Biologically Inspired Sensor-Fusion Algorithm for Self-Motion Detection

eingereichtes
PROJEKTPRAKTIKUM
von

Mark Hartenstein

geb. am 08.03.1990
wohnhaft in:
Riesenfeldstraße 18
80809 München

Lehrstuhl für
STEUERUNGS- UND REGELUNGSTECHNIK
Technische Universität München
Univ.-Prof. Dr.-Ing../Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Betreuer/-in:	M. Sc. Cristian Axenie
Beginn:	22.10.2012
Zwischenbericht:	13.12.2012
Abgabe:	18.01.2013

Abstract

In this coursework a biologically-inspired sensor-fusion model was investigated and successfully implemented. An application was developed that communicates with a robot which is equipped with an inertial measurement unit and an event-based visual sensor. The application fuses the data received from the sensors according to the sensor-fusion model and ouputs an improved heading estimate.

Zusammenfassung

In diesem Projektpraktikum wurde ein biologisch inspiriertes Sensor-Fusions-Modell untersucht und erfolgreich implementiert. Es wurde ein Programm entwickelt, welches mit einem Roboter kommuniziert, der mit einem Inertialnavigationssystem und einem ereignisbasierten visuellen Sensor ausgestattet ist. Das Programm fusioniert die von den Sensoren erhaltenen Daten mithilfe des Sensor-Fusions-Modells und gibt eine verbesserte Ausrichtungsschätzung aus.

Contents

1 Introduction and motivation

In robotics, it is common practice to fuse the data from multiple sensors in order to get a better estimate of the real state that is observed rather than considering the data from individual sensors, as this may be noisy and inaccurate. This synergistic combination of information brings redundancy and complementarity in the perception enhancing the precision and the robustness of the percept. The state of the art methods in sensor fusion are based on probabilistic inference, built on top of Bayes' rule and Bayes' network [HBoR]. Well-known algorithms which use probabilistic models include the Kalman filter, sequential Monte-Carlo-methods and functional density estimates. [HBoR] gives a short overview of the models and algorithms presented above. In probabilistic models the probabilities of both the value of the real state that is being observed and the correctness of the observation by the sensors have to be modelled. Therefore a deep knowledge of the characteristics of the observed process and the sensors are required. Once modelled, the system only works for specific sensors in a specific setting. Furthermore, these models are computationally complex as they require computing a large number of probabilities and in order to extend them to a different sensor configuration the entire model must be redefined. Several multisensory data fusion models were developed to cope with the main drawbacks in probabilistic approaches. These alternative methods include interval calculus, fuzzy logic and theory of evidence, which are offering some advantages in terms of flexibility, generality and computational costs.

In this coursework an alternative, biologically inspired, sensor fusion model was analysed and implemented. The model is fusing vestibular information (orientation) and vision information to get a more precise heading angle. It is based upon a self-motion detection multisensory fusion model of the neural network structure that can be found in the dorsal medial superior temporal area (MSTd) of macaque apes, as described in [NormMod].

It will be shown that this model outputs good state estimation while not being computationally too complex. There is no need for prior information about the sensors, thus providing a good flexibility. The model is easily scalable to enhance the precision of the output.

2 The scenario

The test scenario will try to validate the biologically inspired algorithm using a omnidirectional mobile robot equipped with sensors that will be used to feed data into the model. The "OmniBot" is equipped with an entire range of sensors (odometry, bumpers, IMU, vision sensor) from which relevant for the application are the gyroscope (placed in the housing, in the middle of the robot) to act as the vestibular input in the model, and a vision sensor (event-based camera "eDVS").
See fig. 1.

