motion detector function f be defined as:

$$f(l, r, L, R) = (l \cdot R) - (L \cdot r)$$

To find the strongest response in time all different values for ϵ should be checked, so let $\alpha \in [1, T-1]$. For each such α the following is calculated:

$$g(\alpha) = \frac{\sum_{i=0}^{T-\alpha} f(l_i, r_i, l_{i+\alpha}, r_{i+\alpha})}{T - \alpha}$$

Function $g(\alpha)$ represents the average response of the $T-\alpha$ Reichardt detectors, all with the same $\epsilon = \alpha$. It enables a form of integration over time as the highest response would be expected from the ϵ that, in combination with the fixed $\Delta\varphi$, represents the actual retinal velocity. If for instance due to a non-fitting ϵ temporal aliasing occurs one would expect both negative and positive sign on the found velocities leading to a low overall response. On the other hand, with a perfectly fitting ϵ, despite some non-responding detectors, due for instance to the fact that filtering removed all luminance-information or something similar, this should still lead to a relatively larger response. While it is not assured, it is an integration that can be done easily within a single texture and is thus well-suited to parallel processing as it is structured on a GPU.

The best fitting α, the one with maximum $|g(\alpha)|$, is then the one that best fits a movement between the two receptors and the corresponding velocity v can be found as follows:

$$v = \Delta\varphi \cdot \frac{1.0}{\alpha \cdot sgn(g(\alpha))}$$

$$sgn(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

To be able to store and visualize v better in a texture it is not multiplied by $\Delta\varphi$ in the implementation but left as $-1 <= v <= 1$. It is then multiplied by the sampling base only in the next phase of distance estimation.

One thing to be clarified is that while the aforementioned algorithm operates on light intensity, the imagery used in the implementation has three such values (RGB) and in order to get a useful response from f it is modified to simply add together the responses from the three colour channels, each calculated in the already defined manner.

This way of velocity detection depends on a large swath of time over which it indiscriminately says which velocity best fits all responses and leads to a small side effect. A single detectable contrast translates horizontally if seen on the two-dimensional input texture, leading to responses of detectors over a larger part of the receptive field and this effect is only enhanced when more points cross over. In this way the velocity is correctly detected at a point of the activated receptive field but it also leads to some noise over time. At the end of the following section on distance calculation a possible filter to deal with this will be described.

Distance Calculation

As distance calculation is only necessary for the panoramic task this is logically the part where the shared processing of both tasks ends. This part and the next are then only applicable for the distance task, the rest of the processing for the odometric task can be found afterwards.

This stage has been passed $|V|$ representing the travelled distance over the translation series. The one-dimensional texture representing all velocities and the texture representing the angles of, and between all, points of the panorama are bound and the shader is invoked. For each point a value D (representing distance) shall be found with the following input: the corresponding angle to the endpoint of movement is defined as b, the angle between it and the next point as g and lastly v or the detected velocity. One can see that c follows quite simply from b and could thus be encoded directly in the reference texture instead of b but here this was not done to keep a closer correspondence to the angles of the optical array. As mentioned in the previous section the output of the velocity detector is scaled between -1 and 1 so to get the absolute angular velocity on the retina $s = |v| \cdot T$ is taken. It is multiplied by T because each step in time allows another movement by $\Delta\varphi$ and thus it could reach such a velocity during one cycle. As the agent always moves in a straight line forward and the environment is static this absolute value representing movement on the retina corresponds to an angle $\xi = s \cdot g$ on the optical array. If we assume the optical array to be immobile this angle ξ is the projection of a point in the world that travelled by $-V$. However if $\xi > \pi/2.0$ the results become undependable and in lieu the pixel is discarded. All this defines a triangle as can be seen in figure 11.

With all this in place and as long as $s > 0$ then D can be found with the following formulas

$$
\begin{aligned}
c &= \pi - b \\
d &= \pi - (\xi + c) \\
D &= \frac{|V|}{\sin(\xi)} \cdot \sin(d)
\end{aligned}
$$

If on the other hand $s \leq 0$ then the pixel is discarded and D is left 0.

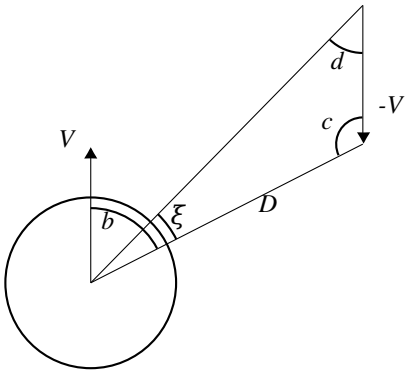


Figure 11: The optical array and the triangle defined through an agent's movement.

The values found this way can be used directly but the mentioned noise leads to a spread of points from better and closer guesses to far away mistakes and for this a small post-processing stage is optionally added. The noise results in a larger and continuous field of distances and thus the post-processing-shader looks at each pixel's neighbours and if it can find, within a preset distance, a pixel not seperated from it by a zero D interval and is predicted to be closer. If it finds such a pixel the heuristic implicates it being more accurate and so the more distant one is removed. It seems to greatly reduces the spread of error in estimations of distance but this will be verified in the evaluations phase by comparing it to the more naive version without this heuristic.

Although distance estimates are calculated for every pixel in the texture and thus every angle visible, some of them are not used. A part of 15° in the field of view around each optical vanishing point is ignored in the surface guessing. This is done for the same purpose as the sine-gradient is, as movement on the retina there correspond to large displacements in the world and any noise there is greatly exaggerated in estimation.

