Appendix

Appendix A: Datasheet sensor EV76C560

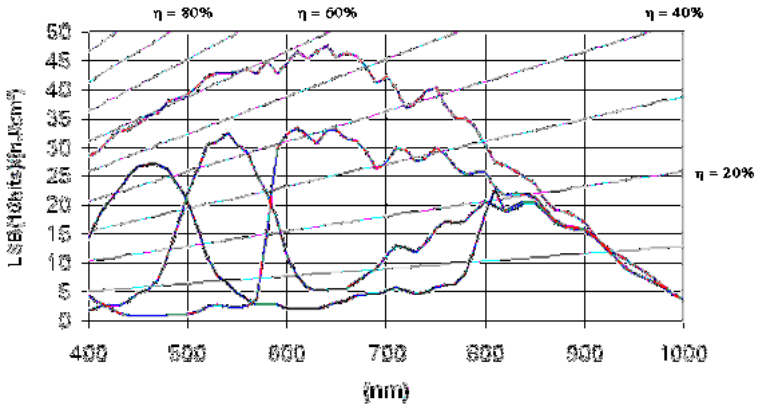
1. Typical Performance Data

Table 1-1. Typical electro-optical performance @ 25°C and 65°C, nominal pixel clock

Parameter		Unit	Typical value	
Sensor characteristics	Resolution	pixels	1280 (H) × 1024 (V)	
	Image size	mm inches	6.9 (H) × 5.5 (V) - 8.7 (diagonal) ≈ 1/1.8	
	Pixel size (square)	µm	5.3 × 5.3	
	Aspect ratio		5 / 4	
	Max frame rate	fps	60 @ full format	
	Pixel rate	Mpixels / s	90 -> 120	
	Bit depth	bits	10	
Pixel performance			**@ T_A 25°C**	**@ T_A 65°C**
	Dynamic range (1)	dB	>62	>57
	Qsat	ke-	12	
	SNR Max	dB	41	39
	MTF at Nyquist, λ=550 nm	%	50	
	Dark signal (2)	LSB_{10}/s	24	420
	DSNU(2)	LSB_{10}/s	6	116
	PRNU (3) (RMS)	%	<1	
	Responsivity(2) (4)	LSB_{10}/(Lux.s)	6600	
Electrical interface	Power supplies	V	3.3 & 1.8	
	Power consumption: Functional (5) Standby	mW µW	< 200 mW 180	

1. In electronic rolling shutter (ERS) mode.
2. Min gain, 10 bits.
3. Measured @ Vsat/2, min gain.
4. 3200K, window without AR coating, IR cutoff filter BG38 2 mm.
5. @ 60 fps, full format, with 10 pF on each output.

