Low-level Stereo Matching using Event-based Silicon Retinas

Semester Thesis, 2006

by Peter Hess[1]

Supervised by Tobi Delbrück[2]

Abstract. This semester thesis examines the task of computing stereo image disparities using a binocular pair of event-based silicon retinas [1]. Unlike conventional video cameras, this silicon retina is asynchronous and event-based. It responds to reflectance change and thus the events primarily encode object movement. Therefore there is no global frame rate. Two different algorithms were devised to do stereo matching by comparing the time differences of events. The matched events are "orientation events" derived from oriented spatio-temporal coincidence of retina events. Additional constraints are used to suppress noise and ensure local and spatial smoothness. The matching is low-level and purely based on events, without any actual object recognition. The first algorithm enforces smoothness by assuming a single frontal object and therefore the same disparity is assigned to all events in a certain period. The second algorithm calculates disparities for every event individually. Smoothness is enforced by restricting the matching search range around the mean disparity of previous events in the neighborhood. Running on a 1.6 GHz Centrino laptop computer, performance ranged from processing 120 keps (kilo events per second) for the global filter to 75 keps for the locally constrained disparity filter.

1 email: pehess@student.ethz.ch
2 email: tobi@ini.phys.ethz.ch, http://www.ini.unizh.ch/~tobi

1. Introduction

By using a pair of binocular cameras, it is possible to derive distance measurements from a scene by comparing the differences between the images. *Fig. 1.1* shows a setting where the cameras are set up parallel with distance b. P_L and P_R are the projections of a 3D point P onto the respective cameras. The difference in pixels between the coordinates of P_L and P_R is called the disparity d. It will only occur along the x-axis as long as the cameras are not tilted. Therefore $d = x_R - x_L$ and the disparity computation can be addressed independently for each image row. $d \geq 0$ is always true for parallel cameras and the connection between disparity and distance is quite simple:

$$z = \frac{bf}{pd} \tag{1.1}$$

Pixel pitch p, focal length f and camera distance b are usually fixed. Therefore distance calculation consists of determining the disparity. Finding corresponding points P_L and P_R (and therefore depth z) is a nontrivial task, which is usually done by feature matching. This is known as the stereo matching problem. The main challenge comes from the possibly large number of false matches (*Fig. 1.2*), although the choice of good features may reduce this problem.

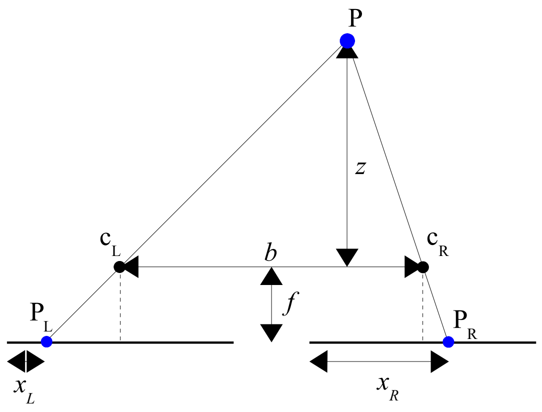


Fig. 1.1: A pair of parallel cameras with focal length f and distance b between camera centers.

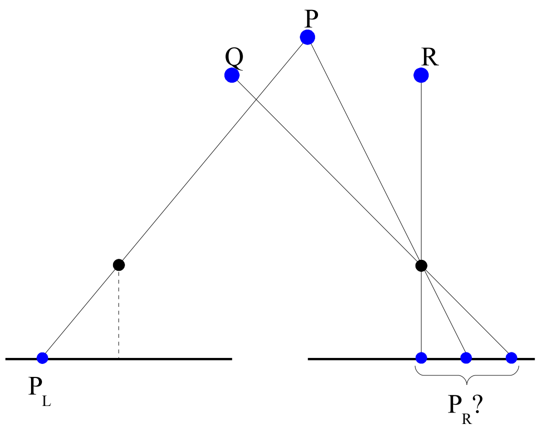


Fig. 1.2: Projection P_L should match to one of the points on the right image. But two of the three points there are false matches.

In this work we didn't use conventional cameras but Silicon Retinas [1]. Unlike normal cameras they respond to reflectance change and thus they primarily encode object movement. They are also asynchronous. This means that pixels are not synchronized to a global frame rate, they rather report a reflectance change (a so-called event) as soon as it occurs. The events produced by the retina contain pixel coordinates, time stamp and polarity information. The polarity is a binary value ON or OFF, denoting whether the reflectance changed from dark to bright or bright to dark. The events are then sent to the computer, where the data stream from the left and right retina are combined into one single stream. This stream is cut into event packets which contain up to a few thousand events. These event packets are then further processed by a number of filters. Filters are user-made algorithms which may alter event packets in multifarious ways. Usually they either drop unwanted events (e.g. background noise) or attach additional information to the events (e.g. disparity values).

Due to these differences in form of input data, stereo matching using retinas is different than classical approaches. Classical feature matching mainly relies on finding and matching edges. The retina reacts to reflectance changes, which basically means that it detects moving edges. Therefore we can directly match events. Further, we can take advantage of exact timing. Instead of matching pixels from two images which were captured at the same time, we can use the time difference between an actual event and previously occurred events.

For reducing the number of false matches we only compare events with same polarity and orientation. Orientation is calculated by a preceding filter which detects whether an event is part of a horizontal, vertical or diagonal line.

Another approach to stereo matching would be to first identify objects on each separate retina and then try to match those objects. This would simplify the matching problem by shifting most of the work towards good object recognition. This topic will be covered in a separate investigation. Here we deliberately chose to do the stereo matching on a low level, because finding useful objects might be difficult if the scene is complex and we wanted to make as few assumptions about the scene as possible.

