Project FoSIBLE
Fostering Social Interactions for a Better Life of the Elderly

Deliverable

D5.2: Software Prototypes according to D5.1 Delivery date: M18

Responsible

AIT (Lead Contractor)

Participants

AIT

Version: 1.0
Date: 28.10.2011
Dissemination level: (PU, PP, RE, CO): PU

Abstract

D5.2 contains the detailed description of the software prototype for the gesture recognition with the UCOS stereo sensor developed in the FoSIBLE project. The details on the software, the underlying algorithms and its modules and the description of a prototype front-end used for a demonstrator are given in this document.

Table of Content

1 Introduction

1.1 Purpose of the Document

The purpose of this document is to report the details of the software prototype for gesture recognition with the UCOS stereo sensor developed in the FoSIBLE project. The report contains the details on the algorithms, their software implementation in the MATLAB programming language, the description of the software modules and their interaction and a prototype front-end to be used for in the project for online demonstration.

1.2 Definitions, Acronyms and Abbreviations

Acronym	Description
FoSIBLE	Fostering Social Interaction for the Well-Being of the Elderly
IF	Interface
PC	Personal Computer
AE(R)	Address Event (Representation)
3D Sensor	Devices that delivers spatial (three dimensional 3D) information of a given scene , typical x,y and depth-z
SW	Software
HW	Hardware
TAE	Timed address event data
OS	Operating System

2 Overview

This chapter describes overall concept and processing steps of the software prototype.

2.1 Concept

The intention of integrating gestures for controlling the FoSIBLE application is to offer an alternative to the tablet used as main input device. The users don't need to grab anything but can immediately start to interact with the system by just moving their hands. This is why a simple trigger for initiating the gesture control is necessary e.g. waving a hand looking directly at the TV.

Gestures are a natural input modality that evolved from real-world interaction styles being more intuitive and easier to learn than indirect input modalities like remote controls.

The prototype software is optimized to allow an online demonstration and evaluation of the gesture user interface running on a PC. It allows navigating a simple menu driven user interface with move-up, down, left and right gestures. The processing chain starts in the UCOS2XL stereo sensor [1] that extracts motion relative to the sensor by an biology inspired "silicon retina" optical sensor [2][4], stereo correlates the motion information yielding depth information and tracks motion nearest to the sensor [3]. In the case of gesture recognition the motion tracked is the hand motion performed by the user.

Tracking the object nearest to the sensor is already implemented in the embedded software of the device and the extracted tracks are also encoded in the UCOS2XL sensors timed address events (TAE) data stream. When moving an arm in front of the device, performing an arm gesture, it will therefore continuously output the position of the hand as being the closed object. Figure 1 shows a sequence of still images of example tracking data (left pane) and synchronous video (right pane) of a person performing a "circle" gesture with the right arm. The continuous track of the hand performing the gesture is then analysed in the later processing stages.

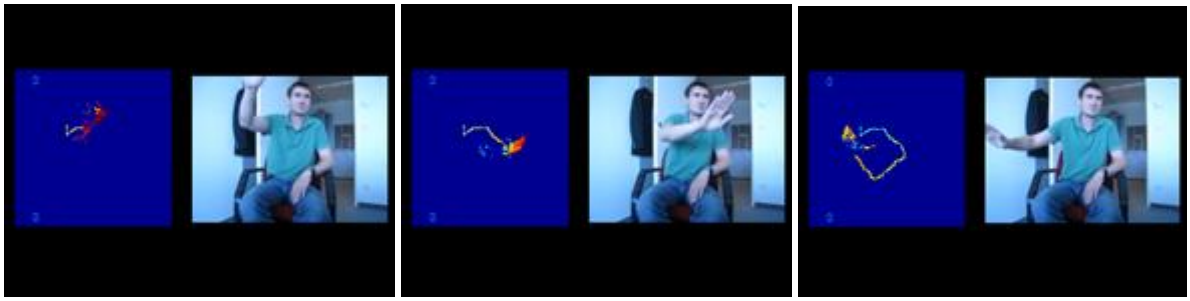

Figure 1 Sequence of still images of an example of raw hand tracking data from the UCOS2 sensor (left pane) and synchronous video (right pane) for a person performing a "circle" gesture with his right arm.

These tracks are then extracted by the embedded software of the UCOS2XL stereo sensor and are sent via Ethernet connection to the gesture processing software implemented on a PC.

2.2 Processing steps and modules

2.2.1 Real time data interface to UCOS Sensor and pre-processing

The motion- and extracted tracking data from the UCOS2XL stereo sensor is received by streaming binary data packets via UDP protocol over an Ethernet connection. Data is transferred via defined UDP ports from the UCOS2XL device to a PC. The pre-processing of these data performs extraction from the track data, containing the actual gesture, from the rest of the motion data and de-noising of the extracted tracks. This includes the removal of outliers and duplicate tracks that may occur in the data.

The begin and the end of each gesture is sensed by monitoring the sensor data rate and a data rate falling below a threshold marks the end of a track. Full tracks are then sent further to the feature extraction module.

2.2.2 Feature extraction

Features are extracted on the same time basis as the UCOS2XL sensor generates the data, i.e. a 100 Hz sampling rate. The feature extraction units use the data received from pre-processing and derive explicit feature from tracks. Typical feature are the track-length, speed, acceleration, orientation, Bounding-Box characteristics or activity (detailed description of the features see “Implementation” chapter). “Strong” features that will allow robust gestures detection are those that are invariant to the size and position of the gesture.

Each gesture is represented by a combination of some of the features. The length of a track is the sum of distance between all points of the track. Since this feature is depends on the amplitude of the gesture-movement length will not be a strong feature. Speed, representing how fast a hand is moved to perform a gesture, is another weak feature, since it has on one side a big variation depending on the person performing the gesture, but there will be no big difference between the gestures. A much stronger feature is the orientation of the gesture movement, since each gesture has a distinguished direction, which can also be detected on sensor data. The Bounding-Box characteristics are the area covered by a box around the gesture movement and the ratio of the side lengths of the box. The Bounding-Box Area is not independent to the amplitude of the gesture-movement, but highly characteristic for each gesture and therefore a strong feature. Another very strong feature is the Bounding-Box ratio, which is very characteristic for each gesture. The acceleration of the gesture-movement might be used to distinguish how intentional a gesture is performed.