Figure 1-1. Spectral response and quantum efficiency



Appendix B: Optical Filters Carl Zeiss

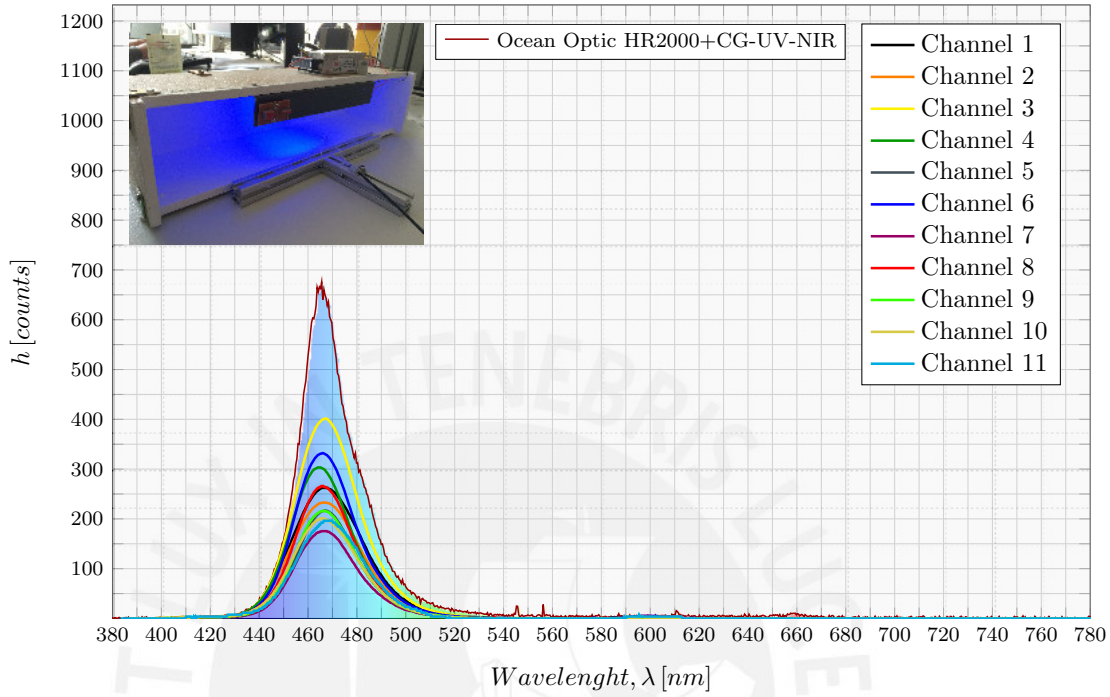
VEB Carl Zeiss JENA

Prüfschein für Metallinterferenzfilter

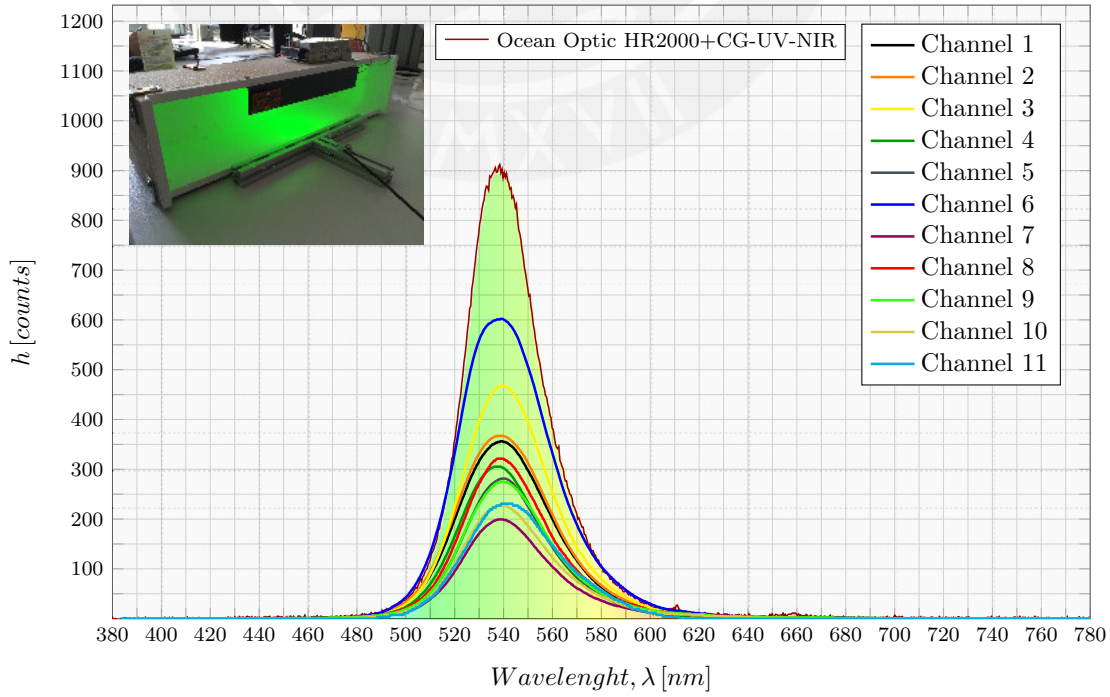
JF	λ_{max}	τ_{max}	FWHM	Typ	Fabr.-Nr
350	350 nm	32%	20 nm	∅50	G6162/9
375	371 nm	32%	16 nm	∅50	G6366/8
400	402 nm	37%	12 nm	∅50	G6655/10
425	424 nm	48%	10 nm	∅50	F3960/4
436	438 nm	42%	8 nm	∅50	G7125/1
450	449 nm	32%	10 nm	∅50	G6407/9
475	476 nm	47%	9.5 nm	∅50	F4010/9
500	499 nm	44%	8.5 nm	∅50	G6424/3
525	525 nm	40%	6 nm	∅50	F4582/8
550	555 nm	43%	5.5 nm	∅50	G7168/9
575	574 nm	40%	7.5 nm	∅50	G6969/2
589	590 nm	43%	11 nm	∅50	G7131/5
600	594 nm	41%	8 nm	∅50	G6464/1
625	620 nm	39%	9 nm	∅50	G7078/1
650	655 nm	39%	7.5 nm	∅50	G6729/8
675	677 nm	42%	10 nm	∅50	G7053/2
700	698 nm	43%	11 nm	∅50	G7141/4
725	729 nm	42%	11.5 nm	∅50	G7186/4
750	749 nm	47%	11 nm	∅50	G7136/10
775	771 nm	37%	8.5 nm	∅50	G7157/6
800	800 nm	46%	12.5 nm	∅50	G6393/5
825	829 nm	47%	15.5 nm	∅50	G6401/9
850	852 nm	42%	10.5 nm	∅50	G6680/5
875	881 nm	42%	12 nm	∅50	G6328/10
900	909 nm	40%	8 nm	∅50	F4391/3
925	925 nm	32%	8.5 nm	∅50	F3843/9
950	942 nm	44%	19.5 nm	∅50	G6364/4
975	983 nm	42%	14 nm	∅50	G6342/2
1000	1005 nm	42%	15 nm	∅50	G7147/9
1025	1028 nm	40%	19.5 nm	∅50	G7133/3
1050	1048 nm	44%	13.5 nm	∅50	G7159/6
1075	1074 nm	30%	15 nm	∅50	G7095/6
1100	1099 nm	30%	15 nm	∅50	G7093/10