2. Global Disparity Filter

2.1. Initial Version of the Global Disparity Filter

Our first approach on computing disparities was motivated by the algorithm described in [2] that was based on the famous Marr-Poggio stereo matching algorithm. It compares a row from one image with the corresponding row from the other image (*Fig. 2.1*). The features from the left and right image are then matched in a 2D matching matrix, where points with same disparity lie on the same diagonal. By simply comparing all possible points, a large number false matches arise (red dots) besides the correct matches (green dots) and it is very hard to distinguish true from false matches by simply looking at one single match. It is clear though, that only one match along the dashed lines can be the true match. For the beginning, lets assume that the retinas are recording a single frontal object. This implies that all true matches should lie on the same diagonal, presumably the one with most matches.

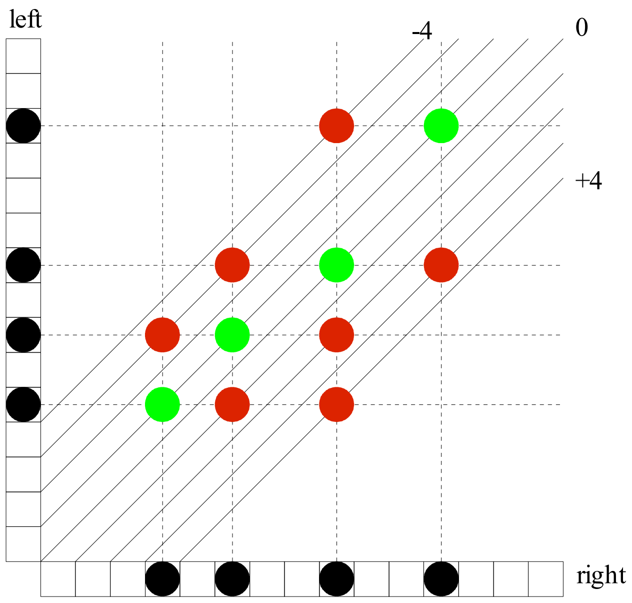


Fig. 2.1: Matching points from the left and right image causes false matches (red dots) which are hard to distinguish from the correct matches (green dots).

Since the retina generates individually timed events, we can't do the matching on a frame by frame basis (Well, we could assign events to artificial frames. But then we would be throwing away timing information). This means that we have to compute a measure of event correlation. Events with differing features (i.e. polarity and orientation) aren't matched at all. Therefore the correlation only depends on the time difference between the events. [3] deals with a topic unrelated to stereo vision, but they also have to correlate events with different timing. To that end they use a Gaussian distribution function. Unfortunately computing exponential functions is rather slow and we have to do several thousand comparisons per second. Therefore we use an inverse linear distance:

$$c(dt) := \frac{1}{a * dt + 1} \qquad (2.1)$$

This hyperbolic function yields a maximal correlation of 1 for $dt = 0$ and slowly decays towards 0 with growing *dt*. The tuning parameter *a* controls the slope. This function computes much faster than the Gaussian and there's no obvious loss in quality (for more details see chapter 5.1).

The overall disparity for two image rows can now be determined by finding the diagonal with the highest sum of correlations. For whole images this extends to finding a diagonal plane in a matching cube. Since the retinas have a resolution of 128x128 pixels, the matching cube would have a size of 128x128x128. This is much too large for efficient computation and therefore we'll make do with a collapsed representation of this cube. This means that we accumulate all correlations with same disparity independent from their y-coordinates into a 128x128 matching matrix. We won't lose any accuracy for determining the global disparity, since we're only interested in the sum of correlations along a diagonal plane.

Concretely, we calculate the disparity of an event packet by first accumulating the correlations in the matching matrix. The search range parameter restricts the maximal disparity we consider for matching. Range restriction will increase computational performance as well as quality since it will cut off matches of events which are too far apart. The disparity is determined by finding the diagonal with highest sum of correlations. We then apply a lowpass filter to ensure temporal smoothness between event packets. The following pseudo code shows how this procedure works:

```
Event[2][128][128] previousEvt;
float[128][128] matchingMtx;


int calculateDisparity() {
   int disparity;
   int eye;
   fill(matchingMtx, 0);  //reset the matching matrix
   foreach(Event e in Eventpacket) {
      if (fromLeftEye(e)) {
         eye = 0;
      } else {
         eye = 1;
      }
      for (int x' = e.x - searchRange; x' < e.x + searchRange; x'++) {
         if (haveSameFeatures(e, previousEvt[1 - eye][x'][e.y])) {
            float c = correlation(e - previousEvt[1 - eye][x'][e.y]);
            if (eye == 0) {
               matchingMtx[x'][e.x] += c;
            } else {
               matchingMtx[e.x][x'] += c;
            }
         }
```

```
        }
        previousEvt[eye][e.x][e.y] = e;
    }
    disparity = maxDiagonal(matchingMtx);
    return lowpassFilter(disparity);
}
```

2.2. Matching Matrix Viewer

The viewer (*Fig. 2.2*) visualizes the binocular retina activity and the matching matrix calculated by the previous algorithm including the diagonal with maximal correlation. The bar below the matrix shows the accumulated correlation values for every diagonal in the matrix. This tool is very helpful for finding good values for the tuning parameter *a* (*equation (2.1)*). It also demonstrates that the ambiguity of the correlations is strongly influenced by the scene. *Fig. 2.2* shows a very well behaved scene where the matching matrix shows almost no false matches. Thus finding a good diagonal is easy.

