Tracking objects and wing beat analysis methods of a fruit fly with the event-based silicon retina

Semesterarbeit

Janick Cardinale
Supervising assistant: Patrick Lichtsteiner
Supervisor: Tobi Delbrück

0 Introduction

This work is separated into two parts. The first part is about tracking objects with a Kalman filter(KF) on the top of a existing clustering algorithm which tracks moving objects. The second part is about tracking wing edges of a fruit fly(drosophila melangulaster) and doing some analysis like reading out amplitude of the wing beat and its frequency.

The framework around this two applications is the same: With the silicon retina objects are recorded. The chip in the camera produces events. An event is a loss or a rise in contrast with a time stamp in the order of microseconds. Through a USB2 interface events are sent to a Java framework where events can be processed. Detailed information about the transient silicon retina can be found in [2]. Movies can of course also be stored and can be played on the screen. The Java framework supports a filter chain where a new filter (in form of a class) can be registered. The framework sends then the data (the amount of data is depending on play-back speed) and informs about special situations like a reset order by the user. A filter can process the data(like filter out noise) and send it ahead the filter chain.

This work is about two new filters for this framework. This documentation about the filters do not contain Java specific implementation aspects. The theory of the used mathematical methods like the Kalman filter isn't covered, but the documentation tries to describe their functioning and practical usage by example. Furthermore this documentation describes what algorithms and procedures were implemented in the Java framework but it's not a manual. All implemented options and parameters are shortly described in the appendix.

Part I

Kalman Filter for Cluster Tracker

1 Introduction

The general goal of this work is to track objects which are recorded with the silicon retina. A very nice, fast and clever way to do this is clustering. The silicon retina detects edges of moving objects in hardware. A very good and simple application of this is to search for clusters and thus tracking objects. Such a "Cluster Tracker" existed already but should be shortly explained here.

With a priori knowledge namely how big the objects we search are, the tracker creates a cluster with the appropriate size where a event arises (if there is no other cluster at this place). If an event arises in an existing cluster, the center of this cluster is moved a little bit in this direction, according to the vector combination:

$$\vec{x}_k = (1 - m)\vec{x}_{k-1} + m\vec{x}_e \tag{1}$$

Where $\vec{x}$ is the position vector of the cluster, $\vec{x}$ is the position of the event and m is a weighting factor that determines how much new events updates the position of the cluster. If a cluster doesn't have enough "support", which means that there are not enough events within this cluster, it dies and make place for a new one, since the number of clusters is limited and belongs to the a priori knowledge. If two clusters are too near to each other, they can collapse to one cluster. A nice application would be to not collapse clusters if they are at the same place but have different velocities.

Based on such a clustering of events the goal is to track objects with a linear Kalman filter(for a detailed explanation see [3]). The clusters then serves as measurements. It is almost impossible to use the Kalman filter directly on the events since we don't know to which object the event belongs to, so the Kalman filter is built on top of the cluster tracker which means, the position of a cluster serves as measurements.

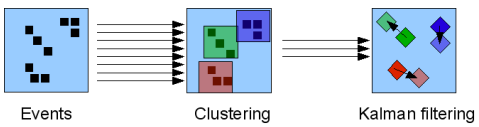

Figure 1: Schematic view of the event processing through the cluster tracker and the kalman filter

2 The Algorithm

For the clusters we would like to filter we assume that they do a not accelerated movement. Acceleration is treated as noise in our model. We would like to track the positions and the velocities so our state vector for one cluster is $\vec{x}_k = (y \quad \dot{y} \quad x \quad \dot{x})^T$ at a certain time k. This depends of course on the previous state at time k-1, so we can define a transition between the state at time $k-1$ and time k as:

$$\hat{x}_{k|k-1} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \Delta t \end{pmatrix}}_{A} \hat{x}_{k-1} \tag{2}$$

where A is called the transition matrix and $\hat{x}_{k|k-1}$ is the prediction just done on apriori knowledge and the last state $\hat{x}_{k-1}$. In this formula we assume an acceleration of the object of 0. The true state would be:

$$x_k = Ax_{k-1} + W \tag{3}$$

where W corresponds to noise and of course this $W \sim \mathcal{N}(0, Q)$ can be dependent in time. So we get in a stochastic process. Of course we don't know the noise, but the Kalman filter uses just it's covariance matrix Q to do an optimal estimate of the true position, optimal in terms of the least squares of the estimated error. So our filter makes a prediction of the new position with knowledge of the old position and the last state according to the tranistion formula (2). From there on it compares this prediction with the new measurement, which is in our case just the x and y position of the cluster. To calculate the residual we need a linear map between the predicted state vector and the measurement:

$$\tilde{y}_k = z_k - \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{H_k} \hat{x}_{k|k-1} \tag{4}$$

with $\tilde{y}_k$ the residual, $z_k (\in \mathbb{R}^2)$ the measurement and H_k the observation model which maps x onto z. We assume that our measurement is noisy with a noise which is Gaussian distributed with mean 0 and variance R. So the correct measurement would be $z_k = H_k x_k + V_k$ where $V_k \sim \mathcal{N}(0, R)$ is the measurement noise.Also here the KF just needs the covariance matrix R. The filter then calculates according to these informations an optimal weighting K between prediction an residual, which is called the Kalman gain and the update of the state vector looks like:

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k \tilde{y}_k \tag{5}$$

The Kalman gain K is the optimal weighting in terms of least squares of the expected error:

$$Min\ \mathbb{E}\left[(x_k - \hat{x}_k)^2\right] \Leftrightarrow \ldots \Leftrightarrow K_k = P_{k|k-1}H_k^T(H_k P_{k|k-1}H_k^T + R_k)^{-1} \quad (6)$$

Which is the same as minimizing the trace of the error covariance matrix $P_k = cov(x_k - \hat{x}_k)$. So elements in the matrix are minimized. (The predicted P is just equal to $cov(x_k - \hat{x}_{k|k-1})$).

2.1 Variances

The entries of Q describe how sure we are about the entries of our state vector. If we assume all our parameters in the state vector as independent, then the matrix Q is diagonal. For example if the element $Q(1,1)$ is increased, then the filter tries to change the first parameter in the state vector more than before (if there is a mismatch between prediction and measurement), in this way we model this parameter as more unsure. In this way we can model which parameters should be adapted faster and which parameters should be more stable. The matrix R, in our case with dimension (2×2), describe how accurate the measurements are. If we know for example that the y-coordinate is for some reason more accurate than the x-position measurement, then we should increase the element on the diagonal which corresponds to the x-position. So for the x-parameter in the state vector the weighting will be more influenced by the prediction than by the measurement.

2.1.1 Dynamic Variances

As mentioned the variance matrices Q and R can change over time. If a cluster is created by the cluster tracker, we don't have some information about the velocity, so we set the velocities for both dimensions to 0. This causes that the filter should adapt very fast its velocity to the movements of the cluster, but only in the beginning, else the filtering effect will be leaved out. And vice versa the measurement variances should be very small in the beginning and grow then till a limit which is set by the user and depends on the application. In our implementation we chose a model as shown in figure (3b) to increase the measurement resp. to decrease the process variance.

Another practical experience made was, that both variances mustn't be too small, else the filter diverges.

3 Results

Figure (2) shows a track of a car and the Kalman filtered version.

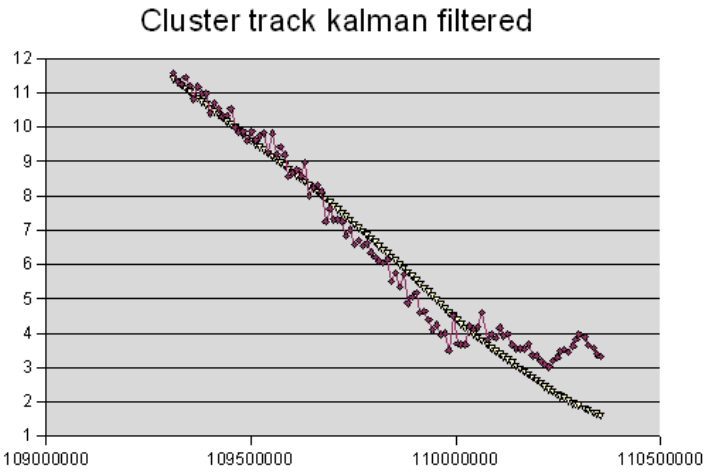

Figure 2: Y-Positions of cluster(red) resp. the measurements and the kalman filter(yellow). The kalman filter track is a lot smoother. Dynamic Variances are used here, see figure (3b). In the end the filter believes more in his prediction than in the measurements.

Figure (3) shows how the Kalman filter minimizes the entries of the matrix P.

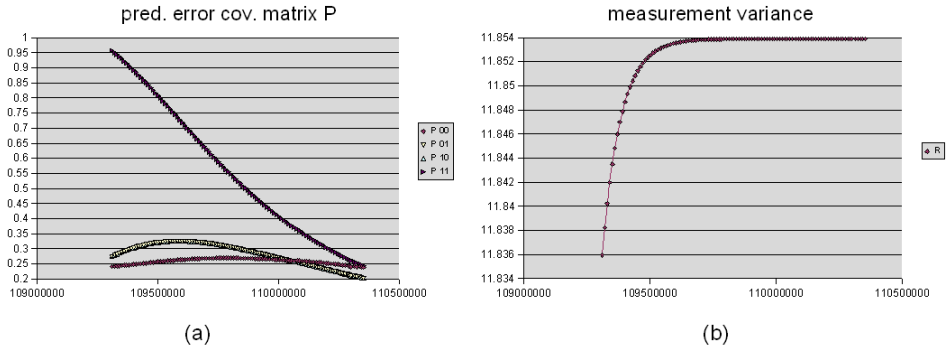

Figure 3: (a) The predicted error covariance entries. The KF tries nicely to minimize these entries. (b) The implementation of dynamic variances. The end value can be set by the user.

Part II
Wing edge tracking

1 Introduction

Goal is to track in a very automatic way the wing edges of a drosophila melangulaster in the described setup. Tracking in this context means to know about the actual position of a wing edge, its amplitude and frequency. There are essentially two problems to solve. The first problem is to find out where the fly is, where the heading is and how big it is. Figure 1 shows how the data look like. In contrast to other visual systems the silicon retina has the advantage, that we don't have to extract edges of objects from an image, we just process events which are, without preprocessing, taken as a measurement. This is a big advantage in terms of computational performance and complexity.