Appendix C: LED tests

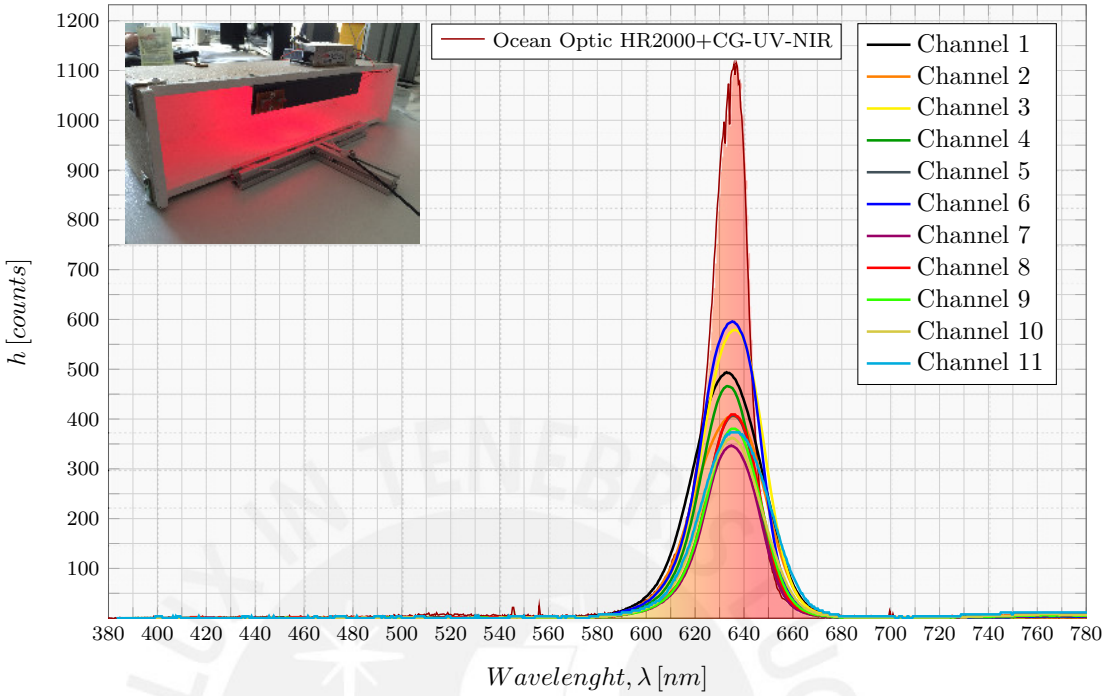
Blue LED



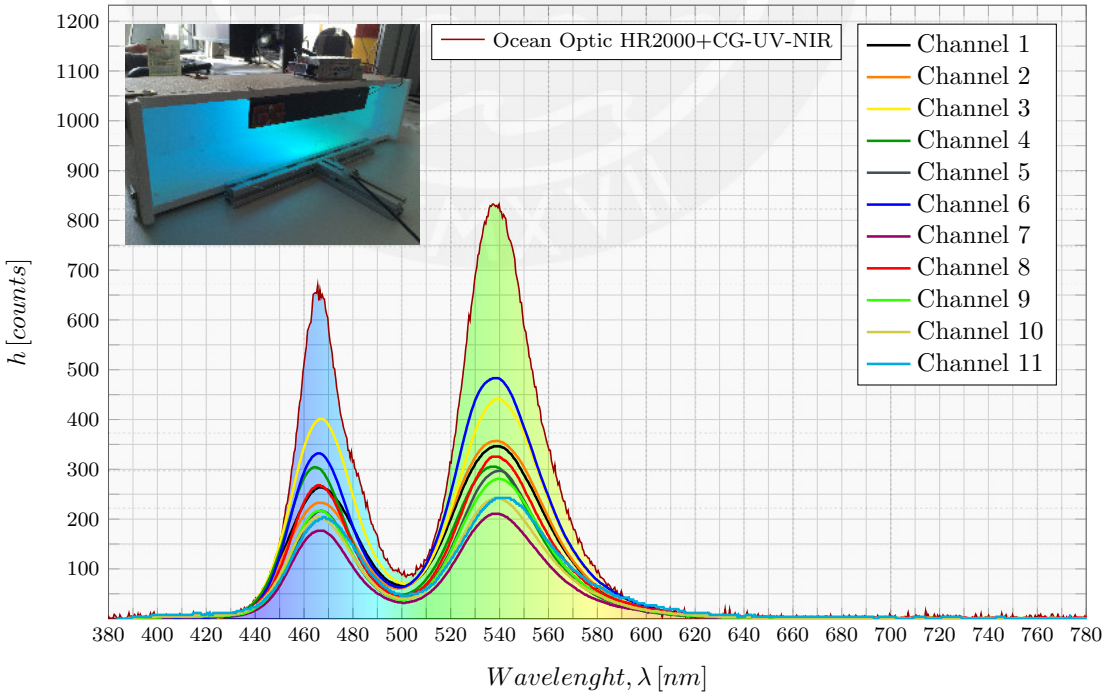
Green LED



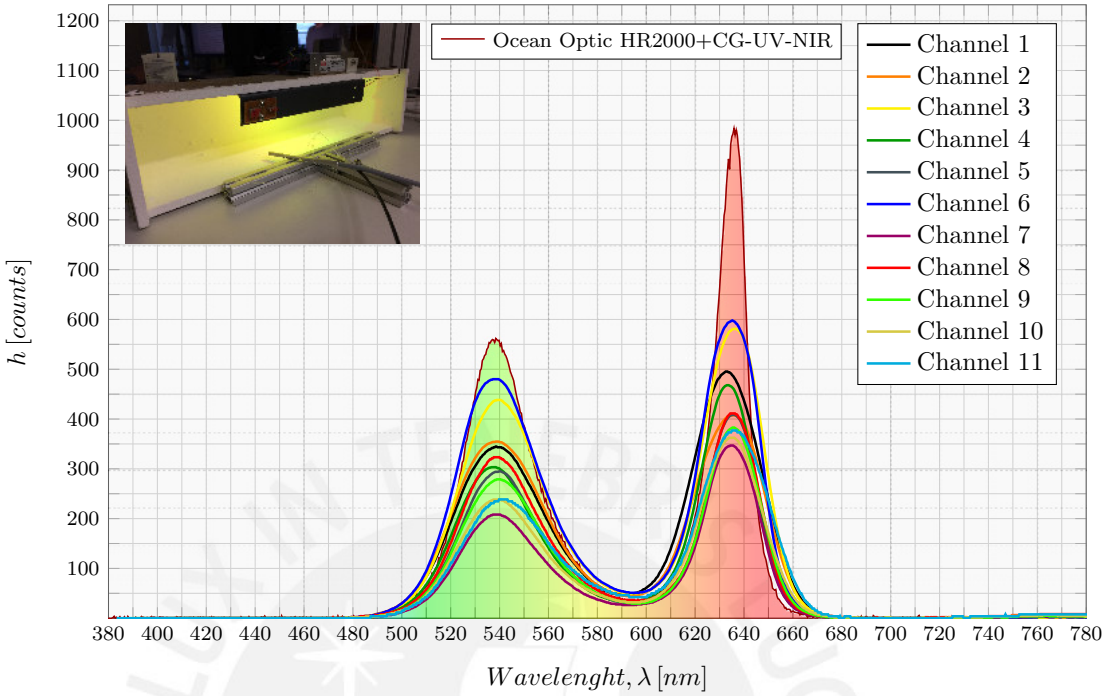
Red LED



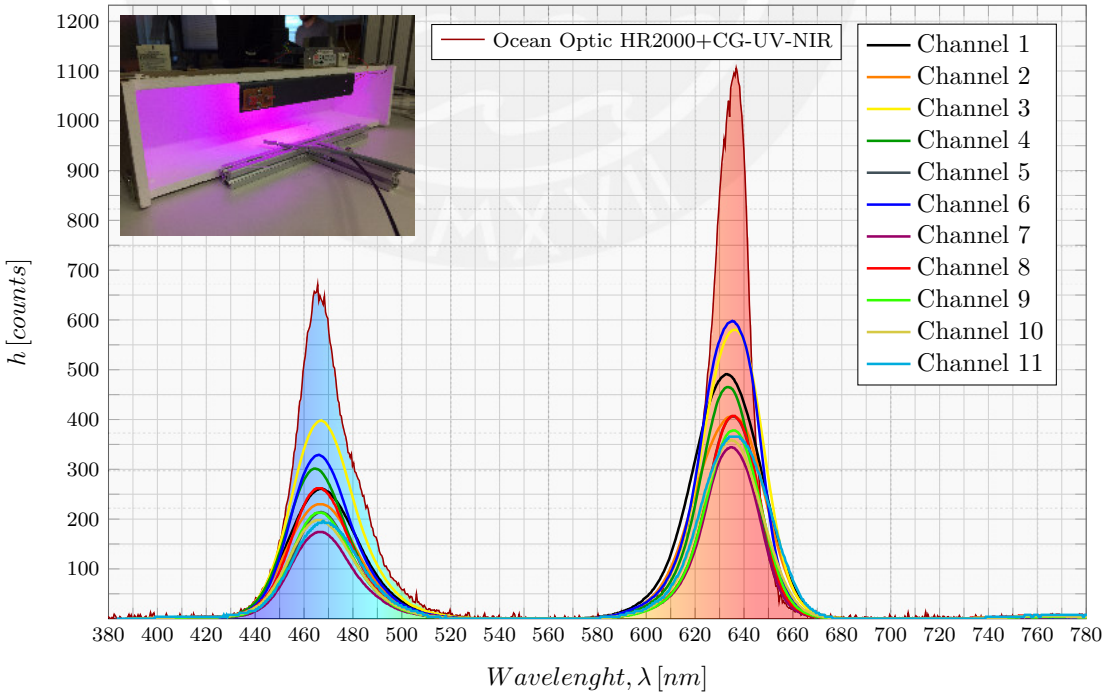