Error Estimation

The error is estimated as a distance, where positive distances denote a guess outside an object and negative distance would denote that behind a surface and sometimes the object as well. It is calculated by passing the position of the agent halfway through its straight segment and a vector representing its guess of the distance and direction of a surface. If this vector passes through any surface than the distance from the closest surface is taken as a negative distance. If the guess-vector does not pass through any surface the distance to the closest surface is calculated and taken as a positive distance. These error values are then averaged and as it approaches zero it represents better guesses.

Odometric Visual System

As the odometric system shares many parts with the panoramic one, for instance, the collection of retinal input is done at the same time as in the panoramic system. The odometric visual system consists of a single render framebuffer pointed downwards from the same agent location as the centre of the panorama. This view does not rotate with the agent, it only translates its position to that of the agent.

This is done to make good use of the series of translations inherent in the update cycle of the agent. That way the same system of movement as in the panoramic view can be used and within the odometry the effects of different T can be evaluated. The rotation updates each cycle are instantaneous and thus cannot be used to gather visual input in high enough frequency. So, instead of tracking, rotation is ignored and a heuristic like that in an optical mouse is used, which is safe because the orientation of the camera itself is assured. The heuristic employed, while in most likelihood not actually that which is used in an optical mouse but chosen because it can use the same pre-stages the panoramic task needs, can be summed up as follows.

During each translation step the ventral camera is translated to the agents position and N slices of the view are taken. Each slice has its centre in the middle of the view, each rotated from the next by $\frac{\pi}{N}$. Each slice has the same length

as the next and in the default setting this length is the same as the horizontal resolution of the (square) view. Because the sampling occurs at an angle as well this could lead to aliasing, to experimentally verify if this is indeed the case one can vary the view's resolution as a multiplication of the slice-resolution. This other option is taken here as fourfold multiplication of the rendered view's resolution compared to the slices resolution, together with anisotropic filtering to assure one can get non-spatial-aliased pixel-data.

Each slice is separately processed in the same manner as the single slice of the panoramic view. One can use the found retinal velocities to see which of the slices corresponds best to the direction in which the agent moved. This is done by counting the non-zero velocities in each list and the orientation of the one with the maximum count is taken as the orientation of the agents movement W. To find the length of it the average of all found velocities on the slice is taken and the normalised W multiplied by it. The sign of this average, when negative, simply means adding π to the orientation of W.

The length of this vector is of course dependent on the velocities found, but to find the exact world distance it represents the field of view needs to be taken into account as well. This is done by calculating the sine of it and multiplying W with it, the possibility of this is assured through the height of the agent that is set to be 1.0.

The structure of the ground texture is still likely to have an influence on the performance of the odometric sense and to evaluate its performance the agent shall track its own position. The error estimation is then the discrepancy between the real and the agent's internal position at the end of the run.

4 Experimental Setup & Results

4.1 Performance

The preceding sections on the factors that influence the efficiency of the optical flow detection have described the most crucial, such as texturing, eye-structure or behavioural patterns and some of the expected interplay. The actual implementation that followed has already roughly mentioned the manner in which performance in each of the two proxy-tasks is calculated. There are simple suitable metrics for each task and are used as follows.

For the distance estimation task the metric consists of the average error distance to the actual surfaces, a positive value would signify a guess that did not intersect a surface and the distance to it, a negative one does intersect and signifies the distance the guess lies beyond the surface. Thus an average response closer to zero is a better response, although if that is not realistic a small negative response is preferable over a small positive one as that would mean that it at least finds the surfaces, something not quite assured with a positive number. It does mean that the true surface lies in front of the estimated one and could make collision more likely when used for navigation, which can be avoided by having it follow a more cautious heuristic in anticipation of this mistake.

The visual odometry task has a metric that is inherent to it and is also the one used here, namely the error or difference between the estimated and the true position. The best response here would naturally be zero and as there can only be positive distance in this case there is no preferable side to err on.

4.2 Experimental Setup

To relate the performance of the algorithm on the typical workstation used to run it to low-processing capabilities possessing platforms a description of modern typical smartphones is given. These specifications change rather quickly and so are meant only as an indication of what would be reasonable to assume viable. After these specifications those of workstation are described, as well as the libraries and environment used.

Considerations on Platform Related Constraints

To check how well an optical flow based approach could work on a small-scale computing platform such as a smartphone one will need to take into account a few important constraints.

While in the past the most important determinant of performance of a computing device would have been the CPU, in this case it is not the bottleneck because it only needs to be used for the overhead of an implementation. It would facilitate control of sensors and actuators or be otherwise involved in using data generated by the GPU.

The GPU in such a limited system is not quite as powerful as one in a typical workstation because a lot of effort in the design is taken to achieve low power consumption, but with an efficient algorithm it should be quite sufficient to run the fragment shaders for the processing of visual information at the same speed as it is offered by the camera. It is accessible through the OpenGL ES (2.0 or higher) API and is a subset of the normal OpenGL API.

The camera itself has two defining relevant characteristics, the first of which is resolution that will usually be at the very least 640x480 pixels and that is a resolution comparable to ten times that of ommatidia in a single compound eye of a dragonfly.[14] Keeping in mind that the dragonfly has one of the most complex and large eyes of all insects and that phonecameras are usually capable of much higher resolutions this means that resolving detail is unlikely to lead to discrepancies in performance.

