POLITECHNIKA GDAŃSKA

Eye tracking

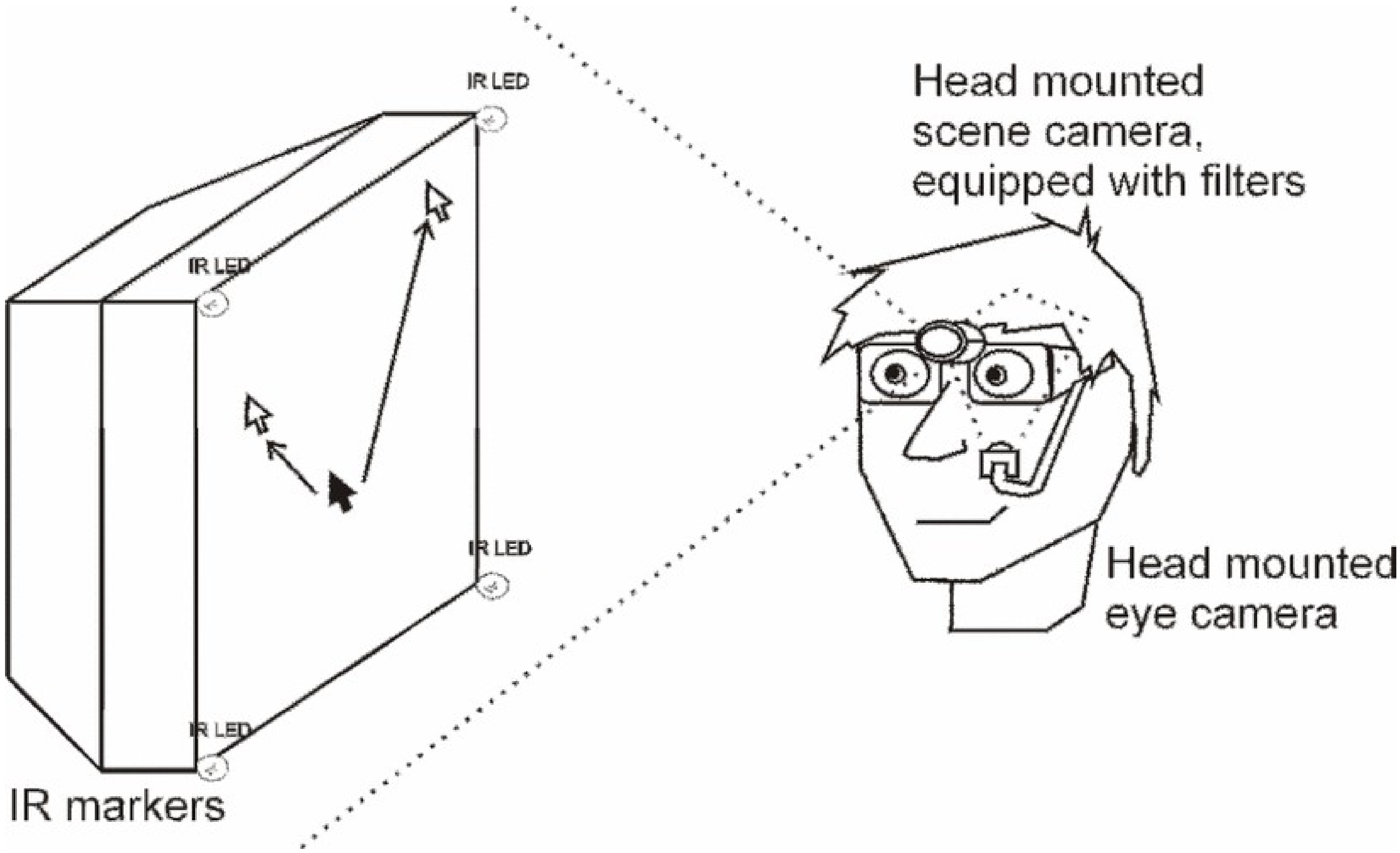
Tomasz Kocejko

POLITECHNIKA GDAŃSKA

Eye tracking

- Operated lights and appliances remotely
- Control infrared devices such as televisions and stereos
- Surf the Web and send emails
- Wirelessly control his own PC or Mac, using the Computer Access program’s on-screen keyboard and mouse
- Store and play music
- Organize photos and home movies and view them
- Read books on Kindle
- Watch YouTube videos
- Use a word processor

Eye tracking interface



Gstreamer support

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    if (!g_thread_supported ())
        g_thread_init (NULL);

    //initiate gstreamera
    gst_init (&argc, &argv);
    QApplication a(argc, argv);
    a.connect(&a, SIGNAL(lastWindowClosed()), &a, SLOT(quit ()));
    MainWindow w;
    w.show();

    return a.exec();
}
```

Register and initiate Gstreamer in main class

POLITECHNIKA GDAŃSKA

Gstreamer support

```
void cam1_thread::initiateGst(QString pipe_name)
{
    qDebug()<<"OK1";
    //The pipelines
    //sprintf(input_str,"v4l2src device=/dev/video0 ! video/x-raw-yuv, framerate=10/1, width=320, height=240 ! ffmpegcolorspace ! xvimagesink");
    //sprintf(entrada_str,"qtkitvideosrc ! video/x-raw-yuv,width=640,height=480 ! ffmpegcolorspace "
    //                      "! video/x-raw-rgb,format=RGB3,width=640,height=480 ! appsink name=sink");

    GError *error = NULL;
    //input_pipe = gst_parse_launch("v4l2src device=/dev/video0 ! video/x-raw-yuv, width=640, height=480 ! appsink name=asink sync=true",&error);
    input_pipe = gst_parse_launch(pipe_name.toLocal8Bit().constData(),&error);
    //input_pipe = gst_parse_launch("v4l2src device=/dev/video1 always-copy=false ! video/x-raw-yuv, width=320, height=240, framerate=20/1, bpp=24 ! ffmpegcolorspace ! app
    //input_pipe = gst_parse_launch("videotestsrc ! video/x-raw-yuv, format=(fourcc)YUY2;video/x-raw-yuv,format=(fourcc)YV12 ! appsink name=asink sync=true",&error);

    sink = gst_bin_get_by_name (GST_BIN (input_pipe), "asink");

    appsink = (GstAppSink *)sink;
    gst_app_sink_set_max_buffers ( appsink, 2); // limit number of buffers queued
    gst_app_sink_set_drop( appsink, true ); // drop old buffers in queue when ful

    gst_app_sink_set_emit_signals(appsink, true);
    //gst_app_sink_set_max_buffers(appsink, 1);

    qDebug()<<"OK2";

    if (error != NULL) {
        g_print ("could not construct pipeline: %s\n", error->message);
        g_error_free (error);
        exit (-1);
      }

qDebug()<<"OK3";

    // prepare the pipeline
    pipeline = gst_pipeline_new ("xvoverlay");

    src = gst_element_factory_make ("v4l2src", NULL);
    g_object_set(G_OBJECT(src),"device",dev_name.toLocal8Bit().constData(),NULL);
qDebug()<<"OK4";
    //capsfilter_1 = gst_element_factory_make("capsfilter","cfilter");
    //g_object_set(G_OBJECT(capsfilter_1),"caps",gst_caps_from_string("video/x-raw-yuv, width=(int)320, height=(int)240, framerate=(fraction)20/1"),NULL);
    qDebug()<<"OK5";
}
```

Initiate and prepare the pipeline, set pipeline sink, set stream properties, set number of buffers queued

Gstreamer support

Check the gstreamer pipeline state and return error in case failure

```
GstStateChangeReturn sret = gst_element_set_state (input_pipe, GST_STATE_PLAYING);
if (sret == GST_STATE_CHANGE_FAILURE) {
    gst_element_set_state (input_pipe, GST_STATE_NULL);
    gst_object_unref (input_pipe);
    // Exit application
    //QTimer::singleShot(0, QApplication::activeWindow(), SLOT(quit()));
    qDebug()<<"cos sie dzieje";
}
```

Gstreamer support

```
if( !gst_app_sink_is_eos(appsink) )
{
    gint fwidth, fheight, fdepth;
    GstBuffer *buffer = gst_app_sink_pull_buffer(appsink);
    uint8_t* data = (uint8_t*)GST_BUFFER_DATA(buffer);

    GstCaps *capss = gst_buffer_get_caps(buffer);
    GstStructure *caps_struct = gst_caps_get_structure(capss,0);
    //GstVideoFormat *vformat;
    gst_structure_get_int(caps_struct, "width", &(fwidth));
    gst_structure_get_int(caps_struct, "height", &(fheight));
    gst_structure_get_int(caps_struct, "depth", &fdepth);
    //gst_video_format_parse_caps(capss, vformat, NULL, NULL);
    int xwidth = 0, xheight = 0;

    GstVideoFormat xformat;
    if (!gst_video_format_parse_caps(capss, &xformat, &xwidth, &xheight)) {
        gst_caps_unref(capss);
        //qDebug()<<gst_video_format_parse_caps(capss, &xformat, &xwidth, &xheight);
    }
    gst_caps_unref(capss);
    //qDebug()<<"Jest "<<gst_video_format_parse_caps(capss, &xformat, &xwidth, &xheight);

    //konwersja
    cv::Size ImageSize;
    ImageSize.height = fheight;
    ImageSize.width = fwidth;

    cv::Mat yuv = yuv2rgb(ImageSize, (uchar*)data);
    img = cv::Mat(ImageSize, CV_8UC3);
    cv::cvtColor(yuv, img, CV_YUV2RGB);

    //zwolnienie bufora
    gst_buffer_unref(buffer);
}
```

Read pipeline properties and image buffer.

Call image decode function and image color conversion function.

Set image for further processing

YUYV format decode

Decode image buffer to YUV (3 channel) image matrix

```
//    YUV yuv;
    for(int i = 0, j=0; i < ImgSize.width * ImgSize.height * 3; i+=6, j+=4)
    {
        img2.data[i] = ImgBuffer[j];            //Y
        img2.data[i+1] = ImgBuffer[j+1];        //U
        img2.data[i+2] = ImgBuffer[j+3];        //V
        img2.data[i+3] = ImgBuffer[j+2];        //Y'
        img2.data[i+4] = ImgBuffer[j+1];        //U
        img2.data[i+5] = ImgBuffer[j+3];        //V
    }
```

YUV to RGB color conversion using LUT (lookup table)

```
for(k = 0; k<img2.channels(); k++)
{
    //j=0;
    //int kk = omp_get_thread_num();
    //omp_set_num_threads(3);

    for(it = (width * height*(k))/6; it < (width * height*(k+1))/6; it++)
    {
        int j = it*4;
        int i = it*6;
        img2.data[i] = ImgBuffer[j] + lut.r[ImgBuffer[j+1]];            //Y to R
        img2.data[i+1] = ImgBuffer[j] - lut.g1[ImgBuffer[j+1]] - lut.g2[ImgBuffer[j+3]];        //U to G
        img2.data[i+2] = ImgBuffer[j]+ lut.b[ImgBuffer[j+3]];         //V to B
        img2.data[i+3] = ImgBuffer[j+2] + lut.r[ImgBuffer[j+1]];        //Y'
        img2.data[i+4] = ImgBuffer[j+2] - lut.g1[ImgBuffer[j+1]] - lut.g2[ImgBuffer[j+3]];        //U to G
        img2.data[i+5] = ImgBuffer[j+2] + lut.b[ImgBuffer[j+3]];        //V to B
        //j+=4;
        //counter++;
    }
}
```

POLITECHNIKA GDAŃSKA

NV21 format decode

Decode image buffer to YUV (3 channel) image matrix

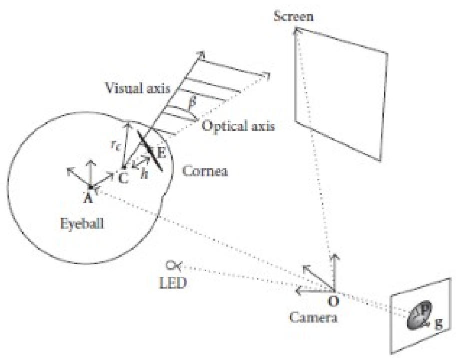
```
int y1,y2, y3,y4, u1,v1;
for(int i=0, kk=0; i<size;i+=2, kk+=2)
{
    y1 = ImgBuffer[i];
    y2 = ImgBuffer[i+1];
    y3 = ImgBuffer[width+i];
    y4 = ImgBuffer[width+i+1];

    u1 = ImgBuffer[size+kk];
    v1 = ImgBuffer[size+kk+1];

    img2.data[i]=y1;
    img2.data[i+1]=y2;
    img2.data[i+width]=y3;
    img2.data[i+width+1]=y4;

    u.data[i]=u1;
    u.data[i+1]=u1;
    u.data[i+width]=u1;
    u.data[i+width+1]=u1;

    v.data[i]=v1;
    v.data[i+1]=v1;
    v.data[i+width]=v1;
    v.data[i+width+1]=v1;

    if(i!=0 && (i+2)%width == 0)
        i +=width;
}

std::vector<cv::Mat> array;
array.push_back(img2);
array.push_back(u);
array.push_back(v);
cv::merge(array, out);
```

YV12 format decode

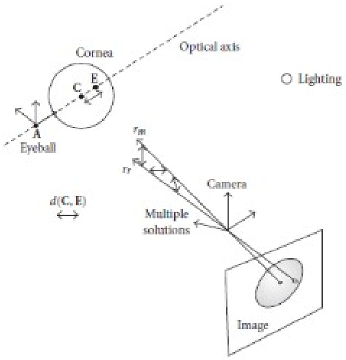
Decode image buffer to YUV (3 channel) image matrix