From there on we can process the incoming events in a more meaningful way: all events that are on the left side of the heading, correspond to the left wing. The assumption here is, that there are no events, except they arise from a wing movement. Internally all events are converted to radians where the body of the fly is the center of a circle and the heading corresponds to 0 rad. So the measurement we use is a one dimensional angular position. The second problem is then to make out the wings parameters, like position, frequency and amplitude. For this we first have to choose a model.

The next subsection describes how to solve the initialization problems, after we discuss about tracking models.

2 Determine the proprieties of the fly - The initialization phase

A human can't recognize a moving object if they see on- or off-contrast events sequentially after each other. We have to buffer or store certain events so we can recognize contours and finally classify the object. The initialization algorithm first buffers an empirically found number of events. On this data (which we loose in an online setup) several steps are processed as shown in fig.3. We perform first a 2-means algorithm to classify the events in 2 clusters: right and left wing.

1. **Initialize** 2 prototypes randomly (for each wing w), i.e. choose 2 pixels. We chose the upper right corner and the bottom left corner, so

the prototypes have maximal distance between each other and in this way we hope that there is faster convergance.

2. **Iterate**

   - With the current prototypes, assign all events $\mathcal{E}$ to a prototype. Choose this prototype which is nearest to the event.
   - With the current assignments ($\mathcal{E} \rightarrow w$), move the prototypes p_w to the position where the center of mass of the assigned events ($\mathcal{E}_w$) is.

$$p_w = \frac{1}{n_w} \sum_{i}^{n_w} e_i \text{ with } e_i \in \mathcal{E}_w$$

$$n_w = |\mathcal{E}_w| = \text{number of events assigned to wing w}$$

If the algorithm succeeds we found the center of mass of each wing. In the middle of the two prototypes we expect the body to be. The heading should be orthogonal to the line which connects the two centroids. So this leaves two possibilities for the heading. A criterion for recognizing head and tail is, that the fly does move the wings nearer to the heading line on her back than on the head side. So one could define regions of interest on the head and at the back of the fly and count the events which appear in these regions. It turned out that the initialization or the heading line has to be very correct that this approach works. Another possibility we implemented is to simulate a retarded track with the "vector combination approach"(which is described below) and record maximal and minimal positions of the wing. The left side of the fly is the wing where the angle α between the recorded positions to the heading line is larger.

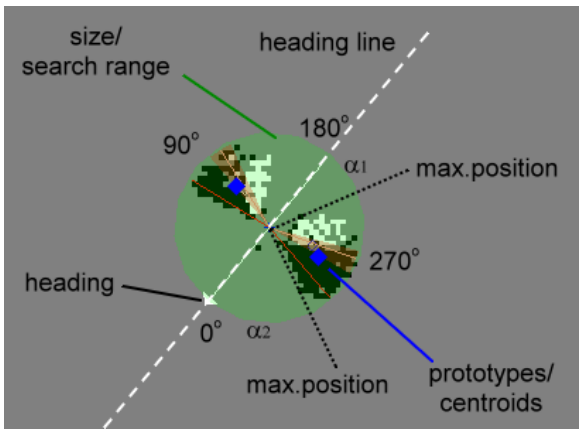

Figure 2: Design of geometry used to track the fly.

For the initialization step it remains to determine the size of the fly. This is an easy task since we found the center of masses of the 2 wings with the 2-means algorithm, the size of the fly is supposed to be twice the distance from the body to one of the centroids. The size is further used to determine a region of interest, all events that arise outside of this region are ignored, so we reduce noise and even ignore disturbing objects(in practice this can be the wire where the fly is fixed on).

After initialization phase the tracker switches the state to one of the following tracker approaches.

3 Wing Edge Tracking

Goal is to find out frequency of wing beat, amplitude and the phase of each wing edge. After initialization we can now classify the different events to a wing edge. The events for left edges are these which have a value(remember the measurements are translated to a 1D angular position in radians) which is smaller than π. All other events correspond to the right wing. Furthermore the "off-polarity events" which correspond to a loss of contrast are supposed to belong to the leading edge(they obstruct the leds). The on-polarity events are classified to belong to the trailing edges. From here on we discuss the tracking only for one edge, we handle all edges in the same way and independently. For both of the following approaches the edges are initialized at the centroid position of the wing.