The features for a gesture are stores in time depended feature value during the gesture and is send further to the gesture analysis module.

2.2.3 Gesture analysis

For the software prototype rule based gesture recognition has been implemented. The time depended feature values are analysed based on a set of simple rules, regarding the occurrence of orientation, bounding box ratio etc. within the feature data. Depending on the result of this rule based analysis the gesture is classified and the respective command is sent

to the GUI to trigger an action. The details of the rule based analysis are described in the chapter “Implementation”.

Note that for the stable software prototype, planned later in the project, the implementation of gesture analysis based on Hidden Markov Model classifiers is foreseen. These will not result in a single selected gesture based on rules but on a probability vector that describes the probability of the gesture to represent each of the possible gestures.

3 Implementation

This chapter describes the detailed implementation of the processing steps of the software prototype including partly the Matlab function code.

3.1.1 Data interface to UCOS Sensor and pre-processing

The AE Stream is recorded from the Sensor is recorded from the Sensor using the Smart Eye Center. This Data is imported into Matlab. The code below shows the complete pre-processing chain for a single set of tracks. The function `ae_bin2mat` is used to import hte binary data into a Matlab structure which enables the further processing steps. In the second step, using the function `ae_trackfilter` all AEs not containing track information are removed and only the tracks are stored. Then the data is converted to the 2x32bit ITN Format using the command `eth2itn`. After this duplicate tracks are removed from the tracks (`removeduplicate tracks`) and all a final filter is run over the converted data to remove short tracks (`trackfinder2`) .

```
%import previously recorded binary data to matlab structure
ae_all_left = ae_bin2mat('C:\Users\zimam\Documents\recordings\left\SmartEye
UCOS2_20110826_144849.bin');
%filter out the AEs representing tracks
ae_tracks_left = ae_trackfilter(ae_all_left,4);
%convert AE to 2x32bit ITN Format
ae_tracks_left_itn=eth2itn(ae_tracks_left,bitmask(129));
%just keep single tracks, with a certain length and remove duplicate tracks
%tracks are separated by “NaN” Markers
Tr_left=trackfinder2(removeduplicate tracks(ae_tracks_left_itn),10,15);
```

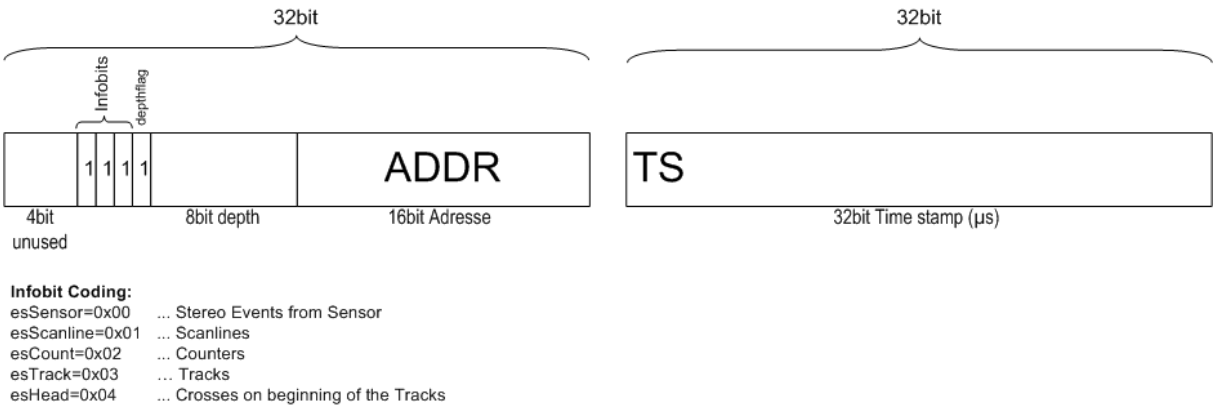

Figure 2 Data Format containing especially coded bits to identify tracks in the AE Stream

The tracks are especially tagged in the bit stream, which makes it easily possible to filter them out. The data format used has this information coded into 3 Bits, which each bit combination standing for another type of AE (Figure 2). In this case the Tracks can be identified as AE type 0x03. To filter the tracks it is necessary to apply a bitmask to the recorded data. AEs, where the masked bits match are kept and returned as reduced set of AEs, just containing the desired data. To filter the tracks out of the mask number four (`maskinfo(4)`) has to be applied to the dataset (as there is no zero element in Matlab).

```
% bitmasks
%                                 10987654321098765432109876543210
%                                  3         2         1         0
maskinfo(1) = uint32(bin2dec('00000000000000000000000000000000')); % Stereo
Evens (do nothing)
maskinfo(2) = uint32(bin2dec('00000010000000000000000000000000')); %
Scanlines
maskinfo(3) = uint32(bin2dec('00000100000000000000000000000000')); %
Counter numbers
maskinfo(4) = uint32(bin2dec('00000110000000000000000000000000')); % Tracks
maskinfo(5) = uint32(bin2dec('00001000000000000000000000000000')); %
Crosses on beginning of tracks

%check the status of the tree masking bits
mymask = uint32(bin2dec('00001110000000000000000000000000'));

%apply mask
index = find (bitand(ae(1,:),mymask)==maskinfo(type));

%if tracks have been found, just store the tracks
if (~isempty(index));
    ae = ae(:,index);
else ae = [];
end;
```

To be able to derive features from the tracks some pre-processing is required. In a first step duplicate tracks of the tracks have to be removed. These duplicate tracks occur, because the tracks coded within the datastream have a certain persistence, which means, that they are

transmitted more than once. To reduce the amount of data these duplicate tracks are removed from the AE data.

```
%find and remove all AEs with the same timestamp
Index = find ((diff(A1.tt)==0));
A1=aeDel(A1,Index);

%find and remove all AEs with the same xy coordinates
%Reason: tracks are drawn more than once...
Index=NaN;
while(~isempty(Index))
    Index = find ((diff(A1.x)==0) & (diff(A1.y)==0));
    A1=aeDel(A1,Index);
end
```