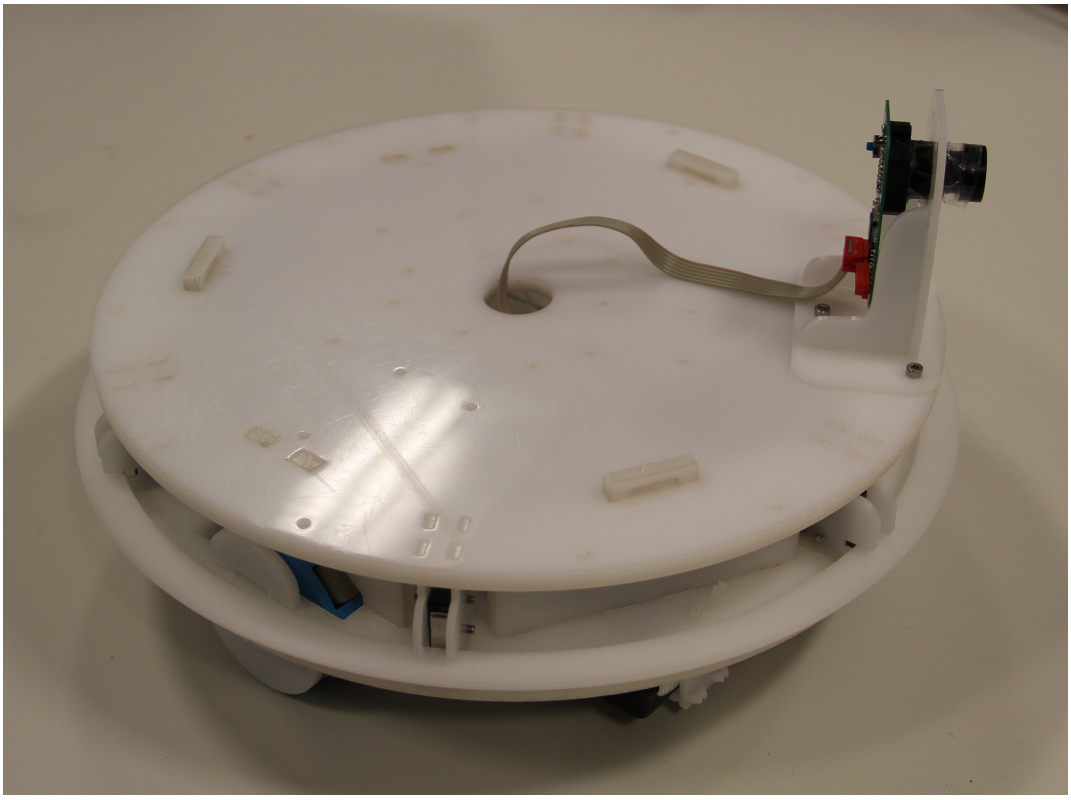

Fig. 1: Robot with mounted eDVS sensor.

Information from both sensors will be fused using the above mentioned biologically inspired model in order to estimate the heading of the robot. In our scenario the robot's movement is constraint to yawing, although the fusion model presented in [NormMod] can also be applied to estimate the pitch angle.
The robot has also a built-in WiFi module that communicates with the application that runs on an external PC via wireless LAN. The raw data from the two embedded sensors is streamed at a certain frequency from the robot to the off board processing machine. The robot can receive commands, e.g. setting the speed of the motors or starting the sensors (see [NomniBot] for the documentation). In this coursework, an application was

developed to control the robot, receive the sensor data and compute the output of the neural model to get a merged heading direction while the robot is moving.

2.1 The eDVS sensor

The eDVS sensor serves as a visual sensor. It is an event-based sensor, i.e. instead of outputting a whole frame as common cameras do, it only reports if the intensity of a certain pixel exceeds a threshold ("on-event") or falls below it ("off-event"). The sensor outputs the type of event and the position of the pixel in the sensor field of view. The concatenation of position and type of event will from now on be referred to as one single "event". Events represent movements of objects in the visual field of the sensor and can be used to compute the optic flow (see chapter 4.24.24.2Retrieving the angular speed from eDVS events) and using the global optic flow velocity the angular speed of the robot around the z-axis can be calculated. As the camera is event-based, it is asynchronous, i.e. reporting events when they occur at unequally spaced time steps. The sensor resolution is 128x128 and the maximum event rate that the sensor can output is around 100k events/s, while the time resolution of the timestamps associated with the events being 1µs.

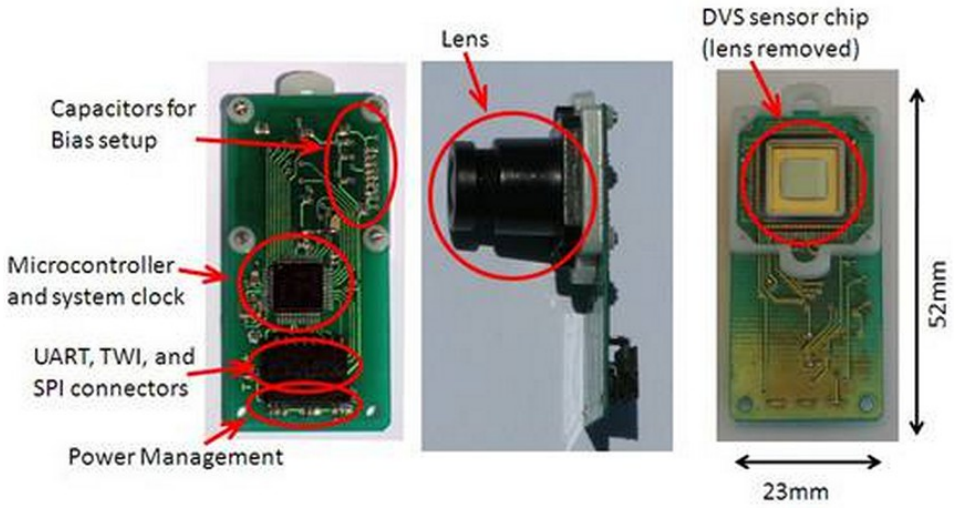

Fig. 2: The eDVS sensor (taken from [eDVS]).

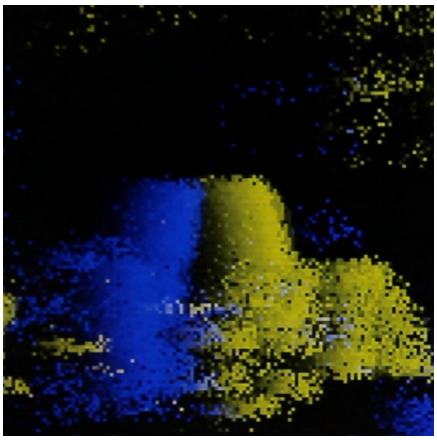

Fig. 3: A sample snapshot of the eDVS events captured from my application.

2.2 The gyroscope

The gyroscope serves as a vestibular sensor and outputs the angular speed around every Cartesian axis in degrees per second. As the movement of the robot is constraint to a yaw movement, only the z-axis is evaluated. The sampling time of the sensor can be adjusted from 1 to 250 Hz; in this coursework it was fixed at 10 Hz. In order to feed the data into the neural model, the raw samples from the sensor must be integrated.

3 The neuronal normalization model of multisensory integration

Ohshiro, Angelaki and DeAngelis developed a model of multisensory integration that can be found in the area MSTd in macaque apes [NormMod]. There were developed two versions of the model, one giving a global divisive normalization representation of the spatial model (as the neurons combine their inputs while the reliability varies over space) and the normalization visual-vestibular integration model. The latter is an extension of the first and is used in the developed application.