3.1 Weighted running average(WRA) approach

3.1.1 Model description

A principle we tried to achieve is to use every information we get in the region of interest. In the following approach every information has the same influence on the track. If an event $e \in \mathbb{N}^2$ resp. a new measurement arise for an edge, the actual angular position is then changed as a vector combination:

$$x_k = (1-m)x_{k-1} + mz_k \tag{7}$$

$$z = \begin{cases} \arccos(e_x) - h, & \text{if } e_y > 0 (\text{quadrant 1or2}) \\ 2\pi - (\arccos(e_x) - h) \mod 2\pi, & \text{else.} \end{cases} \tag{8}$$

where x_k is the new tracker position, x_{k-1} the old one and h is the heading angle. m is the weighting factor which depends a lot on how many events happens per wing beat. In our setup m = 0.1 seems to be reasonable. The parameter determine how much a single event affect the actual wing edge position. If the parameter is too high the wing position bounces to the

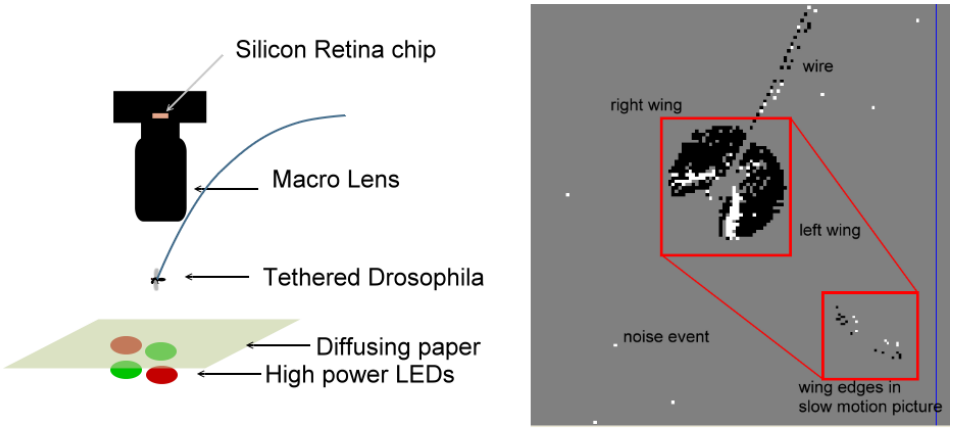

Figure 1: (a) Schematic setup to record data (b) Shows how recorded data looks on the screen. Black pixel correspond to on-events, white pixel to off-events.

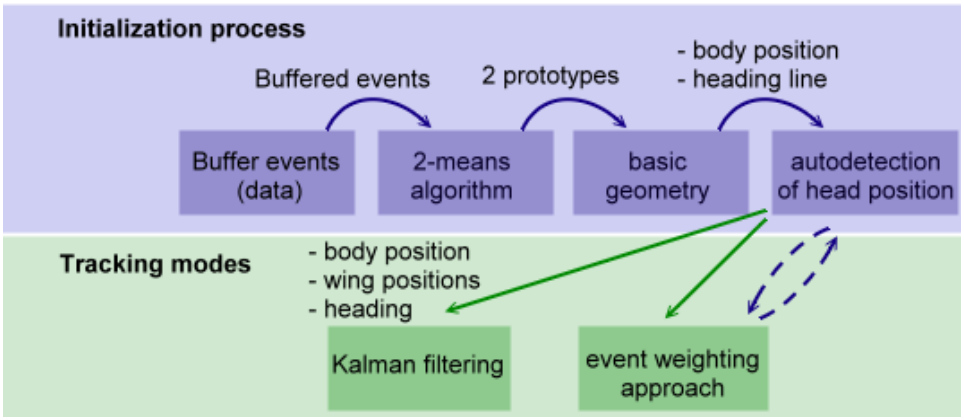

Figure 3: State diagramm of initialization phase. On the transitions we can see input and output parameters of the different states.

events and the filter effect is lost. If m is too small the tracker loses his object.

The amplitude can now be calculated while recording maximal and minimal values of the tracker position. To figure out frequency information we introduced a zero-crossing mechanism with an adaptable hysteresis. Without hysteresis the track has to be very smooth so that the zero can't be crossed multiple times per wing beat. After each beat frequency and amplitude can be updated. In practice we used as crossing detector the position of the centroid. For this the centroids are also updated at every wing beat to the center of the events during the last wing beat. Even if the fly is sticked on a wire it can move a little bit. From there on we can also recalculate the heading and the body position (this updates can be leaved out by option).

3.1.2 Results & Performance

A big advantage of this approach is that there are no big calculations performed per event. So the tracker is very fast and can be easily used for a real time application. Figure (4) and (5) show tracks with good and noisy data. Further results and comparisons can be found in section 3.2.2.

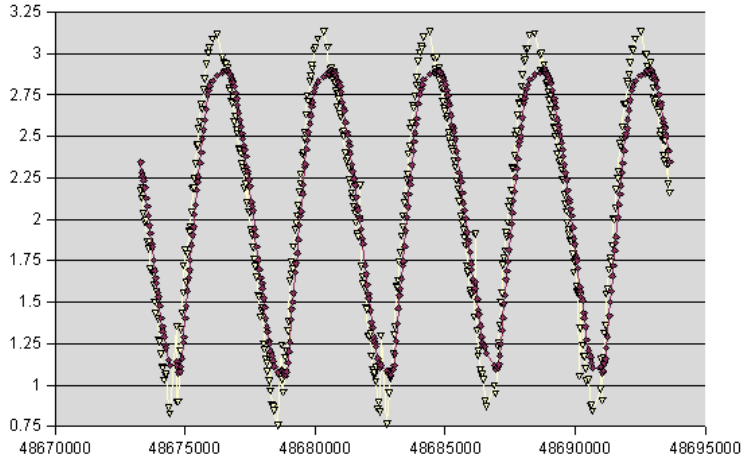

Figure 4: **Red**: tracker position; **Yellow**: measurements. A good track of a wing edge. The tracker isn't able to reach peaks, so amplitude information is always a little too small.

