Robot Control using a Neuromorphic Visual Sensor

Ivan Konstantinov

Master of Informatics

School of Informatics

University of Edinburgh

2012

(Graduation date: June 2012)

Abstract

The goal of this project was to use a neuromorphic camera, the DVS 128, in real-time robot control problems. An optical flow algorithm was developed to handle the pre-processed image output from the silicon retina, and a simple routine was designed to control a robot vehicle. The optical flow algorithm was inspired by observed flight behavior of some birds and insects, such as honeybees, flies, and pigeons. An experiment aimed at analysing honeybee navigation strategies was used as a model for testing the implemented algorithm.

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. J. Michael Herrmann, for all the invaluable help and advice he provided me with during the course of this project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ivan Konstantinov*)

Table of Contents

Chapter 1

Introduction

Frame-based cameras have gained widespread recognition throughout the area of computer vision and robotics as the preferred tool for processing visual input data. These devices acquire the visual information using a continuous stream of "snapshots", or frames, which are recorded at discrete times [1]. Among the many advantages of frame-based devices are the small-sized pixels, which can lead to greater resolutions, low device costs, and a well understood output format [2].

Some research however, has shown that conventional cameras may be unsuitable for robot control problems in real environments [3]. Real environments have a far greater dynamic range than what standard frame-based cameras can provide [4] and its high dynamics and unpredictable change can make the extraction of useful information difficult and resource-consuming. Frame-based imagers would require greater amount of both processing and storage resources as they deal with more "realistic" scenes: ones in which light conditions change rapidly, or events, such as moving objects, occur in unpredictable fashion. In such cases, the imager would be required to operate on high frame rates, have a high dynamic range[1] to reduce its sensitivity to light variations, and produce smaller outputs to allow for quicker scene processing. With increasing frame rates, frame-based cameras produce large amounts of mostly redundant data output, since pixels are repetitively sampled even if their light intensity value has not changed since the last frame was created, which can seriously compromise overall performance.

Based on David Marr's observation that biology has a lot to offer to machine vision

[1] Dynamic Range is defined as the ratio of the brightest and darkest values in an image. Most digital images have a dynamical range of about 100:1, while humans can perceive detail in dark regions when the range is even 10,000:1 [5].

research in terms of using fast, efficient and parallel processing pathways [6], new types of VLSI (Very-Large-Scale-Integration) sensors have been designed based on key properties of biological vision, which is highly complex and extremely efficient, and has experienced millions of years of evolution and development. These neuromorphic devices[2] abandon the frame-based principle of traditional cameras, respond asynchronously to spatial and temporal contrasts [2] and are event-driven, mimicking the way spike patterns are formed within the biological retina [7]. Using such complex circuitry, these imagers achieve greater dynamic range and pixel bandwidth (at the expense of having larger and more sophisticated pixels), small data outputs and very low data redundancy.

A hypothesis exists that states that neuromorphic sensors would be quite useful in real environment situations, which require fast, efficient and reliable processing and analysis of the visual information even in restricted lighting conditions [8]. The benefits of using sparse, event-driven data output allows these sensors to process the visual scene with low computational and resource costs, and with short response latency, making them a good alternative to conventional machine vision systems, which tend to produce huge amounts of data at high frame rates [9].

The goal of this project is to explore this hypothesis by developing a robot navigation system using the Dynamic Vision Sensor (DVS) described in [2], and to assess its potential benefits and advantages when applied in real environment robot control problems. The neuromorphic imager provides a simplified, but accurate and reliable representation of the image preprocessing that happens in a biological retina, before the visual information is forwarded to the brain. It differs considerably from traditional frame-based cameras, employed in machine vision, by abandoning the frame concept, and instead encodes perceived luminance changes asynchronously (i.e. as they occur and not bound by an external clock, such as a frame rate), reducing the size of the data output, while focusing on the more interesting and relevant information. Current implementations of the DVS sensor include real-time tracking of moving objects [10], analyzing fluid dynamics [11], and speed estimation of moving vehicles [12]. This project, therefore, may provide a contribution to the ongoing research in bio-inspired machine vision applications.

To achieve this goal, a navigation system is developed, which consists of an optical

[2]The term “neuromorphic” was first coined by Carver Mead when referring to artificial systems that adopt the form of neural systems [1].

flow algorithm, running on a workstation computer, two controller applications, responsible for issuing commands to a *Bioloid* robot vehicle, and a relay program, which is used to forward messages between the robot and the workstation. The robot is programmed to navigate along the midline of a narrow corridor (roughly 60 cm wide), correcting its position whenever it has drifted away from the middle.. The navigation scenario is inspired by a biological experiment described in [13], which analyzed the flight patterns of honeybees when trained to navigate through a narrow corridor. In the experiment, it was observed that bees tend to fly through the middle of the corridor, at an equidistance from each wall, by balancing the perceived image velocities on their two eyes.

The optical flow algorithm is modeled after the one described in [14], which was inspired by optic flow reduction strategies observed in certain flying animals, such as flies [15], or pigeons [16]. A neural network consisting of software-implemented Leaky Integrate-and-Fire (LIF) neurons is created to process the visual data from the DVS sensor and extract velocity information pertaining to the left and right sides of the image, which correspond to the left and right corridor walls, respectively. The velocity information is then passed to a controller program, which derives a responsive strategy for the robot to execute. The strategy, in the form of a movement command, is transmitted to the robot wirelessly via a relay application, and executed by the robot's own controller (cm510). Whenever the robot is done executing a command, it would inform the workstation and wait for new commands.

The experiments are conducted using a set of rubber grey sections forming a narrow corridor. Each section is taped with vertical white stripes to induce a flow field used in the visual processing. The stripe patterns on both walls, as in [13], is randomly selected and not symmetrical.

It is expected that the DVS camera will contribute to the overall performance of the system, by minimizing the amount of resource-heavy and complex data reduction operations, allowing the optical flow algorithm to use only the relative and important visual data necessary for steering the vehicle. One potential challenge may be the lack of detail resulting from the relatively slow egomotion of the robot, along with the forward positioning of the camera.

This report will proceed in the following way. The neuromorphic approach to machine vision and the underlying biological concepts, on which this approach is based, will

be described in Chapters 2 and 3, respectively. An overview of prior research work is presented in Chapter 4. The project setup will then be outlined in Chapter 5, followed by a detailed description of the actual design and implementation in Chapter 6 and an analysis of the results from conducted experiments and tests in Chapter 7. The report will conclude with a discussion on future improvements and extensions in Chapter 8.

Chapter 2

Machine Vision Concepts

Conventional machine vision systems use CCD or CMOS cameras, which acquire visual information using a continuous stream of frames, taken at discrete periods of time based on an external clock, the frame rate [1]. An image frame can be defined as the two dimensional matrix representation of the spatial locations of intensities of the projected scene and is computed by sampling all pixels[1] for light intensity values. The concept of using image frames, created at a predetermined frame rate, to understand and interpret the environment has such an impact on the field of computer and machine vision, that it is usually taken for granted and has led some researchers to believe that a similar process happens in the biological eye, a belief that has not been supported by a biological proof [17].

2.1 Extracting Image Dynamics using Optical Flow Algorithms

Numerous algorithms have been designed over the years to extract motion information from a scene. Such algorithms derive the dynamical properties of a scene by computing the optical flow field, induced from the viewer's egomotion or from different moving objects. Optical flow is defined as the distribution of apparent velocities of movement of brightness patterns in an image, which arise from the relative motion of objects

[1]The size of a traditional pixel is $7\times7\mu m^2$, which allows for high image resolutions to be achieved. However, it has been observed that some visual tasks require considerably fewer number of pixels [4]. For example, some insects display complex behaviour while using less than 10,000 pixels for visual navigation and object recognition.

and the observer [18]. The flow field of an image can be used for a variety of tasks, such as passive interpretation of scenes, object detection and tracking, or active and autonomous exploration [19].

Traditional methods for computing the optical flow field of a scene use image sequences, recorded at a specific rate (the frame rate), in order to measure the displacement velocities and directions of regions of interest. Many of these methods have been observed to apply three separate processing stages [19] in order to produce the 2-dimensional motion field. Firstly, the signal-to-noise ratio of the image is enchanced by using low-pass or band-pass filtering operations. Then, basic measurements are extracted from the filtered signal (like spatiotemporal derivatives), which are finally integrated in order in to produce the optic flow field. The integration process may require that certain assumptions are made about the underlying flow field, such as the global smoothness constraint [18].

2.2 Limitations of Frame-Based Cameras

Frame-based vision systems benefit from a long history of research and the designed algorithms using frame-based cameras are very well understood. However, several drawbacks have been identified, such as high computational costs, limited dynamic range, high redundancy and large output size among others. Image processing algorithms on a digital computer can be extremely expensive in terms of computational complexity [2], because of the numerous serial operations involved. For example, the computational cost of a filtering procedure has been observed to grow in a linear fashion with the increase of the image resolution [4].

Since all pixels are continuously sampled for luminance change, even if their value has not changed since the last time they were sampled, high amounts of redundant data is produced. The transmission of redundant data can compromise performance, especially in cases in which limited processing resources are available[2], and such data is actually not used in the image processing operations: instead, computationally expensive algorithms are applied to eliminate redundancy.

For real environments with natural lighting, such limitations can decrease the efficiency

[2]It has been estimated that a camera with 352×288 pixel resolution, running at 10 kFPS, has a data output rate of $>$1GB/s [2].

of the visual system. For example, the amplitude of light intensities in the natural world is far greater than a frame-based CCD camera's dynamic range [4]. Such a disparity may cause blooming, which has the property of saturating the CCD pixels and causing a charge overspill to neighboring pixels, thus obscuring fine details in an image [5]. Furthermore, the concept of an image frame does not exist in biology: the world works in asynchronous manner and in continuous-time [1]. If image frames are used, precise temporal localization of events cannot be achieved, and the temporal dynamics of a scene are often lost [17], because things tend to happen between frames and information gets lost.

As an alternative to applying conventional machine vision systems in real-environment problems, researchers have designed sensors which are modeled after observed processes in biological vision.

2.3 Bioinspired Approach to Machine Vision

Since a robot vision system requires a compact, robust and power saving architecture in order to perform well in a real dynamic environment [3], machine vision research should try to keep a common trajectory with biological vision research, because both sides can greatly benefit from one another [6]. Considerable effort has been put into understanding how biological vision works and results have shown that animals possess a highly robust and adaptable sensory systems, capable of quickly and efficiently processing dynamic and complex scenes[20].

An example of a bioinspired approach to machine vision is the design of neuromorphic sensors, which simulate the preprocessing functions of the biological retina using analog CMOS integrated circuits [17]. Neuromorphic imagers abandon the notion of a frame and instead handle visual infocmation on the level of individual pixels. Rather than continuously sampling every pixel for luminance change, each pixel individually and independently transmits its value when a change has been registered. The underlying idea is that a faster and more reliable implementation can be achieved by shifting some of the image processing to the individual pixel [8]. This concept draws a close analogy to how the biological visual system operates: while the visual cortex is responsible for the majority of image processing operations, some of that processing already occurs in the retina circuitry before the image is transmitted to the brain.

2.4 The DVS 128 Sensor

The camera that was used in the project is a neuromorphic sensor, a 128×128 CMOS (*Complementary Metal-Oxide-Semiconductor*) Dynamic Vision Sensor, or *DVS*, which models the biological retina in terms of spatiotemporal filtering, sparse event-based output, response to relative luminance contrasts, and separation of positive (*ON*) and negative (*OFF*) signals into two output channels [2]. Spatiotemporal filtering means that the perceived image is sharpened in both space and time (called *lateral inhibition*) [21], while event-based output, or *Address-Event Representation*, mirrors the spike patterns that are formed in biological vision.

As in other neuromorphic sensors, the concept of an image frame is replaced by a spatio-temporal collection of luminance *events*. Each pixel[3] in the DVS sensor models a simplified 3-layer retina [1] and adopts an abstraction of the photoreceptor-bipolar-ganglion cell information cell. It independently and continuously registers relative changes in light intensity and generates a spike whenever this change has exceeded a predetermined global threshold. Pixels respond asynchronously to intensity changes, in other words they are not bound to a global clock, such as the frame rate. Therefore, information regarding the temporal dynamics in a scene is precisely encoded with the generation of events.

The output of the silicon retina sensor is in the form of an asynchronous stream of pixel address-events (*AEs*) that directly encode scene reflectance changes, and therefore greatly reduce transmitted data redundancy, while preserving the timing information of each event. The sensor puts more emphasis on the pixel design giving it more structure and importance, compared to frame-based cameras. Each pixel continuously monitors its state for any changes in its photocurrent [9]. When such changes occur, it responds with an address-event[4], which can be positive (*ON*) or negative (*OFF*), triggered by positive and negative temporal contrasts respectively.

With the use of a fast logarithmic photoreceptor circuitry, the DVS128 silicon retina achieves low pixel mismatch, a wider dynamic range (compared to both frame-based imagers and some retinomorphic sensors), reduced data redundancy (as only pixels

[3]Research in the field of neuromorphic sensors has not come up with a general definition of how big a single pixel needs to be [2] The DVS128 camera uses pixels of size $40 \times 40 \mu m^2$, while different pixel designs may be found from one report to the other: $69 \times 69 \mu m^2$ [8], $9.4 \times 9.4 \mu m^2$ [22], $25 \times 25 \mu m^2$ [23].

[4]The output of the DVS128 chip is a 15-bit digital address which has the 7-bit x and y coordinates of the event and an ON/OFF polarity bit.

(a) (b)

Figure 2.1: Hardware architecture of the DVS128 silicon retina. (a) Die photo of the process chip with a single pixel layout. Source: http://siliconretina.ini.uzh.ch/wiki/index.php. (b) The DVS128 chip components. Source: http://sourceforge.net/apps/trac/jaer/wiki/Hardware

that have registered luminance changes are processed), high contrast sensitivity and fast response time to changes within the scene [2]. Table 2.1 outlines the funcional specifications of the silicon retina.

Pixel size	$40 \times 40 \mu m^2$
Pixel fill factor (%)	9%
Pixel Array size	128×128
Die size (mm^2)	6.0×6.3
Chip interface	15-bit word-parallel AER
Computer interface	USB 2.0 Windows or Linux driver. Java API
Dynamic range	120 dB
Response latency	15μs at 1 klux chip illumination
Maximum events per second	Approx. 1M events/sec
Lens focal length (mm)	6
Horizontal/Vertical field of view (○)	46.2

Table 2.1: Functional specifications of the DVS 128 sensor. Source: http://siliconretina.ini.uzh.ch/wiki/index.php

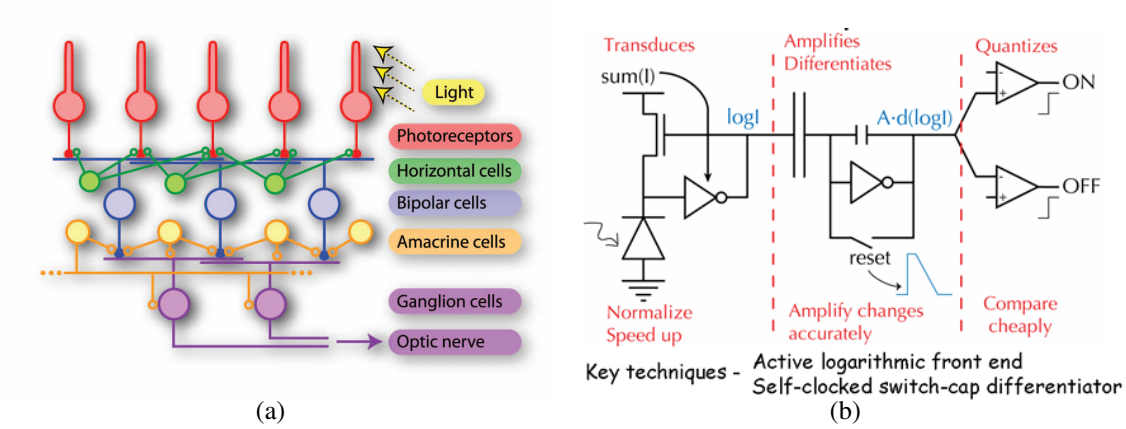
The DVS imager was quite useful for achieving the project objectives, because of its asynchronous and robust properties. There was no need to perform iterative filtering operations, which are both complex and resource expensive, since the visual output is already preprocessed on a lower hardware level. This preprocessing allows the algorithm to only use those portions of the image in which movement has been detected. This feature was helpful when navigating through the corridor, because only the apparent movement of the two walls were required for estimating the velocity difference and

reacting accordingly. On the other hand, the robot was required to constantly move, as it would otherwise be "blind": it would not be possible, for example, to identify the robot's position within the corridor before moving to readjust, as the camera is not applicable in static scene analysis.

Chapter 3

The Biological Vision System

As mentioned in the previous chapter, the DVS camera is a simplified model of a 3-layer vertebrate retina [1], which implements an abstraction of the photoreceptor-bipolar-ganglion cell information flow [Fig. 3.1]. It is then useful to gain a basic understanding of what comprises the retina and how it functions. The following section will provide a brief description the structure of the vertebrate retina.



(a) (b)

Figure 3.1: Comparison between the retina neural circuit and the design schematic of the DVS pixel. (a) Schematic of the retina network. Photoreceptors transduce light stimuli into electrical signals, which are forwarded to the bipolar cells, acting as feed-forward mechanisms from the photoreceptors to the ganglion cells. The ganglion cells form the output layer of the retina. Source: [1]. (b) Design schematic of the DVS pixel, separated in three components which relate to the functions of the photoreceptor, bipolar and ganglion cells, respectively. Source: http://siliconretina.ini.uzh.ch/wiki/index.php

3.1 Structure of the Vertebrate Retina

The retina is a highly complex component of the early visual system[1]. It is even sometimes considered a displaced region of the brain itself [24]. Despite its accessible location which enables more thorough research, however, the retina appears to be more a complex mechanism than what some scientists previously believed [21].

5 different types of neurons have been identified in the retina of most vertebrate animals [Fig. 3.2] and each one has several variations according to morphological characteristics and dendritic stratification patterns [21]. The neuron is the basic information carrying component of the nervous system and its primary function is to transmit information in the form of action potentials, or spikes [Fig.3.3]: a spike is an electrochemical response of a neuron to a present external stimulus [25].