Furthermore tracks, which are too short or who contain a much too little number of points are filtered out to just keep track with enough information for further processing. This is done by counting the number of points which represent the track. If this number is under a certain value the track is considered illegal will be deleted and therefore ignored for feature extraction. A comparison between unfiltered an final track data can be seen in Figure 4.

```
%calculate delax and deltay
deltax=diff(A1.x);
deltay=diff(A1.y);

%calculae distance using pytagoras
dist=sqrt(deltax.^2+deltay.^2);

Index = find ((dist>deltamax));
A1.x(Index+1)=NaN;
A1.y(Index+1)=NaN;

% count number of points between NaN's
% if track is shorter than "tracklength" set point's to NaN
points=0;
NaNcount=0;
elements=size(A1.x);
Index = zeros(1,elements(2));
for ind1=1:elements(2)
    if ~isnan(A1.x(ind1))    %keep count the number of points
        points=points+1;
        NaNcount=0;
    end;
    if(isnan(A1.x(ind1)))
        NaNcount=NaNcount+1;
        if(NaNcount>1)       %keep only one NaN to seperate Tracks
            Index(ind1)=1;
        end;

        if(points>0)
            if (points<=pointcount)
                position=ind1-points;
                for ind2=position:ind1  %remove too short tracks
                    Index(ind2)=1;
                end;
            end;
            points=0;
```

```
        end;
    elseif(ind1==elements(2));
        if(points>0)
            if (points<=pointcount)
                position=ind1-points;
                for ind2=position:ind1  %remove too short tracks
                    Index(ind2)=1;
                end;
            end;
            points=0;
        end;
    end;
end;

Index = find(Index == 1);
A1=AeDel(A1,Index);

elements=size(A1.x);
for ind3=elements(2):-1:1
    if isnan(A1.x(ind3));
        A1=AeDel(A1,ind3);
    else
        break;
    end;
end;
```

3.1.2 Feature extraction

This section describes each feature including the code for the extraction and the data structures.

Each track detected by the sensor has characteristic features. These Features are used to identify the meaning of the track. The more distinctive features could be extracted of a track, the higher is the significance of the track data.

Currently implemented extracted features are length, sped, orientation, Bounding Box Ratio (BBRatio) and Bounding Box Area (BBArea). The calculation of these features is described in detail in the section below.

The Input for all functions are the coordinates of the points representing the track including the relative time from the beginning of the track as timestamp, output off all functions is a feature vector containing the development of the feature over the time .

3.1.2.1 Length

The length of a track is defined as the accumulated sum of all distances between adjacent points. The distance between them is calculates using the Pythagorean Theorem. The result of this calculation is then summarized to the previous calculated distances (`dist`).

$$\Delta x_0 = x_1 - x_0 \, , \Delta x_1 = x_2 - x_1 , \ldots$$

$$\Delta y_0 = y_1 - y_0 , \Delta y_1 = y_2 - y_1 , \ldots$$

The variables Δx (`deltax`) and Δy (`deltay`) represent the difference of the coordinates of the points in an Cartesian coordinate system. They also can be interpreted as an adjacent and

opposite site of a right triangle which enables using the Pythagorean Theorem and basic trigonometric functions for further calculations.

$$tr_{length} = \sum \sqrt{\Delta x_n{}^2 + \Delta y_n{}^2}$$

For the early prototype the calculations are done using Matlab, the code below implements the described functionality. At the moment this code is written to be executed on a set of tracks, which are separated by "NaN"-Elements, which mark the end of a track.

```
%calculate delax and deltay
deltax=diffn(Tr.x,param);
deltay=diffn(Tr.y,param);

%calculae distance using pytagoras
dist=sqrt(deltax.^2+deltay.^2);

%add NaN at beginn of dist (dist has now the length of the original data)
dist=[NaN dist];

elements=size(dist);

F1.tracklenght=zeros(1,elements(2));
NaNfound=false;

for ind1=1:elements(2)
    if isnan(dist(ind1))
         %first NaN marks the beginning of a track, length is 0 at this
         %point
        if(~NaNfound)
            F1.tracklenght(ind1)=0;
            NaNfound=true;
         %second NaN is the track seperator - should be kept
        else
            F1.tracklenght(ind1)=NaN;
            NaNfound=false;
        end;
    else
         F1.tracklenght(ind1)=F1.tracklenght(ind1-1)+dist(ind1);
    end;
end;

%keep the original timestamp in the feature
F1.tt=Tr.tt;
```

3.1.2.2 Orientation

A very important feature to identify the meaning of a track is the orientation. The Orientation of a track can be defined in several ways, in this case two different approaches are chosen. One realized option is to calculate the angle between adjacent points; the other realized option is to calculate the angle from the first point to each other point.

Again the variables Δx (`deltax`) and Δy (`deltay`) represent the difference of the coordinates of the points in an Cartesian coordinate system, calculation is done the same way as for length. These values are used to calculate the angle (`F1.alpha`) in the rectangular triangle

formed between adjacent points. To do so just a basic trigonometric function, the arctangent is used. The result of this function lies in an interval between –π and +π.

$$tr_{orient} = arctangent(\Delta x, \Delta y)$$

For the early prototype the calculations are done using Matlab, the code below implements the described functionality. This code is written to be executed on a set of tracks, which are separated by "NaN"-Elements, which mark the end of a track.

```
%calculate delax and deltay
deltax=diffn(Tr.x,param);
deltay=diffn(Tr.y,param);

%add NaN at end of deltax and deltay (length of the original data)
deltax=[NaN deltax];
deltay=[NaN deltay];

F1.alpha=atan2(deltay,deltax);

elements=size(F1.alpha);

NaNfound=false;

for ind1=1:elements(2)
    if isnan(F1.alpha(ind1))
         %first NaN marks the beginning of a track, angle is 0 at this
         %point
        if(~NaNfound)
            F1.alpha(ind1)=0;
            NaNfound=true;
         %second NaN is the track seperator - should be kept
        else
            F1.alpha(ind1)=NaN;
            NaNfound=false;
        end;
    end;
end;

%keep the original timestamp in the feature
F1.tt=Tr.tt;
```