The framerate on the other hand is a more important and potentially limiting measure as it defines how many images per second can be retrieved. This is then the reciprocal of the ϵ in the Reichardt detector and by its relation to resolution and field of view it determines the optimal and maximum cruising speeds of the agent. Common framerates are around 30hz, but on the newer models it goes up to 60hz or 120hz and all this usually at a resolution of 1280x720 or higher. It is not trivial to measure the framerate of an insect eye but it would be safe to say it is over 120hz.[21] Any aliasing resulting from low framerates could be mitigated by sampling down to a lower retinal resolution for a larger sampling base, although of course this does mean a degradation of resolving capacity.

Experimental Environment

The software has been written in C++ and GLSL (openGL Shading Language) and compiled with GCC for Arch Linux. It makes extensive use of the OpenGL API and related libraries, very closely related to the more limited OpenGL ES API. Software can be easily transferred from one API to the other without great code-modification, GLSL-code for instance can usually be copied almost one-on-one, with just slight additions to it detailing factors that are set by default in the normal full API. The workstation used runs on an AMD quadcore A10-5750M 2.5GHz with 8GB memory and has an AMD Radeon R9 M290X 2GB graphics processor.

This is clearly much more powerful than the typical smartphone described but this setup is capable of running almost real-time while also generating the entire visual world up to a few hundred times per cycle. It can be seen as an indication of the possibility of running just the processing-code on a smartphone, but because an exact comparison between a workstation and a smartphone is not a goal here no exact measurement or rigorous comparison will be made. On the other hand, the parameters actually influencing the capability of the algorithm, such as retinal and temporal resolution, texture-choice and so on will be compared to each other.

4.3 Results

For each task a run of experiments have been done, all with different settings. These proxy-tasks will be described separately. The relevant shared settings in both tasks can be summed up with the agents velocity being fixed at 1.5 units per translation series and the agent's collision-radius set at 0.5 units, but as the agent checks ahead this can also be seen as a collision-distance of 1.5 units. These can be related to the textural resolution of 1024x1024 which is fully copied to each 1.0x1.0 unit of surface, to find corresponding values for real-world agents one would need to relate them to the relevant metrics.

Surface Distance Estimation Task

For the panoramic task 255 runs were performed, all running for roughly 1300 full update cycles. 1300 update cycles was chosen as this is large enough for an agent to random walk through the entire simulated world many times and thus take in a solid subset of all possible perspectives to be taken. In each run the following parameters were varied: retinal resolution, temporal resolution, texture-choice, use of a sine-gradient and the use of post-distance-processing noise removal.

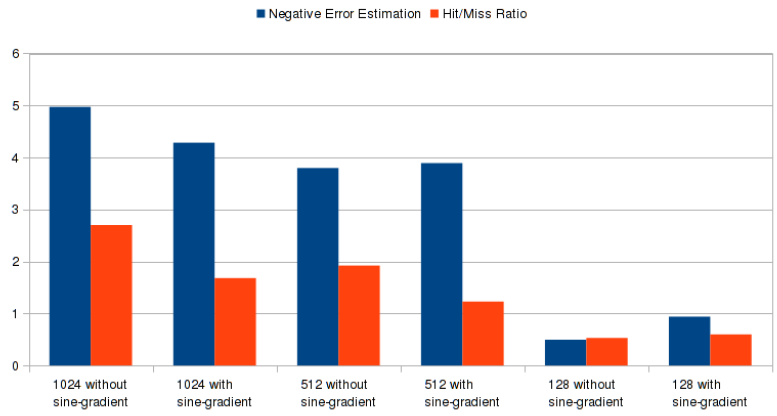


Figure 12: Comparing the average results with or without the sine-gradient enabled.

Figure 12 shows the averaged results of all runs with the stated retinal resolution and use of sine-gradient, while varying all other parameters. Note that the error estimation has been multiplied by -1 for a clearer view, this has been done in every graph of the distance estimation task. The first very noticeable fact is that the use of a sine-gradient in the retina makes a negligible difference and even turns out slightly negative for the lower retinal resolution values. This might be because the greatest advantage of the sine-gradient lies in the points on the retina that are closer to the optical vanishing points and that is also where some data is neglected. This data has been neglected on purpose due to the fact that any noise in that segment of the eye is amplified greatly by the distance calculation algorithm. In any case, it seems that in the current setup applying a sine-gradient has almost no effect and can be disregarded.

Another result in figure 12 that requires a closer look is that, when just the error values are viewed, it seems that the 128 pixel retina performs really well, coming close even to perfectly guessing where surfaces lie. Yet when one takes a look at the hit/miss ratio it becomes clear that out of three guesses two do not even intersect a surface and visual inspection of the resulting maps shows that the 128 retina guesses are spread close to randomly over the entire experimental area available. This seems mostly due to the fact that with 128 pixels to divide 360°, in combination with the chosen method of integrating detector responses over time, leads to heavy noise in the velocity approximations. Different methods of integration might lead to better results, as using a smaller total field of

view might do as well, but in the current setup for this specific retinal resolution the chosen metric does not lead to usable data. As the average hit/miss ratio of 128 retinal resolution is around 0.5 in all cases this resolution shall be disregarded from here on out.

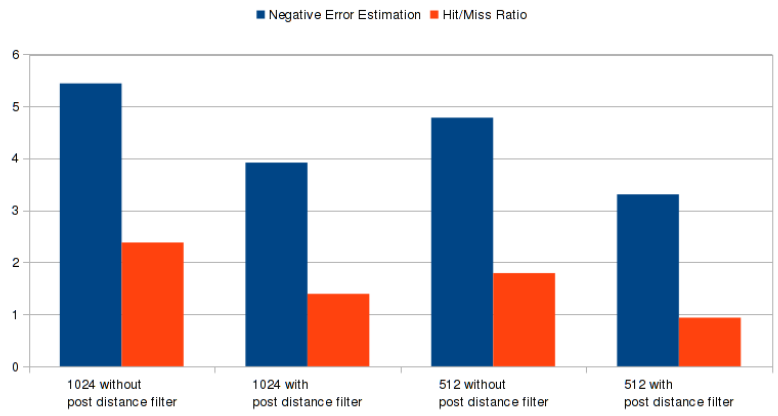


Figure 13: Comparing the average results with or without the post distance calculation filter enabled.