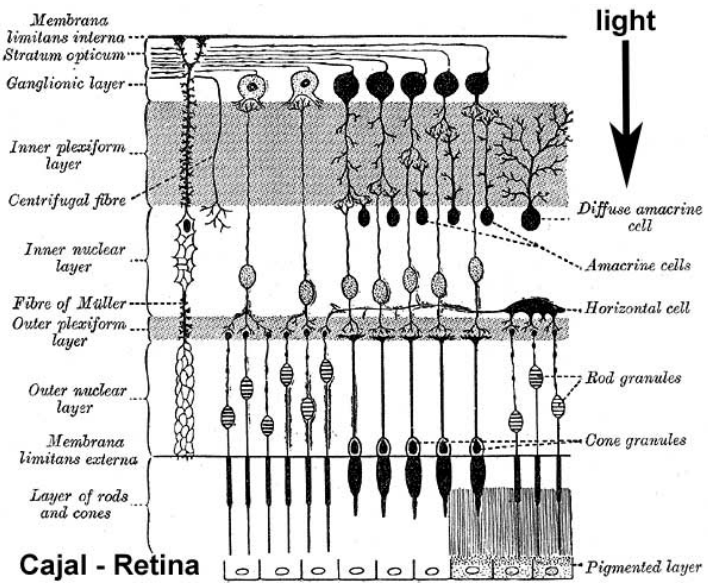


Figure 3.2: *Neural structure of the biological retina, described by Cajal. The last layer to come in contact with incoming light is the one containing the photoreceptor cells, cones and rods. Source: http://embryology.med.unsw.edu.au/Notes/eye11.htm*

The structure of a neuron consists of a *body* (cell soma), *dendrites*, and an *axon*. The function of the dendrites are to transport any incoming information from other neurons to the neuron's soma, while the axon serves as the main transmission line for outbound

[1]The early visual system is defined as the set of processes that make use of two dimensional intensity arrays to recover distance, texture, and other physical properties associated with the surfaces of 3-dimensional objects [4].

stimuli[2]. Axons and dendrites form a vast network which connect numerous neurons together, forming complex bodies, such as the brain or the central nervous system.

It is believed that the retina performs a basic filtering function: it conveys the visual image through the optic nerve to the brain, where complex processes take place to interpret and process the image data [21]. This viewpoint has been criticized, however, due to its failure to provide an explanation as to why a simple spatio-temporal filter would require the use of 5 different and complex types of neurons, some of which are not yet fully understood by scientists[3].

It has been suggested that some form of image pre-processing already takes place in the retina before the visual image is transmitted to the brain (in the form of spike patterns produced by retinal ganglion cells). For example, it has been observed that the retina is capable of manipulating the sharpness and contrast of a visual image (using *lateral inhibition*, which is the ability of a firing neuron to supress, or reduce the activity of, its neighboring neurons [21]), detect object motion [26], discriminate between local and global motion [27], and even anticipate, or predict, the location of a moving object [28]. Therefore, it is argued that the downstream areas of the brain receive an already preprocessed volume of useful information.

There are limitations to our knowledge of the early visual system. The main point raised in [21] was that perhaps we have not fully understood the function of the retina, as it provides a more complex circuitry of neurons than it would be necessary to perform simple filtering and sharpening operations. Another point of view states that an adequate model of visual responses should provide the means to predict responses to arbitrary stimuli [29]. In other words, the claim that we understand the purpose of the early visual system can only be valid if reasonably accurate predictions are made to neural responses to arbitrary stimuli, including those in nature. Therefore, it may not always hold true that neuromrophic visual sensors are modeled after factual biological concepts.

While the neuromorphic camera is modeled after the vertebrate retina, it does provide an interesting analogy to insect vision as well. Its state of "visual blindness" whenever the scene is static resembles the strong relationship between action and perception

[2]There have been identified, however, some exceptions to this rule, such as neurons without any dendrites, or an axon of one neuron connecting to an axon of another neuron.

[3]Amacrine cells are known to provide two types of responses, transient and sustained, but the purpose of those cells is not yet understood [24].

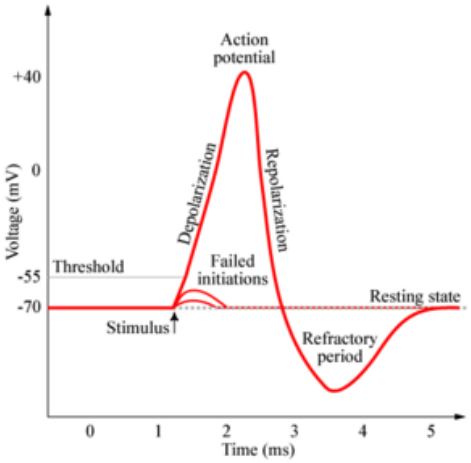


Figure 3.3: *Different stages of an action potential. When a stimulus is present within the receptive field of a neuron, the neuron's membrane potential increases until reaching a threshold value (-55 mV in this case). At this moment, a spike, or action potential, is triggered and the membrane potential quickly rises, or depolarizes (becomes more positive) until it reaches a peak value (+40 mV), then falls down, or repolarizes at the same rate before reaching the initial resting state. Source: http://www.websters-online-dictionary.org*

observed in insect species [20]. The robot, which uses the DVS camera to sense its environment, is required to be an active observer: it cannot derive visual information about its surroundings unless it is constanly moving, similarly to the lack of stereo and depth perception in insects, which requires them to use motion information as their primary visual cue in order to infer distances to objects.

The following section will describe different strategies, which certain insects have been observed to exploit in order to achieve various navigational goals.

3.2 Insect Vision

The project draws inspiration from observed visual strategies of certain flying insects, most notably the "centering" response of bees [30] and the active reduction of the optic flow field to its translational components in houseflies [15] (also observed in pigeons and certain birds [14]). It was therefore essential to also look into the intricate structure of the insect visual system, in order to gain more understanding about how these visual strategies are implemented and why.

Insects have been seen by some scientists as inferior beings, which show nothing

more than a “thermostat-like intelligence” [31]. However, the precision with which insects perform complex navigational tasks, such as flying through the centre of a gap [13], controlling ground speed[4] relative to the apparent rate of optic flow [30], or landing[33], given their limited brain size, is intriguing and outperforms any currently built artificial robot system [34]. Moreover, the structure and position of the insect eyes do not provide the same processing capabilities as the human eyes, for example, and need different strategies to guide navigational behavior.

The compound eyes of insects consist of thousands of small optic units, called *ommatidia* [Fig. 3.4]. Each ommatidium includes a small lens that focuses incoming light onto a rhabdom, which occupies the axis of the ommatidium and is considered to be the site of photoreception [35] using six to nine photoreceptors. The visual field of the lens is usually a few degrees in width[5]. The visual axis of neighboring ommatidia are separated by few degrees, which allows the two compound eyes to provide nearly panoramic view of the environment [20].

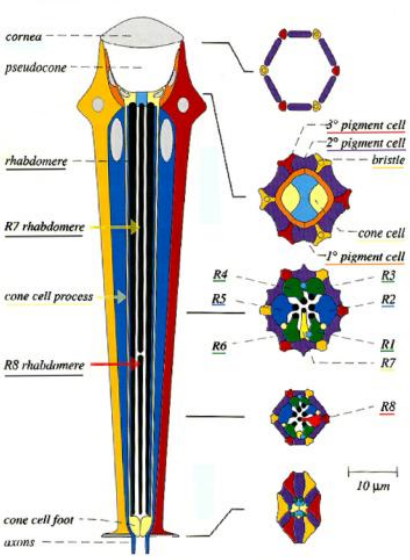


Figure 3.4: Structure of an ommatidium in the eye of the fruitfly. Each ommatidium contains 8 photoreceptor cells (named R1 to R8), and is surrounded by support cells and pigment cells, which separate different ommatidia. The portion of the photoreceptor cells which are located at the central axis of the ommatidium form the rhabdom (or in some fly species, the rhabdom is itself devided in independent components called rhabdomeres). Source: http://crfb.univ-mrs.fr/Crumbs/section/en/Fruitfly_model/107

There are clear differences between the insect and human optics. The human eye contains a single lens and a retina containing millions of photoreceptors (nearly one hundred million) [20]. The arrangement of the eyes provides limited peripheral vision, but good stereo vision, because of the strong overlap between both eyes. Most insects, however, have fixed-focus optics (immobile eyes) and cannot use stereopsis or acco-

[4]Ground speed, in this context, can be defined as the forward speed of the insect with respect to the ground [32].

[5]Some lenses have as low a resolution as about 1°, which results in visual acuity roughly one hundredth the acuity of the human eye [36].

modation to infer distances of objects in their environment [37], because their eyes are positioned much closer together and have low spatial acuity. Therefore, it seems inevitable that insects have evolved different visual strategies for achieving navigational tasks, most of which use visual cues derived from image motion. In this sense, insect vision is an active process, in which perception and action are strongly related.

When an insect moves through its environment, the perceived speeds of the images, that are induced on the two compound eyes, create an optical flow field, which contains rich visual data allowing the insect to achieve its navigational goals. Many strategies have been observed by scientists, in which insects use this optical flow information to achieve complex visuomotor tasks. These strategies include:

- optomotor response: insects use this course stabilization behavior to regain the original course of locomotion when undesired displacement has occurred [38]. For example, if an insect flying along a straight line is blown to one side by wind, the image on their eyes will move in the opposite direction, causing the flight motor system to generate a counteractive yaw torque that brings the insect back on course.

- centering response: when a honeybee flies through a hole in a window, it flies through its centre even if the hole is wide enough to allow flying in irregular trajectories [39]. A theory suggests that the bee achieves such precision by balancing the image motions induced on the two compound eyes: if the image motion induced on its left eye is greater than the image motion induced on its right eye, the bee flies closer to the right side of the hole. The same centering response has also been observed in nocturnal bees (the neotropical sweat bee, *Megalopta genalis*), which are active in dim light conditions, in which reliable visual information is hard to obtain [30].

- flight speed control: Flying insects, such as flies, use optical flow information to control their ground speed [20]. When an insect flies from an open field into a dense and cluttered environment, it decreases its ground speed in order to keep the angular velocity of the perceived image relatively constant. It has been observed that the performance of this speed control response varies with the angle at which the optic flow is measured [32]: the smaller the angle, the earlier the insect detects change in the environment ahead, but the less reliable the visual signal becomes.

- landing: when a flying insect approaches an object to land on it, it carefully monitors the object's expanding image so that its forward speed may be reduced in time [33]. The strength of the landing response has been observed to depend on the spatial frequency characteristics of the image pattern, and the speed and duration of the pattern's expansion.

- inferring distance of objects: Researchers have analysed locusts and grasshoppers regarding their ability to precisely judge the distance to an object or surface to which they want to jump. It has been observed that correct distance perception depends on depth cues obtained from motion parallax[6], which are obtained from peering prior to the jump [41]. Analysis on the peering behavior of locusts showed that, provided the locust knew how fast or far it moved its head during peering, and that it could measure the image displacement while peering, it could then compute the distance to an object of interest [42]. An experiment showed that the locusts could be fooled in misjudging distance when the object was moved horizontally and synchronously with the peering.

- chasing: vision plays a crucial role in the chasin behavior of insects: the target-pursuer combination is modeled as a feedback control mechanism, in which the pursuer attempts to keep the target in his frontal, or straight-ahead, field of view [43]. Additional mechanisms are employed to control the forward speed of the pursuer, so that he does not fall too far behind his target. In flies, the forward speed is correlated with the retinal size of the taget, which the fly tends to keep constant while in pursuit.

The precision and prowess that insects demonstrated when navigating in unconstrained environments surpasses the processing capabilities of any artificial system, and therefore some researchers believe that learning from insect visual processing strategies could prove beneficial and of use to robot control systems [34]. However, it is also pointed out that observing insect navigation performance explains what could be attempted in robot control scenarios, but not how it could be implemented.

The next section will review some of the prior research work, which is related to the concepts and objectives of the project. Such work includes research experiments with neuromoprhic sensors in machine vision problems, including robot control in real en-

[6]The motion parallax is a visual cue for perceiving image depth and can be defined as the optical change of the observer's visual field resulting from the change of his own position, or, in other words, the "apparent motion" of stationary objects which arises during locomotion [40].

vironments, biologically-inspired image processing algorithms and robot navigation strategies based on biological observations.

Chapter 4

Prior work

The field of neuromorphic visual sensors still remains largely unexplored [2], mainly due to unfamiliarity with asynchronous logic, on which such devices are based, and poor uniformity of pixel response characteristics. However, some research has been undertaken on implementing bioinspired visual systems using silicon retina chips and applying those systems to robot control problems in real environment.

In one paper, a reconfigurable hardware implementation of optical flow model was developed, which approximates neuromorphic systems by means of high complexity and robust structure [44]. The model deals efficiently with variable illumination levels, different types of noise and contrast invariance, while minimising the use of algorithms that are not biologically justified, such as matrix inversions and iterative methods. However, a disadvantage of this implementation was identified in its huge computational complexity, due to the larger number of signal processing procedures compared to similar gradient-based models.

Barranco's group [7] have designed a motion detection system using a standard frame-based camera, which adopts the event-driven approach of neuromorphic sensors. Four different neuron responses were considered and used: sustained neurons (*ON-OFF* and *OFF-ON*) and transient neurons (*INCREASING* and *DECREASING*)[1]. This implementation does not use a neuromorphic sensor, but instead tries to model the behaviour of

[1]Sustained neurons model the spatial center-surround response, where ON-OFF neurons tune local regions with more intensity than surrounding regions, and OFF-ON neurons respond when a pixel registers higher light intensity than its neighbours. These neurons are designed using difference of Gaussians with different inner/outer ratios. Transient neurons model temporal responses and are calculated by using two consecutive image frames. INCREASING neurons tune local regions where light intensity increases, and DECREASING neurons tune local regions where intensity decreases.

one. Therefore, image processing is done using conventional methods, such as frame-by-frame analysis amd continuous sampling of all image pixels at a predetermined frame rate.

Another paper describes an optical flow model using a silicon retina sensor, which is based on two physiological observations [3]. Firstly, it has been reported that two separate processing mechanisms encode spatial and temporal changes of the incident light, and, secondly, an approximation of Laplacian-Gauss spatial filter is considered as a common preprocessing technique in biological sensory systems. In the proposed model a Laplacian-Gauss filter is applied to the input image of the silicon retina, using an analog resistor circuit, and the output is fed into an FPGA (*Field-Programmable Gate Array*) circuit, which is able to compute the velocity of a moving object. The resulting system is able to estimate local optical flow in a robust and accurate manner even for scenes in a real environment and has the advantage of low computational complexity due to its use of parallel analog processing in the silicon retina.

A group led by Tetsuya Yagi [45] designed a similar mixed analogue-digital neuromorphic visual system, in which a silicon retina's output is fed into a set of FPGA circuits for extracting image cues, such as direction and velocity of motions. The retina applies two types of image preprocessing: Laplacian-Gauss-like spatial filtering and a subtraction of two consecutive image frames. The authors observed that in spite of being computationally complex and expensive, conventional machine vision systems are quite versatile for their programmable architecture. On the other hand, analog neuromorphic sensors are very efficient in terms of computational complexity, compactness and robustness, but their versatility is limited due to their approach to image computation. Therefore, an approach that combines both analog (in terms of a neuromorphic visual sensor) and digital (FPGA circuitry) models was considered as highly useful for autonomous robotics in real environments.

Several papers describe a robot control system using a neuromorphic sensor for processing visual information, such as [10] and [46]. In the first case, a robot goalie was designed and controlled by a software system, which applied object tracking algorithms on top of an already preprocessed image from the camera. The experiment consisted in a robot goalie blocking balls which were shot at the robot at > 150 ms time to impact. The success rate of the goalie was reported to be between 80% and 90%. Response latency was reported to be about 33 ms for 30Hz frame rate when the image was processed at screen rendering rate, and much shorter when the image was

processed at event packet level.

The second work [46] describes a conducted experiment, in which a toy robot was controlled to mimic perceived object movement by using a silicon retina sensor and two FPGA circuits, *Spartan II* and *Spartan III*, which extract image cues such as motion direction and control the robot's servo motors, respectively.

Considerable research has been conducted on designing visually-guided navigation systems, inspired by the visual processing strategies that some insects, such as the housefly and the honeybee, use in order to achieve navigational goals [47], [31]. In the first paper, a robot control system was presented in which a mobile robot navigated using information from two laterally positioned cameras, so that each camera would point to the opposite direction and the two visual fields would not overlap[2]. The placement of the cameras was based on two observations: firstly, the two eyes of the honeybee are pointing laterally at about 180 degrees and secondly, the frontal position of the eyes, which have a very large binocular field, is mainly motivated by manipulation requirements: for the purpose of navigation control a potentially more effective positioning has been identified in the one adopted by several flying insects like bees and flies, with their eyes pointing laterally [48]. The tasks that the robot performed involved following a textured wall at a certain distance, navigating through a straight corridor, obstacle avoidance, and a door's negotiation. Image processing was done using an optical flow algorithm, which estimated the average optical flow on each side of the robot, and which was inspired by the observation that, when flying through a narrow corridor, honeybees use a similar approach to measure and minimize the difference in image velocities on both eyes, thus flying through the middle of the corridor. Throughout the experiment conventional frame-based cameras were used and the optical flow algorithm was developed using principles defined in [18].

The second paper described a robot control navigation system which uses an artificial compound eye based on the fly's compound eye [31]. The developed visual system was comprised of 100 Elementary Motion Detectors, or EMDs (an example EMD is shown in Figure 4.1), which resemble their biological counterparts found in the fly, and responded only to movement in the scene. Thus the mobile robot, on which the camera was mounted, would become "blind" if no relative motions are present in the scene. The task of the mobile robot was to navigate around a real environment and avoid any obstacles, and it was reported that the robot managed to operate swiftly and

[2]Throughout the paper, this arrangement was referred to as *divergent stereo*.

autonomously in a real world.