As described alternatively the angle between the first point and all other points of a track is used as feature. The calculation of the angle is done mostly the same way as described for the calculation for the orientation between adjacent points, but the variables Δx and Δy are now calculated relatively to the first point.

$$\Delta x_0 = x_1 - x_0\ , \Delta x_1 = x_2 - x_0, \ldots, \Delta x_n = x_n - x_0$$

$$\Delta y_0 = y_1 - y_0, \Delta y_1 = y_2 - y_0, \ldots, \Delta y_n = y_n - y_0$$

For the early prototype the calculations are done using Matlab, the code below implements the described functionality. Note: the especially written function `diff_1toend()` returns the distance between the first coordinate and all other coordinates as an vector.

```
%calculate delax and deltay
deltax=diff_1toend(Tr.x);
deltay=diff_1toend(Tr.y);

%add NaN at end of deltax and deltay (length of the original data)
deltax=[NaN deltax];
deltay=[NaN deltay];

F1.alpha=atan2(deltay,deltax);

elements=size(F1.alpha);

NaNfound=false;

for ind1=1:elements(2)
    if isnan(F1.alpha(ind1))
         %first NaN marks the beginning of a track, angle is 0 at this
         %point
        if(~NaNfound)
            F1.alpha(ind1)=0;
            NaNfound=true;
         %second NaN is the track seperator - should be kept
        else
            F1.alpha(ind1)=NaN;
            NaNfound=false;
        end;
    end;
end;

%keep the original timestamp in the feature
F1.tt=Tr.tt;
```

3.1.2.3 Bounding Box Ratio (BBRatio) and Bounding Box Area (BBArea)

The Bounding-Box characteristics are the area covered by a box around the gesture movement and the ratio of the side lengths of the box. To calculate this characteristics the maximum and minimum value for x and y have to be determined.

This is done by comparing each of the values to another. When a minimum or a maximum is found, the value is stored and becomes the new minimum or maximum value, which is then compared to the rest of the values.

When minimum and maximum for x an y are determined the bounding box characteristics are calculated. This is done after each checked coordinate, so that the bounding box characteristics change, when the maxima/minima change.

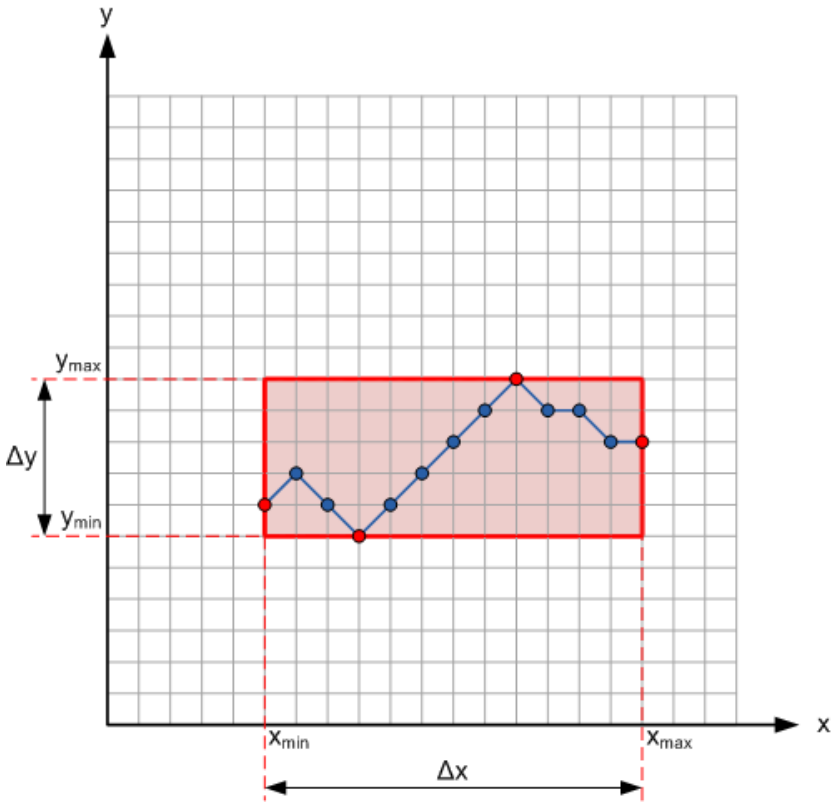

Figure 3 Bounding Box (red) around a track (blue)

Since the Bounding Box is a rectangle (red in Figure 3) the covered Area (light red in Figure 3) is calculated by multiplying the sides forming the rectangle with each other. The length of the sides (Δx, Δy in Figure 3) is represented by the four furthermost coordinates (x_{max}, x_{min}, y_{max}, y_{min} in Figure 3). The Ratio between the sides is then used as feature too. If the side Δy has a side of zero, the Ration between the sides is also set to zero.

$$\Delta x = x_{max} - x_{min}\ ; \Delta y = y_{max} - y_{min}$$

$$tr_{BBArea} = \Delta x * \Delta y$$

$$tr_{BBRatio} = \frac{\Delta x}{\Delta y}\ for\ \Delta y > 0$$

$$tr_{BBRatio} = 0\ for\ \Delta y = 0$$

The Bounding-Box characteristics are calculated for each pair of variables while the vector containing the Track information is stepped trough by a loop. If there is no change, the old value is kept as characteristic, in case of a change the maximum of the BBArea is kept, the BBRatio changes as the BBArea changes.