Blue and Green LEDs



Green and Red LEDs

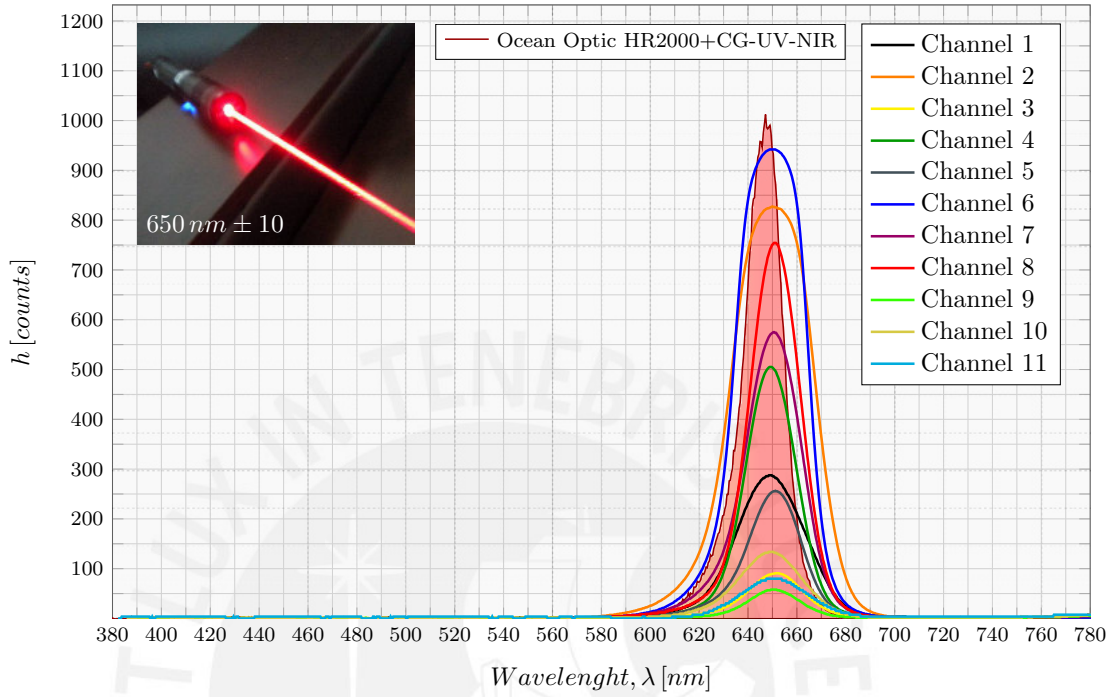


Blue and Red LEDs

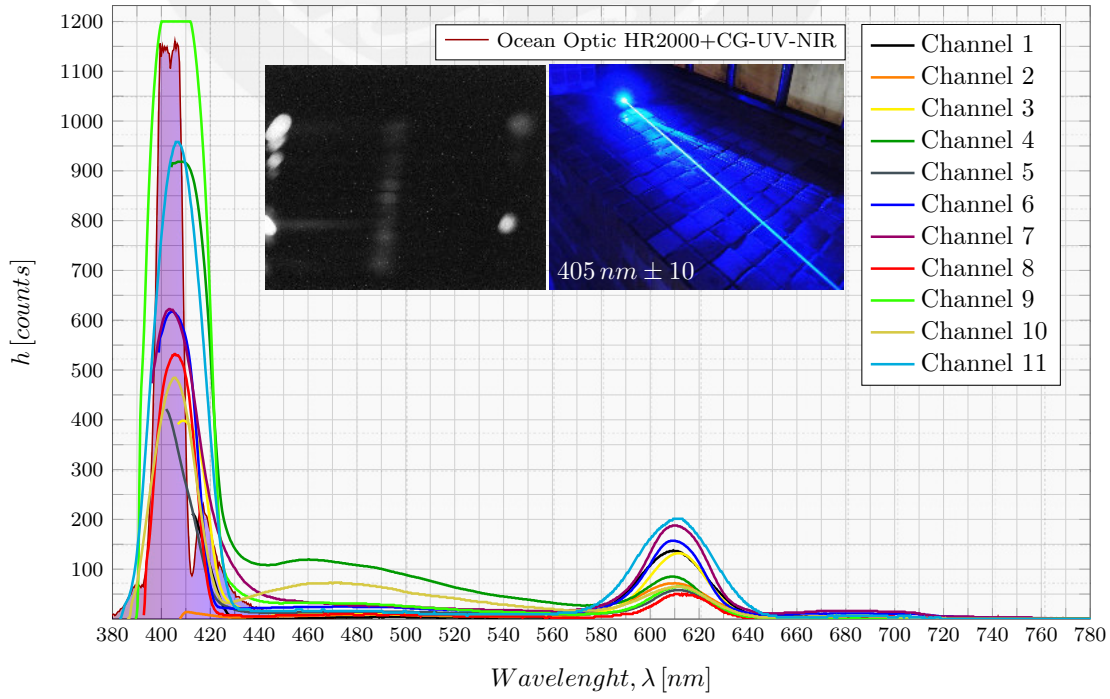


Appendix D: Laser diodes

Red Laser Diode