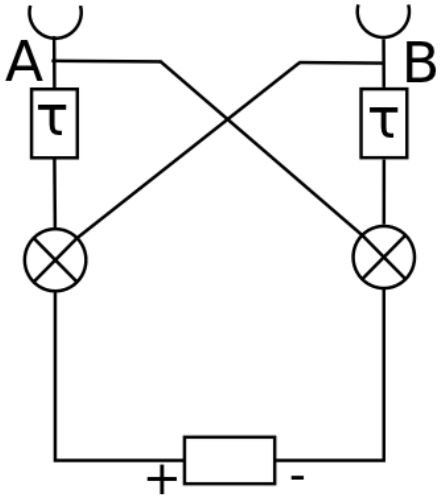


Figure 4.1: An example of an Elementary Motion Detector which has a direction of preference left-to-right. The correlational EMD is one of the most influential models for visual motion detection, which was created by Hassenstein and Rechardt [49]. Its core operation is the correlation of a signal associated with one visual input unit (A) with a delayed signal from a neighboring unit (B). Depending on the speed and direction preference of the EMD, the response to a moving object may be strong - if it moves in the preferred direction, and the travel time to the neighboring photoreceptor is close to the time delay- or weak- if the object does not move in the preferred direction or there is a considerable disparity between the time delay and the object's travel time. Adapted from [49].

Chapter 5

Project Setup

The proposed robot navigation system is tested using an experiment based on the one conducted by Srinivasan [13], in which honeybees were trained to fly through a narrow corridor in order to reach a food source [Fig.5.1]. What the researchers discovered was that, even though the corridor was wide enough to allow the bees to fly in irregular trajectories, they tended to fly around the midline of the corridor. The explanation presented in the paper was that bees use visual cues from the optical flow fields computed from the left and right eyes (which point laterally at nearly 180 degrees) to adjust their position relative to the two walls of the corridor. When the bee is in the middle of the corridor, the computed velocity from the left and right eye is nearly the same, while when the bee is closer to one wall, the velocity from the eye pointing at that wall is larger.

The experiment presented in [13] demonstrated how bees are capable of using optical flow information to achieve complex navigation tasks (flying through the middle of a narrow corridor) with remarkable precision and in real time. Such results have led some researchers to believe that applying similar algorithms in machine vision problems may boost the overall performance, reliability and efficiency of an artificial navigation system.

Although the implemented navigation system is similar to the one presented in [47] and is closely related to the experiment by Srinivasan, there are two major differences which need to be addressed. Firstly, in contrast to a honeybee and the mobile robot in [47] (*Robee*), the robot in this project has a single mounted camera, pointing towards the direction of movement. Because of the camera's narrow field of view, a limited

portion of both walls can be seen at all times, with a wall completely disappearing from sight when the robot is turning to correct its position. Secondly, due to the limited robot speed (around 170 mm/s when servo motors are running at full power) the rate of light intensity change at each pixel is considerably lower then if the sensor was positioned laterally, which produces a smaller amount of address-events, making the walls less detailed. The light conditions also proved to be crucial for the performance of the algorithm, since it affected the wall contrast and how well the wall textures were seen by the sensor. The conducted tests and experiments showed, however, that even with these drawbacks present, the navigation system performed well and the robot managed to correct its position relative to the midline of the corridor fairly accurately.

The second major difference compared to [47] is the use of a neuromorphic sensor for image processing, rather than a frame-based imager. Therefore, the optical flow algorithm, although similar in concept (computing velocity difference from the left and right sides), differs in its implementation and design in order to accommodate the characteristics of a Dynamic Vision Sensor.

The laboratory setup consists of a straight corridor made of gray rubber wall sections, which can be joined together to form a wall. Because of the narrow field of view of the camera, the width of the corridor war restricted to about 60 cm. To produce high contrast edges, the wall sections were covered with white stripes without following any symmetrical patterns (following the observation that the bee's navigation through the corridor was not affected by changing the grating patterns on the corridor walls [13]). As mentioned earlier, the light condition was essential to conducting the experiment, therefore different combination of light sources and positions were tested in order to determine an optimal configuration.

The project uses the DVS128 Dynamic Visual Sensor along with a Java-based software application [50]. The software is developed and maintained by researchers at the Institute of Neuroinformatics at Zurich and provides a framework for extracting and processing visual information from this and other neuromorphic sensors. The software's approach towards extracting the data is somewhat similar to the frame-based concept, as it routinely checks for visual output from the camera, which is merged into an *event packet*. An event packet is a collection of address-events (described in Section 2.4) with an average duration of 16 milliseconds. In other words, every event packet collects address-events that have been created in a 16 ms time window, and is then forwarded for further processing by different image filters (discussed below). A

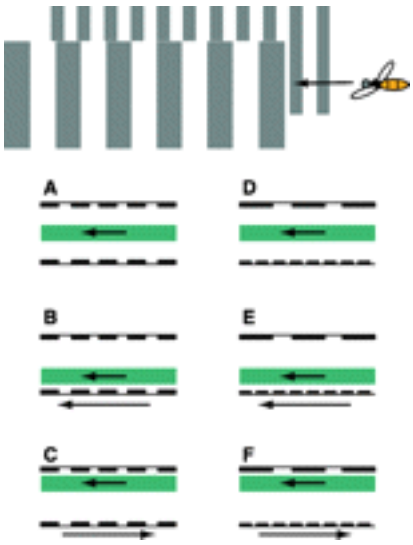


Figure 5.1: The experiment conducted on honeybees. The short arrows depict the direction of flight and the long arrows the direction of grating motion. The shaded areas represent the means and standard deviations of the positions of the flight trajectories, analysed from video recordings of several hundred flights.

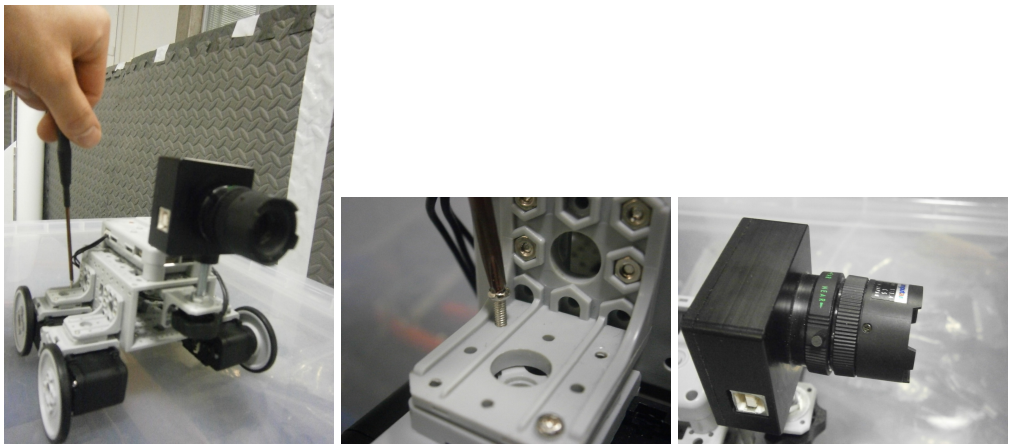
packet can thus be thought of as an image frame: it is a representation of the scene as seen by the camera for a given period of time (16 ms). However, the temporal characteristics of each event are kept within the packet, as each pixel independently responds to luminance change, and therefore each event precisely encodes its time of creation (or *timestamp*).

To perform optical flow computation, a custom "filter"[1] is developed that extracts velocity information from the left and right sides of the image, and passess this information to a controller module, which produces a corresponding response for the robot vehicle to execute. The controller module is running on a separate Java thread and computes the disparity between the image velocities from the left and right image sides in order to estimate the position error of the robot. When the error is computed, a command is transmitted to the robot vehicle for execution: the command contains information about direction of movement, speed of movement and time for executing the command (the time can also be left undefined, thus the robot will run the instruction indefinitely or until told to stop).

A *Bioloid* robot vehicle [Fig. 5.2] is designed to carry the DVS sensor attached to its body, and was controlled by a custom-made firmware. The robot includes a chassis holding the camera and the controller, and four AX12+ servo motors as wheels. The robot is not meant to resemble or model a living being, so simplicity and compactness are the core goals of the design (although one can argue that a better one involving fewer servo motors may be optimal for the setting, since the accumulative movement

[1]The term "filter" within this context refers to a routine which processes the visual data stored in the event packets.

error would be reduced).



(a) (b) (c)

Figure 5.2: The Bioloid robot vehicle. (a) The upper body contains the cm510 controller and the camera, the four AX12+ servo motors are attached to both sides of the chassis. (b) Construction parts of the robot. (c) The Dynamic Vision Sensor, attached to the robot's body.

(a) (b)

Figure 5.3: The Zigbee wireless devices. (a) The ZIG100 and ZIG110 devices. (b) The Zig2Serial adapter used to connect the ZIG100 to the workstation. Source of images: http://support.robotis.com/en/

Communication between the robot and the workstation is achieved using a pair of Zigbee Wireless devices, Zig100 and Zig110 [Fig. 5.3], which use the 2.4GHz band (IEEE 802.15.4). After the image processing application has produced a command for the robot to execute, the command is sent to a program running on the workstation using a TCP-based server-client wireless protocol. The program receiving the command then relays it to the controller running on the Bioloid robot vehicle using an API

developed by the Bioloid designers, which provides greater flexibility in directly controlling the robot components (the servo motors, the wireless device, and the controller in general). The API is slightly modified as some problems were identified during the system development.

Chapter 6

System Implementation

6.1 Optical Flow Using Event Matching

This section will describe the first optical flow algorithm that was developed: an event-matching algorithm in which address-events taken from consecutively acquired event packets were paired together in order to compute each event's displacement. The algorithm was designed with the purpose of better understanding the workings of the camera, but failed to take into consideration the fundamental and distinguishing aspects of an event-driven sensor, and was therefore discarded.

The event matching algorithm was designed to compute the average displacement vector for each address-event, based on probable displacements observed consecutive packets taken at a discrete point in time. Because the pixels do not carry absolute intensity information [1], the nature of luminance change, or polarity[1], was used to determine whether two events can be matched. Two neighboring events, taken from consecutive packets and having the same polarity, are usually part of the same moving edge, except in cases in which the moving edge changes direction, and this change is captured in the second packet.

A more detailed description of the algorithm is presented in Figure 6.1. Events from the first reference packet are stored and are later checked against events from the second reference packet. If the euclidean distance between two events is small (the search window around an event is defined by a custom-set global threshold), and their polarity

[1]The polarity can be either *ON* or *OFF* meaning that light intensity has increased or decreased, respectively.

value is the same, a movement direction vector is computed. After all potential matches for an event have been found, an average direction vector is calculated and used to describe the temporal displacement of that event (or the edge the event belongs to). The choice of event packets should aim to minimize the matching error: with the increase of the time window between two reference packets, the mismatch error increases, as the search window around an event must also increase to allow for fast moving edges to be properly matched.

```
if iteration<count do iteration:=iteration+1;
else do
 iteration=0;
 if phase==1 do
  for each event e in packet do
   if e->timestamp < time_threshold do
   table->store(e);
  phase:=2;

 else do
  for each entry e in table do
   totalx := 0;
   totaly := 0;
   count := 0;
   for each event ev in packet do
        point source := e->location;
        point destination := ev->location;
        if euclid(source, destination)<=threshold do
         point p := destination-source;
         totalx := totalx + p->x;
         totaly := totaly + p->y;
         count := count + 1;
    point mean := (totalx/count,totaly/count);
```

Figure 6.1: Calculating the mean direction vector for each event. The iteration variable defines how many packets to be skipped between storing the events and calculating their direction vectors.

The design of this algorithm contains one crucial flaw. It is based on the frame principle: an event packet is considered as a replacement to an image frame, and it is assumed that consecutive packets are not related to each other, except for their ordering in time, and therefore can be processed independently. The algorithm also assumes that an event packet provides a complete representation of the perceived scene and motion flow between two packets is essentially constant. These assumptions are false and do not relate to the event-driven nature of the DVS imager. An event packet contains only the data from pixels that have responded to a change in light intensity at that particular point in time (in a 16 millisecond time window), thus it is a partial representation of the changes occurring in the scene, in other words it shows only the dynamics in a scene in a time period of 16 milliseconds, and does not give the whole picture. For example, the nearest equivalent to an image frame taken at a rate of 1fps would be the data collected from 62 event packets[2], and this data would exclude the static parts of the scene. Relating event packets to image frames and basing the main assumptions on the use of the latter completely loses the point of using a neuromorphic vision sensor in the first place.

6.2 Optical Flow Using a Radial Neuron Network

This second describes the second algorithmm, which was used to implement the robot navigation system. The algorithm is based on a method for calculating depth information from consecutive image frames using a radial neuron network described in [14]. Given the different nature of the camera used in the project, and the different goals of the project (robot navigation without the use of depth information), some key changes were introduced. However, the overall principles and motivations are the same.

6.2.1 Overview of the Algorithm

Given the asynchronous and bioinspired nature of the visual sensor used, it makes sense to implement a bioinspired approach to computing optical flow, which adapts to and exploits the operational characteristics of the camera. The developed algorithm is based on a proposed parallel and robust algorithm, designed to compute depth information in a scene and taking its motivation from observed animal behavior [14]. Some

[2] 16 milliseconds $\times$ 62 is roughly a second, or 1000 milliseconds.

animal species, such as certain bees, flies ([15]), or pigeons ([16]), demonstrate different strategies reducing the optical flow to a few, or in some cases, a single component while moving. For example, the interesting "head bobbing" motion observed in walking pigeons is thought to serve the purpose of eliminating all optical flow while the bird's is moving under its motionless head, while houseflies tend to fly along straight lines making rather sharp turns with the aim of reducing the flow field to its translational components [15]. These examples show that applying a similar reduction strategy in machine vision problems is a biologically justified approach, and may prove beneficial and useful in robot navigation problems.

The implemented algorithm for this project is loosely based on the algorithm described in [14], with some noticeable differences. Firstly, the type of camera used in this project is entirely different in design considerations: the paper uses a conventional frame-based imager, while the project is concerned with an asynchronous event-based neuromorphic camera. Secondly, the ultimate goal is different: this project is not interested in obtaining depth information from a scene. Nevertheless, the underlying concept is the same: the optical flow field that is induced on the camera from local and global motion is reduced to its radial components using a neural network. During development of the robot navigation system, the following two assumptions were made:

- The robot is assumed to move only along the optical axis of the camera
- The scene, or environment, is considered to be stationary

The first assumption may appear to limit the algorithm to a special case in which the robot is only moving forward. During curves, such as when the robot is turning, the focus of expansion is no longer aligned with the motion trajectory, thus breaking this restriction. However, this is not to say that the robot must refrain at any cost from turning or rotating (it has to be able to correct its heading trajectory), but that such motions may simply be discarded from processing. The period of "visual blindness" experienced by the vehicle during rotation could be minimized by insuring that such motion is completed quickly.

The second assumption states that perceived motion in the environment will be caused only by the robot's own motion. This restriction fits well with the goal of the experiment.

6.2.2 Retina

The core part of the algorithm is comprised of a network of radially arranged neurons, or the retina [Fig. 6.2]. This neural network is composed of a set of radii which emanate from the image center, and the angle between two radii determines the total number of radii and, consequently, the distribution of neurons within the network.

Each radius in the network contains roughly the same number of neurons. The distribution of neurons on a single radius is given by the following equation:

$$D(r_n) = \frac{1}{r_n - r_{n-1}}, n \in [1, ...N] \tag{6.1}$$

where r_n is the location of neuron n on a radius, and $N+1$ is the total number of neurons on every radius. This equation considers the increase of a moving object's virtual speed with its distance from the nodal point in a radial flow. Therefore, the distance between two neighboring neurons on a radius should increase with the distance from the FoE.

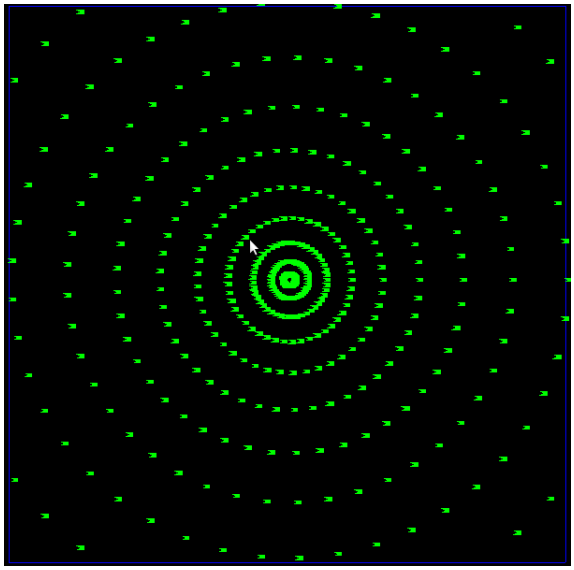


Figure 6.2: The radial neuron network of the algorithm. The blue dots show the location of each neuron. The radial lines can clearly be seen. Notice that some radii have fewer neurons than the rest: this is because any neuron with cartesian (x,y)-coordinates outside the image is not created.

6.2.3 Leaky Integrate-and-Fire (LIF) Neurons

The core unit of the neural network, or retina, is the Leaky Integrate-and-Fire (LIF) neuron. The LIF neuron is a mathematical description of the properties of nerve cells,

that is designed to accurately describe and predict biological processes. The model was developed by Louis Lapicque [51] and is widely used as a modelling tool for processes and responses in the biological neuron.

Using the LIF neuron model in the project, it was possible to achieve visual processing on a different conceptual level. Instead on focusing on individual events and processing them independently, the algorithm now has the means to correlate events based on their spatial position: it can be assumed that address-events that are within the receptive field of the same neuron are strongly correlated.

The neuron model used in the algorithm is an adaptation of an implementation from [50]. While some functionalities were discarded as irrelevant to the navigation system's design[3], the core was kept and reused and some major additions were introduced.