```
if(YV12)
{
        y.data = ImgBuffer;
        int size = ImgSize.width*ImgSize.height;
        //int BuffSize = 1.5*size;
        int u1, v1;
        //int y1, y2, y3, y4;
        int k =0;
        for(int i=0; i<size; i+=2)
        {
            /* ...*/

            u1 = ImgBuffer[size+k];
            v1 = ImgBuffer[size+k+size/4];

            u.data[i] = u1;
            u.data[i+1] = u1;
            u.data[ImgSize.width+i] = u1;
            u.data[ImgSize.width+i+1] = u1;

            v.data[i] = v1;
            v.data[i+1] = v1;
            v.data[ImgSize.width+i] = v1;
            v.data[ImgSize.width+i+1] = v1;

            if (i!=0 && (i+2)%ImgSize.width==0)
                i+=ImgSize.width;

            k+=1;
        }
        cv::vector<cv::Mat> array;
        array.push_back(y);
        array.push_back(u);
        array.push_back(v);
        cv::merge(array, img2);
}
```

Fixation points estimation methods



(a)

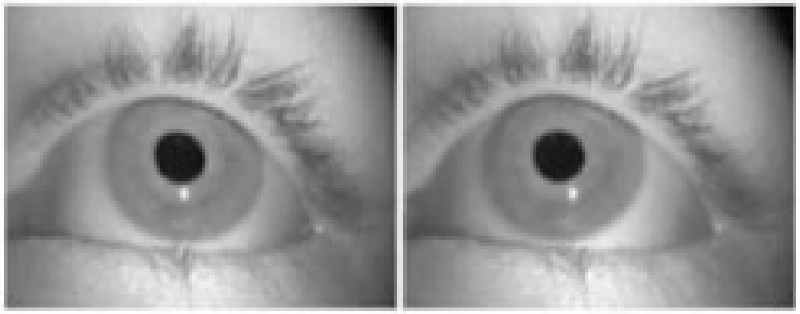


(b)

- Accuracy 0,7° - 1,1°

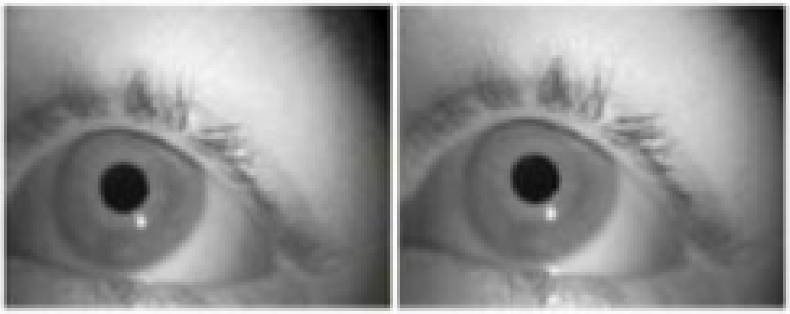
Fixation points estimation methods

$$s_x = a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 + a_5 y^2$$
$$s_y = b_0 + b_1 x + b_2 y + b_3 xy + b_4 x^2 + b_5 y^2$$



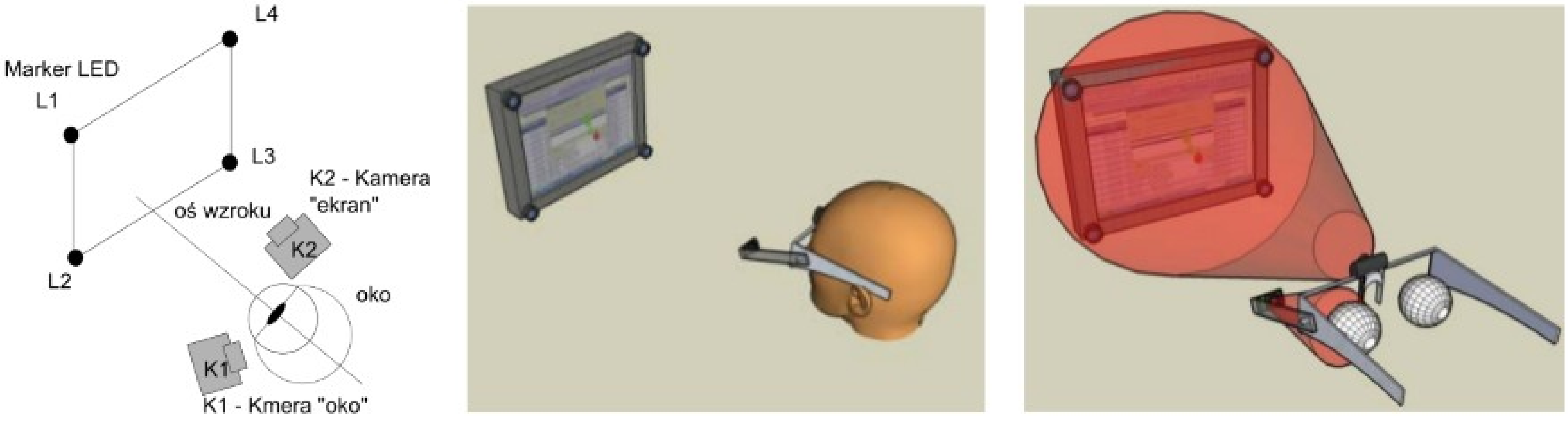
(a)

$$s_x = a_0 + a_1 x_{p\text{-}cr} + a_2 y_{p\text{-}cr} + a_3 x_{p\text{-}cr} y_{p\text{-}cr} + a_4 x_{p\text{-}cr}^2 + a_5 y_{p\text{-}cr}^2,$$
$$s_y = b_0 + b_1 x_{p\text{-}cr} + b_2 y_{p\text{-}cr} + b_3 x_{p\text{-}cr} y_{p\text{-}cr} + b_4 x_{p\text{-}cr}^2 + b_5 y_{p\text{-}cr}^2,$$



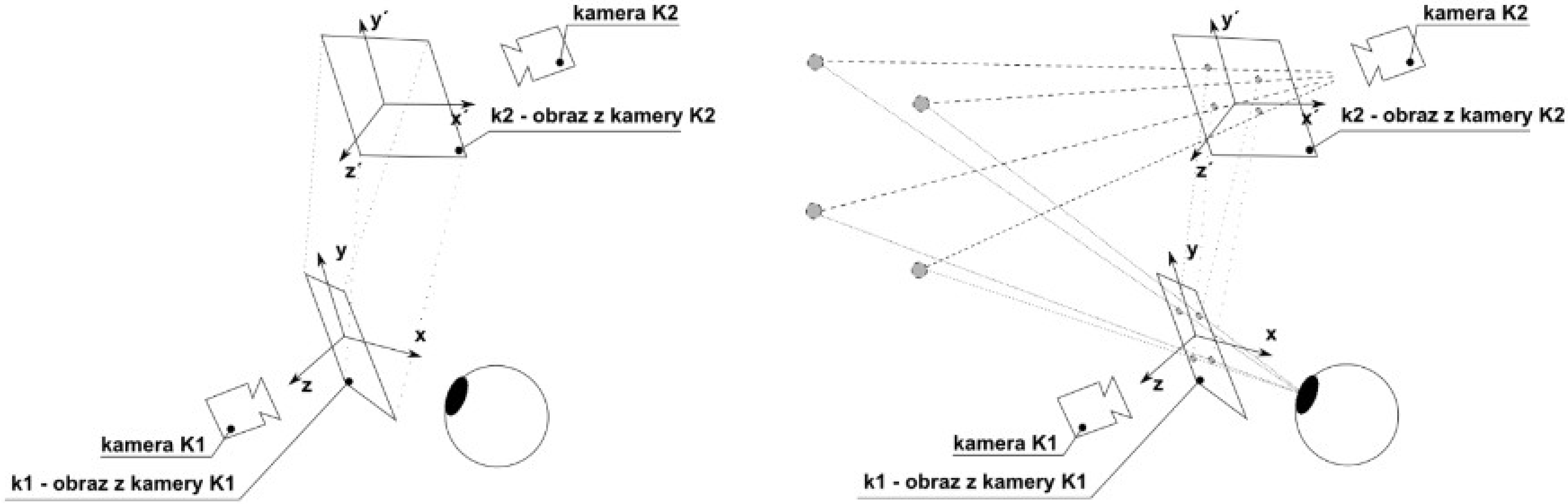
(b)

Fixation points estimation methods

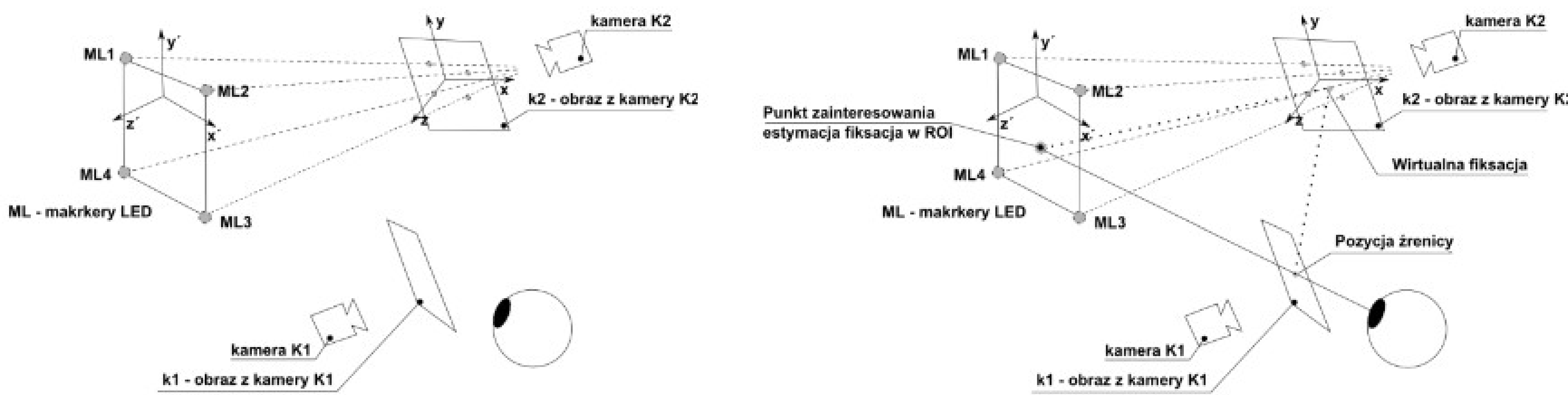


(a) (b) (c)

POLITECHNIKA GDAŃSKA



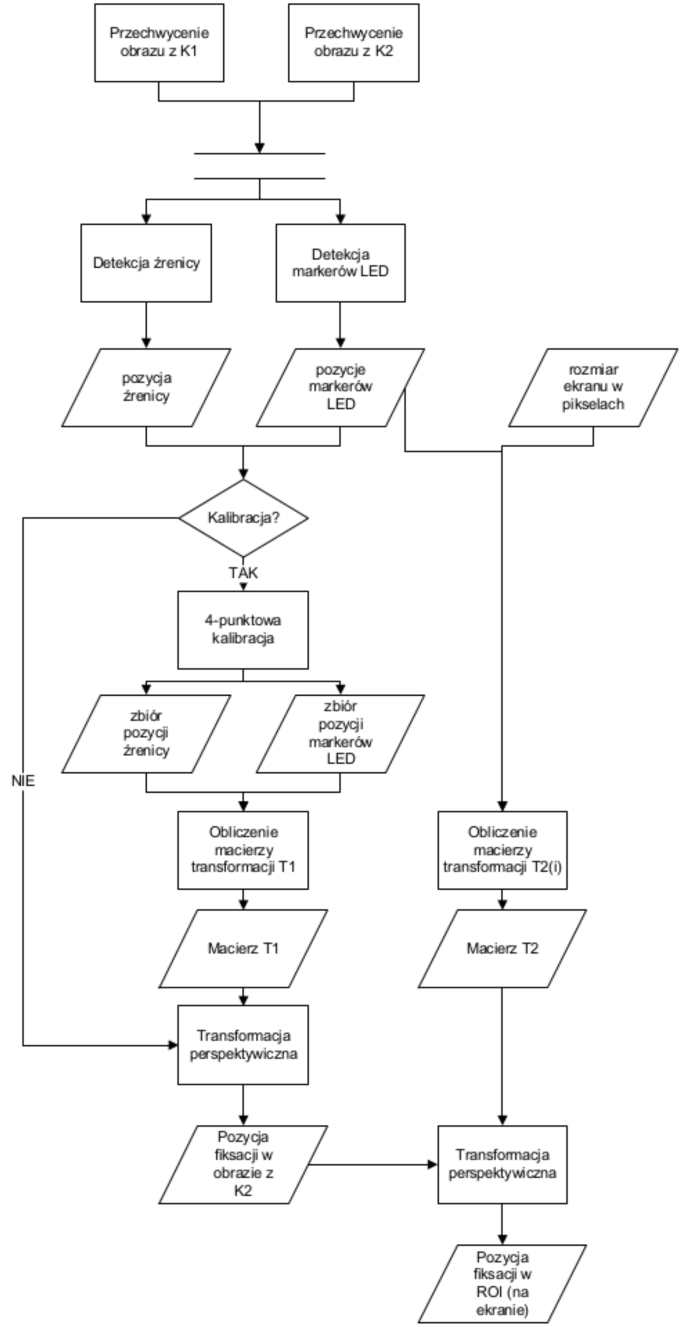
(a) (b)



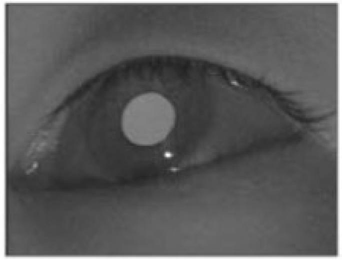
ML1
ML2
ML3
ML4
ML - makrkery LED
kamera K2
k2 - obraz z kamery K2
kamera K1
k1 - obraz z kamery K1
Punkt zainteresowania
estymacja fiksacja w ROI
Wirtualna fiksacja
Pozycja źrenicy

(a) (b)

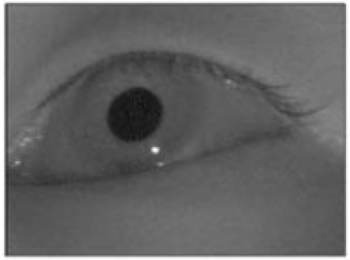
Fixations within ROI



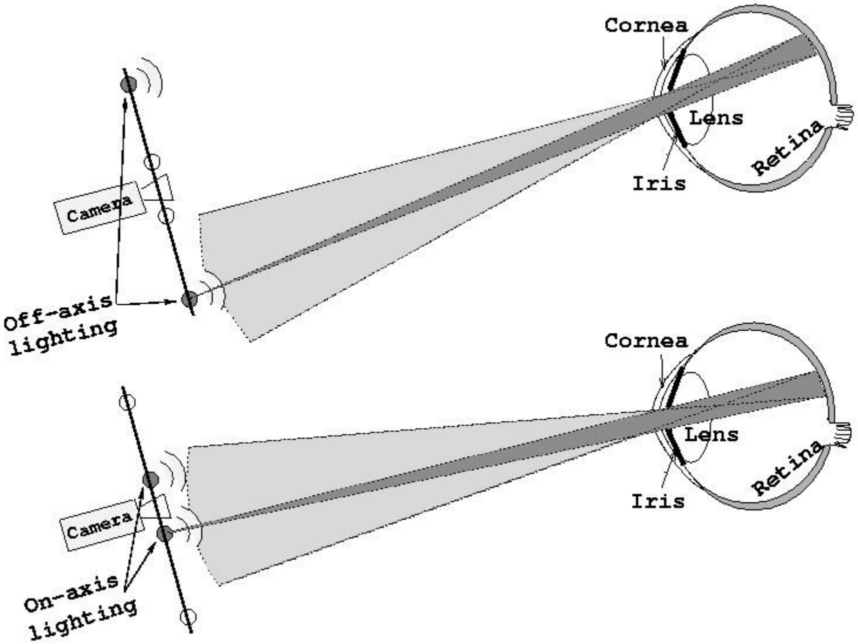
Pupil detection methods

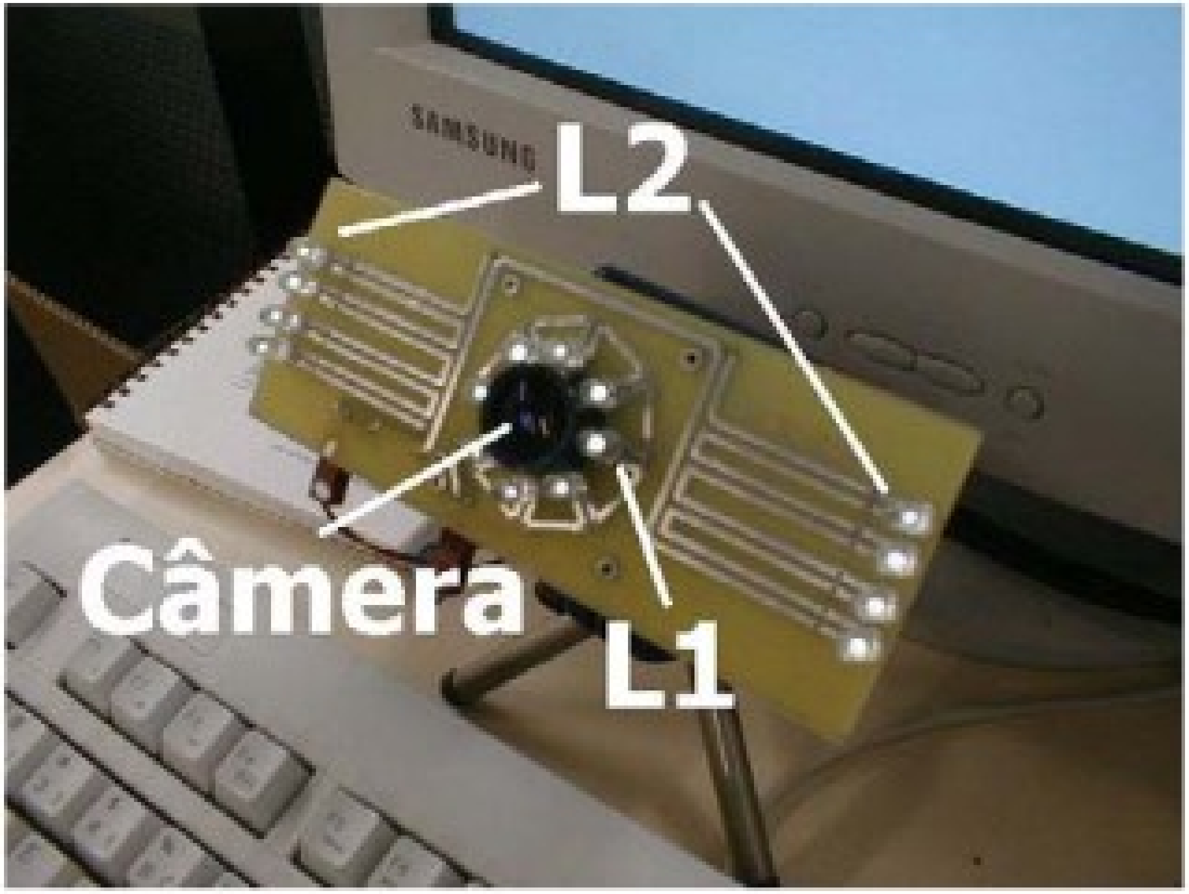


(a)



(b)




SAMSUNG
L2
Câmera
L1

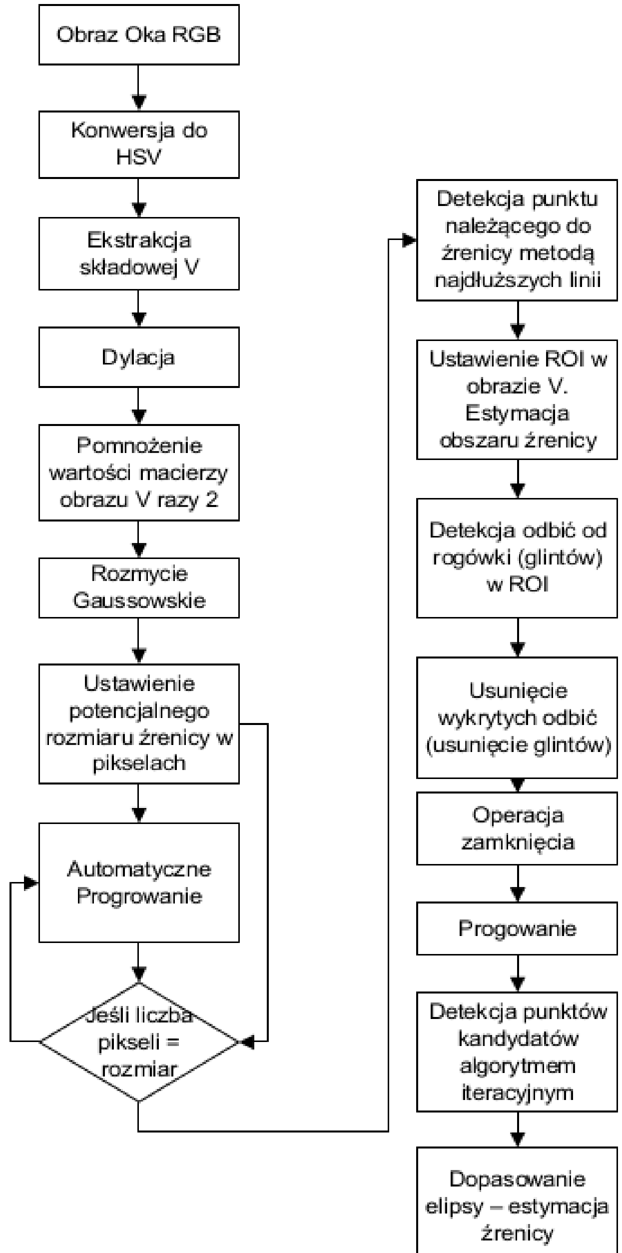
POLITECHNIKA GDAŃSKA

Bright vs dark pupil effect

- The bright pupil method some factors may affect the size of the pupil, such as age and environmental light, may have an impact on trackability of the eye. Ethnicity is also another factor that affects the bright/dark pupil response: the bright pupil method works very well for Hispanics and Caucasians. However, the method has proven to be less suitable when eye tracking Asians for whom the dark pupil method provides better trackability.