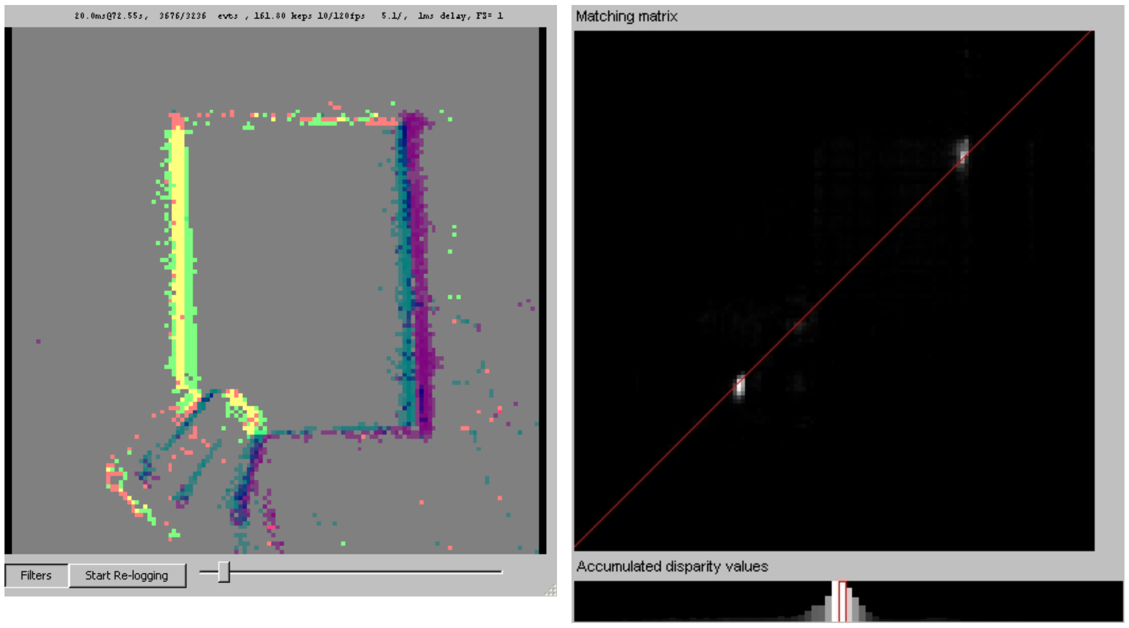
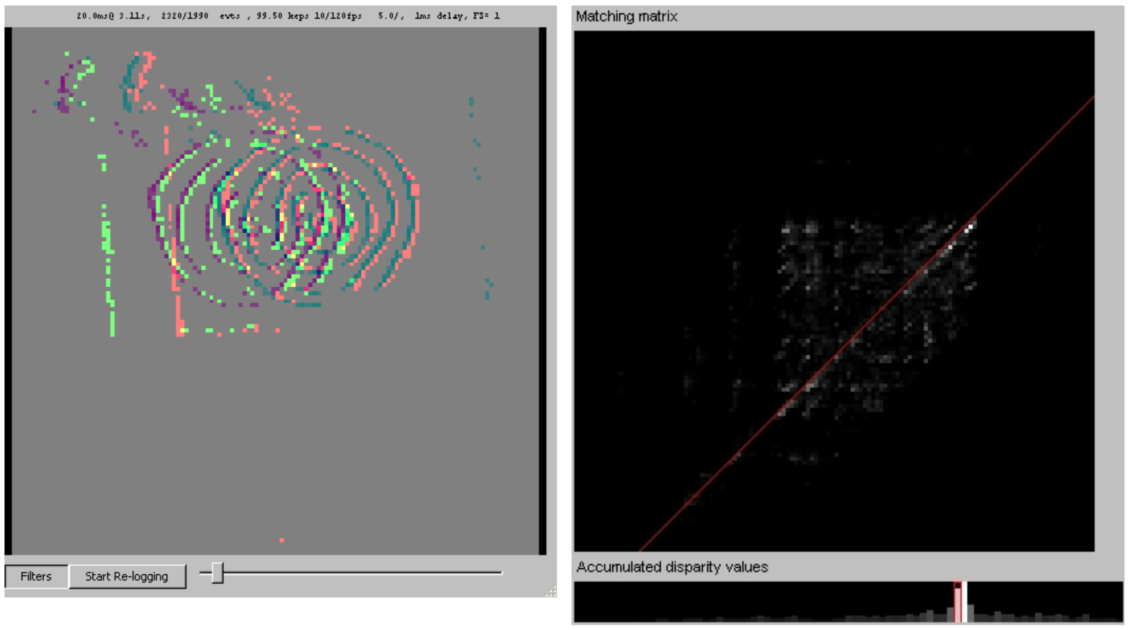

20.0ms@72.55s, 3676/3236 evts , 161.80 keps 10/120fps 5.1/, 1ms delay, FS= 1
Filters
Start Re-logging
Matching matrix
Accumulated disparity values

Fig. 2.2: Shows the binocular view and the matching matrix. The left panel shows events histogrammed over 20 ms with right eye events shown in red and left eye in green. The right panel shows the matching matrix. The histogram below shows the accumulated disparity values along diagonal lines in the matching matrix.
This is a very well behaved scene with unambiguous correlations. Finding the best diagonal is easy.