Major characteristics of the LIF neuron implementation include:

- A neuron has a membrane potential, u, which can be defined as the voltage difference between the interior and the exterior of the neuron
- A receptive field, R, which defines the spatial region in the image in which the presence of a stimulus (in this case address-events) affects the firing of the neuron. By default, the receptive field size of a neuron is set to half the distance to its immediate neighbor [Fig. 6.3(b)]
- Each LIF neuron knows its location in respect to both the neural network (in Cartesian (x, y) coordinates) and the radius it belongs to
- Each neuron has two memory units[4] which store propagated signals (see Section 6.2.5), and are explained in Section 6.2.4
- A neuron can communicate within the neural network only with its direct neighbors, as shown in Figure 6.3(a)

The membrane potential of each LIF neuron in the network is integrated over time and increased with every address-event that falls within its receptive field. A "leak" component is added to gradually decrease the membrane potential. This leak is necessary, since a voltage level below the membrane potential threshold would be retained by the neuron indefinitely (until a stimulus boosts the voltage enough for an action potential

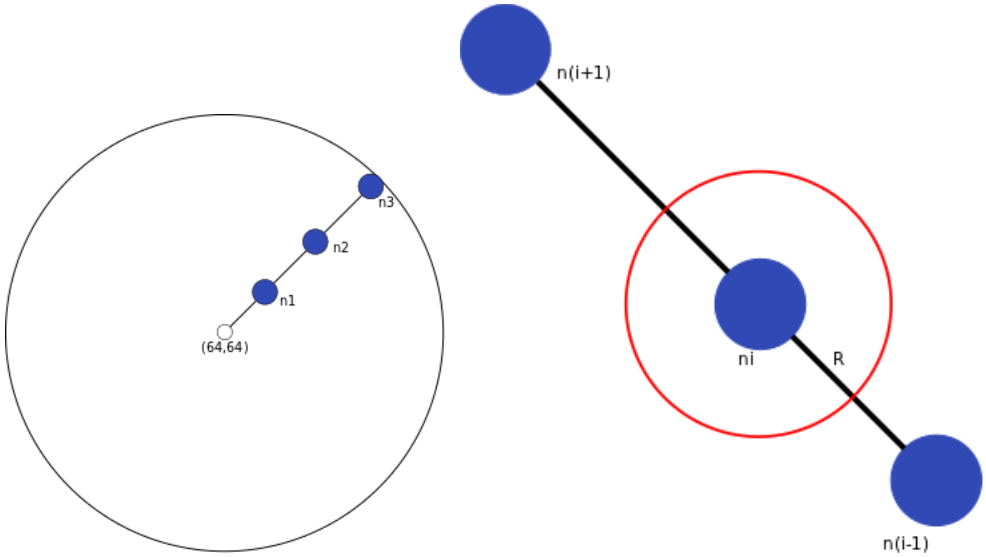
[3] Specifically, the optical flow algorithm uses a different approach than what the original model was created for.

[4] A memory unit simply refers to a data structure for storing information.

to be generated). The membrane potential, or *memPot*, is incremented at time t using the following equation:

$$u(t) = I_0 + u(t_1)e^{-\frac{t-t_1}{\tau_m}} \quad t_1 \leq t \tag{6.2}$$

where I_0 is the input current, $u(t_1)$ is the membrane potential at time t_1 (which is the potential before the current update) and $e^{-\frac{t-t_1}{\tau_m}}$ is the exponential decay (τ is a time constant [52]).



(a) (b)

Figure 6.3: The neuron distribution on a single radius is shown in 6.3(a). In this example, n_1 can send signals only to n_2, while n_2 can communicate with both n_1 and n_3. 6.3(b): The receptive field of a neuron (shown in red) is a circle with radius $\frac{1}{2}\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$.

If the membrane potential of a neuron at time t exceeds the potential threshold, the neuron fires and the membran potential is immediately decreased below the threshold in order to reflect the repolarization phase during neuron spiking. Firing neurons within the neural network are processed by the optical flow algorithm in order to create and propagate signals, as described below (6.2.5).

6.2.4 Neuron Memory

As mentioned earlier, several major additions were introduced to the LIF neuron model, namely the memory units. Each LIF neuron has two memory data structures, the

memory banks, which are used to store information about signals that are propagated throughout the network. Each memory bank stores the signals of stimuli that are expected to excite the host neuron, and are responsible for updating the information on these signals and locate the most feasible match when the neuron fires. Two separate banks are used to disambiguate between the signals, which are propagated "downwards" (towards the nodal point of the retina) and those that are propagated "upwards" (away from the retinal nodal point). Additionally, each memory bank contains two memory units: a temporary unit, used to store newly created or received signals until memory update (see Section 6.2.5), and a "real" unit which stores all signals after the memory updates are complete.

The propagation of signals across memory banks is shown in Figure 6.4. When a signal is propagated away from the nodal point, it is copied from the higher real memory of a neuron to the lower temporary memory of its higher neighbor. During the memory update phase signals from the temporary memory units are transferred to the real memory units in the opposite memory bank. The first and last LIF neurons on a radius lack a pair of memory banks, because they have only one communication channel[5].

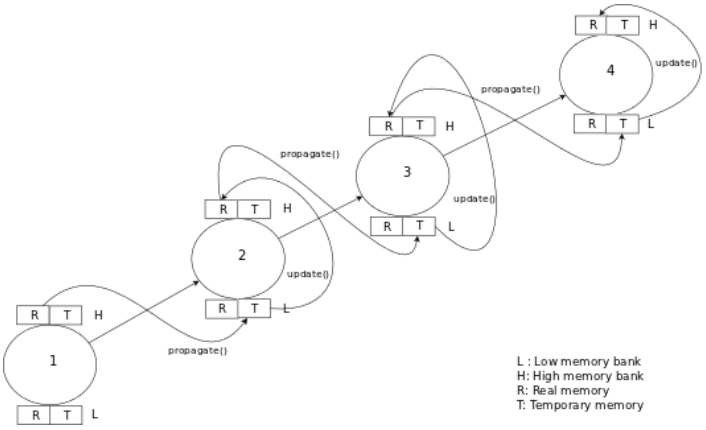


Figure 6.4: Propagation of signals away from the nodal point.

6.2.5 Description of the Algorithm

When an address-event packet is received from the camera, each address-event is added to the neurons belonging to the nearest radius. To find the nearest radius, the angle which the event forms with the X axis (where the origin of the Cartesian coordinate

[5]Note that pair is not symmetrical: the first neuron, for example, has a high real memory and a low temporary memory, while the last neuron has a high temporary and low real memories.

system is $(64,64)$) is computed using equation

$$\alpha = \begin{cases} 360 - \arccos \frac{\vec{d}\cdot\vec{e}}{\|\vec{d}\|\|\vec{e}\|} & \text{if } d_y < 0 \\ \arccos \frac{\vec{d}\cdot\vec{e}}{\|\vec{d}\|\|\vec{e}\|} & \text{if } d_y \geq 0 \end{cases} \tag{6.3}$$

where d is the distance between the event and the origin and e is the elementary vector $(1,0)$. Since the number of radii is known, as well as the angle between two radii, (6.3) is used to find the nearest radius using

$$rad = \frac{\alpha}{\gamma} \mod R \tag{6.4}$$

where γ is the angle between two radii and R is the number of radii.

After finding the nearest radius, the event is added[6] to the closest neurons on that radius. The look-up is done by iterating over all neurons on the radius until the closest ones have been found, starting from the nodal point. In the worst case, in which the event is located near the image boundaries, this look-up would iterate over all neurons until finding the closest neurons.

After processing the event packet the algorithm updates the neural network by checking each neuron for changes in its state. If a neuron has received a sufficient boost to its membrane potential and has exceeded the firing threshold, its state is changed to *fired* (and its membrane potential is repolarized). Every time the membrane potential of a neuron is checked, it is decreased using the exponential decay component $e^{-\frac{t-t_1}{\tau_m}}$.

The algorithm then checks the memory banks of the neuron for any matching signals (if the neuron has fired, otherwise stored signals are updated) that are being stored. A signal is a digital representation of a moving stimulus[7]: it stores information regarding the present location of the stimulus, its direction (in terms of whether the stimulus is moving towards the nodal point or away from it), its estimated velocity and its arrival time at the neighboring neuron. Each signal has a priority and a counter. The priority represents the importance of the signal: once a signal is viewed as late (in terms of the expected arrival time), its priority is gradually decreased until the signal is received, or until its priority reaches a bottom threshold: in this case the signal is removed. The more neurons receive the same signal, the higher the signal's priority becomes.

A signal's counter is used to determine when the signal is reliable enough to be added to the optical flow calculations. If a neuron is excited by noise and fires, it will propa-

[6] The input current for each address-event added to the neuron's membrane potential is 1, irregardless of polarity.

[7] A stimulus, or edge, may be encoded by multiple signals at any time t.

gate a false signal which is not related to an existing stimulus. The probability of the neighboring neuron to receive the same signal is extremely low, therefore the counter value can serve as a mechanism to prevent such misfires from introducing errors in the optical flow calculations by specifying how many neurons are required to receive the same signal before it is considered reliable.

If a LIF neuron has fired, it searches its real memory for any matching signals: those that are expected to arrive within a small time window around the current time, and which have the same polarity as the present stimulus[8]. If a match is found, its counter and priority values are increased, and its expected velocity and expected arrival times are computed. If the signal's counter is large enough (i.e. sufficient number of neurons have received the signal), it is "confirmed": its velocity is added to the corresponding velocity measurement[9].

If no matching signal has been found (no signals are expected to arrive soon, or they have opposite polarities), a new one is created and propagated. When creating a new signal, its velocity cannot be determined as there is no available prior information, and therefore a constant velocity value is set instead, measuring how fast the stimulus is expected to be moving (but not how fast it is actually moving). Tuning the initial velocity value proved to be a crucial factor affecting the overall performance of the algorithm, as discussed in Section 8.6.3.

Since a LIF neuron may be excited from an incoming edge that has not yet reached its location on the image, propagation of the resulting signal is delayed until the edge comes in close proximity to the neuron. A LIF neuron tracks approaching edge in time (or that portion of the edge that is within the neuron's receptive field) and when the edge is sufficiently close to the neuron, the signal is transmitted to the neighboring neuron.

If the LIF neuron has not fired, its memory contents are "silently" updated: the priority of late signals is reduced and signals with priority values below a threshold are removed from the memory.

When each neuron's memory banks are updated, the algorithm performs a second up-

[8]The vertical stripes on the wall are usually seen as having a frontal and a rear edge, each having different polarities. Therefore, to accurately track the motion of an edge, the polarity of stored signals and firing neurons should be monitored.

[9]For the project's goals only approaching motion was considered, as the robot was not expected to move backwards. However, adding retracting motion to the processing should be straightforward.

date: any newly created or propagated signals are copied from the temporary memory of the corresponding memory bank to the real memory of the opposite memory bank. The separation between temporary and real memory banks is necessary in order to prevent a neuron from incorrectly modifying the memory of its neighbor, which has not been processed yet.

Consider the following example. A radius contains 4 neurons is created with neuron n_0 being the closest to the nodal point of the neuron network. At time t_0, neuron n_0 is excited by a strong vertical ON edge and responds by firing a spike. It creates a new signal, which has the following parameters: it has a label α (serves as an identifier), an initial velocity $v_\alpha = v_0$, a priority p, a counter $c = 1$, a timestamp $t_\alpha = t_0$, and carries the polarity of the edge, 1 for ON. Using the initial velocity, the signal's arrival time at n_1 is

$$\theta_\alpha = t_0 + \frac{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}}{v_0} \tag{6.5}$$

After creating the signal, n_0 propagates it to n_1 and switches its state to OFF, meaning that the next signal that will be created will correspond to an incoming edge with and OFF polarity. During the next 16 to 32 milliseconds, n_0 still responds to the same edge, because the edge is within its receptive field, but does not propagate any more signals. At a later point in time, t_i, the same ON edge appears within the receptive field of neuron n_1 and causes n_1 to fire, but because the edge is far from n_1's position in the image, n_1 delays its response.

At time t_j ($j >= i$), the edge has moved closer to n_1 and the neuron checks its memory for any stored signals. If α is expected to arrive soon, in other words $\|\theta_\alpha - t_j\| < \varepsilon$, then signal α is matched to the current edge: its counter is increased to 2, its velocity is updated using the equation

$$v_\alpha = \frac{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}}{t_j - t_\alpha} \tag{6.6}$$

and the arrival time at n_3 is estimated by computing the distance from n_2 to n_3, $d_{2,3}$, and substituting $d_{2,3}$, t_j and (6.6) in (6.5) to obtain

$$\theta_\alpha = t_j + \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{v_\alpha} \tag{6.7}$$

When the ON edge reaches n_3 and is sufficiently close, if the arrival time of the stored signal is close to the current time, the signal is matched yet again. If, say, the threshold

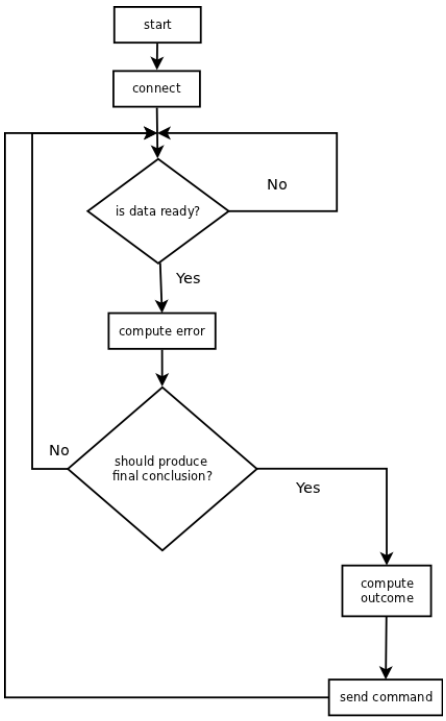


Figure 6.5: A simplified flowchart of the WSC control loop.

for confirming a signals is 3 (meaning that at least 3 neurons have to have received the signal before it is confirmed as reliable), signal α is confirmed: the signal's estimated velocity (after being updated with the newest distance and time) is normalized and added to the corresponding velocity measurement (for the left or right side of the image). The signal's counter is increased by 1, and is propagated onwards, to neuron n_4. Any subsequent match of the signal with an edge results in additional velocities being added to the total estimations.

6.3 Robot Control Implementation

This section will describe the two controllers that were developed for the robot navigation system. The role of the workstation-side controller (WSC), is twofold: firstly, it computes the velocity error from both sides of the image, derived from the processed visual input from the optical flow algorithm. Secondly, based on the velocity error, it creates a control strategy, sends it to the robot for execution, and oversees any changes, or adjustments, to the strategy. The WSC control loop is shown in Figure 6.5.

The robot-side controller is responsible for manipulating the AX12+ servo motors of

the vehicle, based on the received commands. It runs on the Bioloid cm-510 controller [Fig. 6.7] and communicates with the WSC using a pair of Zigbee wireless devices.

6.3.1 Workstation Controller

The WSC runs as a separate thread and has access to all data produced from the optical flow algorithm. When it starts up, the WSC initializes the server side of the communication channel between the workstation and the robot and awaits connection from the relay module (described in Section 6.4.2).

Once connected, the WSC waits for data from the optical flow algorithm to be ready: the data is in the form of the total number of velocity vectors from the left and right image sides. The optical flow algorithm also provides a mechanism to check whether the robot is idle: if a considerably small amount of events are being registered, it is probably because the robot is not moving. If the robot is considered idle for a discrete period of time (i.e. 3 seconds), the WSC instructs it to move forward.

Using the data from the optical flow algorithm, the WSC calculates the velocity error using the difference between the left and right average velocities. The average velocities are given by

$$avg_s = \frac{1}{n}\sum_{i=0}^{n}\frac{v_i}{\|v_i\|} \tag{6.8}$$

where v_i a velocity vector computed at side s (either left or right), and $\|v_i\|$ is the norm of the vector $(x_i - 64, y_i - 64)$ where (x_i, y_i) is the location of the neuron that has confirmed the signal and $(64, 64)$ is the image center. Each velocity vector is normalized using its distance to the nodal point in order to reduce the effect of acceleration of an approaching signal, as described in Section 3.1. The resulting velocity estimate for each side is then normalized by the number of signals to prevent the error estimate from being biased towards the image side containing more confirmed signals.

How would the WSC know when to process the data that is received from the optical flow algorithm? Since the final velocity measurements are updated at the same time as signals are being confirmed, how would the WSC determine if all neurons have been checked and the data is complete? A synchronization mechanism is necessary to force the controller to wait until the optical flow finishes gathering data before proceeding to compute the velocity errors. Such a mechanism, like a binary semaphore, was implemented, which allows the algorithm to signal the controller when the data is ready to

be analysed. The construct also allows the algorithm to gather signal data from multiple event packets consecutively before informing the controller: if more information is gathered, compuations would become more reliable and accurate.

The immediate velocity error at time t is given by

$$\lambda = avg_0(t) - avg_1(t), \quad 0 - left, 1 - right \tag{6.9}$$

If either $avg_0(t)$ or $avg_1(t)$ is 0 (no signals have been confirmed from the left or right sides), then one of the walls cannot be clearly seen: in such case, it is probable that the robot is not close to the midline, but rather closer to the wall that has produced any signals. Therefore, the λ is deliberately biased towards the wall with no signal data.

The WSC can add together several immediate errors λ before deriving a final outcome E. This summation is used to reduce the possibility of producing an incorrect response and its affect on the system performance is discussed in Section 7.1.3. Given a globally defined error threshold ε, the WSC produces a response η using the following rules:

$$\eta = \begin{cases} 0 & \text{if } E \geq \varepsilon \\ 1 & \text{if } E \leq -\varepsilon \\ 2 & \text{if } -\varepsilon \leq E \leq \varepsilon \end{cases} \tag{6.10}$$

where 0 will instruct the robot to turn right, 1 will instruct the robot to turn left, and 2 will instruct the robot to continue forward.

If the robot needs to re-adjust its position, its turning speed is weighted based on the difference between the E and ε:

$$w = \frac{\|E - \varepsilon\|}{\sqrt{E^2 + \varepsilon^2}} \tag{6.11}$$

This weight reflects the disparity between the robot's goal trajectory along the midline, and its actual position: the closer the robot is to the left wall, for example, the higher the weight value, which forces the robot to turn more.

The re-adjustment is done in a series of consecutive instructions, during which the robot is virtually "blind" to its environment: the optical flow algorithm continues to process the visual input, but the WSC ignores the data[10]. The robot makes a forward turn towards the middle of the corridor, moves forward for a short period of time, and makes another forward turn in the opposite direction in order to reposition itself.

[10]This assumption was discussed in Section 6.2.