Figure 13 on the other hand shows the results when compared on the efficiency of the post-distance-processing noise removal and it shows a mixed result. In one regard there is a clear improvement on the error estimate where the guesses lie 1.5 units closer to the actual surface when noise is removed.

The other side of it is that the hit/miss ration decreases and while this might seem like a bad result at first sight, it is not. The noise that is removed consists of large swaths of found velocities all lying close to each other, so the number of hits is decreased through choosing the closest and thus best estimate while ignoring the guesses around it. This means less hits because these less accurate guesses will, in all likelihood, lie even further behind the already estimated surface.

The fact that it responds so well means that further analysis can be restricted to the cases with a post-distance-estimation noise filter enabled.

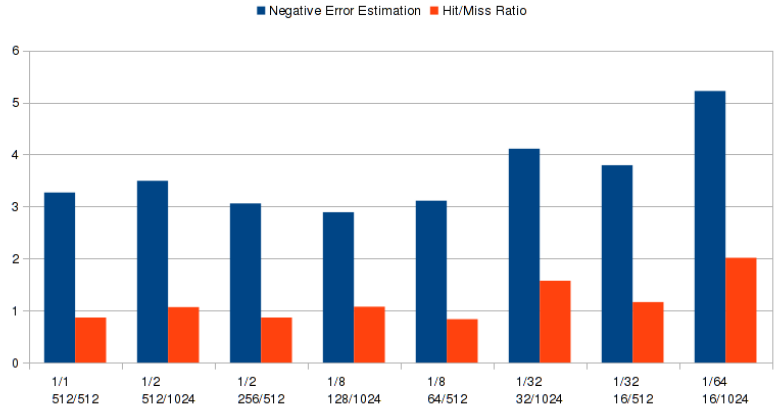


Figure 14: The results of the variation of the ratio between temporal and retinal resolution for 1024 and 512 pixel wide retinas. The x-axis is ordered on the ratio between the resolutions and that ratio is shown with the lowest term on top, with the full ratio of temporal/retinal resolution underneath.

Figure 14 shows the distribution of results when the ratio of the temporal resolution T is varied and its ratio to the retinal resolution. Here it seems that the results hover around -3 units of distance as long as the ratio between the two resolution is greater than $\frac{1}{32}$. With ratios of exactly $\frac{1}{32}$ or $\frac{1}{64}$ one sees that the guesses start to lie further behind the surfaces and although the ratio of hit/miss increases this is not immediately a good thing. Despite the case with retinal resolution 128 where the low ratio was cited as a problem because it indicated a random spread of values, it is different here also for other reasons besides the fact that even this low ratio is still twice that of the 128 case. The higher ratio results from essentially having guesses lie further back, and thus more "hits" behind the surface will be found.

The temporal resolution determines how many intervals of velocity can be found between two photo-receptors and it apparently estimates optimally as long the ratio $\frac{T}{R} \geq \frac{1}{32}$.

Besides that it is also clear that while having a higher resolution does not need to lead to a more correct error estimate it does improve the hit/miss ratio, likely due to more detail becoming perceivable.

Figure 15 shows the difference in results when texture choice is varied, showing the average metrics for all runs $R = 512$ and $R = 1024$ with post-distance-calculation noise removal filter enabled. The results can be grouped together in two categories within which the values do not vary much. The sines can be seen as a group and can be compared to the monochrome "harsher" chequered and noise textures.

The sines together performed worse when looking at the error estimation, despite this their hit/miss ratio is much higher. This can be explained by taking into account the filtering of the texture when an obstacle is at a distance.

The filtering is most strongly noticeable in the noise and chequered textures as there it essentially smooths out visible detail. This leaves visible as only contrast on the retina the outline of the obstacles. So the velocity and distance

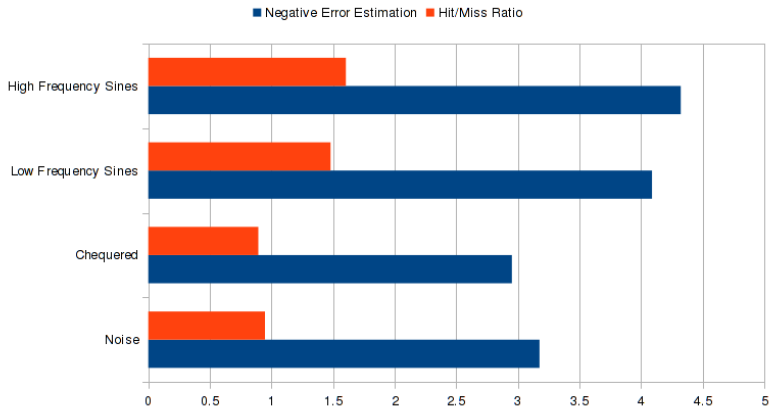


Figure 15: The results of the variation in textural choice.

is calculated only for the outlines, resulting in a low hit/miss ratio. This is due to the fact that the only visible contrast results from a lateral view of a surface and guesses around there will then be distributed close to evenly around and leading to a ratio of around 1.