3.1 The generic spatial model

Assuming two distinct modalities eliciting an output for the same external event (i.e. reacting to the same external event, e.g. movement). Every modality is covered by unisensory neurons with overlapping receptive fields, i.e. they produce an output if they sense an input (e.g. a move in one direction) in their respective receptive field (a nonlinear (e.g. Gaussian) activity region in space). See fig. 4.

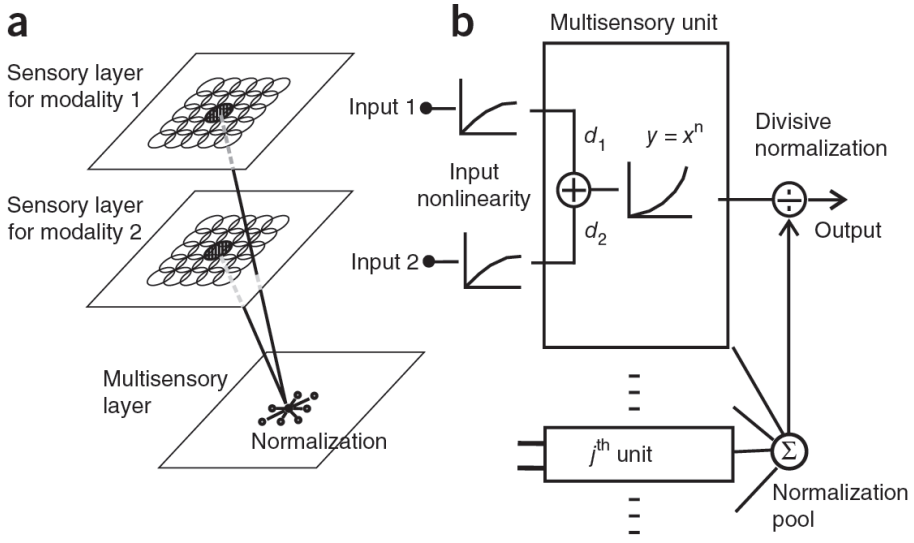

Fig. 4: The spatial normalization model (taken from [NormMod])

The output of every pair of neurons from both modalities whose respective receptive fields overlap serves as an input to a multisensory neuron. Every multisensory neuron produces a weighted sum of the two inputs and applies a nonlinearity onto it (and therefore mimics the membrane dynamics of neurons). The output of every multisensory neuron gets normalized by total net population activity, i.e. the average output of every other multisensory neuron.

The receptive field of a unisensory neuron can be modelled as a two-dimensional Gaussian function

$$G(\hat{\Theta}, \Theta) = \exp \frac{-(\hat{\Theta} - \Theta)^2}{2\sigma^2} \tag{1}$$

where $\hat{\Theta} = (x_{\hat{\Theta}}, y_{\hat{\Theta}})$ represents the center location of the receptive field and σ is the standard deviation. The output of each unisensory neuron can be described as

$$h(c \cdot G(\hat{\Theta}, \Theta)) \tag{2}$$

where c is the stimulus intensity and h a sublinearly increasing function like the square root function. It models the response saturation of the unisensory neurons. The output E of a multisensory neuron before normalization can be described as

$$E = d_1 \cdot h(c_1 \cdot G_1(\hat{\Theta}_1, \Theta_1)) + d_2 \cdot h(c_2 \cdot G_2(\hat{\Theta}_2, \Theta_2)) \tag{3}$$

where the index denote the respective modality. $d_{1,2}$ are the dominance weights that range from 0 to 1 and describe the sensitivity of each multisensory neuron for inputs of a certain modality. Every multisensory neuron has a different combination of those dominance weights. The normalized output of the *ith* multisensory neuron can be represented as

$$R_i = \frac{E_i^n}{\alpha^n + (\frac{1}{N}) \cdot \sum_{j=1}^{N} E_j^n} \tag{4}$$

where α is a semi-saturation constant and N the total number of multisensory neurons. The semi-saturation constant models the neuron's overall sensitivity to stimulus intensity. n is the exponent of an expansive power-law output nonlinearity that simulates the transformation from membrane potential to firing rate. Next a description of the model properties is given. As this is a model for multisensory integration there are some principles that are unanimously accepted as describing the multisensory processes. This model exhibits these features which are also important when analysing and tuning the implementation.

3.2 Biological effects that can be represented with this model

3.2.1 Inverse effectiveness and super-additivity

It has been found in [NormMod] that the output of a multisensory neuron that combines two inputs from two unisensory neurons in area MSTd is stronger when the inputs are weak, and weaker when the inputs are strong. Often the combination of two weak inputs is greater then the sum of both, which is referred to as “super-additivity”, whereas the combination of two strong inputs leads to an output less then the sum of the inputs (“sub-additivity”).