A simple feedback control mechanism was implemented in order to reduce the accumulative turning error during repositioning. This error is a result of the two rotational motions of the robot, one in each direction, which are subject to a variable input current passed to the four servo motors. Additionally, an underestimated transmission timeout value (discussed in Section 6.4.1) may prevent the robot from completing a turn move, in which case the robot ends up facing one of the corridor walls. Given the narrow width of the corridor in the experiment, a fast and efficient mechanism was required to quickly correct the robot's position.

Once the robot has finished repositioning, it enters a "correction phase", during which the WSC uses the processed data from the optical flow algorithm to reduce the turning error. To achieve this task, a second set of LIF neurons[11] were placed in two vertical lines along the far edges of the image [Fig. 6.6]. Since the retina sensor has a limited field of view, it is assumed that if the robot has not positioned itself correctly, once it continues moving forward, one of the walls will start expanding into the opposite side of the image, while the other wall will gradually disappear from view. The feedback control mechanism utilizes this assumption to detect if the robot is positioned incorrectly: if the robot is approaching the left wall of the corridor, response from the neural line at the right end of the image will gradually decrease until it reaches a minimum. After a short period of time, it will start to increase again with the approaching of edges from the left wall.

When the robot enters the "correction phase", the WSC sets a global timer, which is used to control how long it should remain in this phase. It will then issues a forward command to gather visual information, which is used by the optical flow algorithm to process the data along the edges of the image. If one of the neuron lines stops responding, a local timer is started, and if that line remains silent until the timer expires, the algorithm assumes that the corresponding wall has disappeared from view and the robot is not moving paralell to the midline of the corridor. The WSC then tries to correct the robot's position by turning it towards the disappeared wall. This process is then repeated until both neuron lines respond to moving edges, or until the global timer expires.

The designed error correction controller provides a simple mechanism for reducing

[11] These LIF neurons were designed with larger receptive fields, compared to the neurons forming the neural network, but their membrane potential rises at a slower rate when excited by stimuli, in order to minimize misfiring, caused by noise.

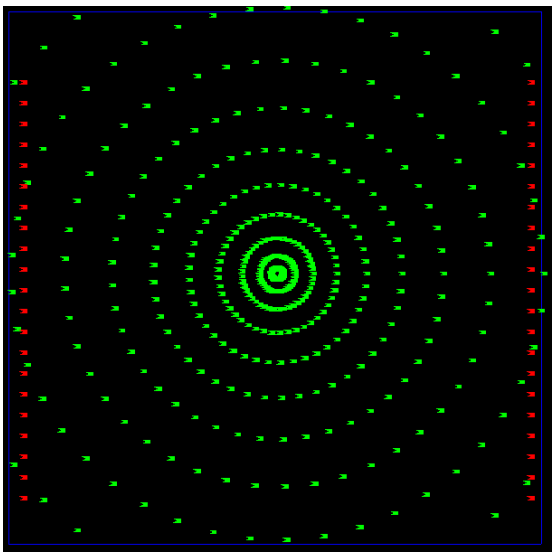


Figure 6.6: The two lines of LIF neurons (green) used during error correction.

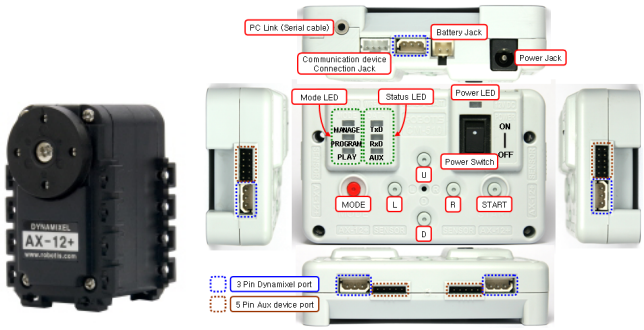
the accumulative turning error of the robot during repositioning, but it was observed during experiments that the strategy was effective: in most cases the robot managed to navigate through the corridor with high precision even when starting close to one wall.

6.3.2 Bioloid Controller

The cm-510 controller [Fig. 6.7] is responsible for manipulating the four servo motors that serve as wheels for the robot vehicle. Its original firmware provides a limited functionality allowing for specific robot motions to be designed and run. Programmers are given limited control over the servo motors and the Zigbee wireless device. Fortunately, it is possible to overwrite the standard firmware with a custom one, written in *Embedded C* (a set of language extensions to the standard C programming language, used in embedded systems). The main control logic was entirely written from scratch and two libraries for directly controlling the AX12+ servo motors and the Zigbee device were used: the Dynamixel SDK and Zigbee SDK, respectively[12].

An interrupt handler was designed to wake up the cm-510 whenever incoming data is received from the Zigbee device, in order to improve the battery life of the controller. The short data packets are combined into a single integer, comprising the message (the

[12]The Zigbee SDK was also used in the workstation relay module (discussed in Section 6.4.2) in order to control the workstation-side Zigbee device, the ZIG-100, but differs in its programming environment: the workstation relay was programmed on a GNU/Linux Operating System, while the cm510 controller was programmed on a Windows XP Operating System.



(a) (b)

Figure 6.7: Main components of the Bioloid robot vehicle. (a) The AX12+ servo motor. (b) the cm510 controller. Source: http://support.robotis.com

communication protocol is described in Section 6.4), and the command parameters are extracted: the direction of movement, the speed, and the time for executing the command. Additional controls were implemented to check whether the message contains a cancel command, which is handled differently from movement commands, or whether the time is specified or not. If the command execution time is left undefined[13] the controller will execute the command until it instructed to stop.

A major challenge during the designing of the cm-510 firmware proved to be the restricted de-bugging capability of the program code, as it was possible to open only one communication channel at any time[14]. The robot would stop responding to transmitted instructions from the workstation: neither by executing them, nor by echoing them back to the workstation (until the controller was reset), and therefore finding the root of the problem proved to be non-trivial. The 7 *LED* lights on the cm-510 were routinely used to identify sections of the code which would render the controller unresponsive. However, they were only useful in finding where the problem ocurred, but could not explain (or hint) why it ocurred. Therefore, solutions to the encountered issues handle the cases in which these issues would arise, not prevent the issues from arising at all.

[13] *Undefined* here refers to a *time* value of 0.

[14] The controller can establish a serial or wireless communication channels with the workstation, but not both simultaneously.

6.4 Robot-Workstation Communication

The following section describes the communication protocol between the robot vehicle and the workstation. The protocol is described in 6.4.1, and the relay module of the system is described in Section 6.4.2.

6.4.1 Communication protocol

An overview of the communication protocol is shown in Figure 6.8. The Workstation-Side Controller (WSC) is responsible for creating and transmitting the motion instructions to the robot. Since transmitted messages may be lost in transmission, the controller needs to be able to resend a given command once it determines that the previous transmission has been unsuccessful.

The instruction packet is a single 16-bit unsigned integer. Since the cm-510 is using an *ATMEL AVR* 8-bit microcontroller, it is essential that the smallest applicable variable type is used for executing a given task [53]. An 8-bit unsigned *char* would be too small for efficiently transmitting instructions, thus a 16-bit unsigned *int* was used to fully encode any instruction passed to the robot[15]. [Fig. 6.9]. Six command types were designed (A 7th, *status*, command type was intended to check if the robot vehicle is moving, but was later discarded) and are shown in Table 6.1.

Code	Command Type
0	Cancel last command
1	Move forward
2	Move backward
3	Turn forward left
4	Turn forward right
5	Turn backward left
6	Turn backward right
7	Status (not used)

Table 6.1: The different command types used in the communication protocol.

The first 10 bits in the instruction are used by the speed component, which defines the movement speed of the robot. The value ranges from 0 to 1023, which is the maximum

[15]Note that *Java* does not provide unsigned types: a *char* is a 8-bit signed type, while *short* is 16-bit signed. Because the range of signed 16-bit types is from $-32,768$ to $32,767$, and the largest instruction that would be transmitted is 28671, a 32-bit *int* was used at the WSC to implement the same functionality

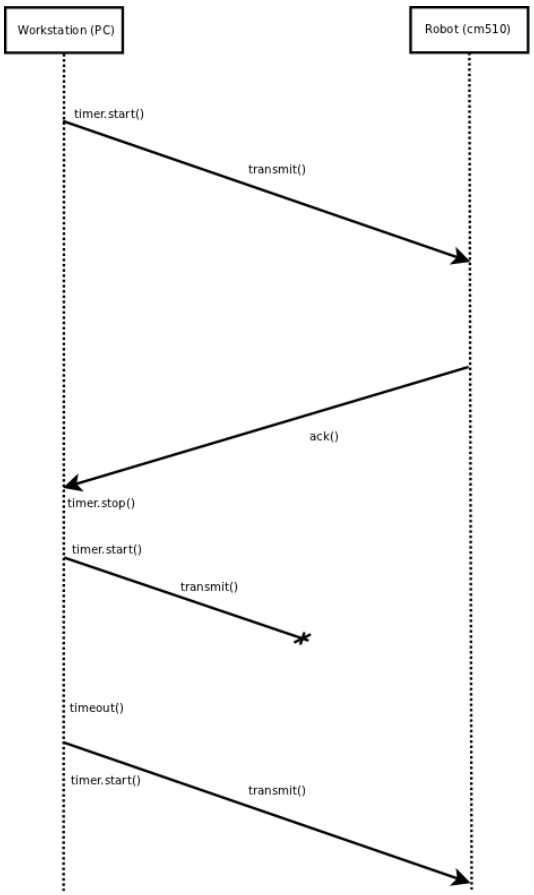


Figure 6.8: Communication protocol used between the workstation and the robot. A timer is started whenever the WSC transmits a message to the robot. If the message is not delivered to the robot (i.e. it is lost), a timeout occurs and the WSC retransmits the message.

rotation speed of an AX12+ servo motor (corresponding to about 114 RPM). The next 3 bits are used by the command component which defines the nature of the motion. The last 2 bits are used by the time of execution of the command, ranging from 0 to 2, where 0 time would mean indefinite execution.

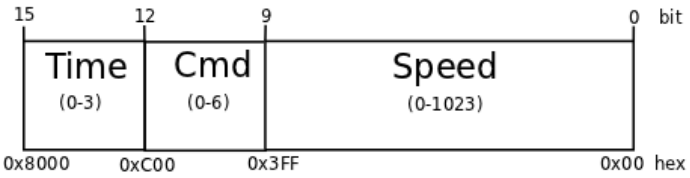


Figure 6.9: Structure of the instruction packet, a 16-bit integer type. The time value is given in seconds.

The WSC has two modes of operation related to instruction transmission: blocking

and non-blocking. Blocking transmission will force the WSC to busy-wait on receiving an acknowledgement message (ACK) from the robot in response to the previously transmitted message. The robot sends an ACK whenever it is done executing an instruction. This is done mainly for efficiency: instead of constantly querying the robot whether it has finished executing a command: a procedure that would increase message traffic (more instructions transmitted), increase the number of lost messages and retransmissions, and increase complexity, the robot simply informs the controller when it is ready to execute a new instruction. A timer is started each time the WSC transmits a message, which is set according to the type of instruction transmitted: for example, if the WSC instructs the robot to move forward for one second, then the timer will be adjusted to expire slightly over one second to allow small transmission delays. If a timeout occurs, the WSC will retransmit the last instruction.

In a non-blocking transmission the WSC will check whether the robot has ACKed the last instruction: if an ACK has been received the WSC will transmit a new instruction; otherwise, it will proceed without transmitting. The two modes require a handler for cases when a message is lost in transmission and timeout occurs[16].

6.4.2 Relay Module

The relay module is written in C and acts as the bridge between the robot controller and the WSC. The relay program connects to the workstation controller using a simple TCP Client-Server protocol (using port 4444, no 3-way handshake, network sockets) and its purpose is to forward messages from the workstation to the robot (the motion instructions) and vice-versa (the ACKs). The relay does not know what the actual contents of the messages mean and does not modify them.

It may be argued that this functionality could be implemented in the WSC itself, but it seemed better practice to reuse code whenever possible: the Zigbee SDK library was already available for controlling the Zigbee device, allowed for easy customization in order to be adapted to the needs of the project, and was written in C.

[16]The timer needs to run for sufficient amount of time to allow delays in transmission and execution: it was previously observed that a timeout occurred while the robot was completing a turn and as a consequence was prevented from properly aligning along the length of the corridor.

Chapter 7

Analysis and Results

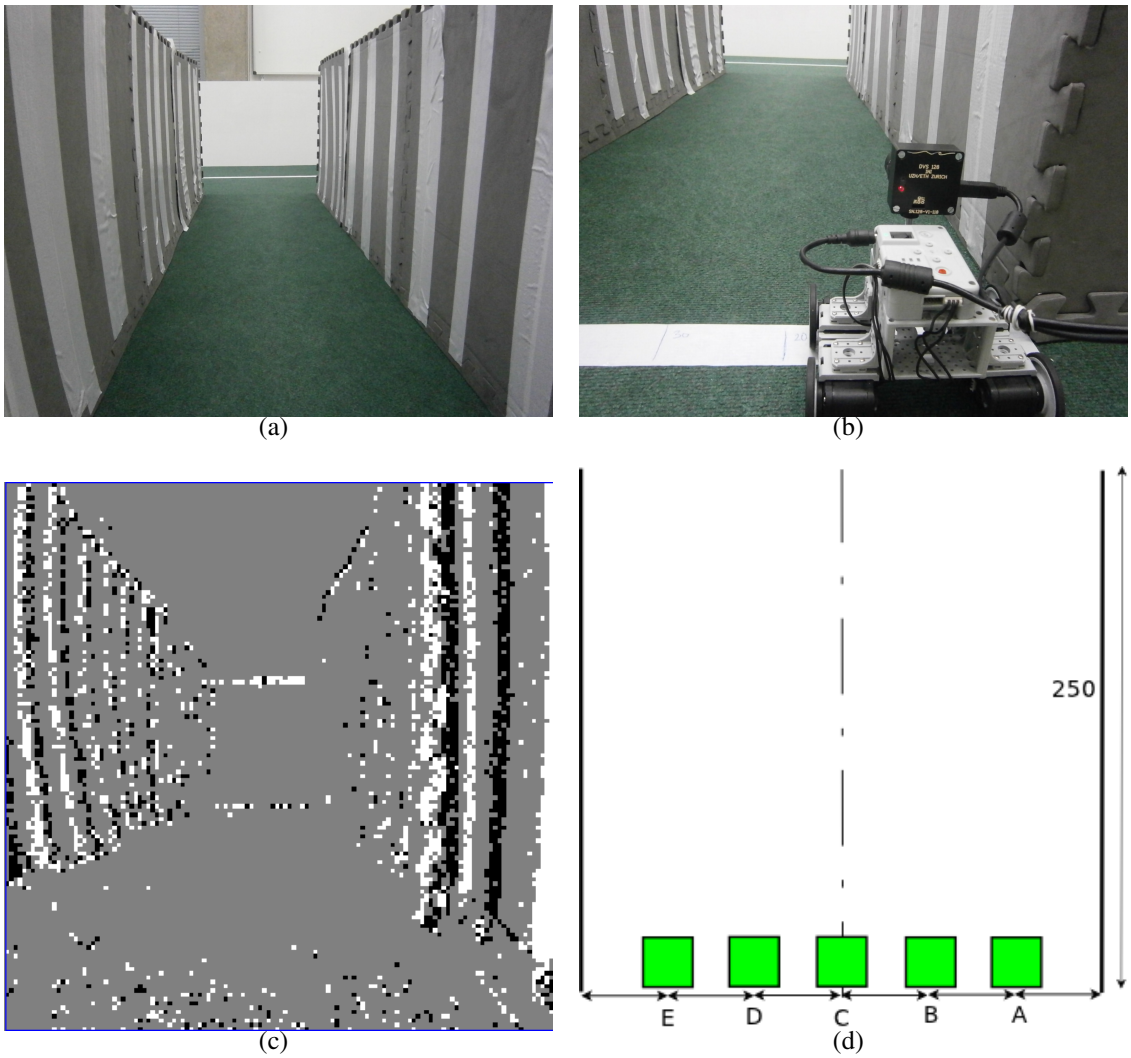
This chapter presents a detailed analysis of the algorithm and experimental results. The implemented algorithm uses numerous adjustable variables, which can be fine-tuned in order to produce satisfactory performance results. These variables are described and analysed is Section 7.1, while the experimental results and subsequent overall analysis of the navigation system are presented in Section 7.2.

7.1 Algorithm Analysis

The algorithm evaluation was performed using a set of pre-recorded sequences, roughly 4 seconds long, of the robot moving forward in the experimental corridor [Fig. 7.1]. Five starting positions of the robot were tested: approximately 10 centimeters from each wall (referred to as “far left” and “far right”), 20 centimeters from each wall (“left” and “right”) and 30 centimeters, or in the center of the corridor (“center”). Along with the three key starting positions (“far left”, “center”, “far right”), the two intermediate ones were analyzed because of their close proximity to the corridor’s midline, making them a difficult starting point, since they are likely to produce noisy results.

Each starting position was evaluated based on the objective response from the control loops as follows:

- Far Left and Left: the robot should turn right
- Far Right and Right: the robot should turn left



(a) (b) (c) (d)

Figure 7.1: The laboratory setup used during evaluation and experiments. 7.1(a): The experimental corridor with dimensions 0.06×2.5 metres. Each wall section was covered with white stripes to enchance contrast. 7.1(a): The robot next to the right wall. 7.1(c): The corridor as seen from the silicon retina. The image has been accumulated over a period of roughly 80 ms for providing better detail. 7.1(d): A diagram of experimental setup. The five starting positions are: “Far Right” (A), “Right” (B), “Center” (C), “Left” (D), “Far Left” (E).

- Center: the robot should continue forward

To get as much visual information as possible, the robot’s speed was set to the maximum of 114 RPM, or about 170 mm/s (at the cost of producing a small movement error). Light sources directly above the corridor (the main laboratory lights and the

lights above the robot football pitch) were used in order to provide a well lit environment. The corridor's width was 60 centimeters wide and about 2.5 meters long. Each starting position was recorded three times and an average result was extracted. Seven variables were identified as key to the performance of the algorithm, described in Table 7.1, and each one was individually evaluated. The analysis for each key variable is provided in the following sections.