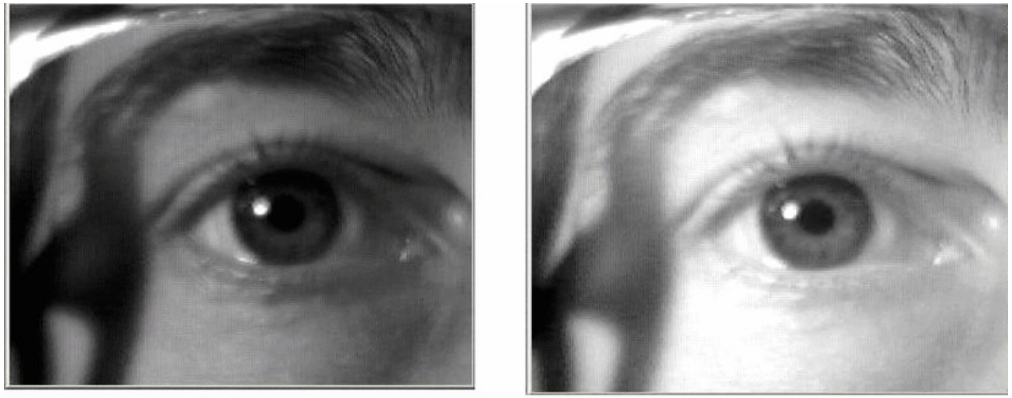
- Dark pupil effect is proved to be more useful under natural light conditions. That is why it is utilized in head mounted devices. The „dark" pupil detection may be easily affected by factors like eyeliners and mascaras

Pupil detection algorithms



Color conversion

- Under IR exposure, to obtain correct pupil position, greyscale image can be processed. Considering pupil detection under ordinary light, it is better to convert captured image to the HSV scale, split image in to three components and extract the Value one.



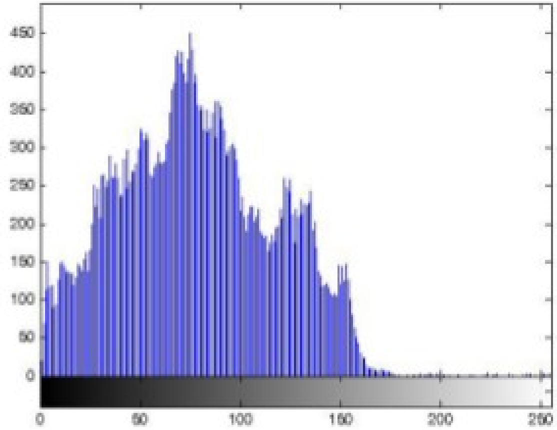
(a) (b)

POLITECHNIKA GDAŃSKA

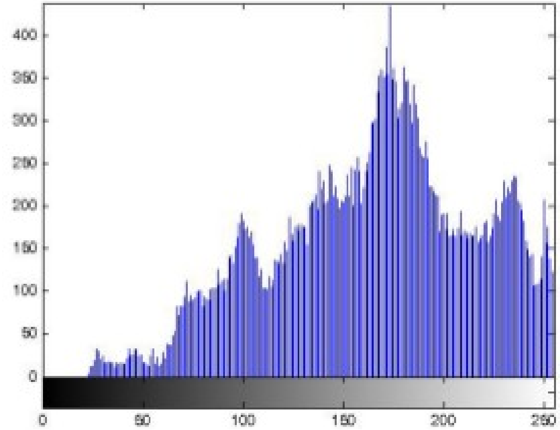
Image segmentation

(a)

(b)



(a)

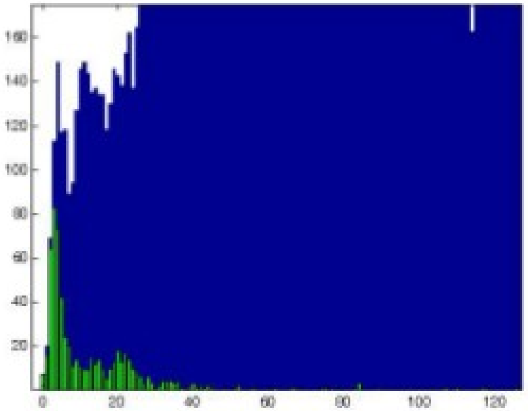


(b)

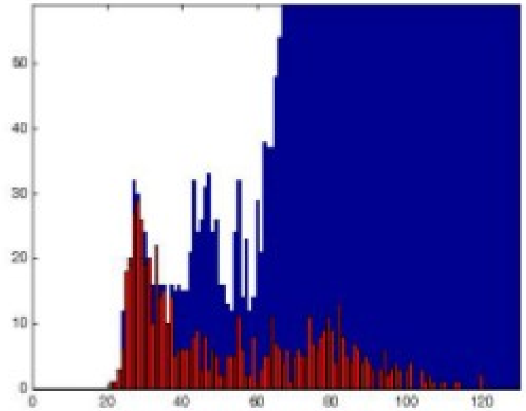
Image segmentation

(a)

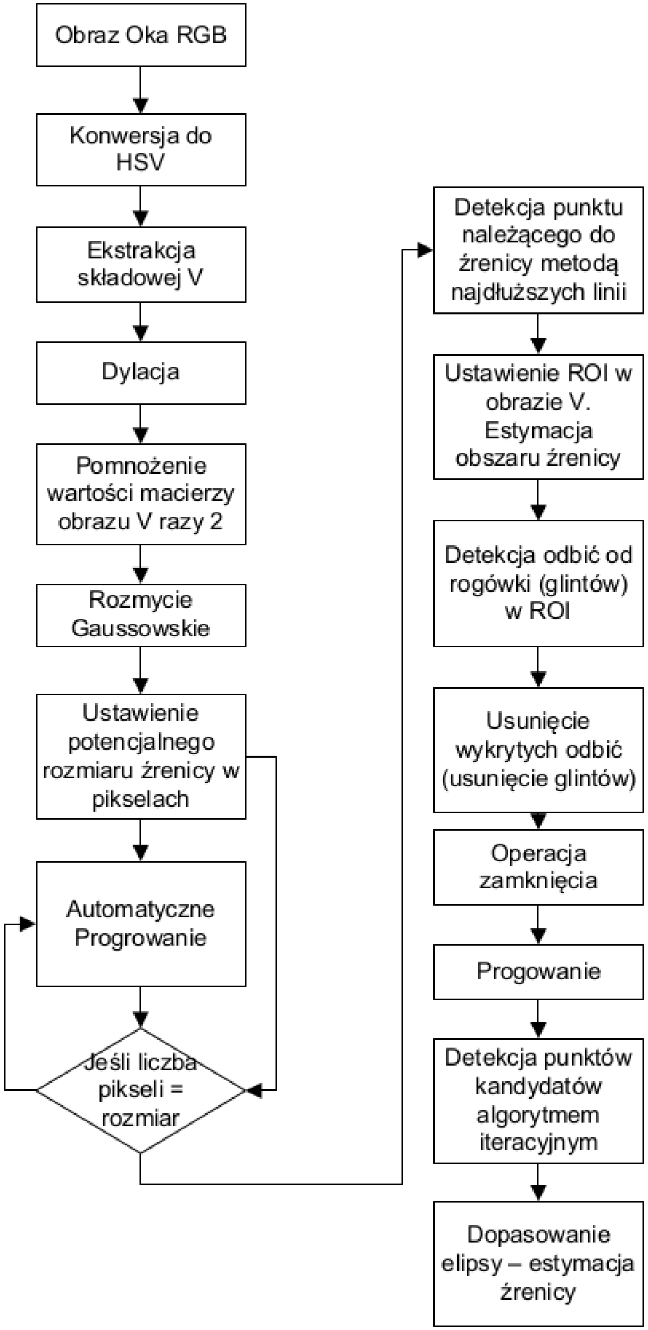
(b)



(a)

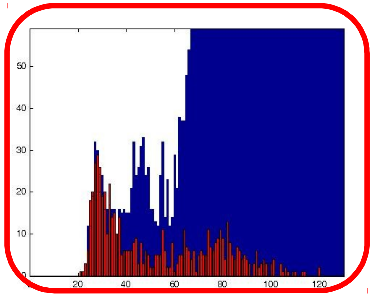


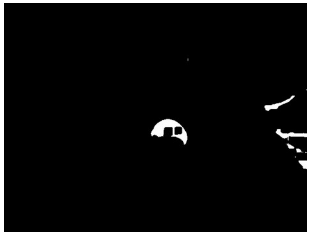
(b)

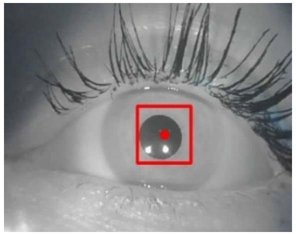

Obraz Oka RGB
Konwersja do HSV
Ekstrakcja składowej V
Dylacja
Pomnożenie wartości macierzy obrazu V razy 2
Rozmycie Gaussowskie
Ustawienie potencjalnego rozmiaru źrenicy w pikselach
Automatyczne Progrowanie
Jeśli liczba pikseli = rozmiar
Detekcja punktu należącego do źrenicy metodą najdłuższych linii
Ustawienie ROI w obrazie V. Estymacja obszaru źrenicy
Detekcja odbić od rogówki (glintów) w ROI
Usunięcie wykrytych odbić (usunięcie glintów)
Operacja zamknięcia
Progowanie
Detekcja punktów kandydatów algorytmem iteracyjnym
Dopasowanie elipsy – estymacja źrenicy

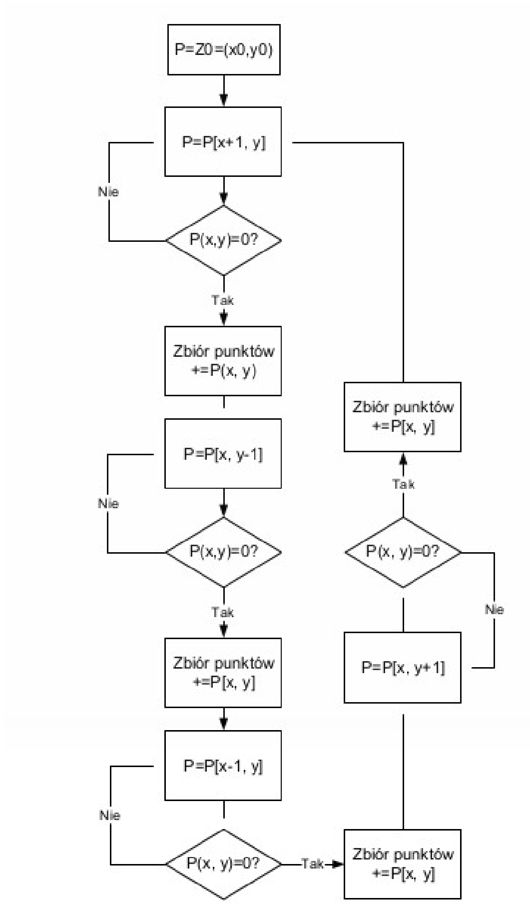
Eye tracking

V component extraction
↓
Morphological operations
↓
Features enhancement
↓
Automatic threshold
↓
Pupil region detection
↓
Glints removal
↓
Pupil detection

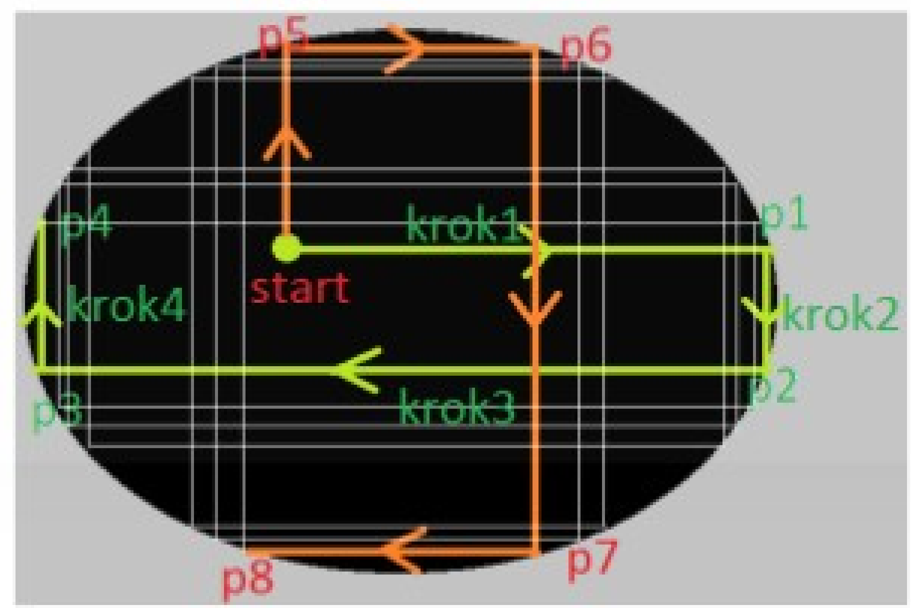








P=Z0=(x0,y0)
P=P[x+1, y]
Nie
P(x,y)=0?
Tak
Zbiór punktów
+=P(x, y)
P=P[x, y-1]
Nie
P(x,y)=0?
Tak
Zbiór punktów
+=P[x, y]
P=P[x-1, y]
Nie
P(x, y)=0?
Tak
Zbiór punktów
+=P[x, y]
P=P[x, y+1]
Nie
P(x, y)=0?
Tak
Zbiór punktów
+=P[x, y]



p5
p6
p1
p4
krok1
start
krok4
krok2
p2
p3
krok3
p7
p8

Eye tracking

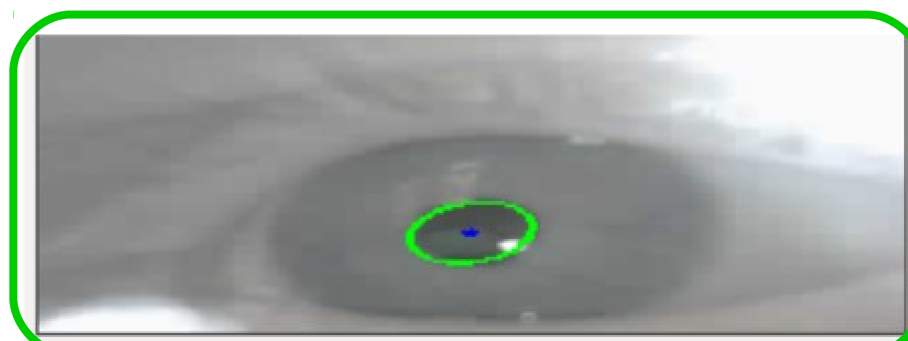
- Pupil detection
- Find contours
- Fill holes

Left branch:

- Approximate size with rectangle
- Look for round shape
- Approximate pupil with circle

Right branch:

- Threshold tunning
- Pupil center candidate det.
- Pupil boundary points dete
- Ellipse fitting

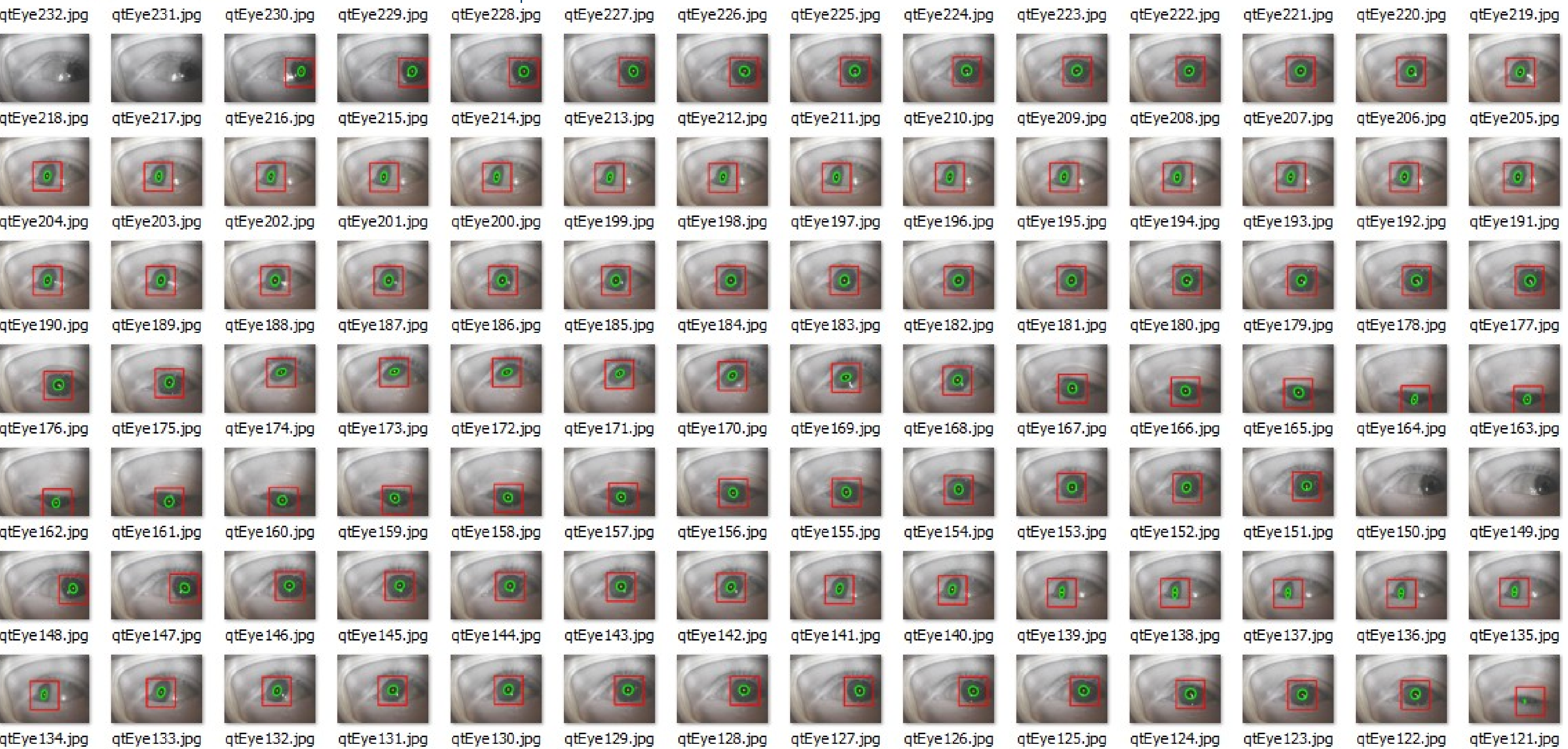