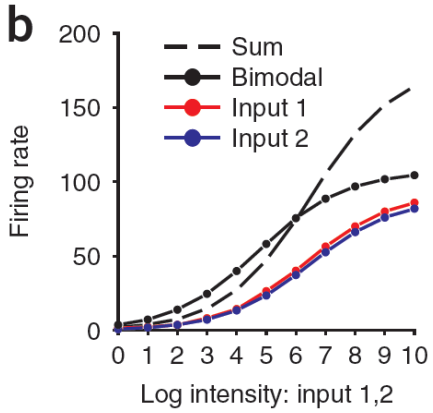

Fig. 5: Super- and sub-additivity (taken from [NormMod])

Fig. 5 shows the output, i.e. the firing rate, of two unisensory neurons as a function of their input stimulus intensity as well as the combined output according to the model, i.e. the output of the multisensory neuron that combines the two ("bimodal"). Additionally, the simple sum of the outputs of the two unisensory neurons is plotted. It can be seen that the bimodal output exceeds the sum for low inputs but falls below it for strong inputs. In the model, super-additivity is described in the expansive power-law nonlinearity with the exponent n in equation (4), whereas sub-additivity is expressed through the divisive normalization in the same equation.

3.2.2 Multisensory suppression

The divisive normalization also accounts for the the effect of multisensory suppression, i.e. that the output of one neuron can suppress the bimodal output of the multisensory neuron. One scenario where this principle can be seen is shown in Fig 6.

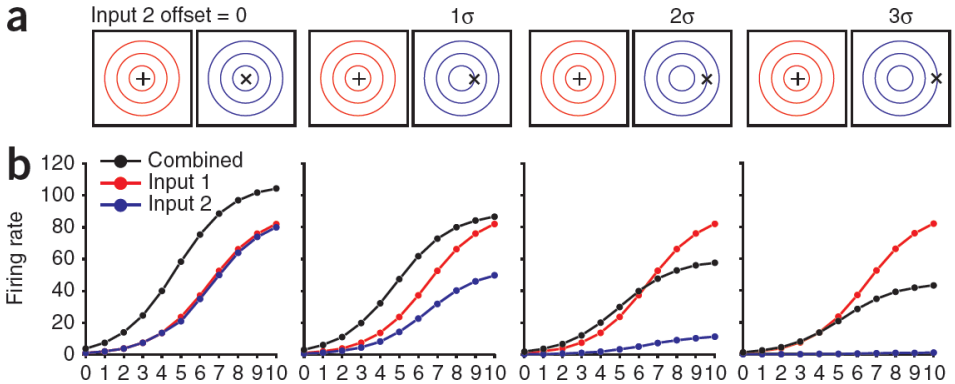

Fig. 6: The red and blue circles represent the respective fields of the two modalities. The location of the input signal is denoted with a "+" and "x". Below the firing rate is plotted against the logarithmic stimulus intensity (taken from [NormMod])

In the first case the inputs two both neurons were located at the centers of their receptive fields. As the input of the second neuron drifts away from the center, the output of the

second neuron becomes weaker, also suppressing the combined, bimodal output. This is due to the fact that the second neuron contributes little to the underlying linear response of the multisensory neuron, but contributes strongly to the normalization pool, as its not-centered input will probably lie near to the center of the receptive field of a different neuron that does produce a strong output.

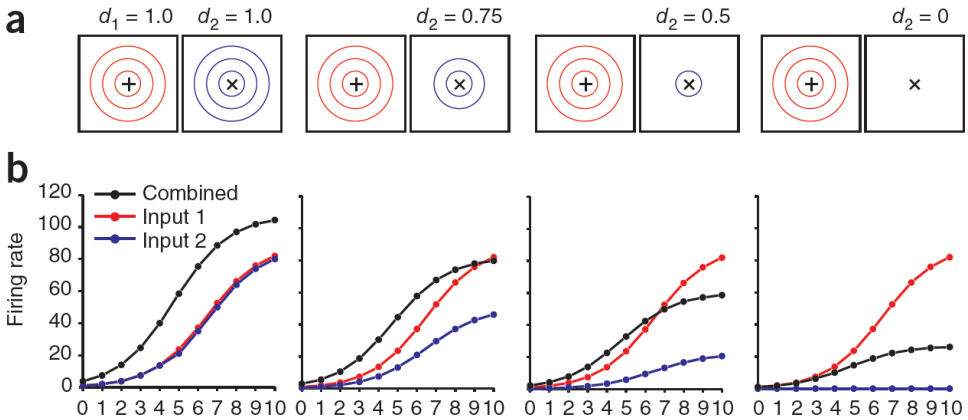

Fig. 7: The size of the red and blue circles represent the respective dominance weights of the two modalities. Below the firing rate is plotted against the logarithmic stimulus intensity (taken from [NormMod])

Another case where multisensory suppression occurs is when regarding different combinations of the input dominance weights $d_{1,2}$. Fig 7 shows again the respective outputs of two unisensory neurons as well as the combined output for different dominance weight sets. As the dominance weight for modality 2 decreases, the output of neuron 2 decreases as well and suppresses also the combined output.

3.2.3 Within-modal suppression

Suppression also occurs within the same modality. Imagine two stimuli at different places in the receptive field of a unisensory neuron. The greater the offset of one of the stimuli from the center of the receptive field, the less the output of the neuron in that case and the more it suppresses the overall output (see fig. 8). The within-modal effect is modelled through the sublinearity $h()$ in equation (3).