In the case of the sine-textures this does not happen as they are much less minutely detailed and yield visible contrast at a much larger percentage of distances and this is reflected in the higher hit/miss ratio, due to the fact that the surface is seen head-on and thus many more guesses will intersect it.

If the agent is close to a surface and little filtering is needed the local features of the texture become important and the fact that the chequered pattern has the best estimate is likely because it has by far the most contrast.

Odometric Task

For the odometric task 327 runs were performed, all running for 100 (or more) full update cycles. This is a much lower number than in the panoramic task because in this task there is only one perspective to be taken and the texture simply repeats which does not leave as much room for convergence over time. Some trials were done first with many more cycles but the differential drift did not change much after some dozens of steps and 100 is thus enough for a good result.

The data collected has as a metric the difference between the actual position of the agent and the agents own position as it was estimated through odometry. The internal position is set to the actual position at the start and after this the odometry algorithm returns an estimated displacement vector which is added to the total. This value depends rather strongly on the amount of cycles taken and so the difference is divided by the cycles to arrive at an average per-cycle-drift.

Before the averages are compared a first glance at the results shows that there were 15 runs that ended up with a difference less than 1.5 units. This is an interesting fact because that is the distance moved each step and thus a total drift less than that is a great result. Especially interesting is that in all but one case it was with the noise texture. This was hypothesized to be the

best performing texture for this task because of the lessened expected effect of the aperture problem.

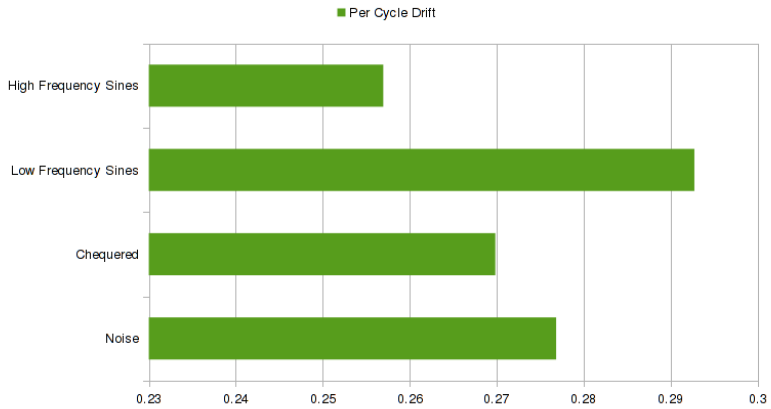


Figure 16: The results of the variation in textural choice.

Now, to be able to compare this to the averages of specific parameters it is useful to keep in mind that these cases had a per-cycle-drift of less than $\frac{1}{1000}$. Unfortunately the bias for the noise texture in the lowest total drift category does not translate to a low average per-cycle drift for it as can be seen in figure 16.

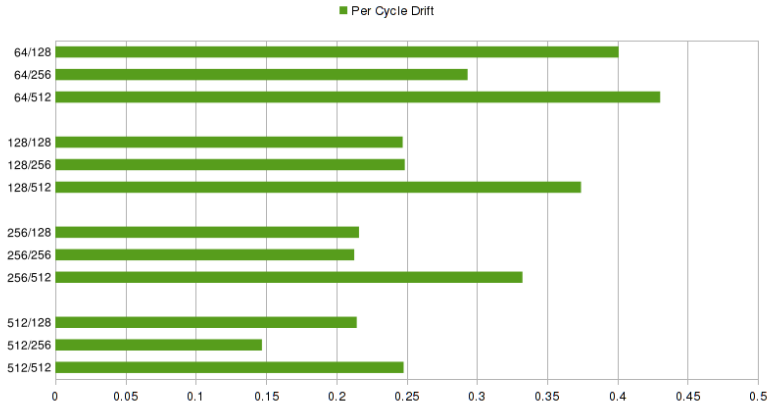


Figure 17: The different average per-cycle-drifts for each combination of resolutions tested, these combinations are listed on the y-axis as temporal/retinal resolution.

Figure 17 shows the average drift decreasing as temporal resolution increases and this is not unexpected. The temporal resolution determines how exact the determination of the retinal velocities can be and is thus important in minimizing the drift.

On the other hand the same list also shows the best results for a retinal resolution of 256 as opposed to the higher resolution as would be expected. Indicating that perhaps somewhere in the process of filtering and sampling some values have not been matched to closely. And thus leading to aliasing of some kind, only to be exacerbated by the aperture problem inherent in two-dimensional motion estimation. As this had been anticipated in a small sense and therefore an extra step had been implemented between the projection to the retina and the sampling of it.

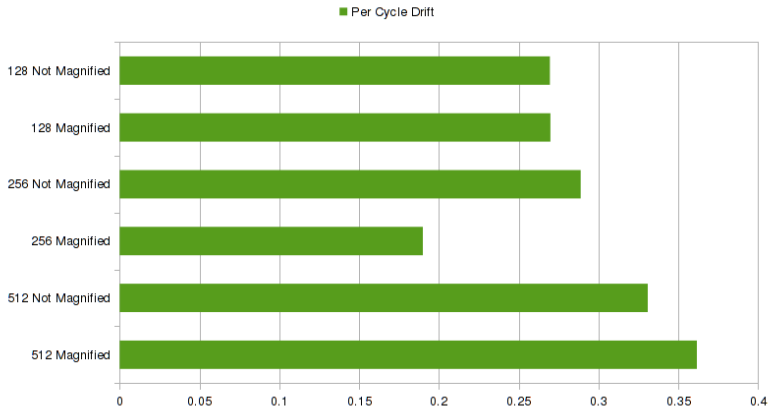


Figure 18: The differences per resolution of using the render magnification or not.