In the code Matlab code below, the variables of the formulas can be identified as followed:

x_{max} (`xmax`); x_{min} (`xmin`); y_{max} (`ymax`); y_{min} (`ymin`); Δx (`deltax`); Δy (`deltay`)

For the early prototype the calculations are done using Matlab, the code below implements the described functionality. This code is written to be executed on a set of tracks, which are separated by “NaN”-Elements, which mark the end of a track.

```
%initialzie help variables:
xmax=Tr.x(1);
xmin=Tr.x(1);
ymax=Tr.y(1);
ymin=Tr.y(1);
F1.BBarea(1)=0;
F1.BBratio(1)=0;

elements=size(Tr.x);

for ind1=2:elements(2)
    if (isnan(Tr.x(ind1)) && (ind1 < elements(2)))
        %if there is a new track reset calculation
        xmax=Tr.x(ind1+1);
        xmin=Tr.x(ind1+1);
        ymax=Tr.y(ind1+1);
        ymin=Tr.y(ind1+1);
        F1.BBarea(ind1)=NaN;
        F1.BBarea(ind1+1)=0;
        F1.BBratio(ind1)=NaN;
        F1.BBratio(ind1+1)=0;
        ind1=ind1+2;
    else
        if(Tr.x(ind1)>xmax) %check if current x value is bigger
            xmax=Tr.x(ind1);
        end;
        if(Tr.x(ind1)<xmin) %check if current x value is smaller
            xmin=Tr.x(ind1);
        end;
        if(Tr.y(ind1)>ymax) %check if current y value is bigger
            ymax=Tr.y(ind1);
        end;
        if(Tr.y(ind1)<ymin) %check if current y value is smaller
            ymin=Tr.y(ind1);
        end;

        %recalculate maxima and minima
        deltax=(xmax-xmin);
        deltay=(ymax-ymin);

        %calculate BBarea
        F1.BBarea(ind1)=deltax*deltay;
        if(deltay~=0) %Capture divide by zero and then calculate BBratio
            F1.BBratio(ind1)=deltax/deltay;
        else
            F1.BBratio(ind1)=0;
        end;
    end;

end;

%keep the original timestamp in the feature
F1.tt=Tr.tt;
```

3.1.2.4 Speed

Another extracted feature is the speed of the expansion of a track. This expansion speed is calculated using the distance between two points and also use the time difference Δt to calculate the speed. Again the variables Δx (`deltax`) and Δy (`deltay`) represent the distance (`dist`) between the points in a Cartesian coordinate system, calculation is done the same way as for length (using Pythagoras Theorem). As the speed is defined as covered distance divided by time the differential timestamp Δt (`Tr.t`) is used exactly for this calculation.

$$tr_{speed} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{\Delta t}$$

For the early prototype the calculations are done using Matlab, the code below implements the described functionality. This code is written to be executed on a set of tracks, which are separated by "NaN"-Elements, which mark the end of a track.

```
%calculate delax and deltay
deltax=diffn(Tr.x,param);
deltay=diffn(Tr.y,param);

%calculae distance using pytagoras
dist=sqrt(deltax.^2+deltay.^2);

%add NaN at end of dist (dist has now the length of the original data)
dist=[NaN dist];

elements=size(dist);

F1.speed=zeros(1,elements(2));
NaNfound=false;

for ind1=1:elements(2)
    if isnan(dist(ind1))
         %first NaN marks the beginning of a track, length is 0 at this
         %point
        if(~NaNfound)
            F1.speed(ind1)=0;
            NaNfound=true;
         %second NaN is the track seperator - should be kept
        else
            F1.speed(ind1)=NaN;
            NaNfound=false;
        end;
    else
         F1.speed(ind1)=(dist(ind1))/Tr.t(ind1);
    end;
end;

%keep the original timestamp in the feature
F1.tt=Tr.tt;
```

3.1.3 Gesture analysis and Visual Inspection

To identify strong features it is required to perform a visual inspection of the collected features for each specific gesture. All collected feature vectors of a certain type of track are therefore overlaid, to distinguish certain accumulations in the features. Therefore all Feature vectors are processed to have a common start point in time – which is zero, marking the beginning of a track.

```
tempt=[NaN FV(1,1:end)];
nansource=[NaN FV(2,1:end)];

elements=size(tempt);

tout=zeros(1,elements(2));
NaNfound=false;

%calculate the timstap relative to the beginning of a track
for ind1=1:elements(2);
    if isnan(nansource(ind1))
        tout(ind1)=NaN;
        NaNfound=true;
    else
        if(NaNfound);
            tout(ind1)=0;
            NaNfound=false;
        else
            tout(ind1)=tout(ind1-1)+(tempt(ind1)-tempt(ind1-1));
        end;
    end;
end;

%keep the modified feature vector.
mod_FV=[tout(2:end);FV(2:end,:)];
```

The so processed data is used to generate a map of containing all the features for all specified kind of gestures. In this map obvious differences in the features are clearly visible. If differences in the features are clearly visible by the human eye, it is highly possible, that these differences can also be used by algorithm to identify a gesture. The following Matlab Code produces such an overview. The results of such a Plot can be seen in Figure 6 and Figure 7.

```
%load names and features
names={'leftdown';'rightup';'rightdown';'roof';'wave'};
FVtoload={F_leftdown1;F_rightup1;F_rightdown1;F_roof1;F_wave1};
tracks={Tr_leftdown;Tr_rightup;Tr_rightdown;Tr_roof;Tr_wave};

elements=size(names);
elements=elements(1);
numplots=7; %number of features, currently 7
count=0;

figure('units','normalized','position',[0,0,1,1]);

for i=1:elements
```

```
    %FV=[F1.tt;F1.tracklenght;F2.alpha;F3.alpha;F4.speed;
    %    F5.BBratio;F5.BBarea];

    FV=FVtoload{i,1};
    FV=Feature_plot_preparation(FV);
    track=tracks{i,1};

    % generate a plot… On top: recorded tracks, below: all extracted
    % features put a certain number of tracks and features next to each
    % other

    subplot(numplots,elements,i);
    plot3(track.x,track.y,track.tt);
    title(names(i));
    subplot(numplots,elements,i+elements);
    plot(FV(1,:),FV(2,:),'.')
    set(gca,'xlim',[0,5000],'ylim',[0,200]);
    title(strcat(names(i),' length'));
    subplot(numplots,elements,i+(elements*2));
    plot(FV(1,:),FV(3,:),'.')
    set(gca,'xlim',[0,5000],'ylim',[-4,4]);
    set(gca,'YTick',[-pi -pi/2 0 pi/2 pi]);
    title(strcat(names(i),' orient'));
    subplot(numplots,elements,i+(elements*3));
    plot(FV(1,:),FV(4,:),'.')
    set(gca,'xlim',[0,5000],'ylim',[-4,4]);
    set(gca,'YTick',[-pi -pi/2 0 pi/2 pi]);
    title(strcat(names(i),' orient (begin to end)'));
    subplot(numplots,elements,i+(elements*4));
    plot(FV(1,:),FV(5,:),'.')
    set(gca,'xlim',[0,5000],'ylim',[0,0.6]);
    title(strcat(names(i),' speed'));
    subplot(numplots,elements,i+(elements*5));
    plot(FV(1,:),FV(6,:),'.')
    set(gca,'xlim',[0,5000],'ylim',[0,10]);
    title(strcat(names(i),' BBratio'));
    subplot(numplots,elements,i+(elements*6));
    plot(FV(1,:),FV(7,:),'.')
    set(gca,'xlim',[0,5000],'ylim',[0,2000]);
    title(strcat(names(i),' BBarea'));

    count=count+numplots;

end
```