Variable	Symbol	Initial Value	Description
Signal Arrival Time Threshold	θ	100000	Defines the time window around a signal's expected arrival time, in which the signal would be matched. In other words, a signal is matched if $\|t - t_s\| \leq \theta$. The threshold is measured in microseconds (μs)
Membrane Potential Threshold	u	2	Defines the firing behavior of the LIF neurons and is measured in mV. Each event that lies within a neuron's receptive field contributes a weight of 1 to the neuron's membrane potential
Signal Confirmation Threshold	α	2	Defines the minimum number of LIF neurons that are required to match a signal, before that signal is confirmed (see Section 6.2.5)
Conclusion Depth	β	8	The number of immediate errors to be added up before creating a conclusion (see Section 6.3.1)
Expected Initial Signal Velocity	v	28	The expected velocity, measured in pixels per second, given to newly created signals.
Minimum Number of Signals per Conclusion	γ	10	The minimum number of signals that must be confirmed before an error measurement can be produced. Defines the semaphore logic described in Section 6.3.1
Velocity Error Threshold	ε	0.5	Determines the control outcome based on the final error estimate.
Neuron Density	N	10	Determines the number of neurons per each radius in the neural network.

Table 7.1: The key variables used in the analysis of the algorithm. The initial values were chosen after several tests were performed, in which different combinations were used.

Each variable was evaluated based on four criteria:

- The average number of created signals per processing step.
- The average number of confirmed signals per processing step.
- The number of correct conclusions, or accuracy, measured in %. It is computed by counting how many produced outcomes correspond accurately to the robot's position divided on the total number of outcomes.

The sample range for each variable is shown in Table 7.2.

Variable	Start Value	Finish Value	Increment
θ	20000	587000	27000
u	1	5	1
α	2	5	1
β	1	11	2
v	6	58	4
γ	1	19	2
ε	0.1	3.8	0.4
N	3	40	4

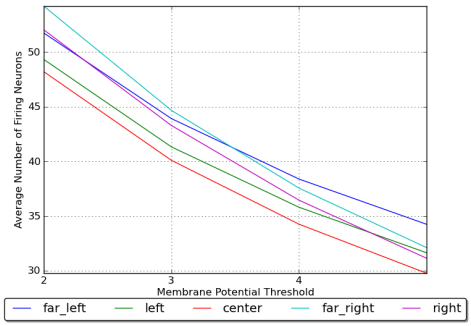
Table 7.2: The sample range for the key variables.

7.1.1 Membrane Potential Threshold

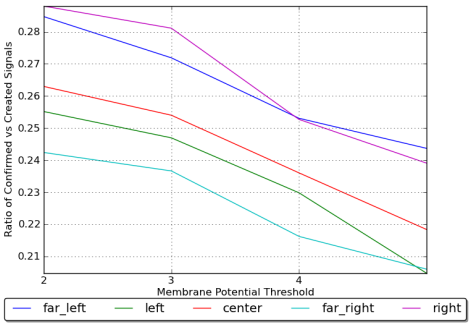
The membrane potential threshold, or u, affects the firing behavior of the LIF neurons in the network [Fig. 7.2].

With the increasing value of u, the number of firing neurons decreases [Fig. 7.2(a)], because fewer neurons will receive the necessary boost to their membrane potential in order to fire. As a result, fewer signals are created, or confirmed [Fig. 7.2(b)]. If the robot is close to one of the walls, the amount of confirmed signals from that side of the image will decrease at a lower rate, as the wall's textures are seen in greater detail (and more neurons respond), hence the accuracy of the algorithm tends to be high (even for large u). At some point, the diminishing number of firing neurons starts to affect the closer wall as well, and accuracy drops.

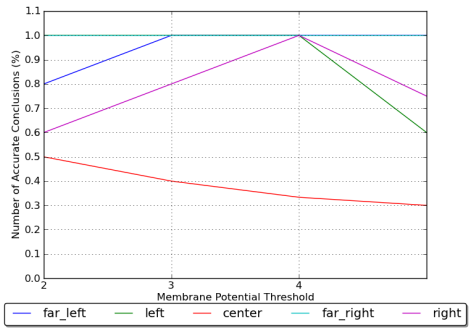
Given the initial system conditions (i.e. the initial values for each variable), the membrane potential threshold can be tuned to achieve a different level of performance, but it is difficult to optimize it for all cases. Therefore, it should be adjusted accordingly



(a)



(b)



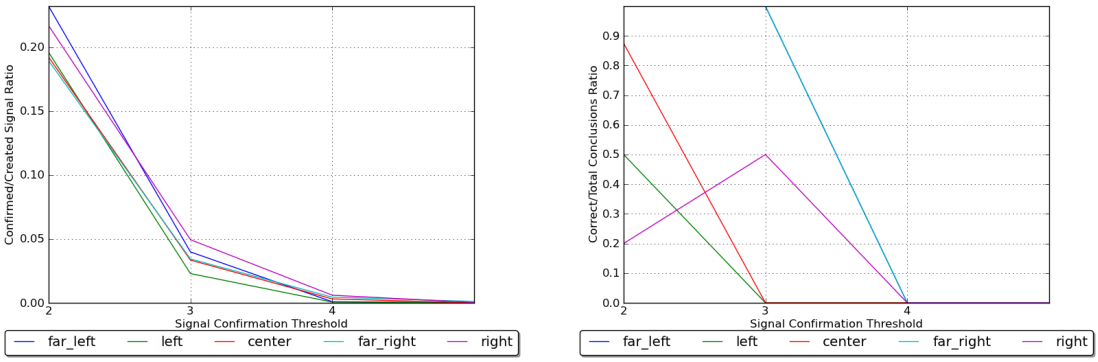
(c)

Figure 7.2: The effect of the membrane potential threshold u on the algorithm performance. (a) the average number of firing LIF neurons decreases when higher thresholds are used, as fewer neurons will gain the necessary boost to their membrane potential in order to fire. (b) The ratio between created and confirmed signals decreases as a consequence to the fewer firing neurons. (c) The number of correct conclusions (%) based on all produced conclusions during the test sequence. The closer the robot is to one of the walls, the less effect the membrane potential threshold has on the precision with which conclusions are produced. The reason is that the closer wall is seen in greater detail and still provides a good amount of visual information even at high values of v, while the information from the opposite wall decreases. The disparity in the response from the left and right produces a larger velocity difference.

in order to meet a specific set of requirements. For example, to increase the probability that the robot will stay around the middle of the corridor, a value between 2 and 4 can be used. If a higher value is set, the robot would tend to oscillate with greater amplitude around the midline.

7.1.2 Signal Confirmation Threshold

The signal confirmation threshold, or α, was briefly analysed, with results shown in Figure 7.3.



(a) (b)

Figure 7.3: Analysis of the signal confirmation threshold α. The threshold, or counter, determines how many LIF neurons a signal must traverse before it is included in the final processing. (a) The average number of confirmed signals decreases exponentially and a new signal is created every time a match is not found. (b) The accuracy of the algorithm (in % based on the whole test sequence) drops sharply for values of α greater than 4 for all cases, because a very small number of signals are confirmed and the controller waits indefinitely for sufficient data to be gathered.

The confirmed-to-total signal ratio decreases exponentially, as signals need to pass through a higher number of neurons before they are confirmed. The more neurons are required to receive a signal, the less likely that signal is going to be confirmed, beacuse of accumulating error when computing the expected arrival time and expected velocity values, and also because of the longer distance each signal has to travel (for example, when α is 5 and each radius has 7 neurons, a signal must first be created very close to the nodal point and then travel all the way up to the last neuron). With a threshold greater than 4, the ratio is close 0, which means that a few signals are confirmed at all.

A nearly linear decrease in the number of correct conclusions was observed when α increases. Because of the fewer number of confirmed signals, the controller is “starved”: it cannot gather enough data to produce a conclusion (how much data needs to be gathered is determined by the threshold γ). With a relatively short test sequence, which is about 4 seconds long, and high γ and β values, the controller needs to repeatedly gather large amounts of data before it can produce a conclusion.

The confirmation threshold determines how reliable a signal is and prevents erroneous signals (in cases when a LIF neuron is exited by noisy events) from affecting the algorithm's performance. If a high requirement is put on the signals, however, even strong signals may not be able to pass through all neurons in time and are discarded from the system. The signal arrival threshold, θ, may be adjusted to allow signal information to be stored in memory for longer periods of time at the cost of lower conclusion accuracy. Therefore, a balance between accuracy and quantity is necessary to ensure that a good amount of right conclusions are produced, while the controller received sufficient data to produce them.

Multiple experiments have shown that a value of 2 for α is enough to prevent noisy signals from affecting the outcome, while keeping a steady flow of information for the controller to process.

7.1.3 Conclusion Depth

The conclusion depth [Fig. 7.4], β, only affects the final outcome of the robot control loop, as it determines the integration period of the immediate errors before a final conclusion is produced.

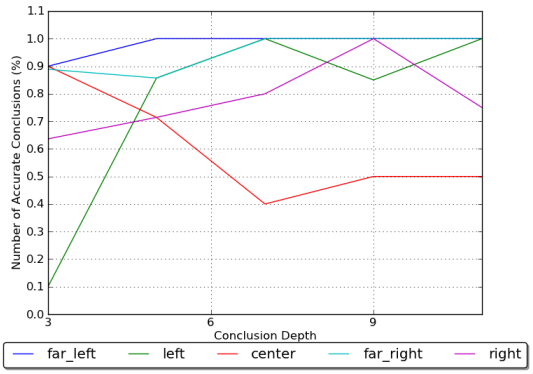


Figure 7.4: The accuracy (in % based on the test sequence) of the final outcome when different value for β is used. The precision of the outcome is high when the robot is closer to one of the walls, because of the longer integration of visual information which produces a large velocity difference. When the robot is close to the middle of the corridor, the controller's response will oscillate between moving forward and turning.

The accuracy of the final outcome tends to grow when the robot is closer to one of the

walls, because the integration of the image velocities over larger periods of time causes a greater velocity difference. When the robot is located in the middle of the corridor (so that it should move forward), a higher β would cause the controller to prefer the wall that displays slightly larger velocity.

Accuracy in the final conclusion is gained at the cost of an increase in response time: the longer the integration period, the longer the controller has to wait until it can derive a conclusion. Coupled with a large γ value (explained in the next section), the wait period may be unfeasibly long.

Expriments have demonstrated that a relatively lower value for β, combined with a low ε and a high γ produces a fairly accurate response from the robot when moving through the midline, and ensures that the robot will eventually reach the midline if it starts close to either wall.

7.1.4 Minimum Number of Signals per Conclusion

The minimum number of signals, γ, determines when the controller has enough data to produce a conclusion [Fig. 7.5]. A counter is applied to check how many signals have been confirmed: if it exceeds γ, then the controller is notified to proceed with processing the data. As is the case with β, this variable only affects the conclusion process.

As seen in Figure 7.5, the accuracy of the final outcome tends to stay at 100% when the robot is closest to either wall, since most of the confirmed signals are part of the closer wall. When the robot approaches the middle of the corridor, the ratio between confirmed signals from the left and right walls tends to even out, decreasing the velocity difference. In such cases, if the robot should be turning towards the middle, it may continue forward instead, while turning to one side when it should be moving along the midline.

The “center” position produces more accurate results with larger values for γ, due to the relatively equal number of signals being confirmed from both sides of the image. A larger γ gives a required minimum of signals that must be confirmed before the controller processes the data, but it does not control the ratio of those signals relative to both sides. Controlling the minimum visual output from both walls may be undesirable in cases when the robot is very close to one wall and the other wall has a different

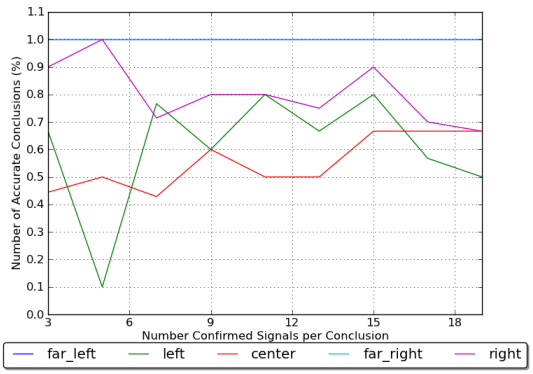


Figure 7.5: The precision of the final outcome in % based on different values for γ, for each of the five starting positions. The precision of the outcome is high at all times for positions closest to a wall, because most confirmed signals will will be generated from edges belonging to that wall. Approaching the middle of the corridor produces a more even distribution of confirmed signals, which also incurs a higher number of incorrect conclusions, especially if the robot is not properly aligned or if the integration period of the immediate errors is small.

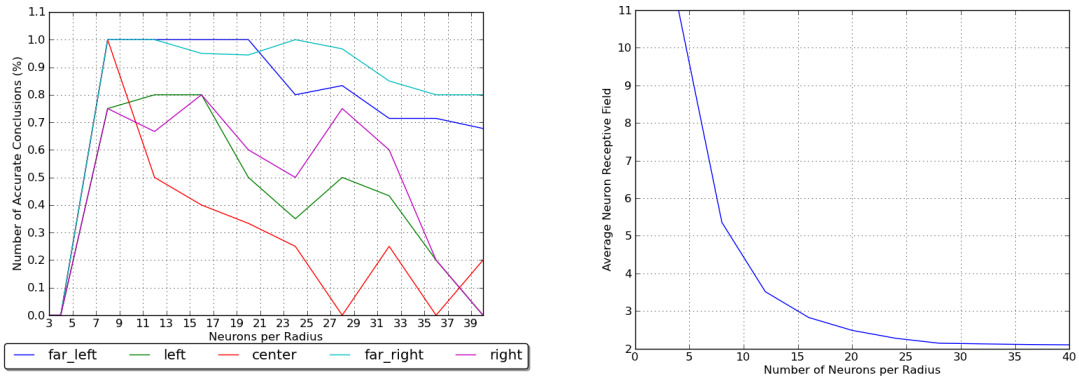
grating pattern (just like in [13], the stripe patterns were not identical on both walls), because the detail on the farther wall will be considerably limited. One side will always produce more confirmed signals than the other and the controller may never be permitted to process the data and produce a conclusion.

A high value for γ, therefore, does not ensure that the accuracy of the final outcome is high, because it does not specify the ratio of confirmed signals from both sides. It does, however, act as a semaphore that enforces synchronization between the controller and the optical flow algorithm: until at least γ signals have been confirmed, the controller should not compute the velocity difference and produce a conclusion. A higher value may also incur larger waiting time for the controller. During experiments using the initial conditions, a value of 10 for γ tends to produce a good balance between response time and accuracy.

7.1.5 Neuron Density

The neuron distribution density determines how many LIF neurons are attached to each radius of the neural network. The density therefore defines the distance between

neighboring neurons and consequently, the size of each neuron's receptive field [Fig. 7.6]. Because neurons in the network are not symmetrically arranged, neurons that are farther apart are given larger receptive fields in order to cover larger portions of the image, which would otherwise be ignored by the algorithm.



(a) (b)

Figure 7.6: Analysis of the neuron density in the network, N, based on the five starting positions.(a) The algorithm performs badly at small values for N, with accuracy close to 0%. When N is between 4 and 8, the number of accurate conclusions grows and reaches above 70%, before it starts to decrease because of the diminishing receptive field size for all neurons. (b) The average receptive field of a LIF neuron based on the neuron density. The size decreases exponentially as distance between neighboring neurons shrinks.

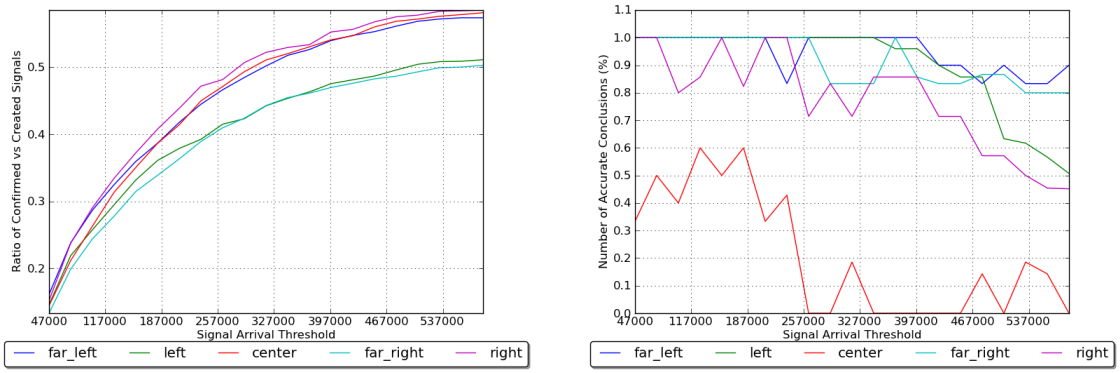
A low amount of LIF neurons (i.e. below 4) for each radius will prevent many signals to be confirmed, as most will fail to traverse the necessary number of neurons, as required by α. With an increase in N, the performance of the system quickly grows, before it starts to decrease again. The second decrease is due to the diminishing receptive field size of all neurons [Fig. 7.6(b)]. As previously mentioned in Section 6.2.3, the receptive field of a neuron is set to half the distance to its closer neighbor (the lower one). When the neuron density grows, that distance shrinks and therefore the receptive field becomes smaller. As a consequence, the LIF neurons are less likely to fire, because fewer events will excite them.

The effect the neuron density has on the accuracy of the algorithm is similar to what was observed when the membrane potential threshold u was analyzed: its influence over the final outcome increases as the robot approaches the middle of the corridor.

From the analysis and subsequent experiments, a neuron density of between 8 and 10 was found to produce good overall results.

7.1.6 Signal Arrival Time Threshold

The signal arrival time threshold, θ, determines how long a signal may be late arriving at a neuron (that is, the difference between the current time and a signal's expected arrival time) before it is discarded and its information is removed from the neuron's memory. At a high threshold, signal information is kept in memory for longer periods of time and therefore more signals are confirmed.



(a) (b)

Figure 7.7: Analysis of the signal arrival time threshold θ based on the five starting positions. (a) The ratio between confirmed and created signals grows logarithmically with higher θ, because the total number of matched signals increases. (b) The percent of accurate conclusions tends to fluctuate around 90% for positions closer to the wall and around 50% when the robot is in the middle. When θ grows too high, the large signal mismatch produces greater velocity computation errors which, as a consequence, affect the reliability of the final outcome. The effect of signal mismactch grows with the approaching of the midline, because of the diminishing strength of signals from the closer wall.