Fig. 2.3 shows a much more complicated scene, where the circular pattern causes many false matches. The matching matrix indicates that it's difficult to determine the disparity by only looking at single events. It's necessary to combine the correlation of many events to get a reliable solution in this situation. Assuming a single frontal object makes finding the best diagonal much easier and more reliable.



```
20.0ms@ 3.11s,  2320/1990  evts , 99.50 keps 10/120fps   5.0/,  1ms delay, FS= 1
```

Fig. 2.3: A scene which causes many false matches. Overall disparity computation is still possible since we know that the object is frontal. The accumulated disparity value histogram shows a clear peak at one disparity.

2.3. Optimized Global Disparity Filter

In the first version of the filter we already simplified the algorithm by only using a reduced matching matrix instead of a whole cube. If we don't care about visualization then we don't even need the matrix. We only have to maintain a one dimensional array where we accumulate the correlations instead of a 128x128 matrix. By doing so we lose no precision, as long as we compute only a single disparity for the whole image:

```
Event[2][128][128] previousEvt;
float[2*searchRange + 1] correlationSums;


int calculateDisparity() {
   int disparity;
   int eye;
   fill(correlationSums, 0); //reset the correlation sum array
   foreach(Event e in Eventpacket) {
      if (fromLeftEye(e)) {
         eye = 0;
      } else {
         eye = 1;
      }
      for (int j = -searchRange; j <= j; x'++) {
         x' = e.x + j;
         if (haveSameFeatures(e, previousEvt[1 - eye][x'][e.y])) {
            float c = correlation(e - previousEvt[1 - eye][x'][e.y]);
            if (eye == 0) {
               correlationSums[searchRange + j] += c;
            } else {
               correlationSums[searchRange - j] += c;
            }
         }
      }
      previousEvt[eye][e.x][e.y] = e;
   }
   disparity = maxIndex(correlationSums) - searchRange; //find index of maximal value
   return lowpassFilter(disparity);
}
```

3. General Disparity Filter

The above-mentioned global filter is fairly robust but its basic assumptions are very restrictive. Nonetheless, they are still very useful for camera alignment, which is the same as finding the vergence of the two cameras and automatically adjusting the average disparity to zero.

As we have seen before, it's very difficult for a single event to find the right match among all the false matches. Instead of averaging over the whole image, we will calculate a local mean disparity value d_{mean} for each new event:

$$d_{mean} = \frac{\sum_{i \in N} c(dt_i) * d_i}{\sum_{i \in N} c(dt_i)} \tag{3.1}$$

d_i is the disparity and $c(dt_i)$ is the correlation between the new event and the i-th previous event from neighborhood N. Due to the weighting old events will have less influence. Additionally we calculate the average time difference of the neighboring points:

$$t_{mean} = \frac{1}{|N|} \sum_{i \in N} dt_i \tag{3.2}$$

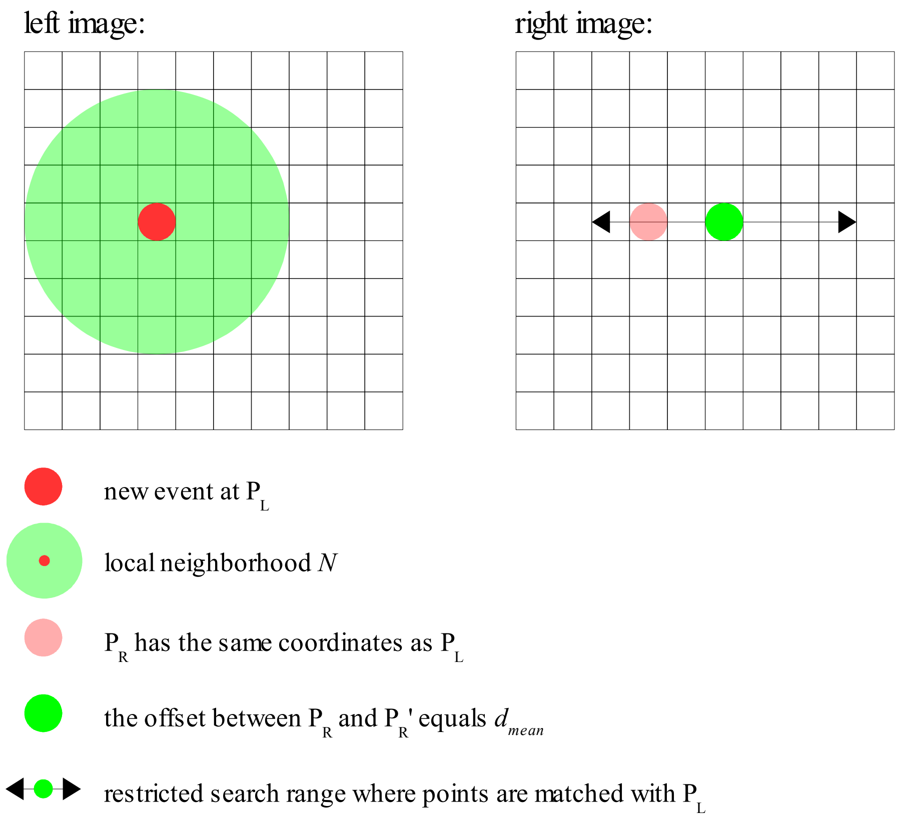

left image:
right image:
new event at P_L
local neighborhood N
P_R has the same coordinates as P_L
the offset between P_R and P_R' equals d_{mean}
restricted search range where points are matched with P_L

Fig. 3.1: Shows how the restricted search range is computed by using neighborhood information.

We now restrict the search range to [0.2*maxDisp, maxDisp]. If the distance between t_{mean} and the new event is small, we are confident that the new disparity should be near d_{mean}. If the time distance is larger, we allow a bigger search range:

$$r = .8 * maxDisp * (1 - \frac{1}{b * dt_{mean}^2 + 1}) + .2 * maxDisp \tag{3.2}$$

b is a tuning parameter and *maxDisp* is the overall maximal disparity which we want to consider for matching. It's important to not restrict the range too much (the algorithm can get stuck if you restrict the search range to zero), so we chose to restrict the range to 20% of maxDisp at top. This value can be seen as tuning parameter.