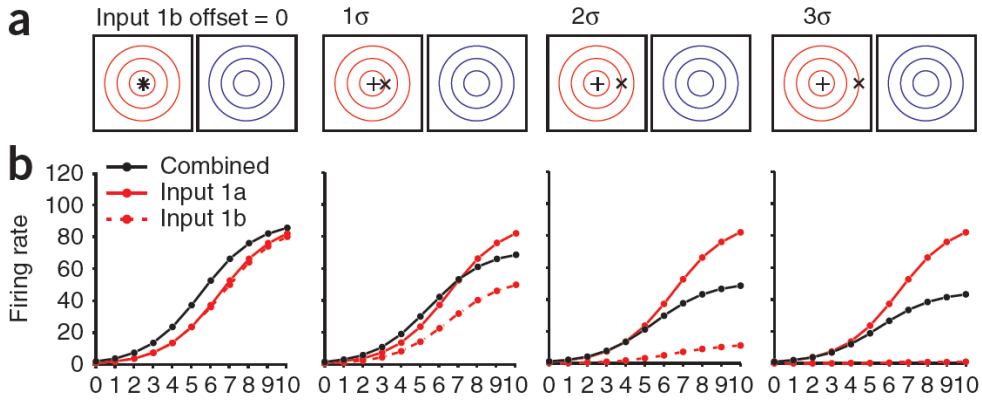

Fig. 8: Two inputs ("x" and "+") occur within the same modality. Below the firing rate is plotted against the logarithmic stimulus intensity (taken from [NormMod]).

Next, a formal description of the visual-vestibular integration model is given. This model follows the same principle of the divisive normalization but it tuned towards fusing visual and vestibular information to get a more precise self-motion detection.

3.3 The visual-vestibular heading estimation model

The authors of [NormMod] found the dominance weights changing for the cue reliability when using this model for the heading estimation through visual and vestibular modalities. As heading in general is a circular variable in three dimensions, overlapping Gaussian receptive fields were not appropriate for heading estimation. Therefore they changed the model and introduced heading preferences of neurons instead of receptive fields and coherences of the inputs.
The output of each unisensory neuron is now described as

$$c \cdot \frac{1}{100} \frac{1+\cos(\Phi)}{2} + \xi \cdot \frac{100-c}{100} \tag{5}$$

where c denotes the coherence of the input and takes values from 0 to 100. ξ ranges from 0 to 0.5 (typically 0.1) and models the fact that the total net population activity increases with the coherence. Φ is the difference between the heading preference of a unisensory neuron and the actual heading that this neuron senses. It can be calculated in three-dimensional space as follows:

$$\Phi = \arccos\left(\begin{pmatrix} \cos(\hat{\theta}) \cdot \cos(\hat{\phi}) \\ \cos(\hat{\theta}) \cdot \sin(\hat{\phi}) \\ \sin(\hat{\theta}) \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) \cdot \cos(\phi) \\ \cos(\theta) \cdot \sin(\phi) \\ \sin(\theta) \end{pmatrix} \right) \tag{6}$$

where θ represents the elevation angle of the preferred heading direction and ϕ represents the azimuth angle of the preferred heading direction. θ, ϕ represent

elevation and azimuth of the actual heading. In our scenario, where the movement is constrained to yawing (no elevation), equation (6) simplifies to

$$\Phi = \arccos(\hat{\phi} - \phi) \tag{7}$$

Each multisensory neuron now produces the following output (before normalization):

$$\begin{aligned} E = d_{vest} \cdot \left(c_{vest} \cdot \frac{1}{100} \frac{1 + \cos(\Phi_{vest})}{2} + \xi \cdot \frac{100 - c_{vest}}{100} \right) \\ + d_{vis} \cdot \left(c_{vis} \cdot \frac{1}{100} \frac{1 + \cos(\Phi_{vis})}{2} + \xi \cdot \frac{100 - c_{vis}}{100} \right) \end{aligned} \tag{8}$$

which then gets normalized as in the spatial model

$$R_i = \frac{E_i^n}{\alpha^n + \left(\frac{1}{N}\right) \cdot \sum_{j=1}^{N} E_j^n} \tag{9}$$

where R_i is the normalized output of the *ith* multisensory neuron.

Ohshiro and his team, who developed the above model, found this model to approximate the real data of the measured multisensory integration mechanisms that are present in the area MSTd in macaque apes very well. Next the implementation details are given in order to validate the viability of the described model and analyse the emerging features of this neuronal computational model.

4 Implementation

4.1 Computing the model output

The paper from Ohshiro et al. only describes how the output of each multisensory neuron is calculated, however it does not mention how the final heading estimate can be derived. I decided to let the final heading estimation be the heading preference of the neuron that produces the greatest output. In the case where the heading preferences of the vestibular and visual modality do not match, an average direction weighted with the dominance weights is calculated:

$$\bar{\hat{\phi}} = \frac{d_{vest} \cdot \hat{\phi}_{vest} + d_{vis} \cdot \hat{\phi}_{vis}}{d_{vest} + d_{vis}} \tag{10}$$

In order to feed in data into the neural model we have to add a pre-processing stage over the raw data from the sensors, to compute an estimate of each unisensory heading estimate. Next, the pre-processing algorithms, which were useful to align the data from the two sensors, are given.

4.2 Retrieving the angular speed from eDVS events

Before the normalization model can be applied, the events of the eDVS sensor have to be converted into an angular speed. This is done by computing the optic flow as described in [OpticFlow] and from that by computing egomotion as described in [AngleOptic].