For each collected feature a histogram is generated and analyzed, searching for the maximum accumulation of data points and the magnitude of the feature. A combination of these results is used for a simple rule based decision mechanism, which will then select the gesture. So for example if a high accumulation around –pi/2 is detected and all the other features show no special behaviour, the gesture “right” is selected.

This decision is then handed over to the demonstration GUI, which will then perform an action according to the received gesture, like moving the selection field on step to the right side. The Demonstration GUI is described in [6].

4 Example results

This chapter shows example data of gestures, tracks and features stemming from the different processing steps of the software prototype as described in the previous chapter. Effects occurring in real data are described and discussed with respect to their influence on the stability and the result of the recognition.

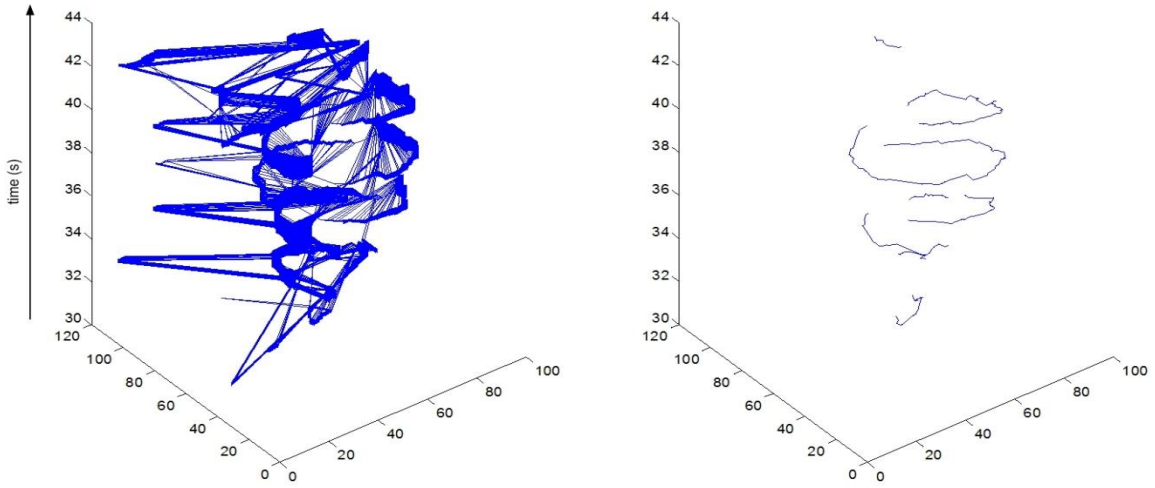

Figure 4 Recorded tracks for a circular gesture, unfiltered data containing duplicate tracks and outliners on the left side, data after processing on the right side

Using a space-time representation Figure 4 shows the unprocessed recorded tracks for a circular gesture on the left side. It is clearly visible, that unfiltered track information contains a lot of useless data and no clear tracks are visible within this enormous amount of track information. Therefore the outliners, duplicate tracks and much too short tracks are removed from the data, just keeping reasonable tracks which allow a clear interpretation in the further processing steps (right side). These tracks are then used for feature extraction as described above.