In the end we will only match events which have a disparity of $d_{mean} \pm r$ (also shown in *Fig. 3.1*). By only looking at those events which have disparity similar to d_{mean}, we ensure local and temporal smoothness. Finally we do one more smoothing step to get the disparity d of the new event:

$$d = (1 - s) * d' + s * d_{mean} \tag{3.3}$$

Here, d' is the disparity of the most recent event in the range $d_{mean} \pm r$ and s is a mixing factor that ranges from 0 to 1. A small s means that the disparity can only change slowly. s is like a time constant but is actually an "event constant" that determines how rapidly new data can be changed by additional events.

The problem with this solution is that the computing performance is very dependent on the size of the neighborhood N. If we look at *Fig. 2.3* we see that the points near the center of the circles are surrounded by other points which have mostly false matches. We have to go quite far to find good points (i.e. the border of the sheet). This tells us that we should use a neighborhood as large as possible for better results. Unfortunately the number of neighboring pixels grows quadratically with the size of the radius. Thus we won't use all adjacent points inside a certain radius but just a representative set of points that we call a *prototype neighborhood*. For the algorithm we use a set of 89 prototype points covering an area with a diameter of 35 pixels (more than ¼ of the whole image). By comparison, with the same number of points we couldn't even cover a filled out area with a diameter of 11 pixels (*Fig. 3.2*). It's difficult to measure the performance of such prototype neighborhoods, but covering a large area tends to give smoother results than evaluating every single point in the vicinity.

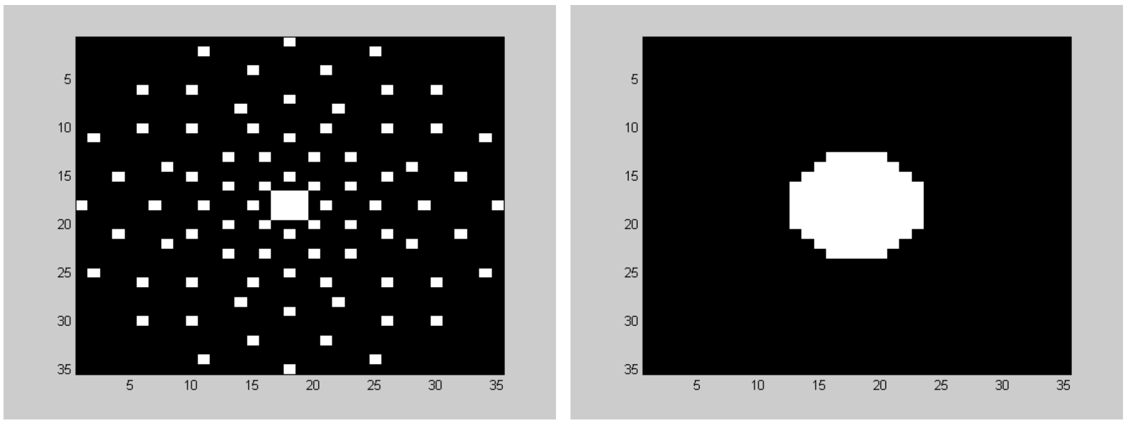


Fig. 3.2: By using a sparse prototype neighborhood a much larger area can be covered with the same number of points.

4. Results

In this chapter we discuss results of stereo matching quality and computational performance of the different algorithms.

4.1. Quality

Since we don't have any error metric, we have to make do with visual results. The Vergence Filter is a simple tool to visualize the quality of any disparity filter. It shifts the right and left image according to the calculated disparities, so that ideally both images should lie maximally on top of each other.

Fig. 4.1 shows a relatively simple scene with low noise and with the hand frontal to the vision system. Therefore only few false matches arise and it's easy for the global filter to determine the disparity with high confidence. *Fig. 4.2* is much more complicated and even the basic assumption about a single frontal object isn't met (the sheet of paper is tilted and the person holding the paper is visible in the background). In addition, the image is noisy and the circular pattern causes many false matches. Nevertheless, the global filter manages to find a good average disparity for the paper. The region around the disparity with maximal correlation is much broader than in the previous example but the solution is still stable since there's only one local maximum. This example shows that the global filter performs fairly well even under suboptimal conditions (i.e. initial conditions not fully met, difficult patterns, noise). Altogether the global filter might have a restricted range of application due to its basic assumptions but it's fairly robust in all scenes.

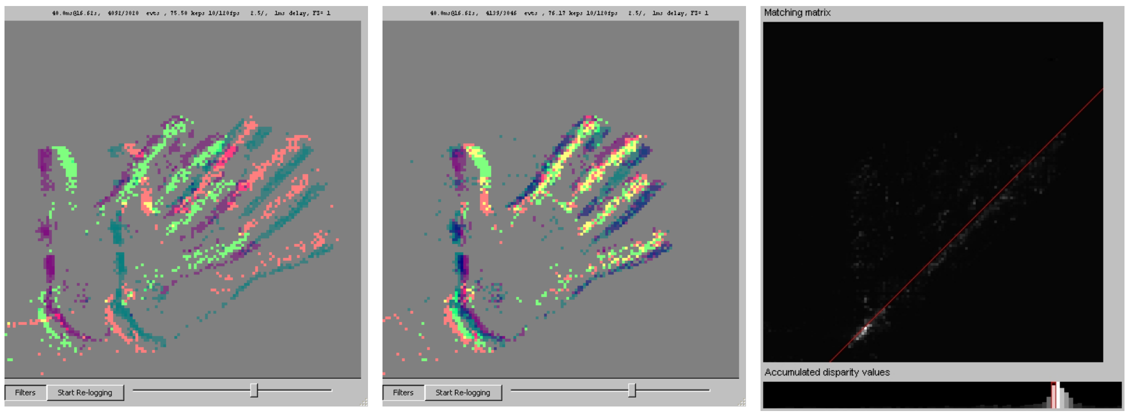


Fig. 4.1: Original scene of a single frontal object (left) and verged images after using the global disparity filter (middle). There is low noise and only few false matches (right).