```
// Find all contours
std::vector<std::vector<cv::Point> > contours;
cv::findContours(g1.clone(), contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);

// Fill holes in each contour
cv::drawContours(g1, contours, -1, CV_RGB(255,255,255), -1);

pdet->PupilDetAlg22(g1,stp, pupilc);

std::vector<cv::Point> ellipse;
for(int i=0;i<pdet->vert_p.size();i++)
{
    ellipse.push_back(pdet->vert_p[i]);
    //qDebug()<<pdet->vert_p[i].x<<" ORAZ "<<pdet->vert_p[i].y;
    //cv::circle(pupilROI, pdet->vert_p[i], 3, CV_RGB(255,0,0), -1);
    //cv::circle(pupilROI, pdet->hor_p[i], 3, CV_RGB(255,255,0), -1);
}
//qDebug()<<pdet->hor_p.size();
for(int i=0;i<pdet->hor_p.size();i++)
{
    ellipse.push_back(pdet->hor_p[i]);
    //qDebug()<<pdet->vert_p[i].x<<" ORAZ "<<pdet->vert_p[i].y;
    //cv::circle(pupilROI, pdet->vert_p[i], 3, CV_RGB(0,255,0), -1);
    //cv::circle(pupilROI, pdet->hor_p[i], 3, CV_RGB(255,255,0), -1);
    //qDebug()<<"p"<<i<<": "<<pdet->hor_p[i].x<<" --- "<<pdet->hor_p[i].y;
}
if(ellipse.size()>6)
{

    cv::RotatedRect r = cv::fitEllipse(ellipse);
    cv::ellipse(pupilROI, r, CV_RGB(0,255,0), 3, 8);
    cv::circle(pupilROI, r.center, 3, CV_RGB(0,0,255), -1);
    pupilc.x = r.center.x + rectBig.x;
    pupilc.y = r.center.y + rectBig.y;
    PupilSzie = r.size.area();
}
else
{

    //qDebug()<<"EYES CLOSED";
    pupilc = roicBig;
}
```

```
// Find all contours
std::vector<std::vector<cv::Point> > contours;
cv::findContours(g1.clone(), contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);

// Fill holes in each contour
cv::drawContours(g1, contours, -1, CV_RGB(255,255,255), -1);

for (int i = 0; i < (int)contours.size(); i++)
{
    double area = cv::contourArea(contours[i]);    // Blob area
    cv::Rect rect = cv::boundingRect(contours[i]); // Bounding box
    int radius = rect.width/2;                     // Approximate radius