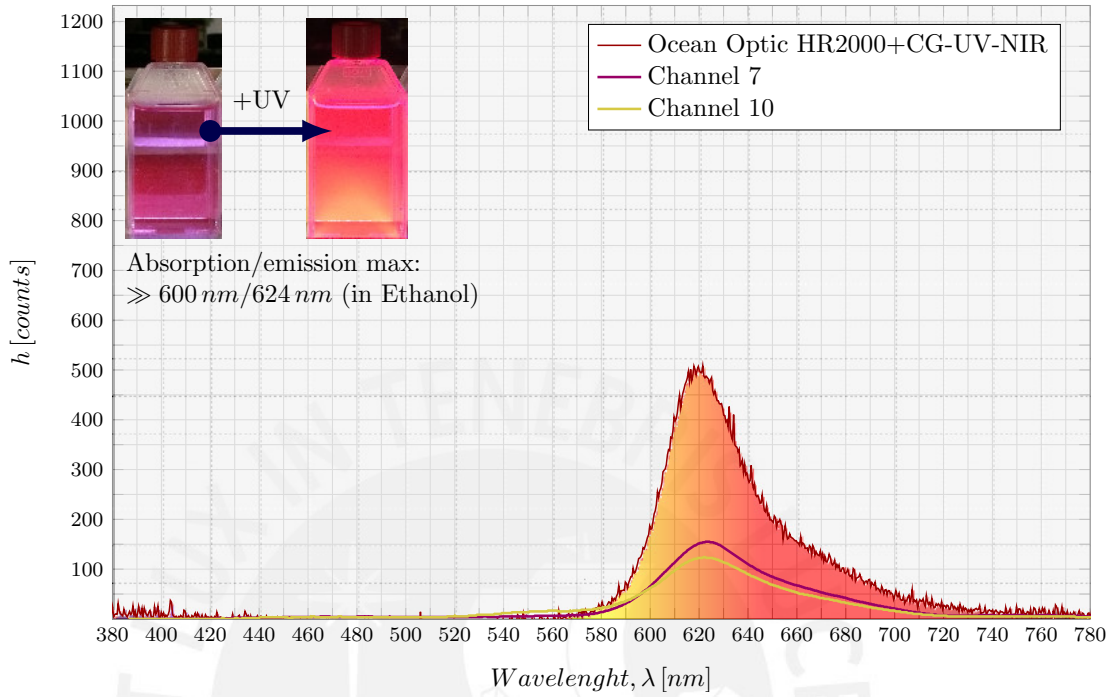


Blue Laser Diode

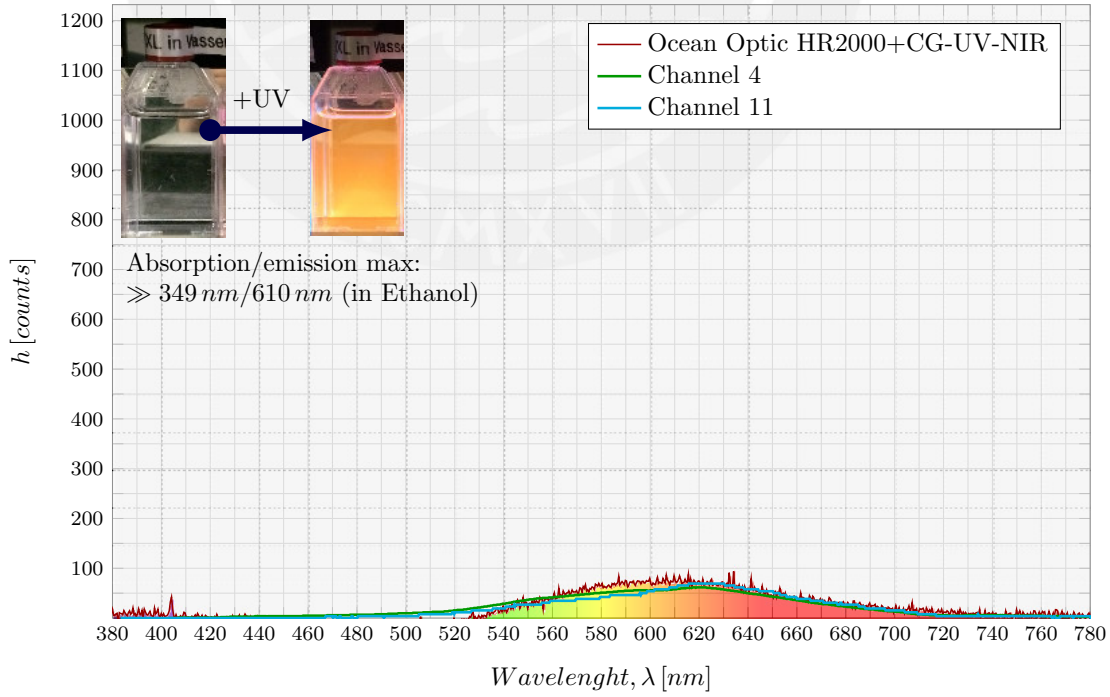


Appendix E: Dyomics Dye Markers

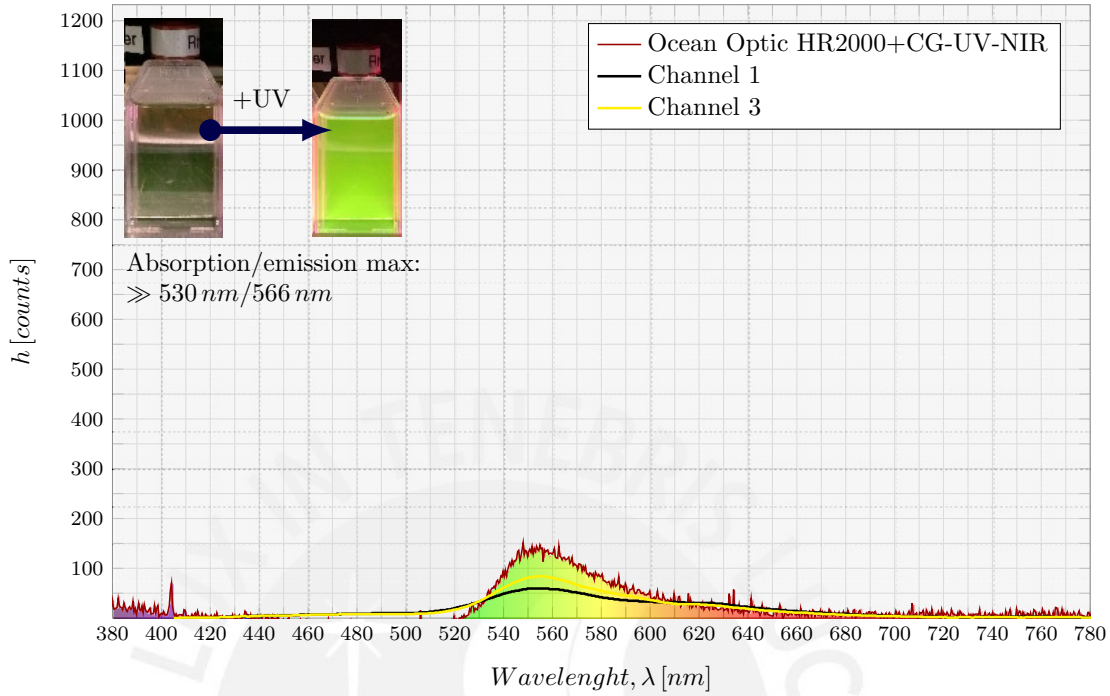
Orange Excitation (DY-605)