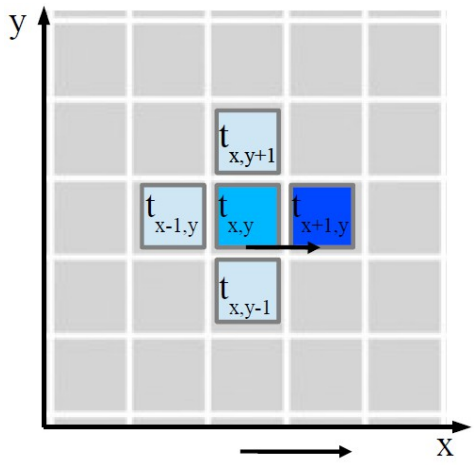

Fig. 9: The optic flow calculation

As the vision sensor eDVS outputs events, we need to design the optic flow algorithm at the pixel level in terms of timing and spatial vicinity of the state changing pixels. In the figure a simple description of the mechanism to compute the (optiv flow) velocity vector is given.
Furthermore, the velocity vector $\vec{v}$ can be computed as follows:

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} \frac{a_{pixel}}{(t_{x,y} - t_{x-1,y})} - \frac{a_{pixel}}{(t_{x,y} - t_{x+1,y})} \\ \frac{a_{pixel}}{(t_{x,y} - t_{x,y-1})} - \frac{a_{pixel}}{(t_{x,y} - t_{x,y+1})} \end{pmatrix} \tag{11}$$

where $t_{x,y}$ denote the point in time when there is an event at location (x,y). a_{pixel} denote the aperture per pixel and is calculated as

$$a_{pixel} = \frac{aperture\,angle\,of\,camera}{number\,of\,pixels} = \frac{70°}{128} \tag{12}$$

as the used vision sensor has 128 x 128 pixels. All single velocity vectors for every event-generating pixel are then summed to form a global velocity vector. As in this scenario only yaw movement is considered, only the velocity in x-direction gets evaluated.

I used an existing implementation of the optic flow algorithm that has been developed by Prof. J. Conradt in [OpticFlow] that I extended by a coherence calculation. The coherence reflects how reliable the visual cue is and serves as an input for the model, although in the paper of Ohshiro et al. it is not mentioned how the visual or vestibular coherence should be calculated. I decided to calculate it as follows:

$$c_{vis} = \left| \frac{n_{vel,r} - n_{vel,l}}{n_{vel,r} + n_{vel,l}} \right| \cdot 100 \tag{13}$$

where $n_{vel,r}$ denote the total number of velocities in the right direction and $n_{vel,l}$ denote the total number of velocities to the left. If the robot spins in a perfect stationary surrounding, there will be only velocities in one direction, so that c_{vis} becomes 100. If the surrounding is not exactly stationary as there a small moving objects in the other direction present, the visual coherence decreases down to 0 at most.
The information from the eDVS sensor is affected by noise in both stationary and dynamic scenarios, i.e. pixels that report events although nothing is moving. Those events are randomly distributed and rarely occur at adjacent pixel positions, so that no proper velocity calculation can be carried out. Occasionally it can happen that two adjacent pixels report events, so that a velocity in one direction can be calculated, whereas the number of velocities in the other direction remains zero. By definition the coherence then turns out to be 1, but as this is due to noise, I added a threshold number of velocities that have to be present before the coherence takes other values than zero. I used this criteria not only as an extension of the coherence calculation for the model, but also as a noise removal filter for the raw angular velocity data from the eDVS sensor, as my application also displays the single, not fused yaw estimation of each sensor separately for comparison.

In order to get the angular velocity from the velocity vector of the optic flow, the position and heading of the eDVS sensor have to be considered (see fig. 10).