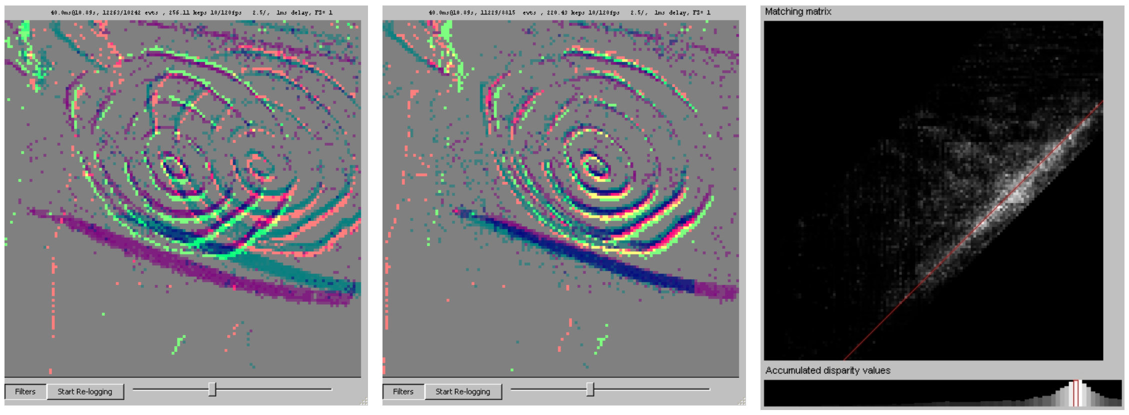


Fig. 4.2: This scene is much more difficult than Fig. 4.1 and and the object is not even frontal. Nevertheless, the Global Disparity Filter manages to find a good average solution.

The results from the general disparity filter (*Fig 4.3*) are noisier than those generated by the global filter (*Fig 4.1*). Of course this doesn't come as a surprise since the global filter averages the disparity over the whole image, while the general filter has to rely on weaker local smoothness constraints.

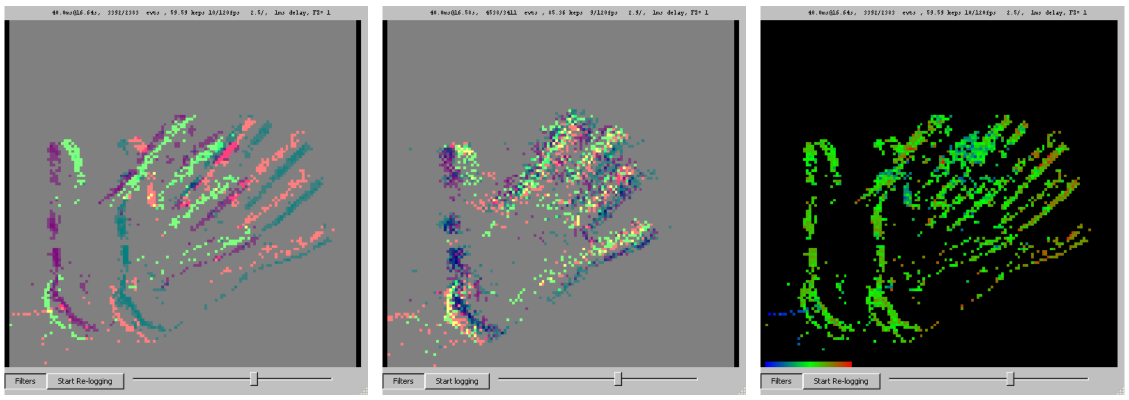


Fig. 4.3: Original scene (left) and verged images after using the general disparity filter (middle). The right image shows a disparity color scale image (blue is far, red is near).

Fig. 4.4 however shows a scene with two objects having clearly different disparities. The global filter verges the front hand correctly but the rear hand is shifted too far (The red hand was initially on the right side. After verging it's far on the left side). The general filter is again noisier but instead it manages to verge both hands separately.

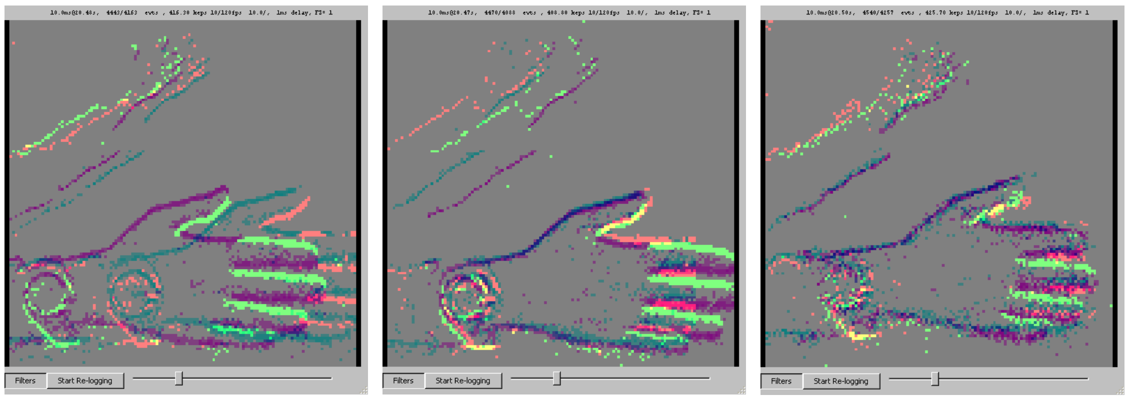


Fig. 4.4: The global filter (middle) matches the front hand well but the rear hand is shifted too far. The general filter (right) is again noisier, but it manages to verge both hands.