This step consisted of rendering first to a magnified texture and sampling the slices with an appropriate filter to avoid problems with diagonally sampling a rectangular grid. The runs were done with this extra step, with a 4x larger texture in between, and without it. Again, as can be seen in figure 18 the results are ambiguous and it seems that in the highest resolution case it even makes performance worse. Although this could simply be due to the fact that the textural resolution of a patch of ground would have a one-on-one correspondence with a retinal resolution of 1024 when viewed from 1.0 unit distance. In this task this is the case and thus a retinal resolution of $512 \cdot 4$ would mean there is not enough detail to be resolved and leading to obvious sampling problems and 1024 does correspond quite nicely to the best performing case of $256 \cdot 4$.

In general one finds that the performance of the algorithm, even with the lowest average 0.189928 drift for the 256 retinal resolution, when seen in light of the $|V| = 1.5$, still means that it under or over-estimates $\frac{1}{8}$ of the movement every update cycle. For a sand-ant like manner of positioning this clearly would not be accurate enough.

5 Conclusion

This paper has aimed to describe the viability of a GPU-based optical flow detection tactic to be used for navigation tactics inspired by nature. To this end a variety of simple behavioural heuristics in the literature have been listed and described. This was followed by a more in-depth look at correlation-based flow detection and how it relates to the similar way insects and other animals see motion.

To find out if such a system could be used on low computational processing power platforms such as smartphones making use of the GPU, an experimental program was written in which relevant parameters are easily varied. This program was then used to evaluate if a simple motion detector implementation on the GPU runs and how well it does within certain constraints, especially those close to those of the mentioned low processing power platforms.

The behavioural heuristics were not implemented directly though, because they are dependent on the functioning of the EMDs, instead some proxies were formulated: distance estimation and odometry with a straightforward metric for each. The results of these experiments were shown and discussed in light of these expectations. Now these results will be interpreted to see how well this algorithm could be expected to work for an autonomous robot running on a smartphone or something alike.

5.1 Panoramic Perception

The results of the panoramic task have shown that the motion detection and its integration over time are not quite exact, in the sense that the error estimates always fall a bit behind the surface that instantiated them. On the other hand, when using appropriate settings, such as a high enough retinal-resolution to field-of-view ratio, the surfaces are found and more importantly: they generate retinal velocities that represent roughly those the surface makes on the optical array.

The fact that the estimation of distance is not exact is not at all a problem because for a scheme such as hall-centring it is only important that the velocities on both lateral sides of the optical array are found with the same heuristic and can be balanced. A proper value, like ω in the ground-hugging algorithm, for a desirable lateral flow speed can then be found experimentally. This could then always be combined with an autonomy-reducing but mapping-enhancing feature such as GPS.

Clocksin's surface orientation scheme can not be used with this motion detection scheme because of the inherent noisiness over larger spans of the velocities-texture, as shown by the improved performance when using the post-distance-processing noise removal. To find surface orientations one would need much more fine-grained temporal responses from the elementary motion detectors, a note on that will follow in the section on future work.

5.2 Odometry

Although a few runs of the odometric task showed an almost perfect tracking of the agents position, this was not the general case and even the best found general per-cycle-drift of $\frac{1}{8}|V|$ each cycle already precludes any home-seeking

scheme. It is not entirely clear if the error results from a mistake in velocity appraisal or the selection of the correct slice, but if the selection is the problem there could be an improvement to the heuristic. In the current implementation the correct slice was found by finding the one with the most responses, instead one could look for the slice where all responses vary the least from the average, otherwise known as the error-function as it is used in statistics.

On the other hand the fraction by which it errs each cycle means that the found movement will only be off from the actual V by $\frac{1}{8}|V|$ and together with an accurate map of the environment this could help improve positioning by giving an estimated range of where the surfaces found are most likely to really be and which landmarks they could represent.

The exact way it has been used in the experiments is not easily translatable to a robotic implementation because of the fact that the ventral camera is locked to the euclidean frame of reference of the virtual world. In the actual world such a thing is not possible and one would need to track the orientation of the agent as much as possible.

This would require an implementation of something similar to the HSE-cell in the fly, combining it with input from the gyroscopic sensors that are part and parcel of most smartphones. This combination of senses is an important one, as it means that using a fast but not exactly perfect algorithm is not a recipe for mistakes as the multiple senses can augment each other. Besides that, the tracking of rotation in such a manner is also rather useful for making sure that the straight segments of the saccade-like flight are kept straight, through the fast compensation of unwanted rotation such a tactic enables.

5.3 Implementation on a Smartphone

The essential part of having the described algorithm running on a smartphone is that even with the rather naive scheme of simply copying slices of the visible world over time into a texture and simply iterating over them leads to actual motion recognition and is sufficient for finding moving surfaces. It runs on a GPU easily and does not require very sophisticated software. In fact most of the program consists of code for generating the world itself and projecting it to the simulated retinas.

A smartphone on the other hand would be receiving images from its onboard camera, likely copied directly to the memory directly accessible to the GPU and then processing them. Each cycle requires only two processing steps before motion can be calculated, each new frame only means a slice needs to be copied and during this copying part of the band-pass filter can already be applied. Before a possible improvement is discussed that would be mandatory for a flying agent it is important to note that the crude implementation of integration over time here already fits well with the characteristics of a smartphone as the brain of a ground-bound robot.