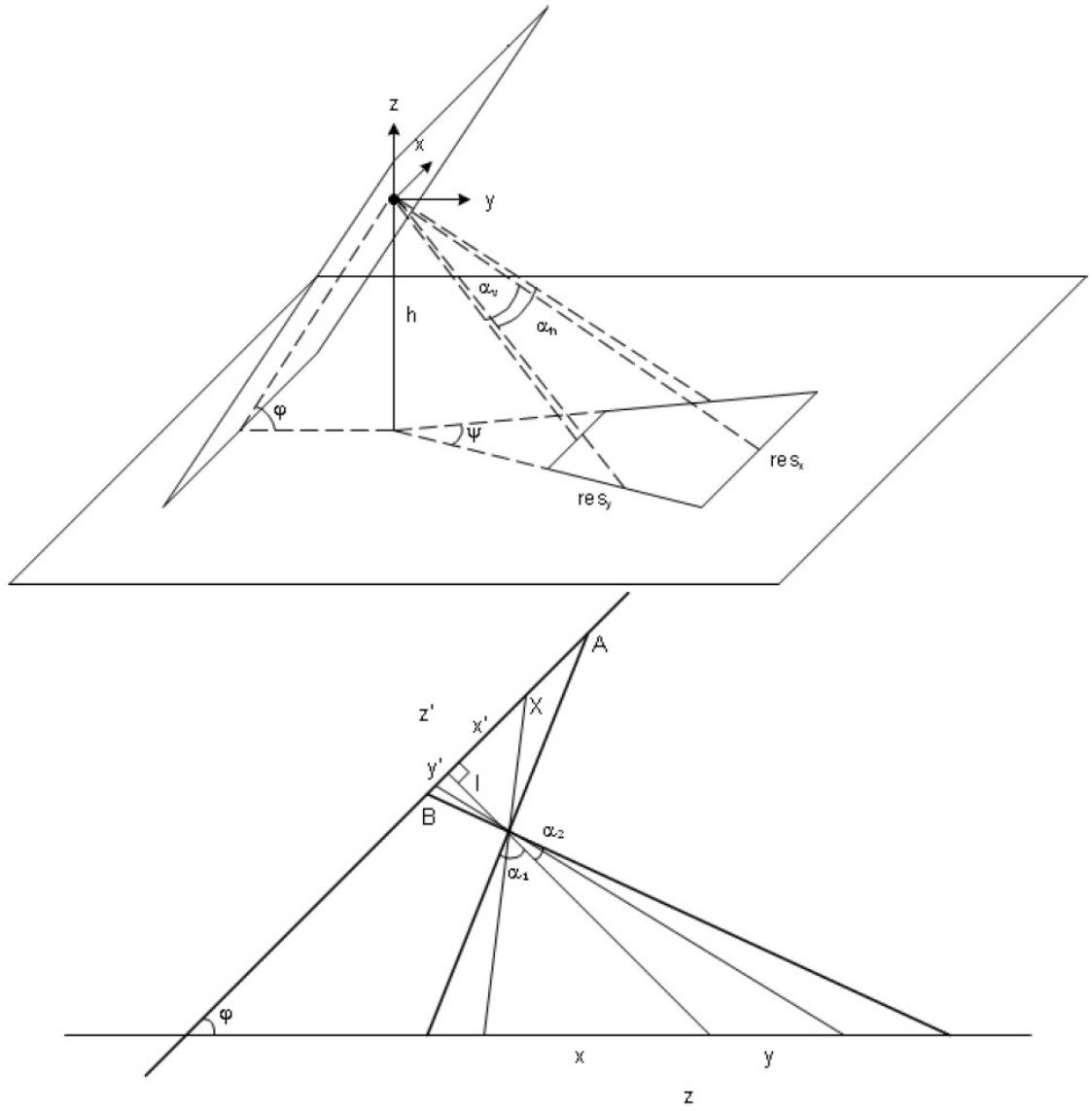

Fig. 10: Estimation of egomotion from optic flow. The camera with the aperture angle α is tilted to the ground with the angle ϕ (taken from [AngleOptic]).

As described in [AngleOptic], the rotation angular speed Ω can be computed as follows:

$$\Omega = \frac{res_x \cdot v_x}{\psi} \tag{14}$$

where ψ is calculated as

$$\psi = 2 \cdot \arctan\left(\sin(\phi) \cdot \tan\left(\frac{\alpha_h}{2}\right)\right) \tag{15}$$

These formulas do not consider the fact that in our case the sensor is not placed in the middle of the robot, so equation (14) must be modified to

$$\Omega = \frac{res_x \cdot v_x \cdot \frac{1}{r}}{\psi} \tag{16}$$

where r denotes the offset from the middle of the robot.
In order to get the angle of the angular speed, an integration stage must be carried out, which is done using the trapezoidal method of integration. Each time the application receives an event, it is assigned a timestamp. The difference of two consecutive timestamps form the timestep needed for integration.

4.3 Processing the output of the gyroscope

The gyroscope already outputs the angular speed in degrees per second and the noise level is not that high although the values are drifiting over time. As the model contains also a vestibular coherence, I regard it as the consistency of the angular speed. The coherence should be 100 if the angular speed remains constant or increases linearly. If the steepness of the velocity increase or decrease changes compared to the one before, the coherence will go down.

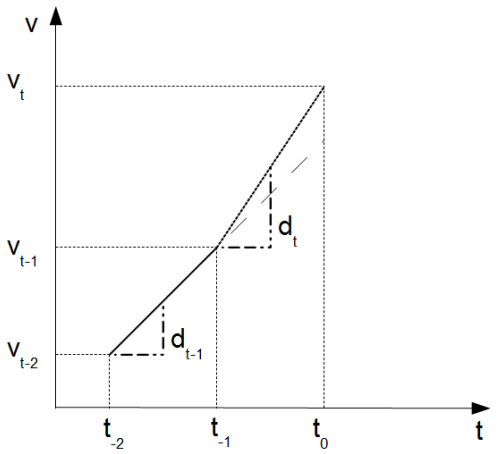

Fig. 11: Change of velocity over time. t_0 represents the current timestamp, t_{-1} the one before.

Considering fig. 11. The coherence is calculated as follows:

$$c_{vest} = \left(1 - \frac{|d_t - d_{t-1}|}{d_{max}}\right) \cdot 100 \tag{17}$$

with d_{max} being the maximum velocity increase that is possible from the point t_{-1} before the maximum velocity the robot can spin with is reached.
As with the data from the eDVS sensor, the angular speed gets integrated to yield the absolute angle.

4.4 Communication with eDVS sensor and robot

My application communicates with the robot and the eDVS sensor via wireless LAN using the TCP/IP protocol. The robot thereby acts as the server and my application as the client. The data from the gyroscope and the eDVS sensor is streamed through the same WiFi module, so they share the same IP adress, but have different ports.