4.2. Performance

The following performance tests were made from recorded data on a 1.6GHz Centrino-based laptop with 512Mb RAM and ATI Mobility 9000 graphic card. Calculation of event orientation is a preceding task which is needed for all disparity filters. Since this calculation is quite costly, disparity and orientation computation are measured separately. These are average results and they may slightly vary, depending on the scene.

For the optimized global filter, orientation feature extraction is the most costly part in the whole algorithm, taking more than twice the time of actual disparity computation:

	microseconds/event	**events/second**
total	8.1	120'000
disparity only	2.3	430'000
orientation only	5.9	170'000

Computation time also depends on how many previous events are correlated with each new event. The upper measurements were taken with a search range of 40 pixels to the left and right. Changing search range has a linear influence on the algorithms performance (*Fig 4.3*).

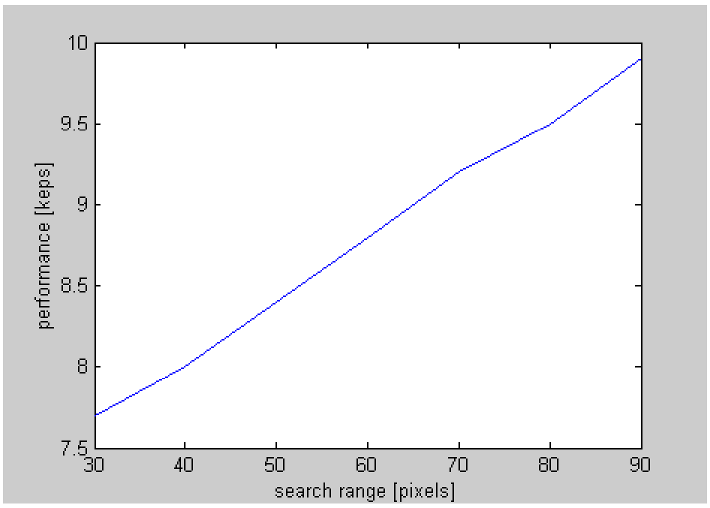


Fig 4.3: Linear relation between performance and maximal search range.

The general disparity filter is slower than the global filter, mainly due to the high number of neighbors which are needed for calculating mean disparities. Even so, a considerable amount of time is spent for computing orientation information. Maximal search range doesn't influence performance much because of the range restriction.

	microseconds/event	**events/second**
total	13	74'000
disparity only	8.1	120'000
orientation only	5.5	180'000

5. Further Questions

In this section we discuss remaining problems and possible improvements to the work.

5.1. Correlation Functions

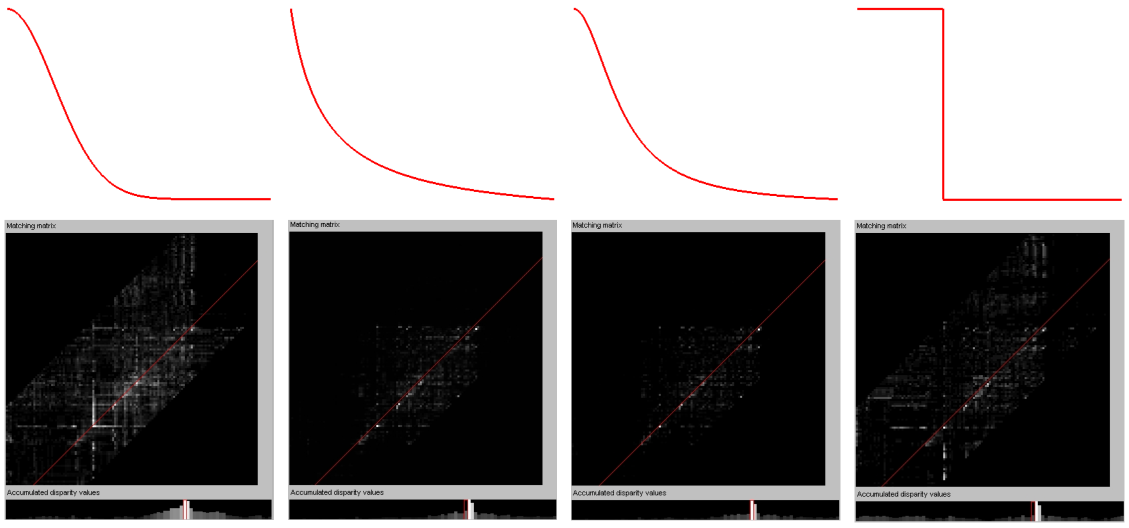
We mentioned in chapter 2.1 that we chose a inverse linear correlation function over a Gaussian, mainly due to computational performance. But since we don't have any error metric we can't numerically determine the influence of these functions on quality. Therefore we had to rely on the visual output generated by the matching matrix viewer which admittedly is fairly inaccurate. Nevertheless we tested different correlation functions, all depending on a single tuning parameter *a* and normalized to the interval [0,1]. At least it seems like there are no obvious discrepancies in quality since we can always tune *a* in such a way that the correlations conform with the other results.