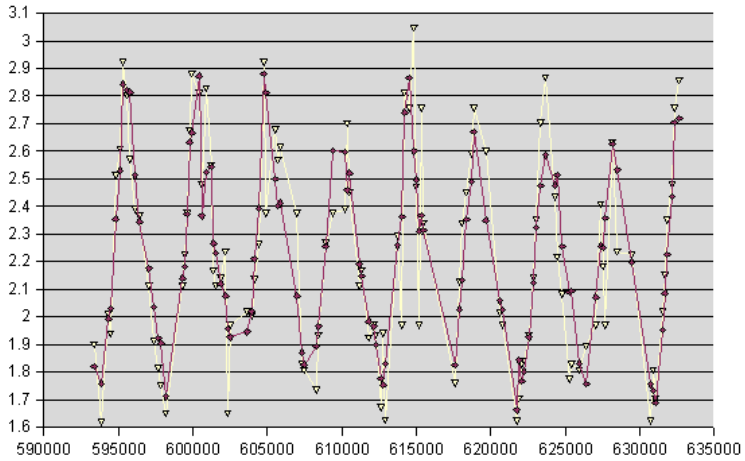

Figure 5: The track still works even with bad data(yellow). The track graph(red) is not that smooth but it tracks correctly. With the hysteresis construct also frequency information is correct.

3.2 Extended Kalman Filter(EKF)

3.2.1 Model description

We use the Kalman filter to include a priori knowledge in the system, so we are able to filter in a more clever way. A Kalman filter is a meaningful approach since the wings perform a very regular and predictable movement. This subsection serves just as a short description of what happens, the whole algorithm can be found in the appendix or in [2].

In our case a priori knowledge means that we would like to track a sinusoidal movement h:

$$h = a + M\sin(\xi), \text{ where } \xi = \omega t \tag{9}$$

where a is the wing edge position, M the amplitude, ξ the phase and ω the angular velocity. A possible enhancement would be to expand the function to a fourier serie (see [1]) to better approximate the wing edge movement function.

So we define our state vector like $x = (a \quad M \quad \xi \quad \omega)^T$. The goal of the EKF is to track the edge while updating this state vector every event so that the output of (9) with the current state vector matches the wing edge. Note that h is a nonlinear map of the state vector onto a one dimensional angular position. We'll use that to compare the measurements with the current state vector.

Prediction Step The filter consists of two steps, a prediction and an update step. In the prediction step no measurement is used, the filter tries to predict the state vector by just it's a priori knowledge and the state vector found at the last step. So the filter uses the knowledge of the past (this is what is called recursive in this context). It assumes that the a priori knowledge resp. our model is not capable to describe the process exactly. So the KF assumes a process-noise which is Gaussian distributed. We can write down the true state as follows:

$$x_k = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{A} x_{k-1} + W_k \tag{10}$$

where $W_k \sim \mathcal{N}(0, Q)$ is the process noise. This is obviously a linear transition with transition matrix A. A updates just the phase according to the time difference of the actual time and the time of the last step. Of course we don't know the noise, but we just need to specify Q for the EKF. With an element on the diagonal of Q we can specify which parameter should have which variance. If a variance of a parameter is small, the Filter adapt faster the other parameters, because it believes more in the parameters with small variances. For example it is favorable if the frequency changes faster than the phase, else the filter will try to compensate an increase of the wing beat frequency of the fly with increasing the phase, instead of the frequency. In the same manner we would like to have a fast variable amplitude and a more stable wing position a. Q (also R) can be found empirically and the magnitude of the elements in Q depend on how many measurements arise per unit, in our case a unit can be a wing beat.

Process noise W in our context represents just the non-regularity of the wing beats or even a movement of the fly. We can see that the prediction .

Update Step In the update step the measurement and the predicted state are compared resp. the residual is calculated and the EKF chooses for the new state(a posteriori) a weighting for this residual which is added to the predicted state vector. This weighting matrix is called the Kalman gain.

Furthermore the filter assumes that the measurements z_k (our events translated in 1D angular positions in radians) are noisy and Gaussian distributed:

$$z_k = h(x_k + V_k), \quad V_k \sim \mathcal{N}(0, R). \tag{11}$$

In our case the measurement variance R is a scalar because the measurement is just one dimensional. With the value of R we can specify how

noisy measurements are or how accurate they are. The residual can thus be formulated:

$$\tilde{y}_k = z_k - h(\hat{x}_{k|k-1}) \tag{12}$$

As mentioned, the map h is nonlinear (in our case the tranistion map with matrix A is linear, the EKF provides actually nonlinear transitions too). This is why we use the extended Kalman filter: to calculate the optimal Kalman gain K_k and error covariances(see below) one needs to linearize h. The matrix H is the Jacobian matrix of h w.r.t. the predicted state vector. The measurement mapping function h is linearized at point $\hat{x}_{k|k-1}$.