For instance, it was noted that the optimal performance of the surface estimation task was generally achieved when the ratio of $\frac{T}{R} \geq \frac{1}{32}$. If we translate this to a smartphones characteristics it is helpful to assume the camera of paramount importance and derive a desirable retinal resolution from this. With a camera framerate of 60 frames per second one could have straight segments of duration 1s and the resolution for optimal detection would be just 960 with a ratio of $\frac{1}{16}$.

For a one-dimensional retina, sufficient for a ground-bound robot, it is certain it will run fast enough because the only action necessary each consecutive camera frame is the copying of a slice. This means that one could take 1280 pixel wide frames at 120Hz and have more than sufficient resolution for a full 360° panoramic view. And if one lowers the resolution there could even be multiple updates each second of for instance $T = 40$ with $R = 640$ while still having a sufficient resolution for a panoramic view and the short duration of each cycle would make it safe to neglect any rotation during one.

For a flying agent on the other hand one would need a two-dimensional retina and it is helpful to see why this would lead to some performance problems. A frame of 960x960 for instance has many more pixels than the compound eyes of even dragonflies and is easily registered by an inbuilt camera. Such a resolution could be lowered tenfold before it drops to a level comparable with that found in a fly, yet without also realizing the high framerates they posses and which helps explain their capabilities. Down-sampling through Gaussian convolutions could simulate the fixed-lens optical properties of the ommatidia and as shown by Borst and Egelhaaf this specific blur is very effective in avoiding spatial aliasing, as long as the resulting sensitivity distribution is matched correctly to the sampling base $\Delta\varphi$.[2] Thus the first stage of the velocity detection process is instantiated, but would require heavy use of the GPU each frame and whether this is within the capabilities of an inbuilt graphics processor is something that remains to be seen.

The simple accretion of slices in a two-dimensional texture would not suffice for it, here a three-dimensional storage texture would be needed. Storing in that three-dimensional texture even only one second worth of footage would mean roughly 165MB of space is necessary for a retinal resolution of 960x960. Besides bandwidth problems when a one-on-one copy of the visual input is required there is the fact that this much data to process is unlikely to be within the capabilities of the GPUs in typical smartphones. Decreasing the framesize as described would be a possibility as long as the blur of an ommatidia can be performed because otherwise the boundary, with 360° field of view, is somewhere upward of 128 pixels in each dimension. This would bring the required storage and bandwidth within the more reasonable domain of just dozens of megabytes but it seems better to improve the algorithm instead of lowering the resolution.

5.4 Future Work

A more mature algorithm for integration of velocities over time is thus important because of the stated problems with keeping a straight path in a three-dimensional world and the cubic increase in data to be processed when one raises R or T. Combined with the fact that two-dimensional detection of motion requires some heuristic that determines the non-relevant part of the response of the EMD that is due to structure and the relevant part that is due to motion gives possible directions in which the algorithm could be improved.

The aperture problem could be overcome in a variety of manners, such as figure-ground discrimination as per Reichardt or other biologically inspired heuristics such as those responsible for false-motion illusions in humans for instance. Combining this with output from an imitation HSE-cell-like algorithm to define the position of previous frames in relation to the current frame would

make it possible to disregard small rotations and have a more continuous perception.

A real-time perception can perhaps be facilitated by incrementally compressing older frames in the temporal dimension so that the fastest motions can be found between the current frame and the previous frame, while slower motions would then be found be comparing the current frame to a temporally compressed older frame. For instance there could be a buffer of only 16 frames, each twice as long ago as the frame before was from the current frame. This would require a low processing framerate and only 44MB of storage for 960x960 frames while still allowing for responses within $\frac{1}{60}$s without losing the slower-movement-detection with much larger ϵ.

Besides these possible improvements to the algorithm it would of course also be interesting to use the existing algorithm as it is and see how well it can be used for the possible navigation tasks and see if the velocity detection is indeed robust enough for an agent to manoeuvre through obstacle-rich environments with minimal collisions and to see if the behaviour it then displays corresponds to those found in natural agents. It would also be helpful to see if the complete indifference of the algorithm is due to a too high C by comparing the results for a number of different sizes for C.

Of course an actual robotic implementation, flying or otherwise, is something that would be a great experimental tool to see if conclusions from the simulation hold in reality as well. Besides that one can then also find the actual runtime on smartphones or comparable devices.

5.5 Conclusion

This paper has shown that even a crude implementation of a perceptual system for motion can use visual data to build a representation of the world, as long as action and perception are coupled. Such a coupling has long been understood as crucial in cognitive research where studies of the brain never find clear boundaries between systems involved in perceiving and acting within modalities.

Considering this, a crucial notion for the future of Artificial Intelligence then would be the need for coherent autonomous entities, by which agents are meant that use the responses of their sensors completely coupled with their actions, which then produces and constitutes perception. An agent that has access to such data can then use it for other heuristics such as, but not limited to, path-finding that can be implemented on top of that data in a more traditional symbolic manner. This would allow it to build its own representation if it misses one by making use of hall-centring, ground-hugging and other surface or object avoidance tactics for the exploration of space and use its own constructed knowledge for further tasks or share this information with an operator or other agents. Of course this is only a single example, it does however already indicate the possible use of such a perceptual system and it is very likely that other tasks could benefit as well.

Motion is of course only a small part of the information contained within the visual modality and this could be extended to include form and other features of visual perception. Multiple subsystems each working within a modality could be constructed and combined with each other, for instance the briefly mentioned gyroscopic sensors could be used for proprioceptive perception by imitation of a vestibular system, that way a coherent perceptive system can be created that

then creates data structures out of the world for use by other "higher" level algorithms.