Gaussian-like: $e^{\frac{-dt^2}{a^2}}$

inverse linear: $\frac{1}{a*dt+1}$

inverse squared: $\frac{1}{a^2*dt^2+1}$

Heaviside-like: $1-\Theta(dt-a)$



a = 10'000 a = 0.001 a = 0.0007 a = 9'500

Fig. 5.1: Different correlation functions depending on time difference between events and tuning parameter a. The matching matrix examples are taken from the same data as in Fig. 2.3.

In fact the Gaussian seems to perform worst among the functions in *Fig. 5.1*. It creates more false matches and unlike the other functions the area around the maximal disparity is quite broad. The two inverse functions seem to perform best in this scene. Remarkably even the simple Heaviside function has comparable results. This might indicate that exact time difference between events is not that important at all.

Of course this is only a single example which doesn't give us much insight about general robustness of these functions. It would also be interesting to know how the parameter *a* could be computed automatically and whether it should be constant or adaptive.

5.2. Prototype Neighborhoods

Concerning the choice of the prototype neighborhood, there are still many open questions. Shape, range and distribution of points are important for the quality of the algorithm, but the influence of these parameters is still widely unexplored. The total number of points for a prototype is a trade-off between computing performance and reliability of the d_{mean}. As for now, it's not clear what a well-performing prototype should look like. Again, the main difficulty is the lack of an appropriate error metric and even then, extensive testing would be needed to get any useful results. Besides, one could imagine to combine multiple prototypes of different shape to enhance the quality of the algorithm.

As for future work, one could use compact and wide prototypes (*Fig. 5.2*) and chose the prototype with minimal variance for further computations. Or use corner- and edge-sensitive prototypes like in *Fig. 5.3*. These would be interesting because as for now, disparities are averaged uniformly around the center. This blurs disparities in an area where two objects with different distances meet.

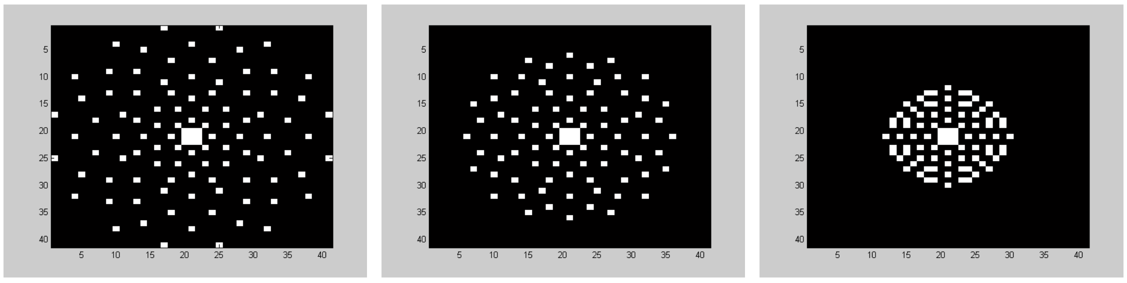
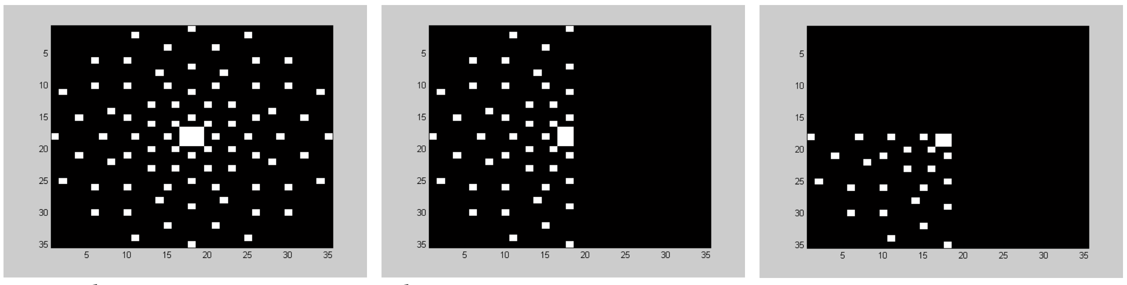


Fig. 5.2: Use prototypes of different size and use the one with minimal variance.



centered *edge-sensitive* *corner-sensitive*

Fig. 5.3: Use edge- and corner-detecting prototypes to better adapt to the local geometry.

6. Summary

The main achievements of this work are two low-level event-based stereo matching algorithms. The Global Disparity filter is a fast and robust algorithm for situations where you have only a single frontal object. For the General Disparity filter, the situation is far more challenging because we don't have any additional information about the scene. The only constraint we make, is that the computed disparities should be spatially and temporally smooth. Therefore the General Disparity filter gives noisier results and is also not as robust as the General Disparity filter. In the future we could improve the performance of the General Disparity filter by using different prototype neighborhoods which would detect edges and corners.

References

[1] P. Lichtsteiner, C. Posch, and T. Delbrück, **A 128×128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change**, *ISSCC Digest of Technical Papers 2006 508-509*, 2006

[2] M. Mahowald and T. Delbrück. **Cooperative stereo matching using static and dynamic image features**. *In C. Mead and M. Ismail (eds.) Analog VLSI Implementation of Neural Systems. Kluwer Academic Publishers: Boston. pp 213-238*, 1989

[3] M. Boerlin, K. Eng, **Getting to know your neighbors: reconstructing topology from real-world input**, March 2, 2006