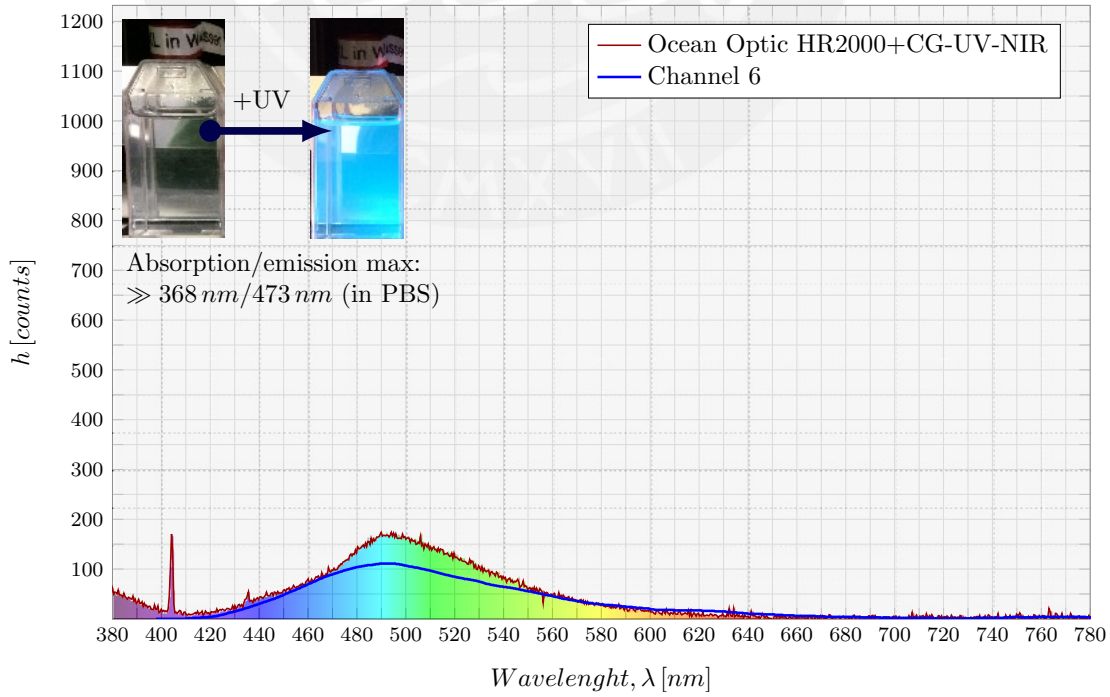
UV-Megastokes (DY-350XL)



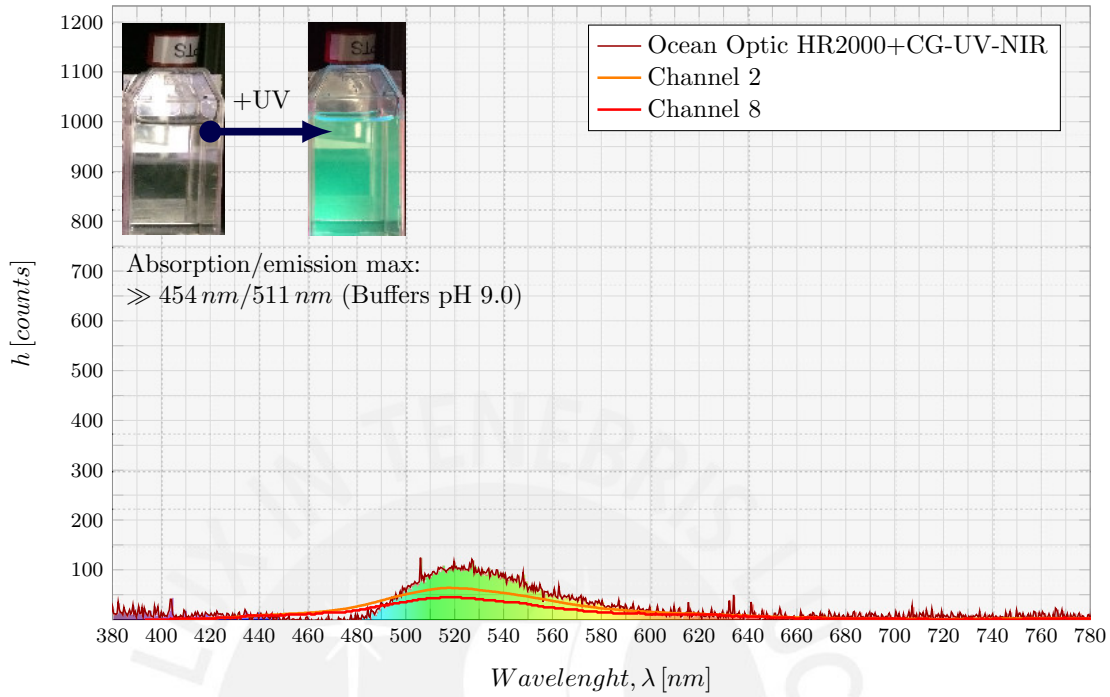
Rhodamine 6G (RH-6G)