To have the project of Artificial Intelligence lead to actually intelligent agents then, I believe it is necessary to recreate all the unconscious perceptual underpinnings that are common in the animal kingdom, commonly called subconscious in humans, for without these an intelligence would miss grounding in reality and any reasoning without that is fruitless. I hope that the reader is convinced of the viability of using the GPU for the small part of such a subsystem that motion detection comprises.

References

[1] J.L. Barron, D.J. Fleet, S.S. Beauchemin, *Performance of Optical Flow Techniques*. International Journal of Computer Vision, Vol. 12, No. 1, pp. 43-77 (1994).

[2] A. Borst, M. Egelhaaf, *Principles of visual motion detection*. Trends in Neuroscience, Vol. 12, No. 8 (1989).

[3] W.F. Clocksin, *Perception of surface slant and edge labels from optical flow: a computational approach*. Perception, Vol. 9, pp. 253-269 (1980).

[4] M. Egelhaaf, A. Borst, W. Reichardt, *Computational structure of a biological motiondetection system as revealed by local detector analysis in the flys nervous system*. Journal of the Optical Society of America A, Vol. 6, No. 7, pp. 1070-1087 (1989).

[5] M. Egelhaaf, R. Kern, *Vision in flying insects*. Current Opinion in Neurobiology, Vol. 12, pp. 699–706 (2002).

[6] N. Franceschi, J.M. Pichon, C. Blanes, *From insect vision to robot vision*, Philosophical Transactions: Biological Sciences, Vol. 337, No. 1281, pp. 283-294 (1992).

[7] N. Franceschini, *Visual guidance based on optic flow: a biorobotic approach*. Journal of Physiology - Paris 98, pp. 281–292 (2004).

[8] N. Franceschini, F. Ruffier, J. Serres, *A Bio-Inspired Flying Robot Sheds Light on Insect Piloting Abilities*. Current Biology, Vol. 17, pp. 329–335 (2007).

[9] M.O. Franz, H.A. Mallot, *Biomimetic robot navigation*. Robotics and Autonomous Systems, Vol. 30, pp. 133–153 (2000).

[10] J.J. Gibson, *The Problem of temporal order in simulation and perception*. Journal of Experimental Psychology, Vol. 62, pp. 141–149 (1966).

[11] K.G. Gotz, *Die optischen Übertragungseigenschaffen der Komplexaugen von Drosophila*. Kybernetik Vol. 2, No. 5, pp. 215-221 (1964).

[12] F. Iida, *Biologically inspired visual odometer for navigation of a flying robot*. Robotics and Autonomous Systems, Vol. 44, pp. 201–208, (2003).

[13] J.J. Koenderink, *Optical flow*. Vision Res Vol. 26, No I, pp. 161-180 (1986).

[14] M.F. Land, *Visual Acuity In Insects*. Annual Review of Entomology, Vol. 42, pp. 147–77 (1997).

[15] D. Lambrinos, R. Möller, T. Labhart, R. Pfeifer, R. Wehner, *A mobile robot employing insect strategies for navigation*. Robotics and Autonomous Systems, Vol. 30, pp. 39–64 (2000).

[16] C. McCarthy, N. Barnes, *Performance of Optical Flow Techniques for Indoor Navigation with a Mobile Robot*. 2004.

[17] H. Nyquist, *Certain Topics in Telegraph Transmission Theory*. 1928.

[18] W. Reichardt, *Evaluation of optical motion information by movement detectors*. Journal of Comparative Physiology A, Vol. 161, pp. 533-547 (1987).

[19] W. Reichardt, R. W. Schlögl, *A Two Dimensional Field Theory for Motion Computation*. Biological Cybernetics, Vol. 60, pp. 23-35 (1988).

[20] K. Perlin, *An Image Synthesizer*. Computer Graphics, Vol. 19, No. 3, pp. 287-296 (1985).

[21] R. de Ruyter van Steveninck, A. Borst, W. Bialek, *Real time encoding of motion: Answerable questions and questionable answers from the fly's visual system*. 2000.

[22] A. Sherman, M.H. Dickinson, *A comparison of visual and haltere-mediated equilibrium reflexes in the fruit fly Drosophila melanogaster*. The Journal of Experimental Biology, Vol. 206, pp. 295-302 (2002).

[23] M.V. Srinivasan, J.S. Chahl, K. Weber, S. Venkatesh, M.G. Nagle, S.W. Zhang, *Robot navigation inspired by principles of insect vision*. Robotics and Autonomous Systems, Vol. 26, pp. 203-216 (1999).

[24] M.V. Srinivasan, *Small brains, smart computations: Vision and navigation in honeybees, and applications to robotics*. International Congress Series, Vol. 1291, pp. 30–37 (2006).

[25] A.D. Straw, S. Lee, M.H. Dickinson, *Visual Control of Altitude in Flying Drosophila*. Current Biology, Vol. 20, pp. 1550–1556, (2010).

[26] D'Arcy Wentworth Thompson, *On Growth and Form*, page 45. University Press, Cambridge. 1917.

[27] E.J. Warrant, P.D. McIntyre, *Arthropod Eye Design and Physical Limits to Spatial Resolving Power*. Progress in Neurobiology, Vol. 40, pp. 413-461 (1993).

[28] A.M. Wenner, *The Elusive Honey Bee Dance "Language" Hypothesis*. Journal of Insect Behaviour, Vol. 15, No. 6 (2002).

`The code of the implementation and result-data can be found at:`
`https://github.com/Falafelski/Bachelorthesis`
`http://jorisgoosen.nl/Thesis/Sourcecode-Bachelor-Thesis-JC-Goosen.tar.gz`