$$H = \left.\frac{\partial h}{\partial x}\right|_{x_{k|k-1}} = \begin{pmatrix} \frac{\partial h}{\partial a} & \frac{\partial h}{\partial M} & \frac{\partial h}{\partial \xi} & \frac{\partial h}{\partial \omega} \end{pmatrix} \tag{13}$$

$$= \begin{pmatrix} 1 & a\sin(\xi) & M\cos(\xi) & Mt\cos\xi \end{pmatrix} \tag{14}$$

The update formula is then the following weighting between the predicted state and the residual $\tilde{y}$:

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k\tilde{y}_k \tag{15}$$

Furthermore the filter internally carries an expected error covariance matrix P, which tells about how accurate the single parameters and their covariances are. Actually the Kalman gain is exactly this weighting, which minimizes the trace of this matrix $P_k = cov(x_k - \hat{x}_k)$ which is equally to minimize the least squares of the expected error: $Min\ \mathbb{E}[(x_k - \hat{x}_k)^2]$

Implementation aspects In our case Δt is not constant, so we wait with the prediction till an event arises and update Δt . Δt can be even 0, if two events happen at the same time. One possibility would be to first average these events that happen at the same time. The other one is to leave just the prediction phase out and do the update for each of them which was our choice. This option corresponds not to a prediction with $\Delta t = 0$, this would be a third possibility.

3.2.2 Results and Performance

The filter performs a lot of computation, so it's almost impossible to iterate the filter for every event on a standard CPU. A profiling showed that the matrix operations take about 40% of the CPU time. Another 24% are used to convert an event to a angular position in radians. A possibility to increase performance is to average some events of an edge and then update the filter. This serves just as an accelerator but influences the track dramatically it too much events are averaged. Another idea would be to model a hybrid system between WRA and Kalman approach.

The tracker takes a long time to "find" a wing edge if the initial frequency (set by the user) is not in the range of $\pm 5\%$ or can't even succeed. A big advantage compared to the WRA approach is that the filter doesn't depend too much on the quality and amount of measurements. Results and analysis of a track are shortly discussed on figures (6),(7) and (8). Results of both approaches are shown and compared in figures (9) and (10).

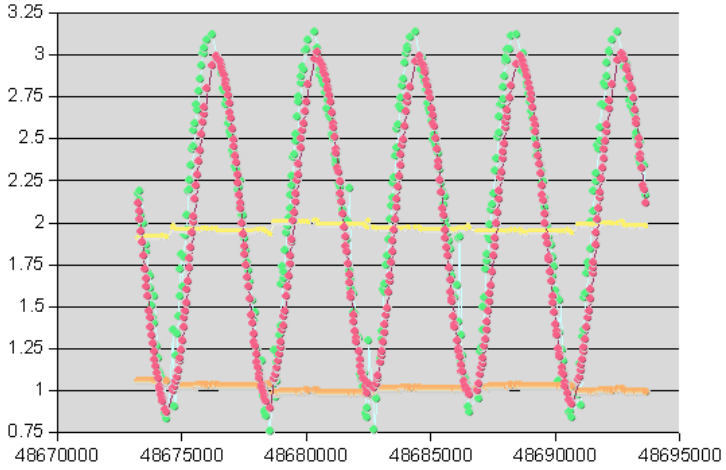

Figure 6: **Pink**: The Kalman track position; **Green**: measurements; **Yellow**: wing position; **Orange**: amplitude. Also here the tracker isn't able to track the edge correctly on the peaks. This is probably because of the too simple model which should be enhanced to a fourier serie.

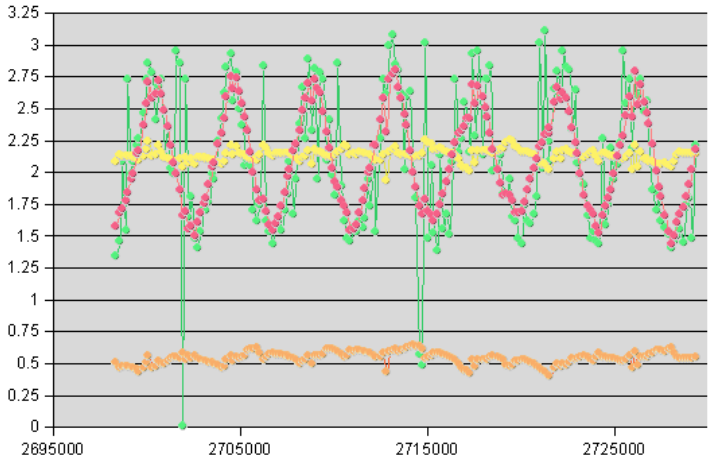

Figure 7: **Pink**: The Kalman track position; **Green**: measurements; **Yellow**: wing position; **Orange**: amplitude. A track of very noisy data. The Kalman filter filters in a very powerful and meaningful way.