    // Look for round shaped blob
    if (area >= 30 &&
        std::abs(1 - ((double)rect.width / (double)rect.height)) <= 0.4 &&
        std::abs(1 - (area / (CV_PI * std::pow(radius, 2)))) <= 0.4)
    {
        //cv::circle(pupilROI_dst, cv::Point(rect.x + radius, rect.y + radius), radius, CV_RGB(255,0,0), 2);
        cv::circle(img, cv::Point((rect.x + radius)+rectBig.x, (rect.y + radius)+rectBig.y), radius, CV_RGB(255,0,0), 2);
        pupilc.x = (rect.x + radius)+rectBig.x;
        pupilc.y = (rect.y + radius)+rectBig.y;
    }
}
```

POLITECHNIKA GDAŃSKA

Eye tracking



For each 10 tested subjects 100 random samples were selected. Sensitivity = 98%

POLITECHNIKA GDAŃSKA

Algorytmy kompensacji ruchów głowy

ALGI

Wyznaczenie macierzy T1 raz podczas kalibracji

Estymacja pozycji źrenicy w obrazie K2

Wyznaczenie macierzy T2 dla każdej zarejestrowanej klatki

Estymacja pozycji fiksacji w ROI

$$\begin{bmatrix} X_f Z_r \\ Y_f Z_r \\ Z_r \end{bmatrix}_i = T2 \cdot \begin{bmatrix} x_v \\ y_v \\ 1 \end{bmatrix}_i = T2(i) * T1 \cdot \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix}_i$$

i – numer przechwyconej klatki

ALGII

Rejestracja danych do obliczenia macierzy T1 podczas kalibracji

Wyznaczenie przesunięcia na podstawie obrazu z K2

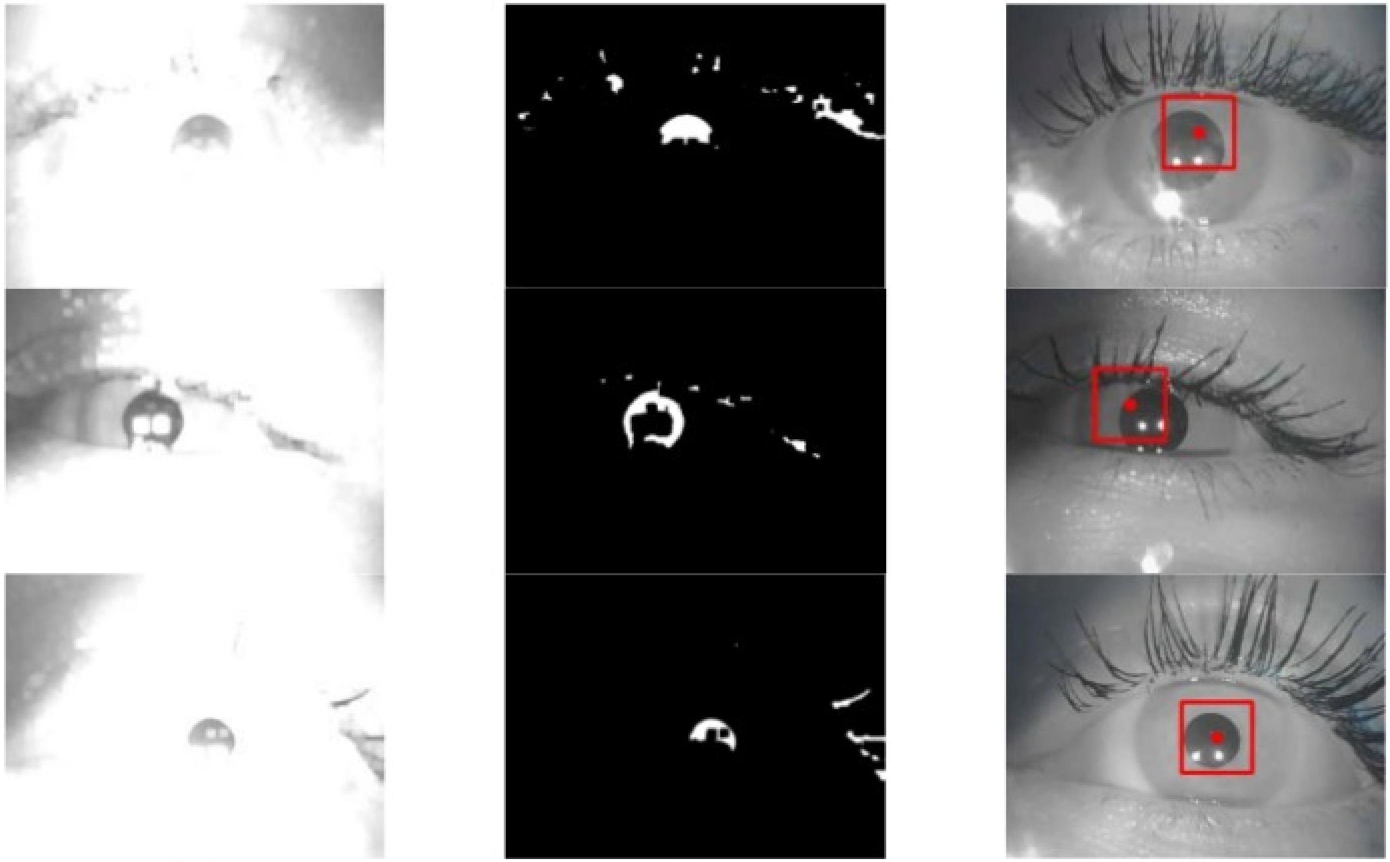
Estymacja pozycji źrenicy odpowiadających danym z K2

Estymacja pozycji źrenicy w obrazie K2

Wyznaczenie macierzy T2 dla każdej zarejestrowanej klatki

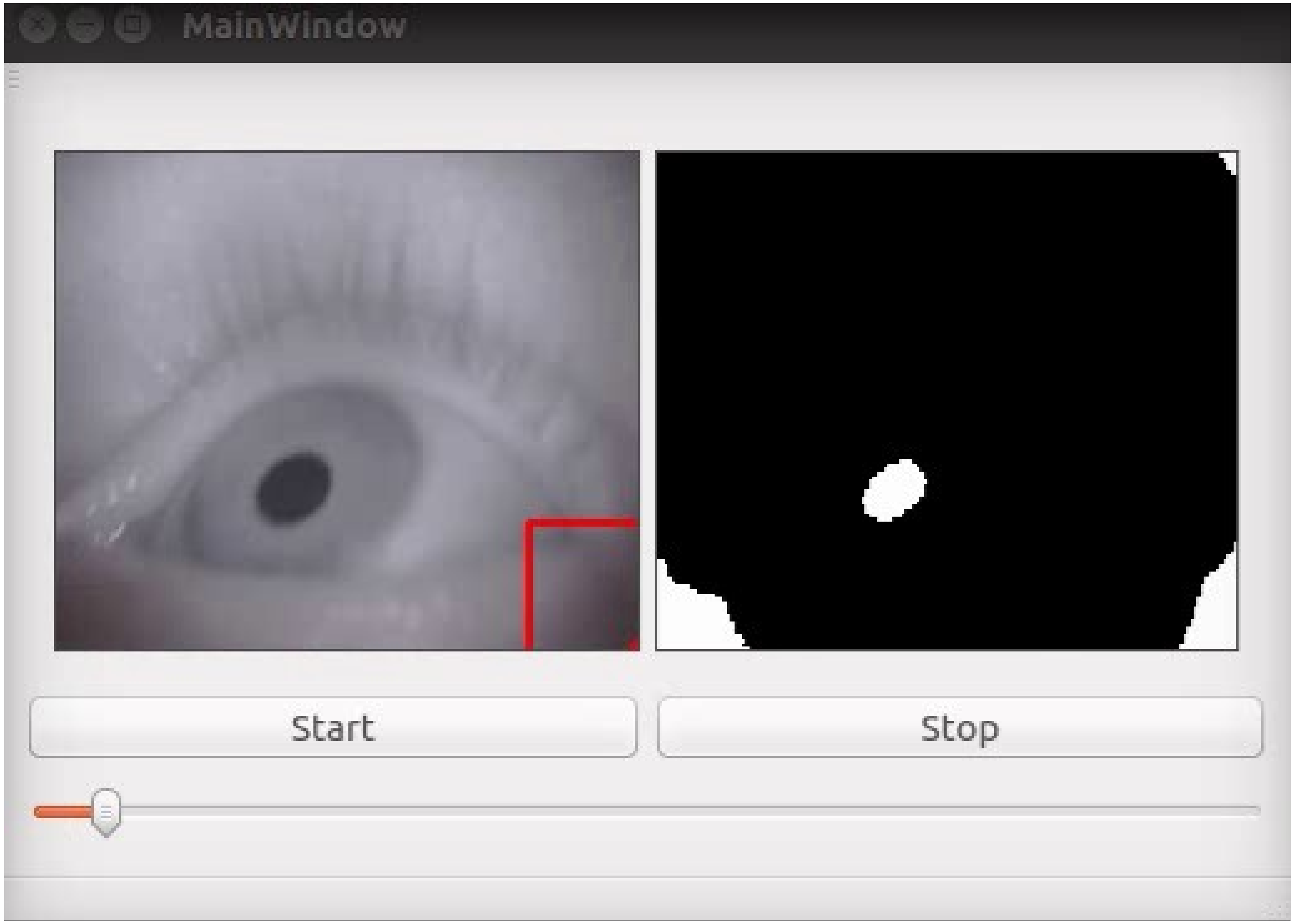
Estymacja pozycji fiksacji w ROI