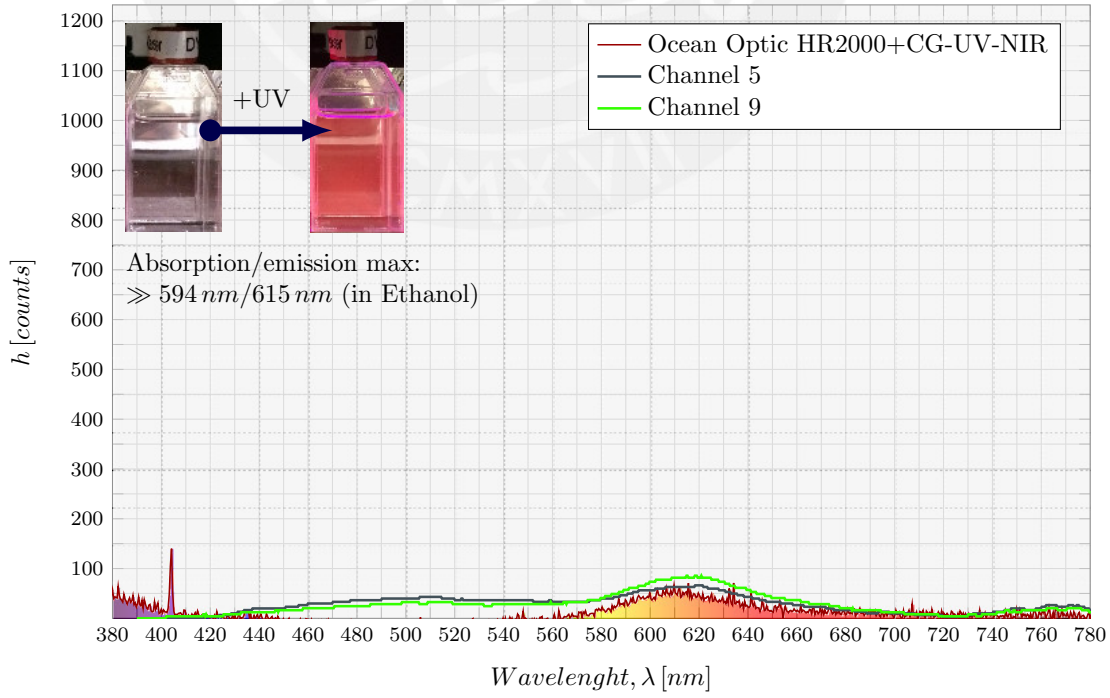
UV-Megastokes (DY-370XL)



Pyranine (HPTS)



Orange Excitation (DY-594)



Appendix F: Code: Find rotation

```
void findRotation(Mat &image, Mat &mRgbRescale, Point2f &P1t ,Point2f &
    P2t ,double &resultDegrees,double posPeak){
  Mat blurr,threshold_output,ttadjMap;    blur(image,blurr,Size(3,3));
  threshold(blurr,threshold_output,(float)(posPeak+posPeak/2)/1023.0,
                    1.0,THRESH_BINARY);
  vector<vector<Point> > contours;    vector<Vec4i> hierarchy;
  //Put in 8C3 image to use "findContours"
  double max,min;         minMaxIdx(threshold_output, &min, &max);
  convertScaleAbs(threshold_output, ttadjMap, 255/max);
  findContours(ttadjMap, contours, hierarchy, CV_RETR_TREE,
               CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
  vector<RotatedRect> minRect(contours.size()); //Rectangle
  vector<Moments> mu(contours.size());          //Get the moments
  vector<Point2f> mc(contours.size());          //Get the mass centers
  double maxArea=0;  int posMax;  //Find the biggest blob
  for( uint i=0;i<contours.size();i++){
     minRect[i]=minAreaRect(Mat(contours[i]));
     mu[i]=moments(contours[i],false);//Moments, Center of mass
     mc[i]=Point2f(mu[i].m10/mu[i].m00,mu[i].m01/mu[i].m00);
     double area=contourArea(contours[i]);
     if(area>maxArea){maxArea=area; posMax=i;} }//for //Save i
   Point2f rect_points[4];  minRect[posMax].points(rect_points);
   double maxres=0; //Meassure the four sides of the rectangle
   for(int j=0; j<4;j++){
     double res=cv::norm(cv::Mat(rect_points[j]),
                         cv::Mat(rect_points[(j+1)%4]));
     if(res>maxres){maxres=res;} }//for //Save the largest
   bool inMaxLine=true;
   for(int j=0; j<4;j++){
     double res=cv::norm(cv::Mat(rect_points[j]),
                         cv::Mat(rect_points[(j+1)%4]));
     if(res==maxres && inMaxLine){ //Check only the largest
       inMaxLine=false; //Horizontal line as reference
       Point2f P1=rect_points[j];   Point2f P2=rect_points[(j+1)%4];
       if (P1.x<=P2.x){resultDegrees=atan2((P2.y-P1.y),(P2.x-P1.x));}
       else{resultDegrees=atan2((P1.y-P2.y),(P1.x-P2.x));}
       //Move the line to where the canal begins
       double dt=maxres/3; double h=-dt*cos(resultDegrees);
       //Calculate equation from the first line, between P1 and P2
       double m=(P2.y-P1.y)/(P2.x-P1.x);
       double C=mc[posMax].y-m*mc[posMax].x;
       double ncmy= m*(mc[posMax].x+h)+C;// y=mx+C
       Point2f NewCM(mc[posMax].x+h,ncmy); //Return orthogonal line
       PerpendicularLine(image,P1,P2,P1t,P2t,NewCM); } } }//if,for,void
```

Appendix G: Code: Create floating points

```
//Create floating points (x,y) between two points
void CreatePoints2(Point2f P1t,Point2f P2t,vector<Point2f>&PLine){
    //Now the points between borders in P1t,P2t are created
    double res = cv::norm(cv::Mat(P1t),cv::Mat(P2t));
    PLine.resize(floor(res)-1); //Round the next integer num.
    vector<double> xx;          //Position x
    vector<double> yy;          //Position y
    vector<double> da(2);       //InputArray x
    vector<double> db(2);       //InputArray y
    vector<double> man(2);      //InputArray Size,length
    vector<double> angle(2);    //Angle between P1t and P2t
    da[0]=0;
    da[1]=P2t.x-P1t.x;
    db[0]=0;
    db[1]=P2t.y-P1t.y;
    phase(da,db,angle,true);  //true:[degrees] false: [radians]
    man[0]=0;
    for (int i=0;i<floor(res-1);i++){
        man[1]=i;               //Unit vector, jump 1 pixel
        //Calculates (x,y) coordinates of 2D vector
        //from their magnitude and angle.
        polarToCart(man,angle,xx,yy,true);
        PLine[i]=Point2f(P1t.x+xx[1],P1t.y+yy[1]);
    }//for
}//void
```