As the threshold increases, more signals are confirmed and used to compute the velocity difference from the left and right. The number of newly created signals in the neural network decreases exponentially as a larger quantity of signals are successfully matched against stored information in the neurons' memory banks.

The problem with a very high θ value is the increasing signal mismatch: the longer a signal's information remains in memory, the higher the chances that the signal will be matched against the wrong edge. The signal mismatch causes larger errors when computing the velocities of both walls (an edge may appear to be moving faster or slower than it actually is) and the accuracy of the outcome deteriorates at a rate based on the robot's position: the rate is relatively low when the robot is closest to the walls and increases when the robot moves away from the walls and is around the midline.

It was difficult to tune θ in order to get the best system performance, because of its strong relation to the expected initial velocity, discussed in the following section. A low threshold would produce the best results, but would require a very precise estimation of v, which is not a straightforward task, because motion is perceived differently from different distance to both walls. A higher threshold would be less sensitive to the accuracy of v, but the signal mismatch and subsequent errors in the image velocities would be high.

Fortunately, this strong dependence between θ and v also means that different combinations can produce good results overall. For example, several tests used 28 pix/s for v and 131000 μs for θ and produced accurate results, while similar results were obtained using 30 pix/s and 120,000 μs.

7.1.7 Expected Initial Signal Velocity

When a new edge excites a LIF neuron (i.e. the edge has just entered the camera's visual field), the neuron has no prior information about the edge's motion speed and cannot determine when the edge is going to reach the next neuron in the network. Therefore, the neuron guesses the speed and computes an estimate of the travel time. If the guessed speed does not correspond to the actual speed of the edge, the expected arrival time at the neighboring neuron would differ from the actual time of arrival. A high value of θ will increase the chances of receiving, and consequently confirming, signals that display such deviations.

When the robot is close to the midline, the virtual velocities of both walls are lower the conclusions are more accurate with a small value for v. When v increases, mismatch between the estimate and the real velocity of a moving edge grows and causes the expected arrival time of the edge to deviate considerably from its actual time of arrival.

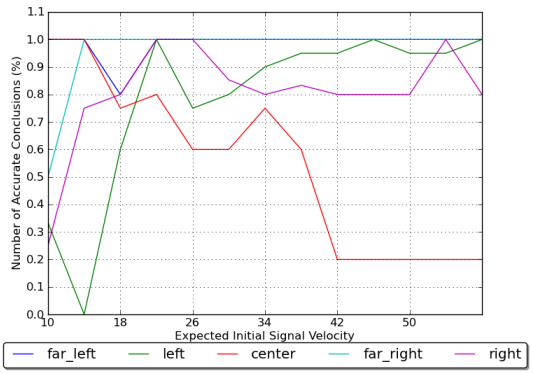


Figure 7.8: Number of accurate conclusions in % based on different values of v for each starting position. A single value of v may not be optimal for each case. For example, when v is 42 pix/s, the four starting positions away from the middle (blue, green, cyan and purple lines) were observed to produce adequate results, while performance of the system when the robot was in the center (red) was low. During experiments, a good compromise was found in the range between 22 and 30 pix/s.

In cases when θ is small, the edge will not be matched to its stored signal (however, the same signal may be matched against a different edge which excites the neuron at an earlier or later time).

As mentioned in the previous section, different combinations of v and θ may be used to produce a reliable and accurate response of the robot. During experiments, such combinations included values for v between 26 pix/s and 30 pix/s and values for θ between 120,000 μs and 140,000 μs. When lower movement speeds were applied,

7.1.8 Velocity Error Threshold

The velocity error threhold, or ε, is used to determine when the velocity difference from the left and right is high enough to force the robot to re-adjust its position. If the difference is greater than ε, than the robot should turn depending on the sign of the difference: if it is positive, the velocity from the left is greater and the robot should turn right, while if it is negative, the robot should turn left.

Increasing values of ε tend to decrease the probability that the robot will turn, because the velocity difference rarely exceeds ε. Therefore, a high value for the error threshold would produce desirable results when the robot starts in the middle of the corridor, as

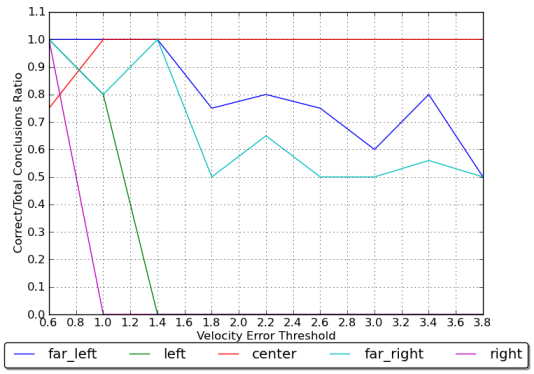


Figure 7.9: Number of accurate conclusions (in %) based on different values for ε. With larger values of ε the final computed difference between the left and right velocities is rarely great enough to produce a turning response. Therefore, a robot would keep moving forward even if it is close to one side of the corridor. The precision of the starting positions different from the center (red) tend to decrease as ε increases. The effect of ε depends on the distance to the middle: the closer the robot is to a wall, the bigger the visual information that is received from that wall and as a consequence the more accurate conclusion is derived.

the robot would most often decide to continue forward. The outcome also depends on the conclusion depth β, because the velocity difference tends to grow with the additional immediate errors.

Using the intial conditions, a value of around 0.6 for ε gives good overall results.

7.1.9 Discussion on the Performed Analysis

The conducted analysis does not provide a specific recipe for how to tweak the control system in order to achieve the best results. Optimizing all variables for all different cases (even for basic three) is not straightforward as there are intricate relationships between the different key variables that have to be considered: changing a single variable may not provide the best performance in general, while simultaneously changing several varibles may not always guarantee that the system will behave the same way as predicted when variables were independently observed.

The selection of the initial values for each variable was a product of multiple experiments, in which different combinations were tested and analyzed in a similar fashion.

However, some of these values were based on simple observations: for example, since a very small amount of signals are usually confirmed when the signal confirmation threshold is greater than 3, there is no point setting α to a larger value. In the same way, using a membrane potential threshold greater than 4 would not make sense for performing the analysis.

Even if the analysis would produce different results given a different combination of initial values, it did provide a clearer picture on how the variables are connected to each other and what dependencies between those variables exist. The following section will describe how the results of the analysis were applied to the actual control system and what outcome was observed.

7.2 Experiment and Results

Multiple experiments were conducted to test the different combinations of values for the key variables and observe the performance of the system. Additionally, different timeout values, used in the correction phase of the controller (described in Section 6.3.1), were tried to determine which one would produce a good mix of quick response and effective correction of the movement error.

Lastly, different speeds of the robot were tested to see the effect of the robot's movement speed on the system's response accuracy. The robot's minimum effective speed (which would still produce sufficient optic flow for the algorithm to operate with reasonable accuracy) was estimated at about 58 mm/s and was used for the lower bound when the turning speed was computed.

Using the analysis described above, the values for the key variables were picked with the aim to maximize the forward response of the robot (without significantly compromising the other two responses), as evaluated using the "center" starting position. The reason for this preference was to try and reduce the oscillating behavior that the robot demonstrated when it was close to the midline, while ensuring that the robot would still turn to adjust its position when it was farther away from the middle of the corridor. The values that were used are shown in table 7.3.

The robot used the maximum specified speed V_r to move forward and its turning speed was decreased by applying a weight to V_t, equal to the difference between the computed velocity difference and ε. Before turning back to position itself paralel to the corridor,

Variable	High speed	Low speed
θ	120,000 - 140,000 μs	120,000 - 131,000 μs
v	22 - 30 pix/s μs	8.4 pix/s μs
u	2 or 3	2 or 3
α	2	2
β	4	4
γ	10	4 or 5
ε	0.5 - 0.8	0.35 - 0.45
Global timeout T	1500 - 3000 ms	2000 ms
Maximum robot speed V_r	170 mm/s	83 mm/s

Table 7.3: The values that were used during the experiment, depending on the robot's speed. Some values, such as the membrane potential u and the confirmation threshold α remain unchanged, while others were adapted to the slower movement of the robot.

the robot moved forward for a short period of time with the turning speed. The final turn was executed in backwards direction (i.e. the robot would turn while moving backwards) to prevent it from colliding with a wall, and also to increase the experiment time (given the short length of the corridor).

The corridor was set with the same dimensions as the ones used in the evaluation. The width was again 60 cm, as a smaller width did not provide enough space for the robot to maneuver and therefore proved difficult to analyze. Larger widths were also ineffective because of the restricted field of view of the camera (46.2 deg. horizontal angular field of view, see Table 2.1).

The conducted experiments showed that the navigation system could be adjusted in order to achieve good and accurate results, as shown in Figure 7.10. The low values for β and γ produced relatively quick responses, allowing the robot to detect its movement deviation from the goal trajectory along the corridor midline early, and to have more time to re-adjust. Smaller error thresholds ε caused the robot to oscillate around the middle with greater amplitude, as the turning speed would be larger [Fig. 7.10(a)]. This behavior was more obvious when higher robot speeds were used, while its was dampened with lower (or minimum) speeds [Fig. 7.10(b)]. High values for ε reduced the accurate response of the robot, as it would prefer to move forward even if it should be turning towards the middle.

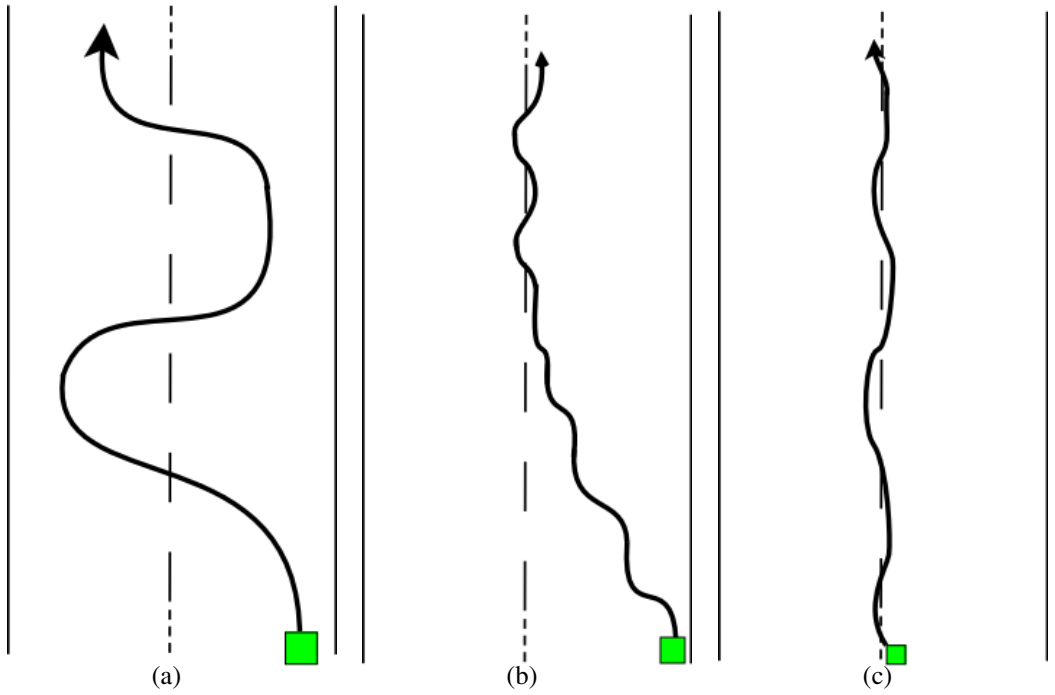


Figure 7.10: The movement trajectory of the robot based on different settings. 7.10(a): Using maximum movement speeds and a low ε value (around 0.6) forces the robot to oscillate with a greater amplitude around the corridor midline. 7.10(b): Observed robot trajectory when starting next to the right corridor wall. The robot's maximum speed is set to 83 mm/s and ε, v and γ were decreased accordingly. The robot's motion is more smooth, as it slowly approaches the middle of the corridor. Once there, it tends to stay on track until it loses the walls from sight (i.e. it has reached the end of the corridor). 7.10(c): The same setting as 7.10(b) but with higher ε value. The robot tends to move along the midline without swerving.

A large global timeout value T milliseconds, which is used during the correction phase, allowed the robot to detect small turning errors earlier [Fig. 7.11], as visual information during the phase would be integrated for longer periods of time. The problem with larger timeouts was the longer period of visual "blindness" with respect to the robot's position in the corridor, because the controller would ignore the data produced by the neural network and uses only information from the two neuron lines.

Smaller timeout values resulted in a shorter correction phase, and therefore the controller would quickly return to process the visual information from the neural network, but smaller movement errors would not be detected early until the error has grown. For a timeout of about 2500 milliseconds, the robot managed to correct its error after the

second position re-adjustment.

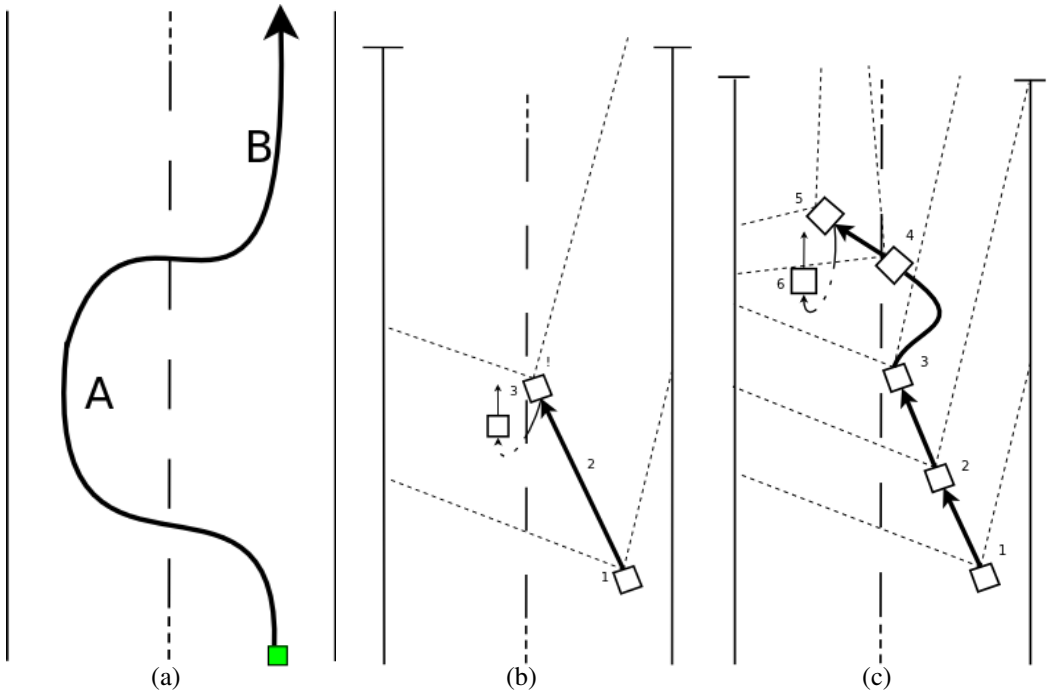


Figure 7.11: The movement trajectory of the robot using different timeout values for the correction phase (the dotted lines show the camera's field of view, which is not a precise representation of the actual field of view of the sensor). 7.11(a): Using maximum robot speed and a large timeout value. After the robot has readjusted it will enter in a long correction phase, (A) and (B), in which it will move forward, but won't respond to perceived deviation from the goal position. 7.11(b): Diagram of the robot's response using a high timeout value. If the robot's readjustment has caused a slight deviation from the goal alignment relative to the corridor (1), the robot will be able to detect it, because it will remain in the correction phase longer (2). Thus, it will eventually detect the error and re-align (3). 7.11(c): Diagram of the robot's response using a small timeout value. The robot will fail to detect the motion error (2) and will proceed with normal operation. The misplacement of the image will force the robot to turn right (3), and after it has readjusted (4), it will eventually detect the error (5) and re-align (6).

At lower speeds, the robot performed quite well and with very good precision. A speed of roughly 80 mm/s was set for forward motion[1] and lower values for v, γ and ε were used (γ was decreased because fewer LIF neurons were expected to fire, thus the number of confirmed signals will be lower). While it would take longer for the robot to

[1] Similar to the forward motion of *Robee* in [47]

reach the middle of the corridor when starting close to a wall, the oscillations observed around the midline were with a very low amplitude, thus the robot managed to stay on track for extended periods of time [Fig. 7.10(b)].

7.3 Conclusion

The conducted analysis demonstrated how the algorithm may be adjusted to increase its effectiveness and reliability based on an established initial set of values. It also demonstrated that different combinations of the key variables may produce similar results in terms of accuracy and response rate.

Using the values derived from the analysis, a set of experiments were performed which demonstrated the actual performance of the robot navigation system when applied to the corridor scenario. Overall results were positive, as the robot managed to perform well in all different cases (based on the starting position), and even demonstrated high precision when it was moving at the lowest effective speeds.

As a conclusion, the next chapter will discuss different possible modifications, improvements and extensions to the project, which are left as future work.

Chapter 8

Discussion

The robot navigation system is based on the centring response of honeybees described in [13]. Just like the biological experiment, the two corridor walls were covered with vertical white stripes forming asymmetrical texture patterns, which are used to induce an optical flow field, allowing the robot to navigate through the corridor. The robot used the optic flow field induced from its own motion to balance the apparent velocities of both walls, while remaining independent of the contrast frequencies produced by the successive ON and OFF edges of the stripes. However, the navigation system depended on the amount of contrasting edges that were visible by the camera, in order to compute the velocities.

The horizontal movement of wall gratings that were used to shift the honeybees' trajectory closer to one of the walls, was simulated by applying different weights when computing the final outcome. For example, when more weight was put on the velocity estimate of the left side of the image, the robot would shift its goal trajectory closer to the right wall, and as a consequence would move away from the midline of the corridor. Because no mechanism was designed to compute the distance between the robot and the walls, however, the robot would often collide with the wall as it turns towards it, instead of keeping some distance away from it.

While the lack of a second camera presented a considerable challenge, and did not fully represent the biological model on which the system was based, the overall positive results from the conducted experiments demonstrated that even with a constrained setup, the robot navigation system can produce accurate results, which resemble centring behavioral response of honeybees. Possible improvements to the current system

are presented and discussed in Section 8.6.