4.5 The Graphical User Interface

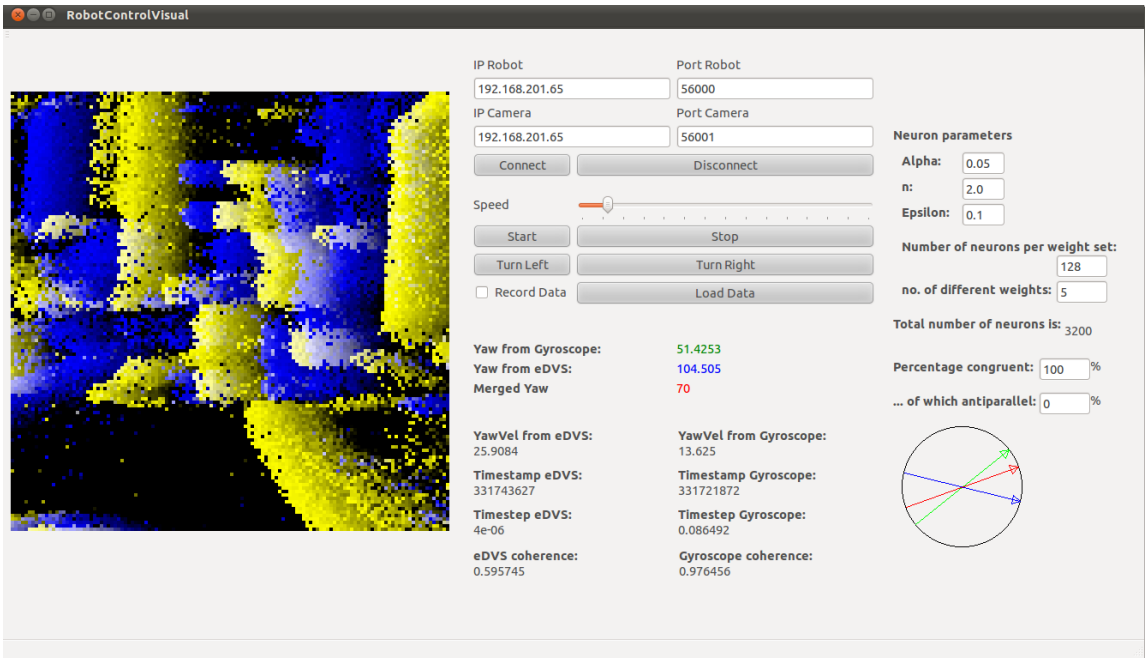

Fig. 12: The graphical user interface.

Fig. 12 shows the GUI of my application. In the left panel the events of the eDVS sensor get visualized, where on- and off-events are assigned different colours. The middle part contains buttons to connect to the robot, to start the sensors and to steer the robot. The received data can also be written to a file. Below the buttons, the angular velocity that each sensor outputs, the coherences of the sensors, the timestamps and -steps as well as the integrated angle of each sensor, i.e. the heading estimates of each sensor on its own, are displayed. Furthermore, the fused heading estimate obtained by the model is displayed. Those three heading estimates are also visualized in the lower right corner of the application for better comparison. On the right side of the application the user can adjust the parameters of the model. These include the semi-saturation constant α , the exponent of the expansive power-law nonlinearity n and ξ (see formula (9). Furthermore, the total number of neurons can be set, the percentage of which have parallel heading preferences (meaning visual and vestibular heading preferences point in the same direction), the percentage of neurons whose heading preferences point in

opposite directions (antiparallel neurons), the percentage of neurons whose heading preferences point in arbitrary directions and the number of different weight sets (see formula (8)). A weight set number of 5 means that both d_{vest} and d_{vis} can take the values [0; 0.25; 0.5; 0.75; 1.0].

4.6 Sampling time and event processing

The eDVS sensor does not have a fixed data transmission rate since it is event-based. On average, it sends events at the order of 1kevents/second. Everytime the application receives an eDVS event in its TCP/IP buffer, it visualizes it in the left panel and computes the integrated angle. The robot is adjusted to send strings containing the angular speed of the gyroscope at a sampling rate of 10 Hz. Only when the application receives a packet from the gyroscope, the output of the normalization model is calculated. This measure ensures that the visual and vestibular inputs are synchronous in time, which is important for the model to work. These considerations taken into account when processing the data from the sensors ensure that the empirical principles of multisensory integration are fulfilled in terms of spatial and temporal congruency of the sensors. This means that in order to properly integrate the two measurements we need to have the sensors closely positioned in space and the data from each sensor must be synchronized in order to mark the processing of the same movement.

5 Results

I've tested the model with the neuronal parameters that Ohshrio et al. proposed in their paper and used 128 neurons that had a parallel preferences tuning with 5 different weight sets, so that the model consisted of 3200 neurons in total.

The eDVS sensor data is noisier than the gyroscope data, resulting in a decreased visual coherence compared to the gyroscope. Therefore the combined yaw is biased towards the yaw computed from the gyroscope. Probably due to communication and data acquisition latencies, the eDVS sensor can only detect angular speeds up to a certain limit, above which it outputs a very low angular speed that cannot be plausible. As in this case the visual coherence also decreases, the model is robust against it. When the robot is standing still and the coherence of both modalities equals zero, the combined output is simply the mean of the two. The more multisensory neurons are created, the higher is the output resolution, as there exist more neurons whose heading directions cover the whole 360° direction space.

The application was executed on a Ubuntu Linux operating system in a virtual machine that limited the processing power of the underlying hardware significantly (only 1 out of 8 cores were detected). Still the model comprising 3200 neurons could be executed in real-time. Apart from the noise removal in the eDVS data when standing still, I did not have to know any special characteristics of the used sensors, which makes the model easily adaptable to other sensors.

Summarizing, this biologically inspired model of divisive normalization is a good alternative to existing sensor fusion models because of its simplicity and because it invokes a functional operation that has been observed in the cortical function. This model of divisive normalization was also able to account for the empirical principles of multisensory integration and the effects of cue reliability on multi-modal integration.

Bibliography

[eDVS]: 2012, https://wiki.lsr.ei.tum.de/nst/programming/edvsgettingstarted
[HBoR]: B. Siciliano, O. Khatib, Handbook of Robotics,Multisensor Data Fusion, Springer, 2008
[AngleOptic]: Angle from optic flow, paper from NST
[OpticFlow]: J. Conradt, Optic Flow from Miniature Event-Based Vision Sensors, IEEE, 2012
[NomniBot]: C. Hubmann, Nomni Bot User Manual, TUM NST, 2012
[NormMod]: T. Ohshiro, D. E. Angelaki, G. C. DeAngelis, A normalization model of multisensory integration, Nature America, 2011