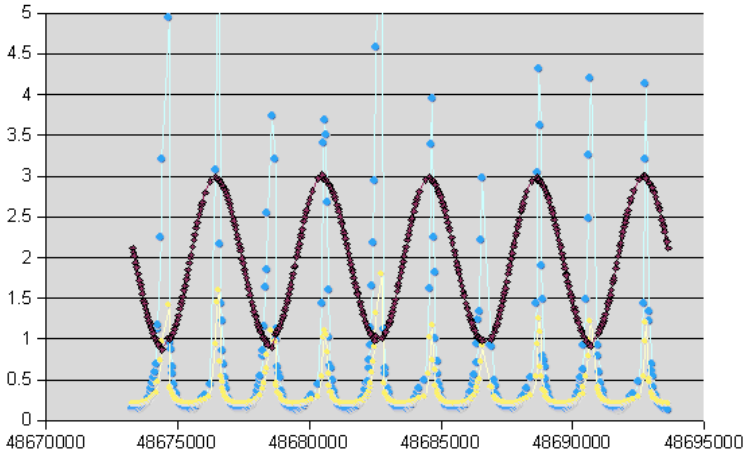

Figure 8: This shows the entries of the error covariance matrix for the phase(yellow) and the frequency(blue) on the same data as in figure 6. One can see that the filter is pretty unsure about his state vectors on the peaks of the track(red). This is because of the too simple sinusoidal model. After a critical phase the entries of the estimated covariance error matrix P are minimized again till the next peak.

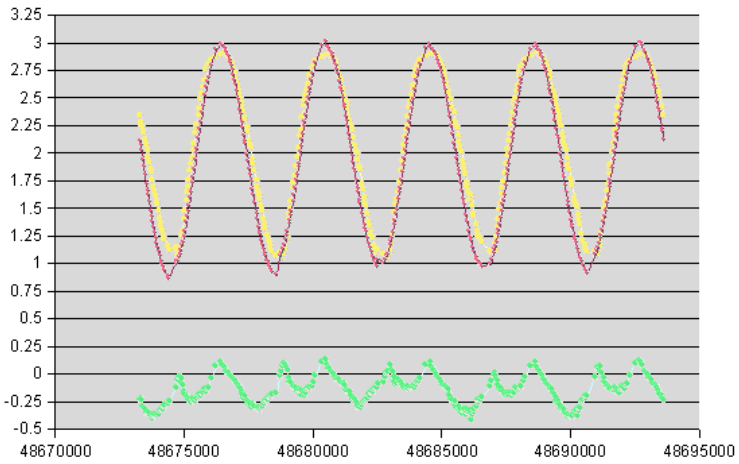

Figure 9: Kalman(red) vs. WRA(yellow) approach and the difference of the track position ($p_{Kalman} - p_{WRA}$)(green) where the same data as in fig.(6) and (8) was used. Note that it is impossible that the WRA approach can be exactly on the wing edge position or ahead of the real wing position. Not the Kalman track. This is also the reason why the amplitude matches better on the peaks for the Kalman track.

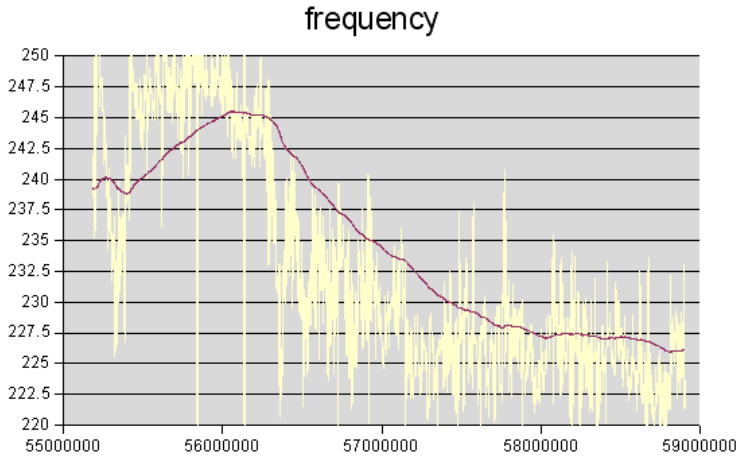

Figure 10: The frequency output of the WRA approach(yellow) and of the EKF(red). The fly performs a maneuver and increases its wing beat frequency. The WRA result is very jittered. This is because of the the low spatial resolution of the retina: The hysteresis is modeled as a floating point variable, an event just as an integer. The integer is then transformed to an angular position w.r.t. to the flies heading. So it is a discrete(dependent on the pixel grid) entity which decides if the hysteresis boundary is crossed.

Appendix

A Description of the Cluster Tracker Kalman Filter Options

feedback to cluster With this option on, the position of a cluster is set to the actual state vector position after each update step of the Kalman filter. The goal was that the cluster searches for events just around the state vector, a side-effect is that the clusters die because they don't have support if the KF estimate is not at a similar position as the cluster.

use dynamic variances With this option on, the measurement variance increases to the parameter value `maxMeasurementVariance` as described in 2.1.1 in part 1. In a similar way the process variance decreases from 10× minProcessVariance to `minProcessVariance`. If the option is not selected the variances used are just constant and equal to `maxMeasurementVariance` and `minProcessVariance`. Note that the current variance value is set to all diagonal elements of Q resp. R.

number of events till track After at least this number of events the Kalman Filters for all the clusters are launched. Note that this counter is not per cluster which is simple but the clusters aren't treated in a same way, which for the filter doesn't make a difference. This parameter serves to increase performance.