$$\begin{bmatrix} X_f Z_r \\ Y_f Z_r \\ Z_r \end{bmatrix}_i = T2(i) \cdot \begin{bmatrix} x_v \\ y_v \\ 1 \end{bmatrix}_i = T2(i) * T1(i) \cdot \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix}_i$$



(a) (b) (c)

POLITECHNIKA GDAŃSKA



Head movement compensation

Constant geometric relationship between eye, display and a camera

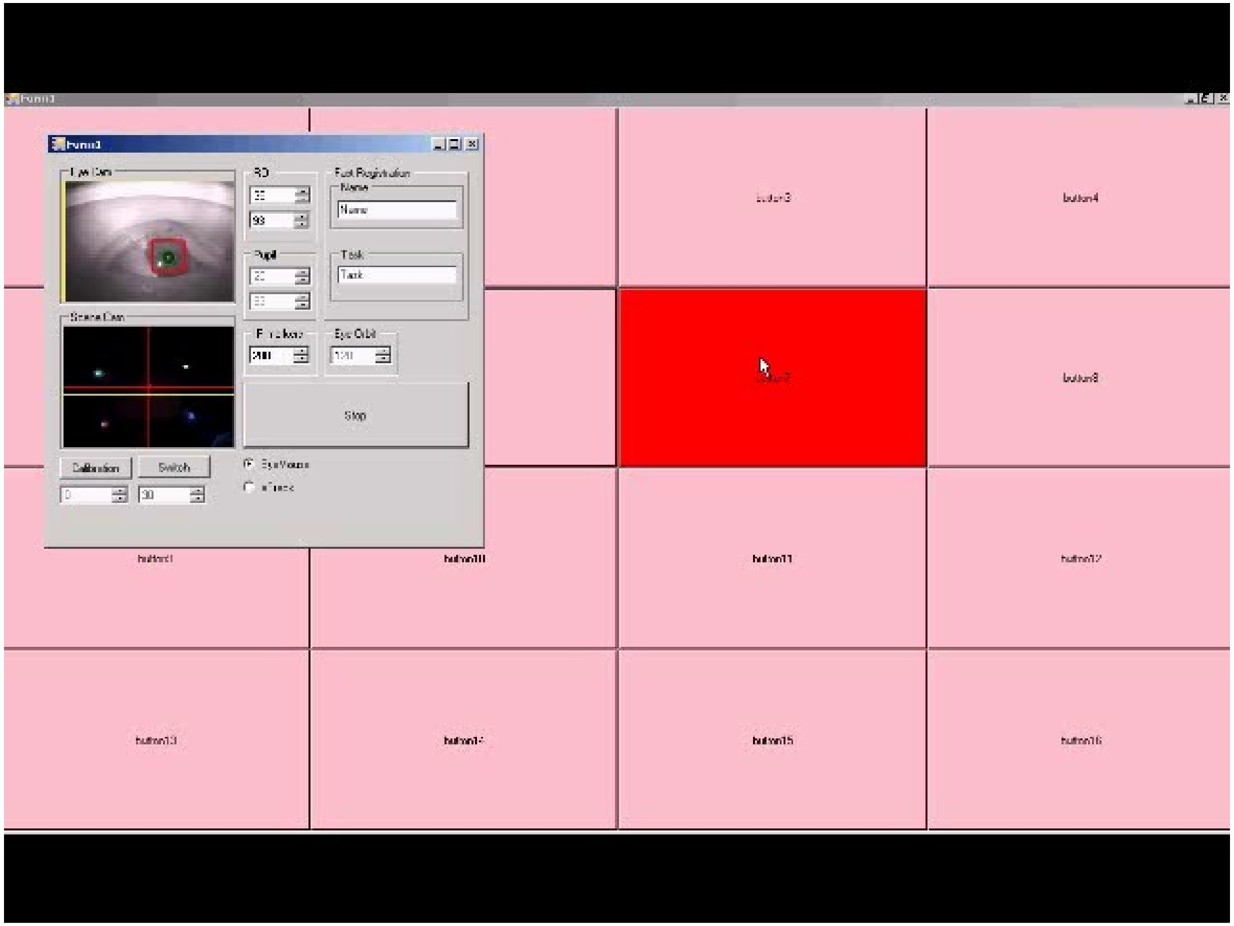
$$\begin{bmatrix} x_v Z \\ y_v Z \\ Z \end{bmatrix} = T \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix}$$

$$T = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & 1 \end{bmatrix}$$

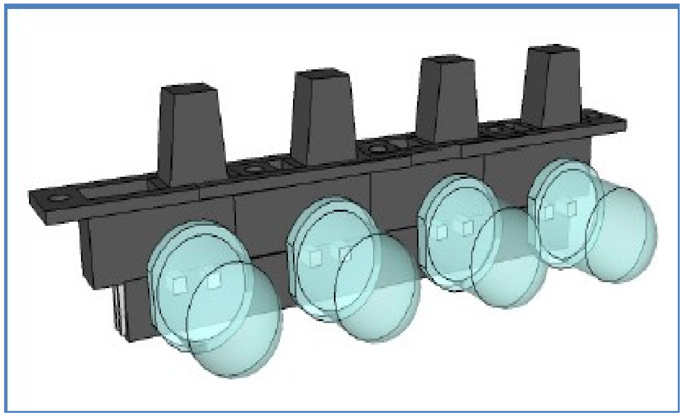
$$\begin{bmatrix} x_{cp} \\ y_{cp} \end{bmatrix} = \begin{bmatrix} x_p & y_p & 1 & 0 & 0 & 0 & -x_{cp} x_p & -x_{cp} y_p \\ 0 & 0 & 0 & x_p & y_p & 1 & -y_{cp} x_p & -y_{cp} y_p \end{bmatrix} \begin{bmatrix} a1 \\ a2 \\ a3 \\ a4 \\ a5 \\ a6 \\ a7 \\ a8 \end{bmatrix}$$

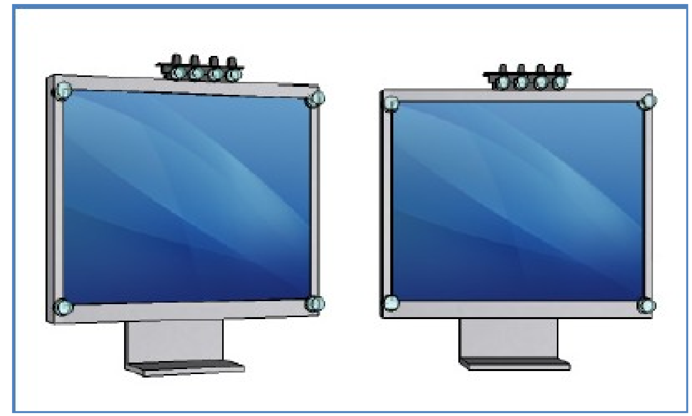
Constant geometric relationship between display and a camera

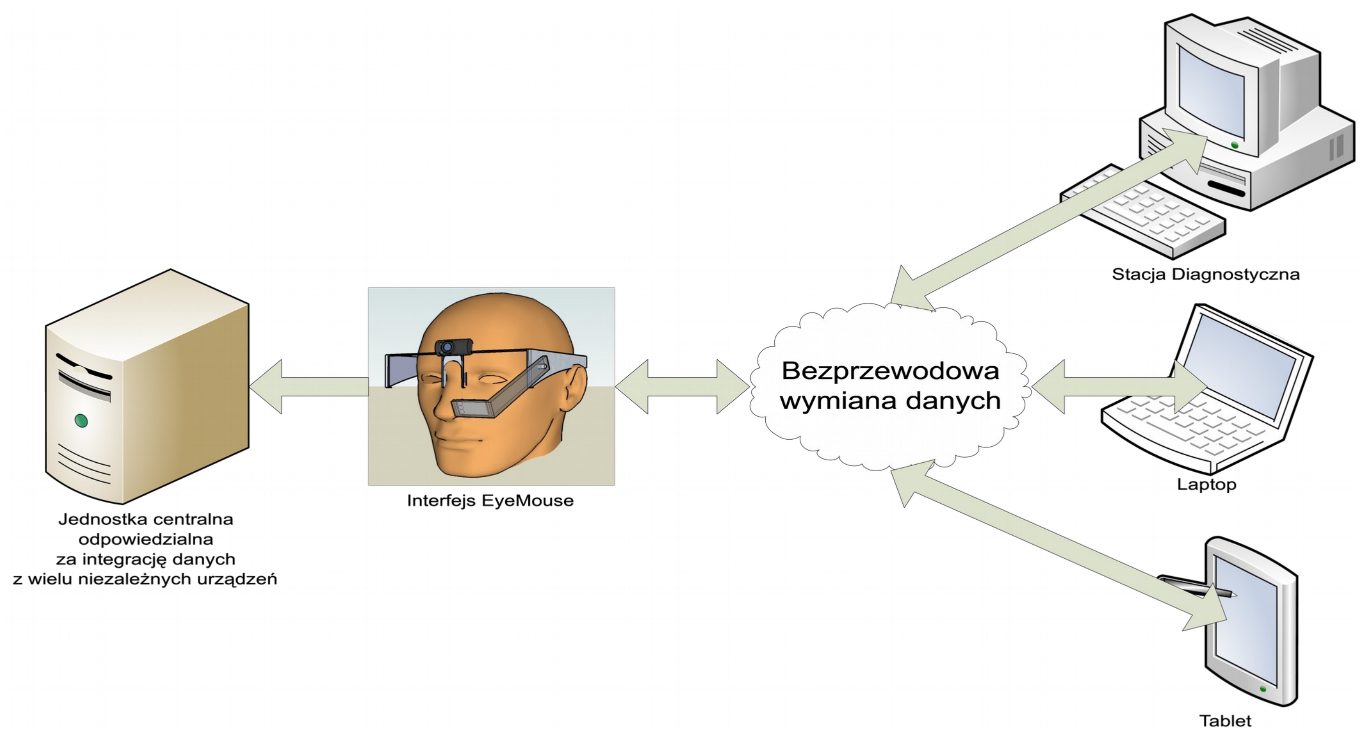
$$\begin{bmatrix} x_v Z \\ y_v Z \\ Z \end{bmatrix} = T \begin{bmatrix} x_p - x_g \\ y_p - y_g \\ 1 \end{bmatrix}$$


Name
Task
Stop
Calibration
Switch
button4
button8
button10
button11
button12
button13
button14
button15
button16

Detekcja punktów fiksacji w środowisku wieloekranowym

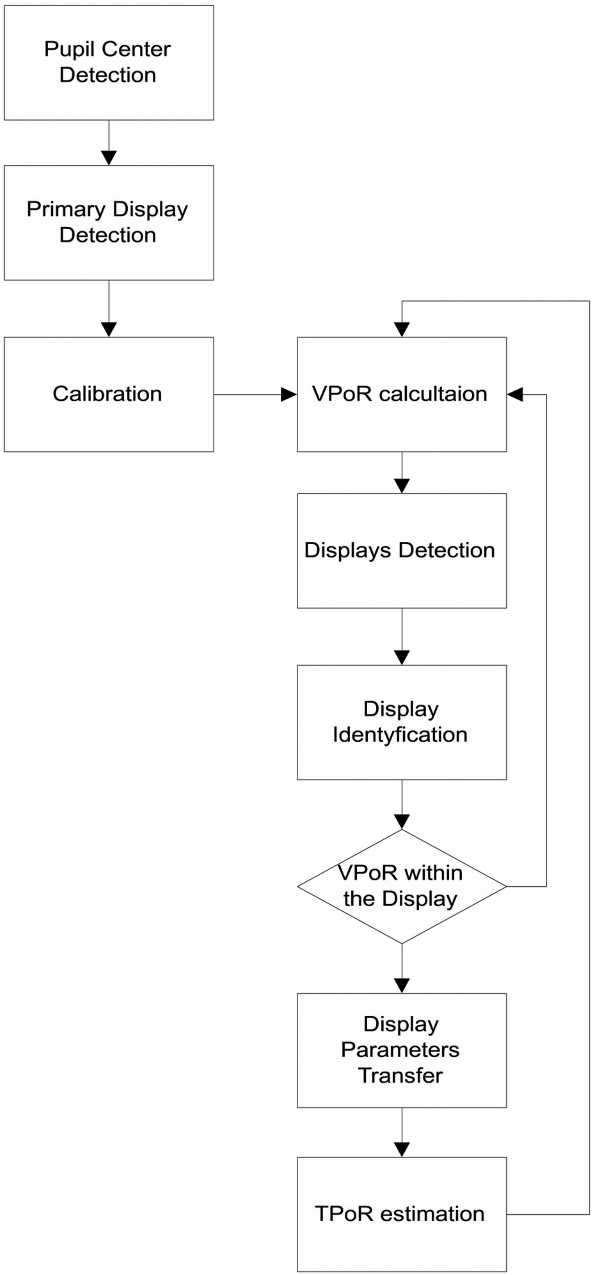


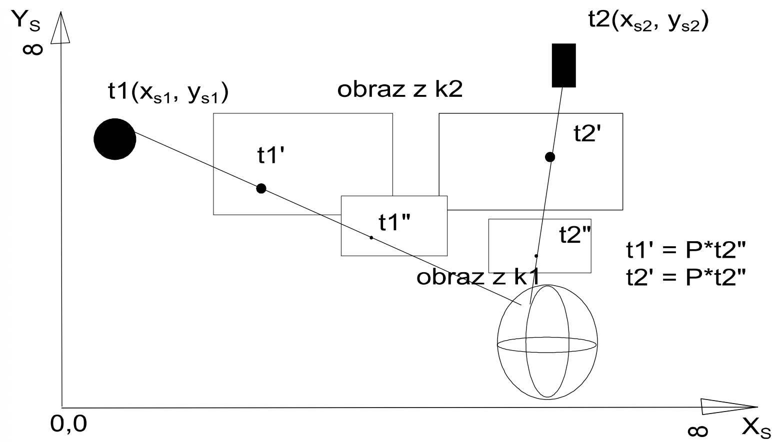




*Kocejko, T. & Wtorek, J. (2013), Gaze tracking in multi-display environment, in 'Human System Interaction (HSI), 2013 The 6th International Conference on', pp. 626--631.