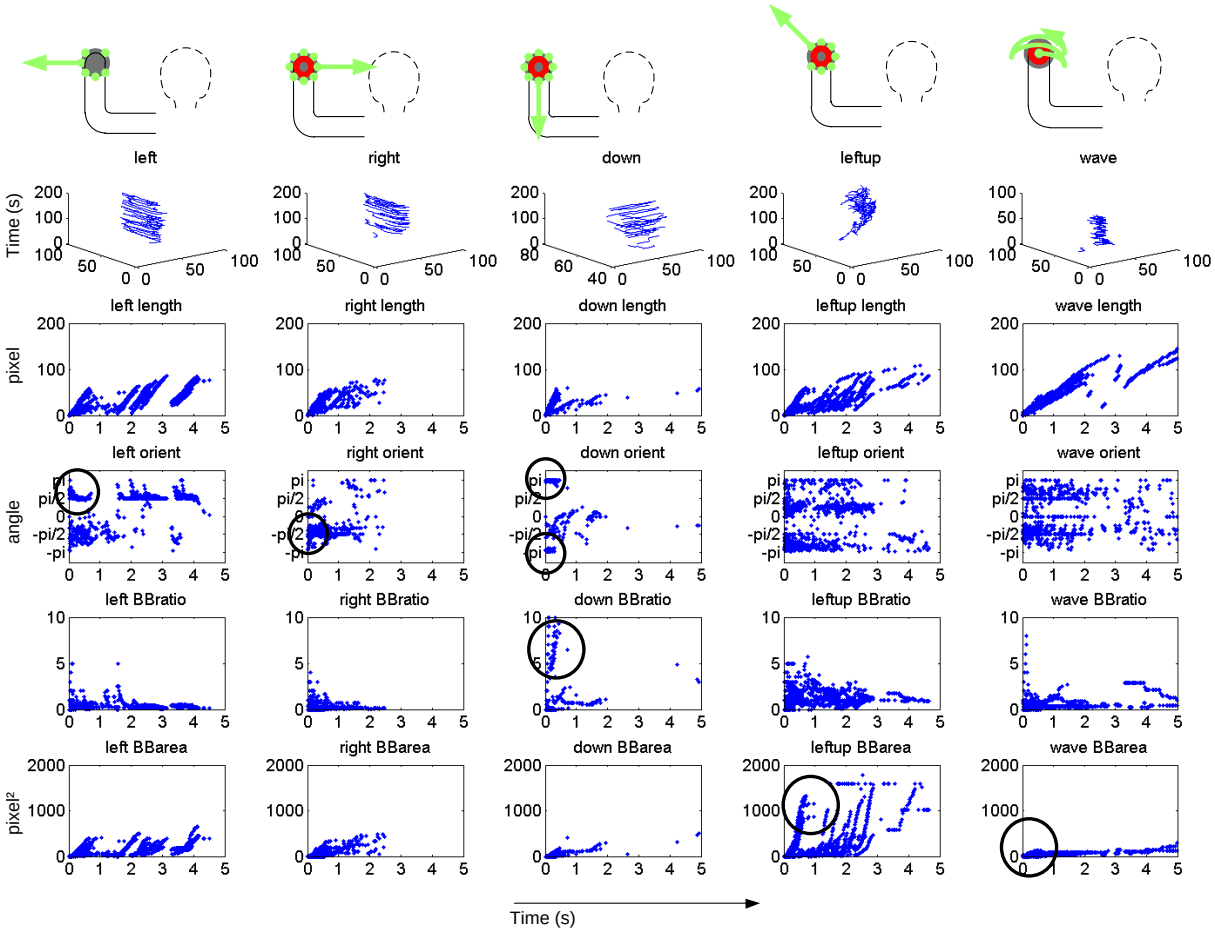

Figure 5 Extracted features for 5 different gestures.

Gesture data are organized in columns in the Figure 5. The previously recorded ideal test tracks are shown in a space-time presentation the top row axes. The extracted features track length, orientation, bounding box ratio and bounding box area for the first 5 seconds of each track are shown in the four rows below. Each gesture is represented by a set of significant values of some of the features. Significant feature values are indicated by black circles.

The length of a track is the sum of distance between all points of the track. Since this feature depends on the amplitude of the gesture-movement length will not be a strong feature.

A much stronger feature is the orientation of the gesture movement, since each gesture has a distinguished direction, which can also be detected on sensor data. In Figure 5 the orientation is labelled as “angle” and as marked, accumulations

The Bounding-Box Area is not independent to the amplitude of the gesture-movement, but highly characteristic for each gesture and therefore a strong feature. Another very strong feature is the Bounding-Box ratio, which is also very characteristic for each gesture.

5 References

[1] UCOS2 data sheet - http://www.ait.ac.at/research-services/research-services-safety-security/new-sensor-technologies/development-of-embedded-systems-for-customer-specific-solutions/smart-eye-ucos-universal-counting-sensor/?L=1 as of 2011-08-31

[2] "Lichtsteiner, P.; Posch, C.; Delbruck,; " A 128x128 120dB 30mW asynchronous vision sensor that responds to relative intensity change"; IEEE International Solid-State Circuits Conference, ISSCC 2006; San Francisco; S³ Digital Publishing, Inc., Lisbon Falls, Maine; ISBN:1-4244-0079-1; pp. 508-509, 669; February, 5.-9., 2006.

[3] Milosevic, N.; Schraml, S.; Schön, P.; "Smartcam for real-time stereo vision – address-event based Stereo Vision"; INSTICC – Inst. f. systems and technologies of information, control & communication; INSTICC Press, Portugal; ISBN: 978-972-8865-74-0; p. 466-471, March, 8-11, 2007

[4] Mead, Carver Silicon retina. In: Mead C, editor. Analog VLSI and neural systems. Reading, Mass: Addison-Wesley, 1989:257–78.

[5] FoSIBLE Deliverable D5.1: "Report on requirements of software components and tools for observation and tracking in laboratory and Living Lab environment." Delivery date M16

[6] FoSIBLE Deliverable D5.5: "Prototypes of evaluation tools to test software components and sensor." Delivery date M18