max. measurement variance and min process variance see at `use dynamic variances`.

map to road This option is only usable for a highway overpass application. It maps the pixels to a distance according to the following parameters before the Kalman filter treat the data. The scaling within the Kalman filter changes from pixel to meters.

bridge height This parameter is only for use in `map to road` mode. It determines the height of the bridge in meters.

distance to 1st Pixel This parameter is only for use in `map to road` mode. It determines the distance in meters to the 1st(bottom) y-coordinate pixel. This is used to calculate then the opening angle of the camera.

distance to vanishing point This parameter is only for use in `map to road` mode. It determines the distance in meters to the vanishing point set by a mouse click. If no vanishing point is set it is then by default the vanishing point is set at the top of the window.

hysteresis This parameter determines the size of the hysteresis.

B Description of the Wing Tracker Options

B.1 Options and parameters only for WRA mode

do body update After each wing beat, parameters are updated. With this parameter on also the body position is changed to the middle point $\vec{p}$ of the center of masses of the wings. If the user set a body position $\vec{u}$, the offset $\vec{u} - \vec{p}$ is stored and added to the updated $\vec{p}_{new}$.

do heading update The heading is corrected. This correction compensates a bad initialization or a movement of the the fly. It is very important that the heading line just separates the wings.

mixing factor See section 3.1.1 in part 2.

B.2 Options and parameters only in Kalman filtering mode

use Kalman filtering With this option on, the Kalman filtering approach is used to track. A window pops up where for each wing edge the covariance matrices Q and R can be modified. The window shows also the current state vector (click on a matrix element to refresh the entries) and the error covariance matrix P. Individual edges can be turned off to increase performance for other edges.

show EKF Parameter Window see above at `use Kalman filtering`.

number events to collect per Edge For each edge, this number of events are averaged before the Kalman filter machinery starts. The goal is to increase performance, but if too much events are averaged, the quality of the track suffers. The magnitude of the paramter depends on the quality of the data.

B.3 Options and parameters for both approaches

flip heading If the initialization didn't work correctly one can mirror the heading direction (rotate 180 degrees). Note that the Kalman filter then have to refind the edges.

do log While this option is set, data are recorded and written out into a human readable text file in the home directory of the "usb2aemon" application.

C Kalman Filter algorithm

C.1 Description of the variables

x_k	The true state vector.
$\hat{x}_k$	The current estimated state vector.
$\hat{x}_{k\|k-1}$	The predicted state vector after the prediction step.
A	The tranisition matrix. In our case it is constant.
H_k	The observation model. It maps x on z.
z_k	The measurement used for the update step.
$\tilde{y}_k$	The residual. The difference between the prediction and the measurement.
P_k	The error covariance matrix($= cov(x_k - \hat{x_k})$). It is a measure of the estimated accuracy of the state vector.
$P_{k\|k-1}$	The predicted error covariance matrix($= cov(x_k - \hat{x}_{k\|k-1})$.
Q	The process covariance matrix. One can define the accuracy of the individual parameters in the state vector and their dependencies. More stable parameters should have a less big variance.
R	The measurement covariance matrix. One can define how accurate the individual measurement parameters are.
K	The Kalman gain. The weighting between prediction and residual.

C.2 Kalman Filter without control input model

The prediciton step

$$\begin{aligned} x_{k|k-1} &= Ax_{k-1|k-1} \\ P_{k|k-1} &= AP_{k-1|k-1}A_k^T \end{aligned}$$

The update step

$$\begin{aligned} y_k &= z_k - H_k x_{k|k-1} \\ S_k &= H_k P_{k|k-1} H_k^T + R_k \\ K_k &= P_{k|k-1} H_k^T S^{-1} \end{aligned}$$

$$\begin{aligned} x_k &= x_{k|k-1} + K_k y_k \\ P_k &= (I - KH) P_{k|k-1} \end{aligned}$$

C.3 EKF for Wing Tracking

Note that in contrast to a standard EKF only the measurement map is non-linear. The prediciton step

$$\begin{aligned} x_{k|k-1} &= A x_{k-1|k-1} \\ P_{k|k-1} &= A P_{k-1|k-1} A_k^T \end{aligned}$$

The update step

$$\begin{aligned} y_k &= z_k - h(x_{k|k-1}) \\ S_k &= H_k P_{k|k-1} H_k^T + R_k \\ K_k &= P_{k|k-1} H_k^T S^{-1} \end{aligned}$$

$$\begin{aligned} x_{k|k} &= x_{k|k-1} + K_k y_k \\ P_{k|k} &= (I - KH) P_{k|k-1} \end{aligned}$$

where $H = \frac{\delta h}{\delta x}|_{x_{k|k-1}}$

References

[1] C. F. Graetzel, S. Fry, and B. Nelson. A 6000 hz computer vision system for real-time wing beat analysis of drosophila. In *Biorob 2006*, 2006.

[2] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128 x 128 120db 30mw asynchronous vision sensor that responds to relative intensity change. In *IEEE ISSCC Digest of Technical Papers*, pages 508 – 509, 2006.

[3] P. S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. 1979.