8.1 Visual Blindness Assumption

The "visual blindness" assumption, that holds when the robot is turning, was necessary for two reasons. Firstly, a single-camera configuration is not a flexible approach to individually process information from both walls: the corridor is narrow in order to allow the camera to see the walls with sufficient detail, and when the robot turns towards a wall, that wall will gradually expand onto the whole image, while the opposite wall will disappear from sight. Therefore, even with the slightest turning of the robot (but yet sufficient enough to be noticed), the assumption that each side of the image corresponds to a separate corridor wall will quickly become invalid. Secondly, and as a consequence, the algorithm is based on the observation that certain animals apply optical flow reduction techniques, in which only the radial flow components are used for navigation purposes. In this case, rotational and translational motions would be minimized (such as the sharp turns observed in flies [15]) and excluded from visual processing.

An improvement in this case would be the introduction of a second camera and repositioning both cameras laterally, which is left as possible future work (see Section 8.6).

8.2 Processing Boundaries

The corridor that was built for the experiment is not a perfectly isolated laboratory environment (nor it should be) and certain objects or textures, such as a taped white line on the floor, are visible during robot navigation. Such textures produce a flow field that is not related to any of the corridor walls and should be excluded from the algorithm's compuations. For this reason, "processing boundaries" were introduced which define the parts of the image included in the optical flow computations. These boundaries are given by a pair of parabolas with vertical axis and focus point $(64, 64)$, given by the formula

$$x = f(y) = \pm \frac{64}{sMin^2}(y-64)^2 + 64$$

where *sMin* is the semi-minor axis. Any signal that is confirmed within these conic sections are added to the final velocity estimations, while other signals are ignored. These conic sections do not always cover the whole of the corridor walls (especially in cases in which the robot is very close to a wall, portions of the wall appear beyond the boundaries and are not processed), but reduce the number of "noisy" signals and velocity compuations.

8.3 Edge Polarity

Along with using polarity information for edge matching, polarity is used by neurons to restrict the number of propagated signals. During development it was observed that the wall textures are presented by alternating ON and OFF edges (or vice-versa). Since the moving speed of an edge is less than the algorithm's rate of processing event packets (if a packet's duration is rougly 16 milliseconds, than the rate of processing for the algorithm would be 62.5 packets per second), the same edge may keep a neuron constantly excited and firing before moving beyond its receptive field. Instead of generating a new signal each time the neuron fires (assuming its memory is empty), the neuron can keep track of the polarity of the edges and only fire when the polarity changes. Thus, instead of firing multiple times when an ON edge is present[1], the neuron will fire once and then use a switching mechanism that will prevent it from propagating additional signals (the neuron will continue firing) until an OFF edge excites it. This way, the number of propagated signals and the number of stored signals overall is kept to a minimum.

The neuron may misfire, or it may confuse the polarity of an incoming edge. In those cases, it may ignore the next incoming edge, treating it as the same one (carrying the same polarity even though it actually saw an opposite-polarity edge). Thus, a time threshold is used to reset the polarity switch for each neuron which has not transmitted a signal for an extended period of time (0.5 second)

[1] Polarity of an edge represents the polarity of the majority of address-events that form the edge.

8.4 Feedback Control

A second feedback control mechanism was tried, aiming at minimizing the turning error of each servo motor due to voltage and current variances. Initially, it was discovered that the voltage for the servo motors could not be manipulated, and no current sensing was build within the motor. The only way to gauge the turning error was the rotation speed of the servo motor, which was being measured by the controller. Thus, the intended feedback controller would compute the goal travel distance for each motor (using the command parameters for time and speed) and would try to approximate the actual travelled distance to the goal distance. The time of execution for each motor would then be manipulated using the difference between the goal and actual distance travelled. While this feedback control mechanism performed well on certain occasions, it was quickly found that the rotation speed measurement contained a 60-degree dead band (between 300-360 degrees) in which the measurement was completely inaccurate. In the context of full rotating servo motors, the existence of this dead band means that every time the servo motor's rotation angle reaches 300 degree, the actual speed measurement becomes unreliable. Therefore, for the indended purposes, this mechanism would not be useful.

8.5 Extending the Algorithm

The connections between the LIF neurons in the neural network are not hard-coded, therefore they can be changed and configured in a custom way, changing the structure and possibly the function of the network as a whole. A modified network configuration may be applied to obstacle detection problems. Instead of linking neurons to their immediate neighbors on the same radius, neurons can be linked together based on their position on a radius. In other words, the neurons can be grouped by their distance from the nodal point, forming rings around the image center. Such rings may then be connected to their immediate neighbors. Neural responses can be handled on a ring level: if a sufficient number of a ring's neurons fire, then the ring fires. The firing behavior of all rings can be monitored to determine is there is expanding or contracting motion within the scene.

8.6 Improvements and Future Work

Several possible improvements to the optical flow algorithm, or the robot navigation system as a whole, are presented and described below.

8.6.1 Adding a Second Camera

The introducton of a second camera would contribute greatly to the designed navigation system. When the two cameras are placed laterally, the induced optical flows would be greater and would allow for higher precision when computing the average velocities for both walls. The optic flow field from a single camera pointing in the direction of movement gives less edge detail and the distance between edges is affected by the angular position of the wall with respect to the camera. In a lateral positioning of the eyes, greater detail of the wall textures would be achieved, allowing for more accurate computations of the velocity differences.

It is possible and straighforward to connect two DVS sensors and use the visual output from both, and the optical flow algorithm would require certain modifications to accomodate the second camera. Instead of dividing the visual information from each camera into data related to the left and right sides of the image, each camera would produce output for one side. The optical flow field would no longer be reduced to its radial component, since the motion would be mostly translational and rotational (when the robot is turning away or towards a wall). Therefore, the neural network would need to be reconfigured with new connections between neurons and this configuration would not follow the distribution described in Section 6.2.2. The signal propagation mechanism would not need any modifications, as its function is not dependent on the network structure. The Workstation-Side Controller would then use the processed data from both cameras to compute the velocity error as before.

With a two-camera approach, the visual “blindness” assumption would no longer be necessary. As the two corridor walls would be in constant view, the movement speed and direction of the robot can be directly controlled using the computed velocity disparity from both cameras. If, for example, the velocity from the left side is greater, the robot’s left servo motors would run at a faster rotation speed, turning the robot in direction of the right wall. The velocity on the left side will start to gradually decrease while the velocity on the right side will gradually increase, until the velocity disparity

is minimized, in which case the robot would be positioned along the midline of the corridor. Furthermore, the robot would be able to discern the distance to each wall, so that it would be able to prevent possible collisions.

8.6.2 Robot Speed Control

The ground speed control that is observed in flying insects [20] could also be applied to the robot navigation system in order to control the amount of visual input that the optical flow algorithm receives. The system would increase or decrease the robot vehicle's speed to meet a given target optic flow (in terms of the number of confirmed signals per certain period of time, for example). This response to the strength of external stimuli may be useful when the visual system does not receive enough input in order to function properly, or when it is being overburdened with too much information (or it needs to slow down to avoid colliding with objects). The change of movement speed should be followed by appropriate adjustment to certain parameters in order to adapt to the new optical flow. Such parameters, as discussed in Section 7.2, include the initial velocity v and the velocity error ε[2].

8.6.3 Flexible Computation of Edge Velocity

The flaw of assigning a static estimate to the velocity of edges when no prior information is available, is that it assumes that the apparent motion velocity remains the same iregardless of the distance to the walls. This assumption may compromise the performance of the system, if an adequate and balanced velocity estimate is not used, because large disparities between the expected and real velocities of perceived edges can incur large computational errors when measuring the velocity difference. The static assignment of an expected initial velocity for new signals could be replaced by a more flexible and local to each neuron mechanism.

Such a mechanism could involve each LIF neuron to locally compute the velocity of perceived edges and use the local estimate to communicate the expected arrival time of the edge to its neighboring neurons. The precision of this local computation may depend on the integration period of the velocity signal, but would produce more reliable

[2]The third parameter that was decreased when the system was tested at lower speeds, γ, could remain unchanged and be used to determine the target optic flow.

estimates compared to assigning it a single static value. An example implementation would combine the delay mechanism used by the LIF neurons, and the local velocity computation: while the neuron is waiting for the edge to come sufficiently close to its virtual position in the image before transmitting a signal to its neighbors, it would also calculate its movement speed.

Chapter 9

Conclusion

A robot navigation system was designed and implemented using a neuromorphic visual sensor, the DVS 128, to extract optic flow information from the perceived scene. An optical flow algorithm, which employes a radial network of LIF neurons, was developed to use the pre-processed visual output from the camera and navigate a Bioloid robot vehicle along the midline of a narrow corridor, modeling the centering response observed in flying insects.

The optical flow algorithm was designed to be flexible, allowing core properties to be adjusted and tuned dynamically, in order to meet certain requirements, such as stability when moving along the midline, short response time, or more reliable detection of accumulating motion errors. It was also conceived to be easily extensible, as the underlying neuron network can be modified and re-configured with ease to achieve different navigation tasks (such as collision avoidance).

The experimental setup, which was used to assess the navigation system, was based on a biological experiment described in [13], which analyzed the centering response of honeybees. Results from the conducted tests demonstrated that the robot navigation system performs well when steering the robot along the middle of the corridor. Oscillations around the midline were observed to vary depending on the configured parameters of the algorithm, and could be reduced to produce smoother trajectory.

The neuromorphic camera was successfuly integrated into the robot control system, and provided an efficient and robust means for processing the relevant visual information. With its asynchronous event-driven characteristics, the sensor allowed a more efficient and bio-inspired approach to machine vision to be adopted and applied.

Further improvements to the developed system, which would enhance and extend its functionality, were presented and described, and are left as future work.

Bibliography

[1] Posch C. Bio-inspired vision. *Topical Workshop On Electronics For Particle Physics*, 2011.

[2] Delbrück T. Lichtsteiner T., Posch C. A 128x128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2), 2008.

[3] Onishi M. Yagi T. Hosoe S. Akiyama K., Luo Z. Local computation of optical flow using a silicon retina. *Systems and Computers in Japan*, 36(11), 2005.

[4] Mathur B. Koch C. Neuromorphic vision chips. *Systems and Computers in Japan*, 33(5), 1996.

[5] Fitzgibbon A. Robertson C. Trucco E. Fisher R., Dawson-Howe K. *Illustrated Dictionary of Computer Vision*. Wiley-Blackwell, 2005.

[6] Marr D. *Vision: A computational Investigation Into the Human Representation and Processing of Visual Information*. New York: Freeman, 1982.

[7] Ros E. del Piño B. Barranco F., Diaz J. Visual system based on artificial retina for motion detection. *IEEE Transactions on Systems, Man, and Cybernetics- PART B:Cybernetics*, 39(3), 2009.

[8] Kaess F. Grenet E. Heitger F. Burgi P.Y. Gyger S. Nussbaum P. Ruedi P.F., Heim P. A 128x128 pixel 120-db dynamic-range vision-sensor chip for image contrast and orientation extraction. *IEEE Solid-State Circuits*, 89, 2003.

[9] Delbrück T. Lichtsteiner T., Posch C. A 128x128 120db 30mw asynchronous vision sensor that responds to relative intensity change. *IEEE International Solid-State Circuits Conference*, 89, 2006.

[10] Lichtsteiner T. Delbrück T. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. *IEEE International Symposium on Circuits and Systems*, 2007.

[11] Häfliger P. Delbrück T. Jensen A. Drazen D., Lichtsteiner T. Toward real-time particle tracking using an event-based dynamic vision sensor. *Experiments in Fluids*, 51(5), 2011.

[12] Donath N. Gritsch G. Kohn B. Litzenberger M. Posch C. Schñ P. Bauer D., Belbachir A.N. and Schraml S. Embedded vehicle speed estimation system using an asynchronous temporal contrast vision sensor. *EURASIP Journal on Embedded Systems*, 2007, 2007.

[13] Lehrer M. Collet T. Srinivasan M. V., Zhang S. Honeybee navigation en route to the goal: Visual flight control and odometry. *The Journal of Experimental Biology*, 199, 1996.

[14] Gerdes V. Wörgötter F, Cozzi A. A parallel noise-robust algorithm to recover depth information from radial flow fields. *Neural Computation*, 11, 1999.

[15] Wagner H. Flight performance and visual control of flight of the free-flying housefly (musca domestica l.).i. organizaton of the flight motor. *Journal of Experimental Biology*, 138, 1988.

[16] Green P.R. Davies M.N.O. Head-bobbing during walking, running and flying: Relative motion perception in the pigeon. *Journal of Experimental Biology*, 138, 1988.

[17] Benosman R. Vision without frames: A semiotic paradigm of event based computer vision. *Biosemiotics*, 3(1), 2009.

[18] Schunk B.G. Horn B.K.P. Determining optical flow. *Artificial Intelligence*, 17(1-3), 1981.

[19] Beauchemin S.S. Barron J.L., Fleet D.J. Performance of optical flow techniques. *International Journal Of Computer Vision*, 12, 1994.

[20] Zhang S. Srinivasan M. V. Visual motor computations in insects. *Annual Review of Neuroscience*, 127, 2004.

[21] Meister M. Gollisch T. Eye smarter than scientists believed: Neural computations in circuits of the retina. *Neuron*, 65, 2010.

[22] Liu X. El Gamal A. Kleinfelder S., Lim S. A 10 000 frames/s cmos digital pixel sensor. *IEEE Journal of Solid-State Circuits*, 36(12), 2001.

[23] Choi E. Cauwenberghs G. Etienne-Cummings R. Mallik U., Clapp M. Temporal change threshold detection imager. *IEEE International Solid-State Circuits Conference*, 1, 2005.

[24] Poumplun M. Garaas T.W. Retina-inspired visual processing. In *Bio-Inspired Models of Network, Information and Computing Systems*, 2007.

[25] Meister M. Gollisch T. Rapid neural coding in the retina with relative spike latencies. *Science*, 319, 2008.

[26] Manu M. Meister M. Baccus S.A., Ölvezcky B.P. A retinal circuit that computes object motion. *Journal of Neuroscience*, 28, 2008.

[27] Meister M. Ölvezcky B.P., Baccus S.A. Segregation of object and background motion in the retina. *Nature*, 423, 2003.

[28] Jordan T.A. Meister M. Berry M.J., Brivanlou I.H. Anticipation of moving stimuli by the retina. *Nature*, 398, 1999.

[29] Mante V. Tolhurst D.J. Dan Y. Olshausen B.A. Carandini M., Demb J.B. Do we know what the early visual system does? *Journal of Neuroscience*, 25, 2005.

[30] Wcislo W. Warrant E. Dacke M. Baird E., Kreiss E. Nocturnal insects use optic flow for flight control. *Biology Letters*, 7(4), 2011.

[31] Blanes C. Brady J.M. Franceschini N., Pichon J.M. From insect vision to robot vision. *Philosophical Transactions: Biological Sciences*, 337(1281), 1992.

[32] Dacke M. Baird E., Kornfeldt T. Minimum viewing angle for visually guided ground speed control in bumblebees. *Journal of Experimental Biology*, 213, 2010.

[33] Chahl J.S. Barth E. Venkatesh S. Srinivasan M.V., Zhang S.W. How honeybees make grazing landings on flat surfaces. *Biological Cybernetics*, 83, 2000.

[34] Longuet-Higgins H.C. Horridge G.A. What can engineers learn from insect vision? *Philosophical Transactions: Biological Sciences*, 337(1281), 1993.

[35] Goldsmith T.H. Fine structure of the retinulae in the compound eye of the honeybee. *Journal of Cell Biology*, 14(3), 1962.

[36] Land M.F. Visual acuity in insects. *Annual Review of Entomology*, 42(1), 1992.

[37] M.V. Srinivasan Kirchner W.H. Freely flying honeybees use image motion to estimate object distance. *Naturwissenschaften*, 76(6), 1989.

[38] Kral K. Srinivasan M.V., Poteser M. Motion detection in insect orientation and navigation. *Vision Research*, 39(16), 1999.

[39] Gregory R.L. Srinivasan M.V. How bees exploit optic flow: Behavioural experiments and neural models. *Philosophical Transactions: Biological Sciences*, 337(1281), 1992.

[40] Smith O.W. Flock H. Gibson E.J., Gibson J.J. Motion parallax as a determinant of peceived depth. *Journal of Experimental Psychology*, 58(1), 1959.

[41] Eriksson E.S. Movement parallax and distance perception in the grasshopper (phaulacridium vittatum (sjöstedt)). *Journal of Experimental Biology*, 86, 1979.

[42] Collett T.S. Peering - a locust behaviour pattern for obtaining motion parallax information. *Journal of Experimental Biology*, 76, 1978.

[43] Egelhaaf M. Boeddeker N., Kern R. Chasing a dummy target: Smooth pursuit and velocity control in male blowflies. *Proceedings: Biological Sciences*, 270(1513), 2003.

[44] Rodríguez-Álvarez M. Ros E. Meyer-Baese U. Molina M.C. Botella G., García A. Robust bioinspired architecture for optical-flow computation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(4), 2010.

[45] Yagi T. Inoue K., Kameda S. Real-time motion detection with a mixed analoguedigital neuromorphic vision system. *International Congress Series*, 1301, 2007.

[46] Jímenez-Fernández A. Delbruck T. Lichtsteiner P. Linares-Barranco A., Gómez-Rodríguez F. Using fpga for visuo-motor control with a silicon retina and a humanoid robot. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007.

[47] Curotto F.-Garibaldi S. Santos-Victor J., Sandini G. Divergent stereo for robot navigation: Learning from bees. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1993.

[48] Curotto F.-Garibaldi S. Santos-Victor J., Sandini G. Divergent stereo in autonomous navigation: from bees to robots. *International Journal of Computer Vision*, 14(2), 1995.

[49] Humbert J.S. Shoemaker P.A., Hyslop M.A. Optic flow estimation on trajectories generated by bio-inspired close-loop flight. *Biological Cybernetics*, 104(4-5), 2011.

[50] The project is open source and can be found here:. `http://sourceforge.net/apps/trac/jaer/wiki`.

[51] Abbott L.F. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5-6), 1999.

[52] Kistler W.M. Gerstner W. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[53] Avr035: Efficient c coding for avr. available online:. `http://www.atmel.com/Images/doc1497.pdf`.