6 Annex A

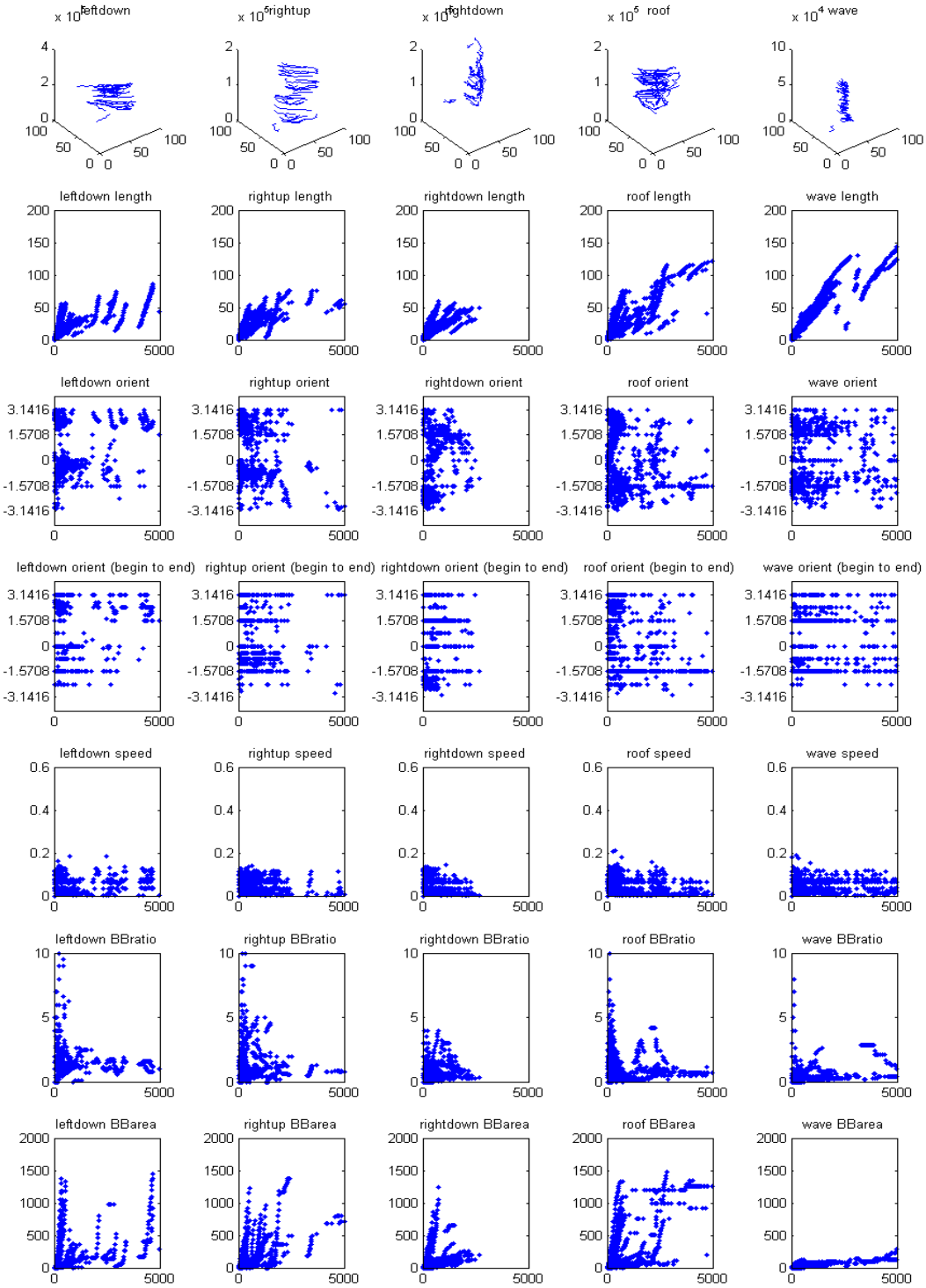

Figure 6 Feature overview for visual inspection for the first set of gestures

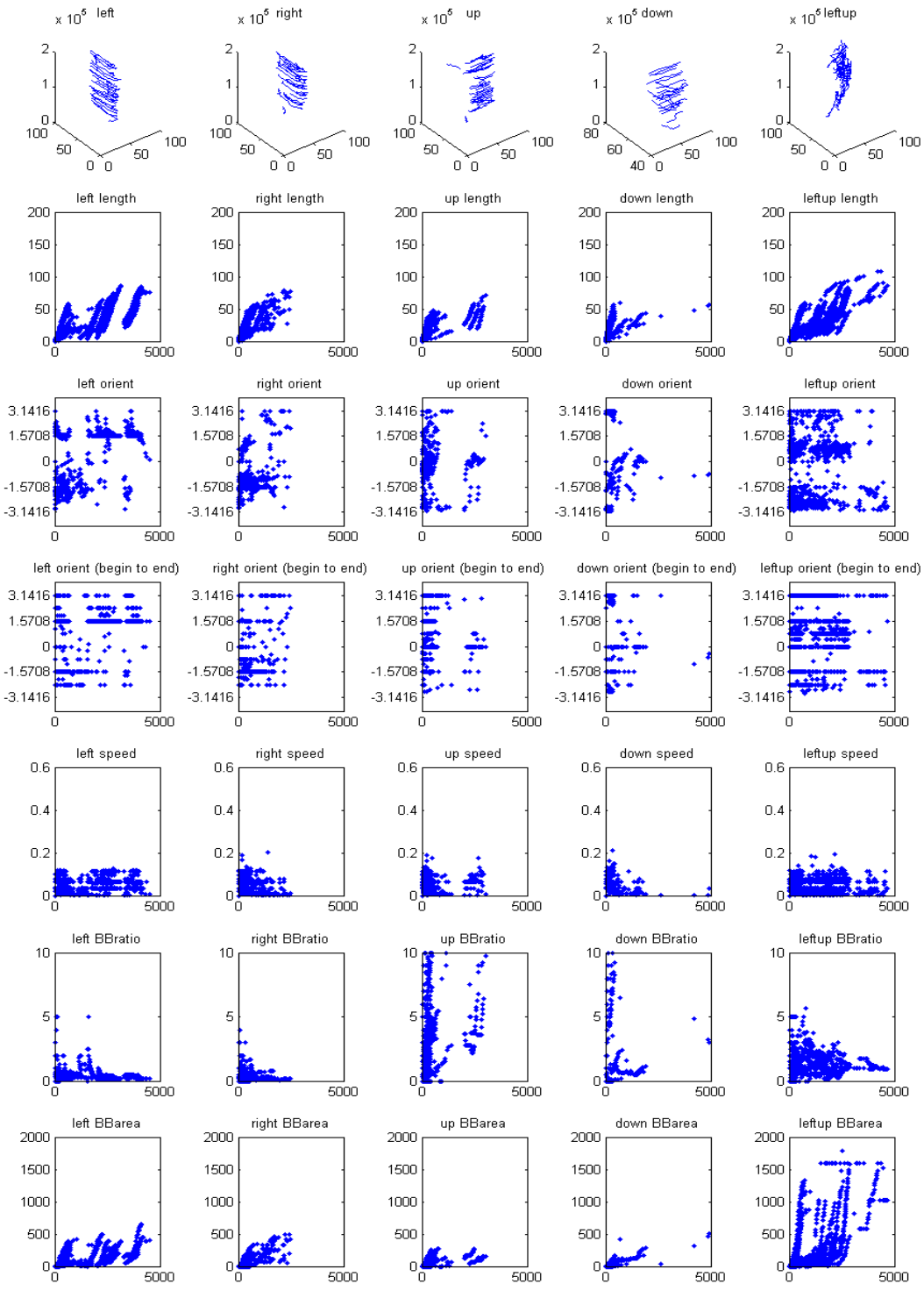

Figure 7 Feature overview for visual inspection for the second set of gestures