gaze estimation





(1) $$VPoR = \Omega_1^T * P$$

where:

- Ω_1 - is transformation matrix computed from pupil positions stored in matrix M1 related with calibration points stored in M2
- P - is a vector containing absolute pupil center position registered by eye camera
- VPoR - Virtual Point of Regards vector containing the fixation position correlated and represented in the same space as images captured by scene camera.

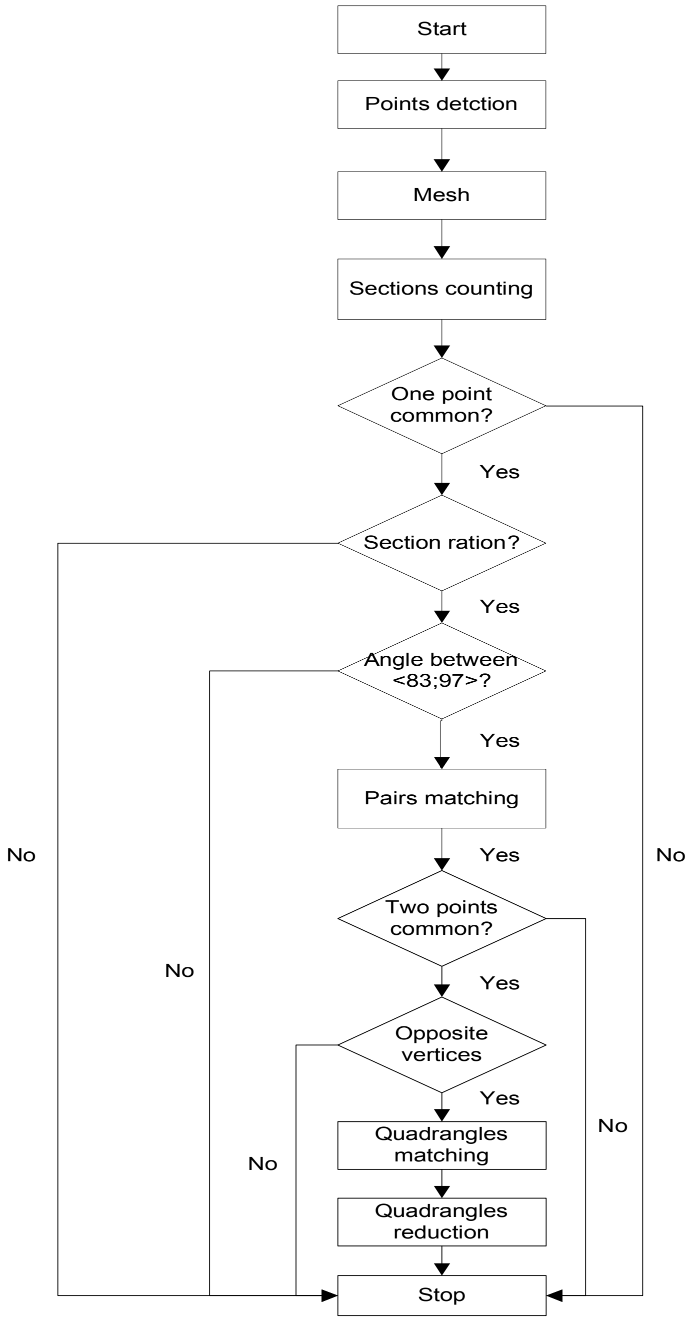
(2) $$TPoR = \Omega_2^T * VPoR$$

where:

- Ω_2 - is transformation matrix computed from virtual positions of IR LEDs markers dynamically captured by scene camera stored in matrix Mi and their position represented in pixel space - MRi
- VPoR - Virtual Point of Regards vector containing the fixation position correlated and represented in the same space as images captured by scene camera.

multiple display tracking algorithm

- Points detection
- Mesh application
- Section counting
- Section properties check
- Pairs matching
- Pairs properties check
- Quadrangles matching
- Quadrangles reduction

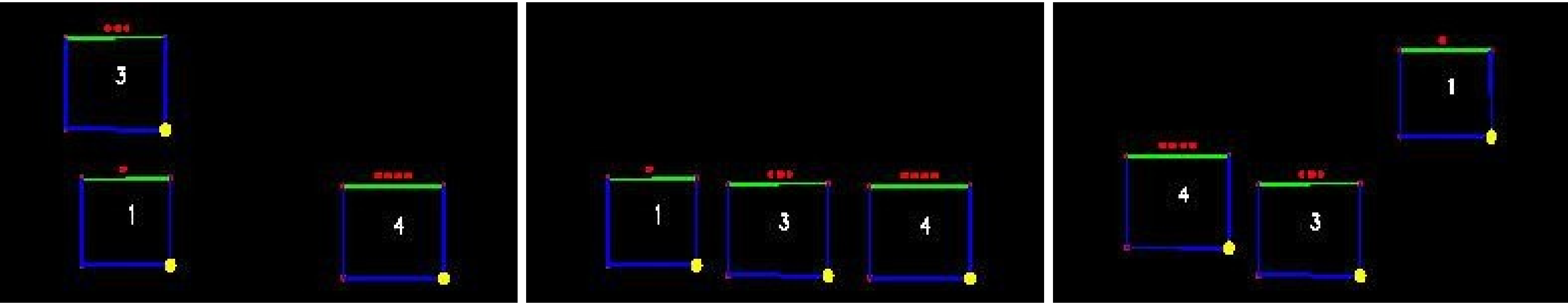


POLITECHNIKA GDAŃSKA

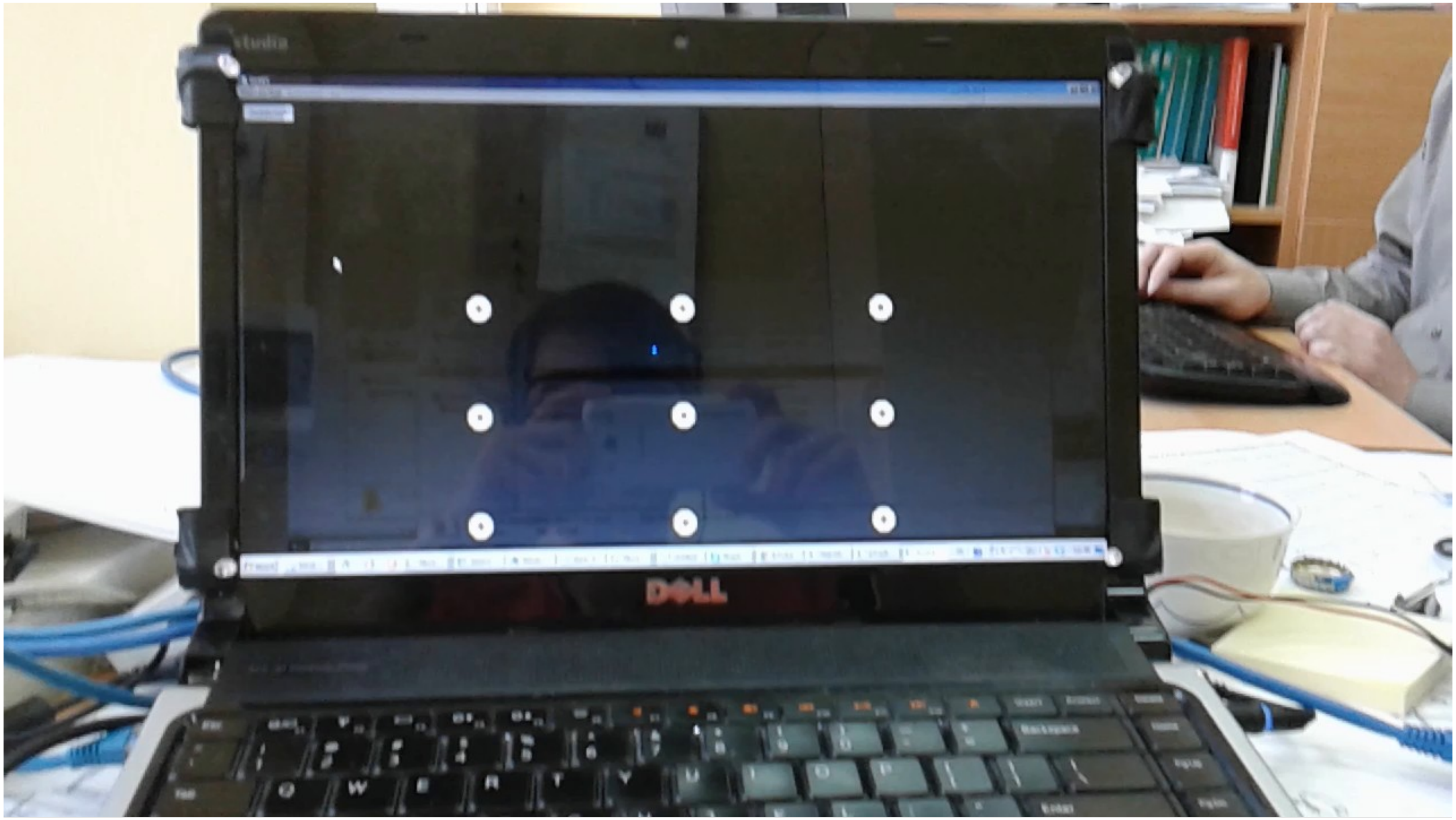
Results

Cloud of points

Detection result

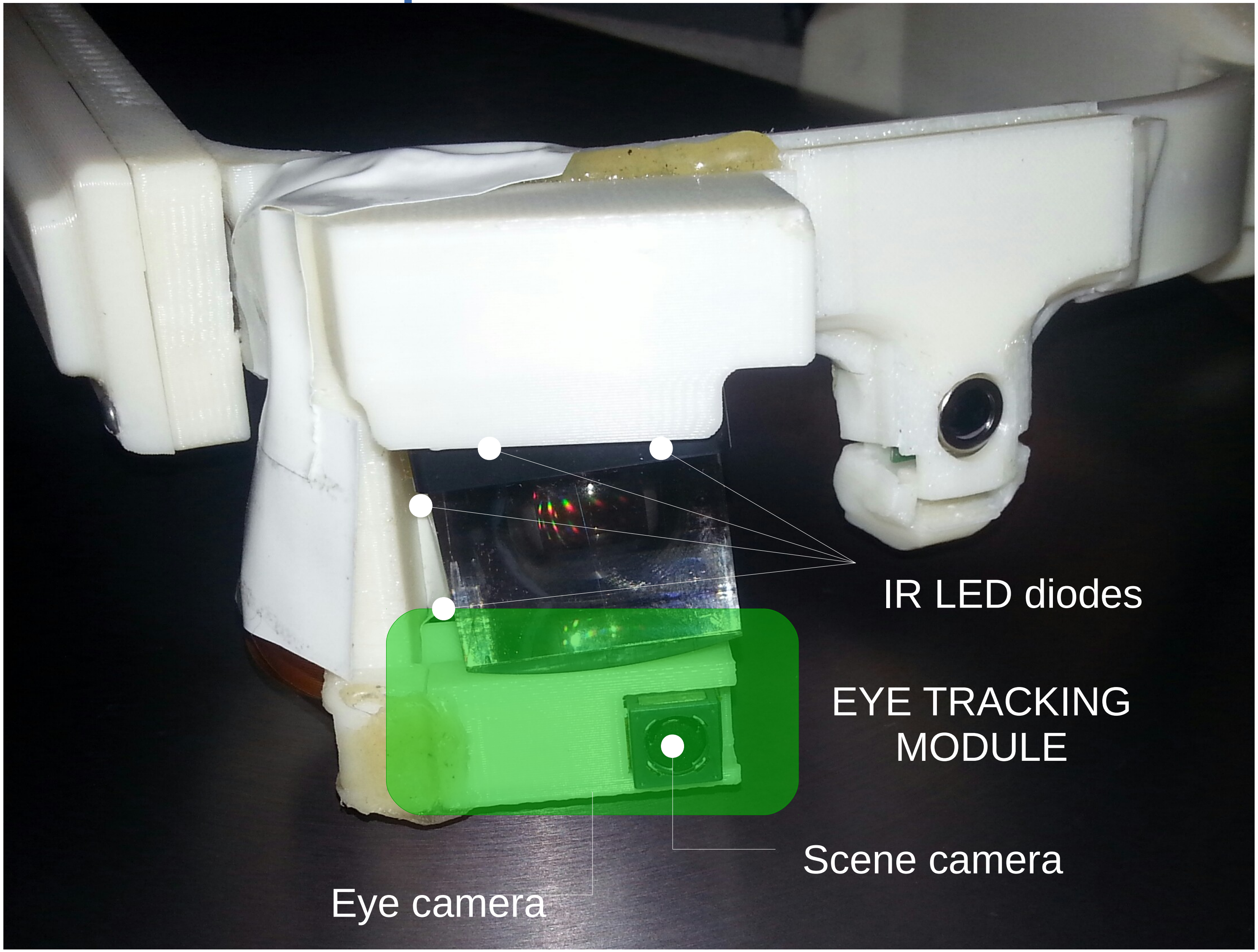


POLITECHNIKA
GDAŃSKA



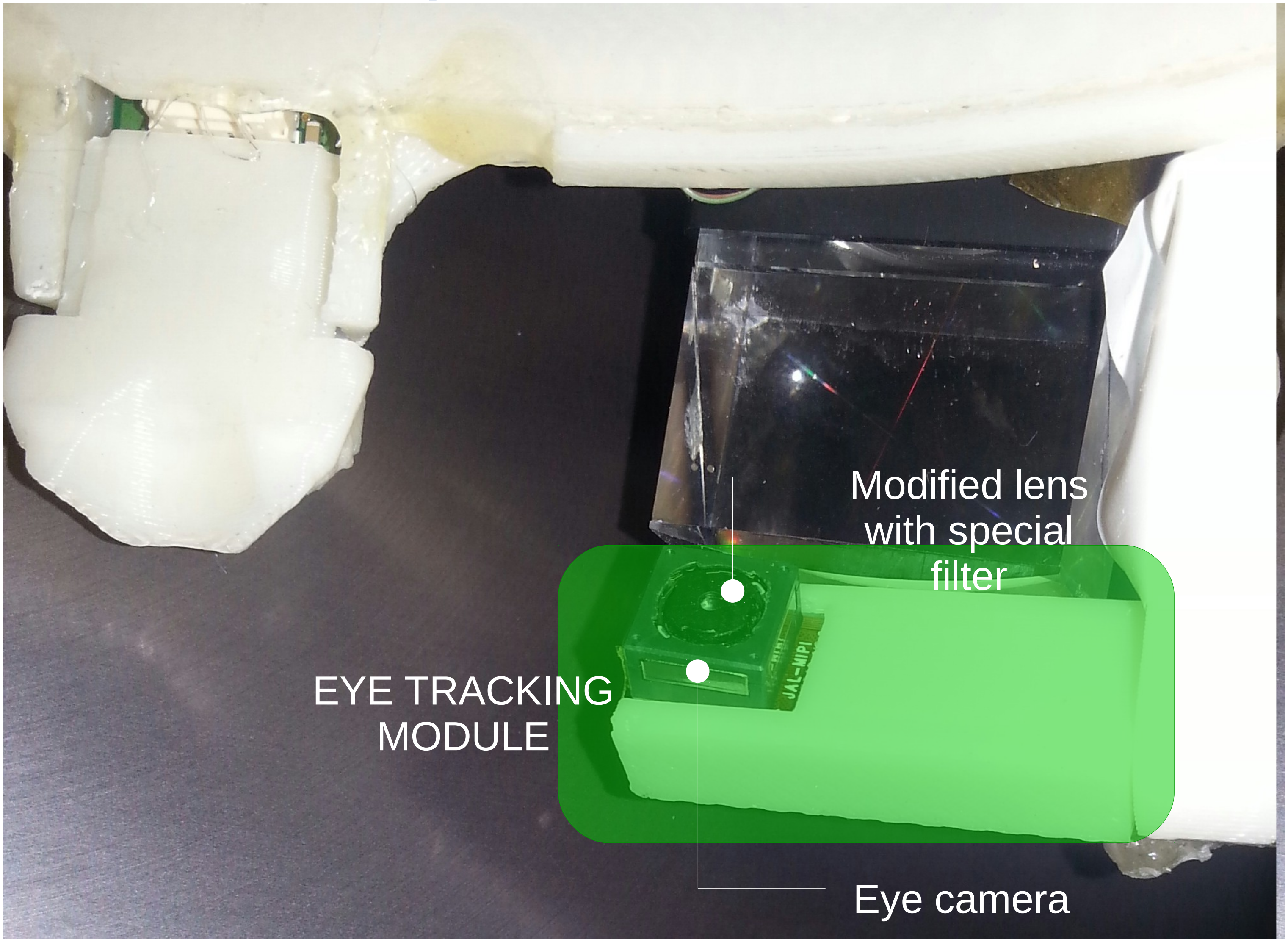
POLITECHNIKA
GDAŃSKA

Hardware setup

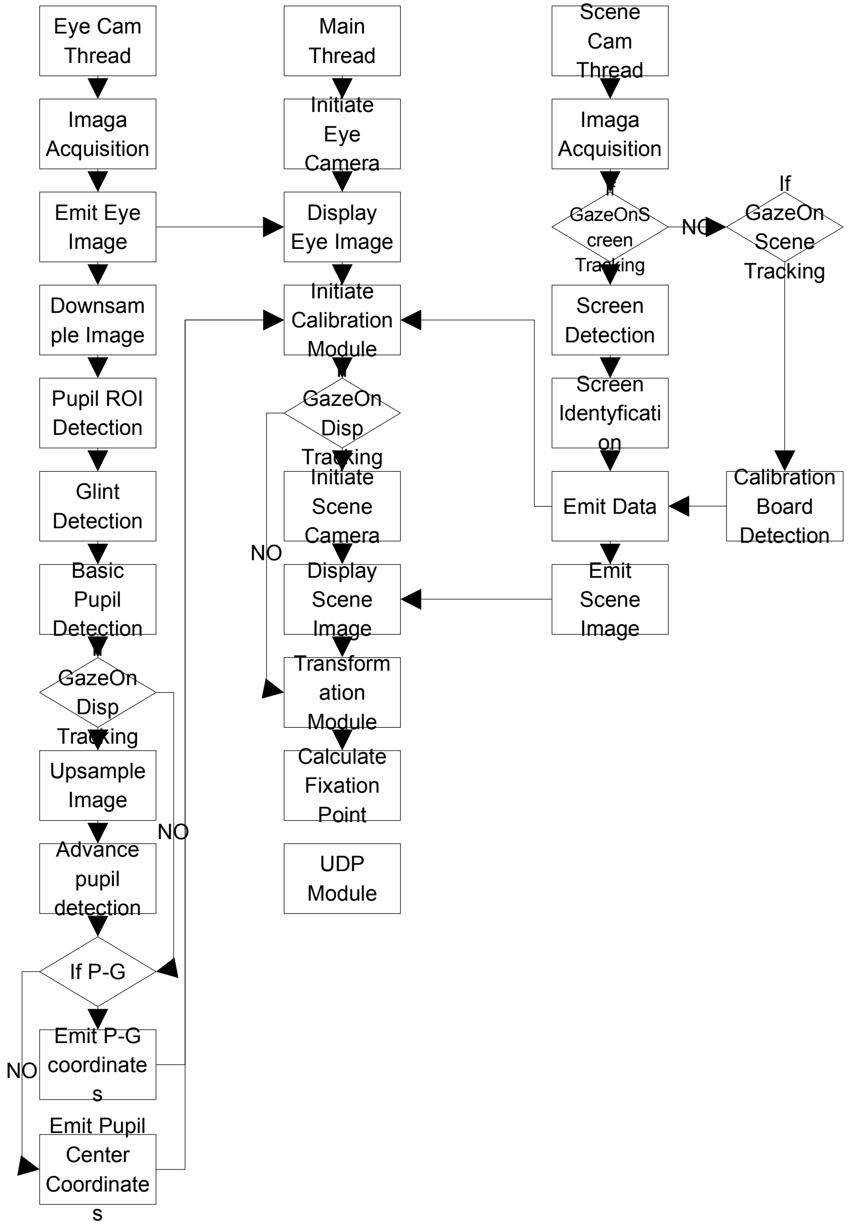


POLITECHNIKA
GDAŃSKA

Hardware setup



Software architecture



Multiple camera support

Different primary settings for each defined camera:

```
cam1->cam_no = 1;
cam1->GrabInV4l2 = true;
cam1->v4lgrab->vidfo = V4L2_PIX_FMT_MJPEG;
cam1->dev_name = "/dev/video1";
cam1->res_w = 320;
cam1->res_h = 240;
```

```
camX->cam_no = 2;
camX->GrabInV4l2 = false;
camX->v4lgrab->vidfo = V4L2_PIX_FMT_YUYV;
camX->dev_name = "/dev/video2";
camX->res_w = 320;
camX->res_h = 240;
```

```
cam2->cam_no = 0;
cam2->GrabInV4l2 = true;
cam2->v4lgrab->vidfo = V4L2_PIX_FMT_MJPEG;
cam2->GrabInGst = true;
cam2->dev_name = "/dev/video0";
cam2->res_w = 640;
cam2->res_h = 480;
```

Default options are presets in main thread constructor and maybe changed from other classes

The eGlasses consortium receives the funding support of NCBiR, FWF, SNSF, ANR, and FNR in the framework of the ERA-NET CHIST-ERA II.

POLITECHNIKA GDAŃSKA

Pupil detection

- Image downsize
- Pupil region detection
- Precise pupil detection algorithm 1
- Precise pupil detection algorithm 2

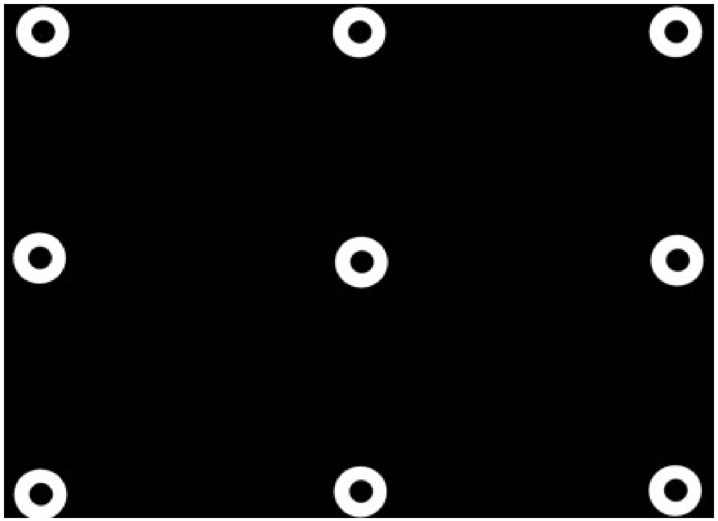
The eGlasses consortium receives the funding support of NCBiR, FWF, SNSF, ANR, and FNR in the framework of the ERA-NET CHIST-ERA II.

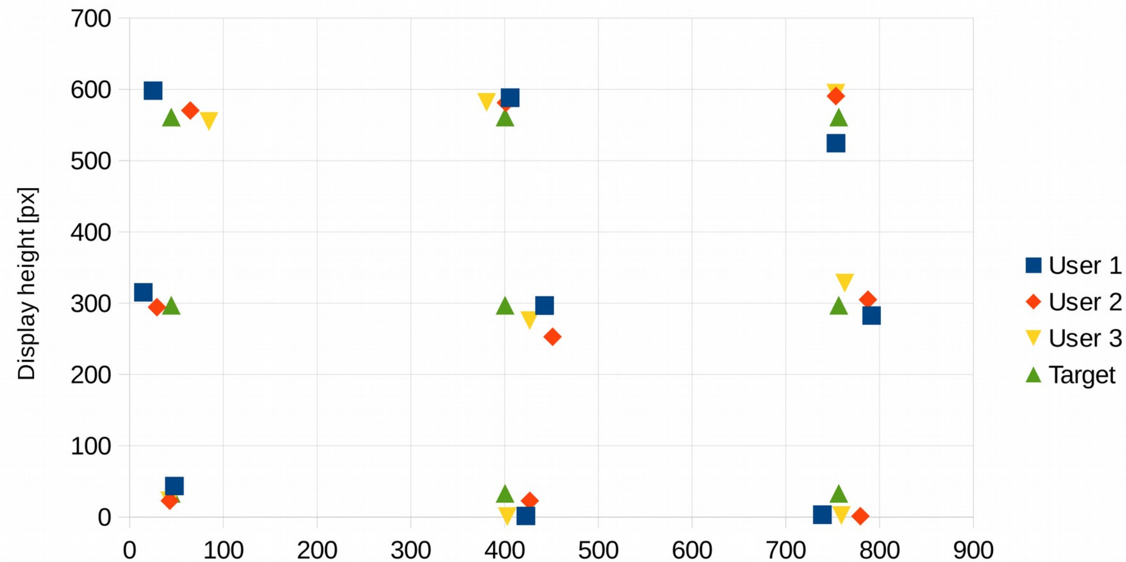
Pupil ROI detection/Fast pupil detection

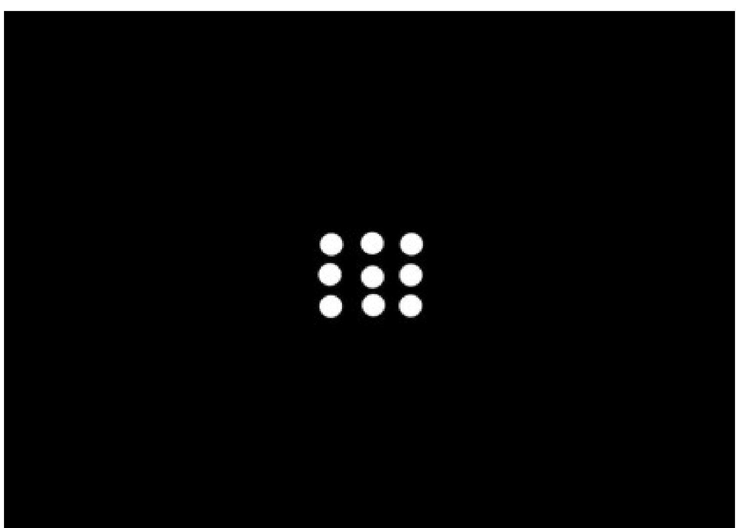
The result of this function is approximate center of pupil or pupil region

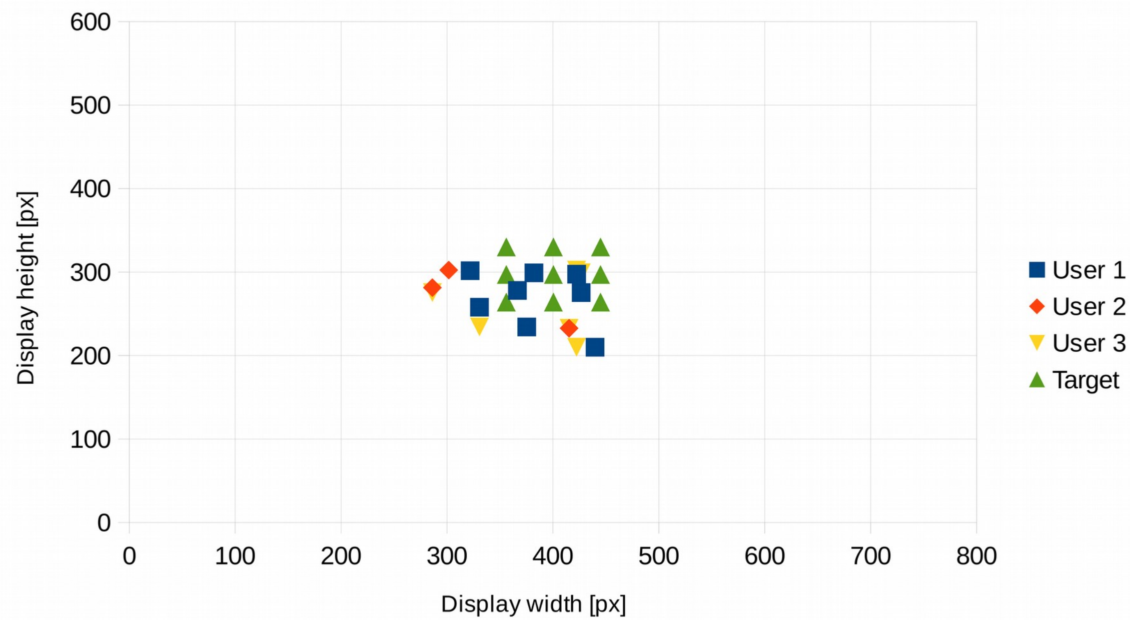
```
cv::Point roic; // = pdet->pupilROI(binImg);

cv::Mat maxCol = cv::Mat::zeros(1, small_img2.cols, CV_32F);
cv::Mat maxRow = cv::Mat::zeros(1, small_img2.rows, CV_32F);

if(binImg.channels()>1)
{
    qDebug()<<"wrong no. of channels";
}
else
{
    int r = 0;
    double cmax = 0;
    double rmax = 0;
    for(int i=0+r; i<binImg.rows-r; i++)
    {
        double temp = 0;
        //double cmax = 0;
        for(int j=0; j<binImg.cols-1; j++)
        {
            if(binImg.at<uchar>(i,j) == 255) {...}
            if(binImg.at<uchar>(i,j+1) == 0) {...}
            if(j == binImg.cols-2) {...}
        }
        maxRow.at<float>(0,i) = cmax;
    }
    for(int j=0+r; j<binImg.cols-r; j++)
    {
        double temp = 0;
        for(int i=0; i<binImg.rows-1; i++) {...}
        maxCol.at<float>(0,j) = rmax;
    }
}

double min, max, min2, max2;
cv::Point min_p, max_p, min_p2, max_p2;
cv::minMaxLoc(maxCol, &min, &max, &min_p, &max_p);
cv::minMaxLoc(maxRow, &min2, &max2, &min_p2, &max_p2);

roic.x = max_p.x;
roic.y = max_p2.x;
```

The eGlasses consortium receives the funding support of NCBiR, FWF, SNSF, ANR, and FNR in the framework of the ERA-NET CHIST-ERA II.





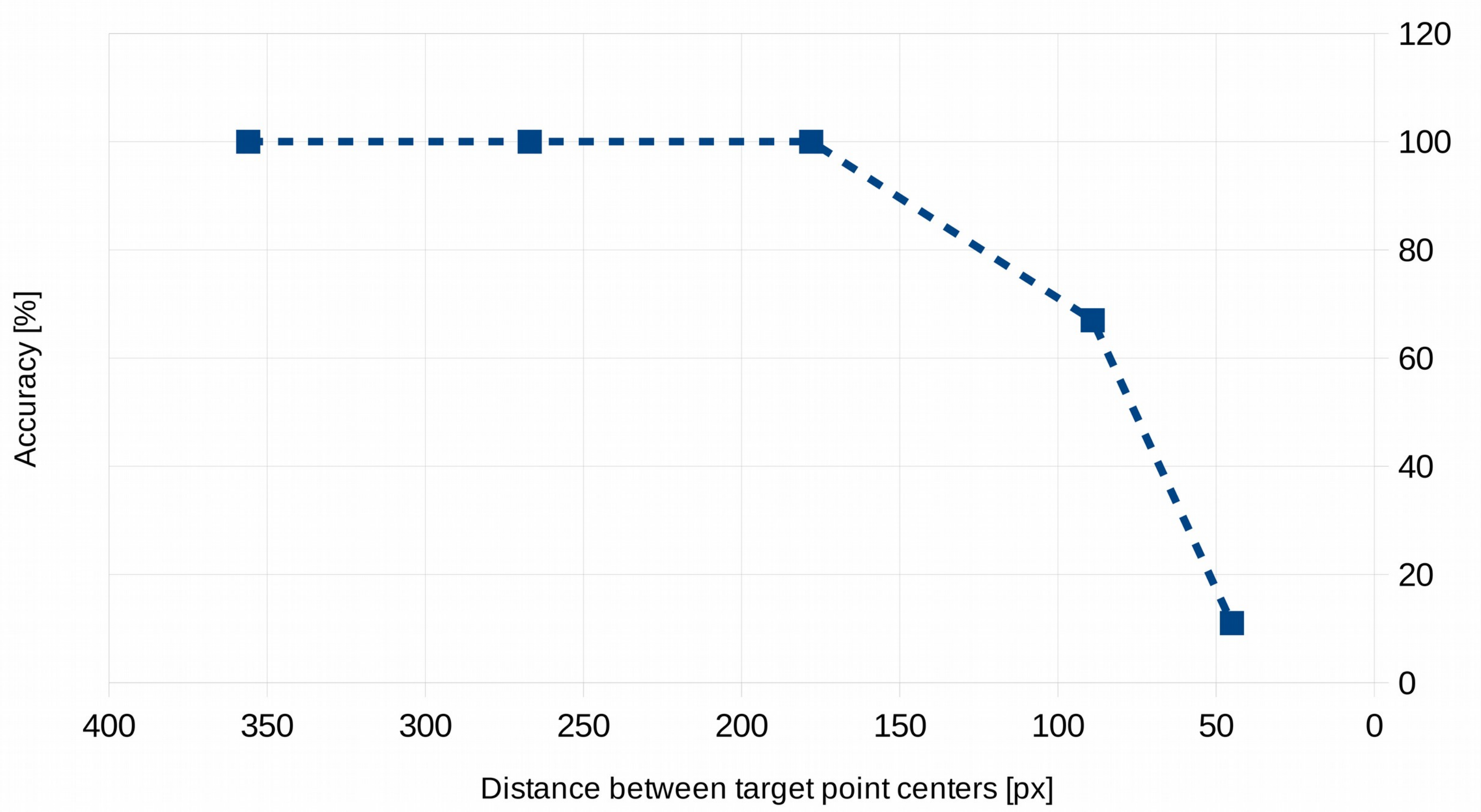




The eGlasses consortium receives the funding support of NCBiR, FWF, SNSF, ANR, and FNR in the framework of the ERA-NET CHIST-ERA II.

Results

The accuracy of a hit test (gaze based selection) in regards to the distance between target point centers



The eGlasses consortium receives the funding support of NCBiR, FWF, SNSF, ANR, and FNR in the framework of the ERA-NET CHIST-ERA II.

Conclusions

eye tracking interface in smart glasses is reliable

Proposed system was 100% accurate for test tables containing target points spread in a distance up to 150px

Eye tracking module can be use as a pointing device for the correctly designed interface

The eGlasses consortium receives the funding support of NCBiR, FWF, SNSF, ANR, and FNR in the framework of the ERA-NET CHIST-ERA II.