Appendix H: Code: Check test Line (L_n)

```
//Acces test line (Ln) and print key points in New Image
//Mat salPo: Image to find points
//double limit=min_value+40: Low window
//Mat image: Original Image
//Ins vector: Floating points (PLine)
//double n: External loop, for (uint n=0;n<PLine.size()-1;n++){}
void PrintLine(Mat &image, Mat &sal, Mat &salPo, vector<Point2f> &Ins,
               double limit, float &m0, bool &Switch, double n,
               vector <double> &dat){
    double factor=cv::norm(cv::Mat(Ins[0]),cv::Mat(Ins[n]));
    double factor1=cv::norm(cv::Mat(Ins[0]),cv::Mat(Ins[n+1]));
    double m;           //Slope between to consecutive points
    //Here it only sees the change in rows in only one column
    Mat A;  //Point 1 and 2, bilinear interpolation
    getRectSubPix(image,Size(1,1),Ins[n],A);
    float val= A.at<float>(Point(0,0));
    getRectSubPix(image,Size(1,1),Ins[n+1],A);
    float val1= A.at<float>(Point(0,0));
    double PoY0=cvRound(val*1024);
    double PoY1=cvRound(val1*1024);
    m=(PoY1-PoY0)/1.0000;           //Current line slope
    //Condition 1: between 1 and -1 and on the low limit
    if (PoY0>limit &&  m==0){
        //Image to find points (orange)
        circle(salPo, Point(cvRound(val*1024),factor),
            2, Scalar(230,165,0,100),20,4);
    } //if
    //Condition 2: slope crossing 0
    if (PoY0>limit && m>=0 && m0<=0 || PoY0>limit && m<=0 && m0>=0){
        //Image to find points (light blue)
        circle(salPo, Point(cvRound(val*1024),factor1),
            2, Scalar(0,77,102,100),20,4);} //if
    //Condition 3: when the curve cross the low limit
    if (Switch && cvRound(val*1024)>limit){
        circle(salPo, Point(cvRound(val*1024),factor),
            2, Scalar(255,255,255),20,4); //(z0)
        Switch =false;} //if
    if (!Switch && cvRound(val*1024)<limit){
        circle(salPo, Point(cvRound(val*1024),factor),
            2, Scalar(255,255,255),20,4); //(zn)
        Switch =true;} //if
    m0=m; //Save the last value of line-slope
}//void
```

Appendix I: Code: Crop channels

```
void CropChannels(Mat src,Mat mOriginal,double beta,
            vector<vector <Point2f>> pointsChannel,
            vector<double> widthChannel,vector<vector<Mat>> &SChannel){
 Point2f P1,P2,P3,P4;      vector<double> angle0;
 vector<double> angle90(2);vector<double> xx;    vector<double> yy;
 vector<double> da(2);     vector<double> db(2); vector<double> man(2);
 da[0]=0; db[0]=0; man[0]=0; angle90[0]=0; //Fix initial values
 for (int cc=1;cc<12;cc++){ //Accessing each channel
   P1=pointsChannel[0][cc];   P2=pointsChannel[1][cc];   //Line 1
   P3=pointsChannel[0][cc-1]; P4=pointsChannel[1][cc-1]; //Line 2
   da[1]=P2.x-P1.x;  db[1]=P2.y-P1.y;
   phase(da,db,angle0,true); //360-angle0[1] -> alpha [degrees]
   double alpha=360-angle0[1];    double theta=180-beta; //[degrees]
   double gama=(90+alpha-beta);   double limit=cos(gama*PI/180);
   vector<Mat> ForeplusChannel; //Vector with new matrices
   double res = cv::norm(cv::Mat(P1),cv::Mat(P2)); //Line norm
   Mat final2 = Mat::zeros(Size(int(res),
                      floor((widthChannel[cc]/limit))),src.type());
   Mat final3 = Mat::zeros(src.size(),src.type());
   for(int i=0;i<res;i++){ //Following P1-P2
     man[1]=i;   //Unit vector, jump 1 pixel
     //Calculates (x,y) coordinates
     polarToCart(man,angle0,xx,yy, true); //With angle in degrees
     Point2f Pn1,Pn2;    Pn1=Point2f(P1.x+xx[1],P1.y+yy[1]);
     for (int j=0;j<floor((widthChannel[cc]/limit))-1;j++){
       man[1]=j; //Unit vector, jump 1 pixel
       angle90[1]=theta; polarToCart(man,angle90,xx,yy, true);
       Pn2=Point2f(Pn1.x+xx[1],Pn1.y+yy[1]);
       if(Pn1.y+yy[1]>src.rows-1){break;}
       if (Pn2.x>0 && Pn2.x<mOriginal.cols &&
               Pn2.y>0 && Pn2.y<mOriginal.rows){
         Mat B;
         if (src.depth() == CV_32FC1){
           //Take values from the original Image
           getRectSubPix(src,Size(1,1),Pn2,B);//Bilinear interpolation
           float valn= B.at<float>(Point(0,0));//Read the new value
           final2.at<float>(Point(i,j))=valn;  //Remake a new matrix
           float valc= src.at<float>(Pn2);
           final3.at<float>(Pn2)=valc; } } } } //if,if,for,for
   Rect myROI2(0, 0, final2.cols ,widthChannel[cc]-1);
   Mat Channel2;   Channel2=final2(myROI2); //Image with 32FC1
   ForeplusChannel.push_back(Channel2);//Save a new matrix per channel
   ForeplusChannel.push_back(final3); //New image only with one channel
   SChannel.push_back(ForeplusChannel);  } } //for,void
```

Appendix J: Code: Wavelength calibration

```
//Find Center of mass per channel, findCM
//Input: SChannel, matrices with only one channel
//Output: vector with center of mass from all the blobs
void findCM(vector<vector<Mat>> SChannel,vector<Point2f> &CenterMass){
  for(uint i=0; i<SChannel.size();i++){
      Mat B,blurr,threshold_output; //final3 --> B
      SChannel[i][3].copyTo(B);     //Image only with one channel
      blur(B,blurr,Size(9,9));
      double max,min;  //Check max and min values
      minMaxIdx(blurr,&min,&max);
      if(max-float(80.0/1023) >= min+float(100.0/1023)){
        threshold(blurr,threshold_output,max-float(80.0/1023),
                  1.0,THRESH_BINARY);
        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;
        Mat adjMap;
        minMaxIdx(threshold_output, &min, &max);
        //Put in 8C3 image to use "findContours"
        convertScaleAbs(threshold_output, ttadjMap, 255/max);
        findContours(adjMap, contours, hierarchy, CV_RETR_TREE,
                          CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
        //Get the moments
        vector<Moments> mu(contours.size());
        //Get the mass centers:
        vector<Point2f> mc(contours.size());
        double maxArea=0;
        int posMax=0;
        for(uint i=0;i<contours.size();i++){
            //Moments
            mu[i]=moments(contours[i],false);
            //Center of mass
            mc[i]=Point2f(mu[i].m10/mu[i].m00,mu[i].m01/mu[i].m00);
            drawContours(mRgb,contours,i,Scalar(50,205,50),
                              4,8,hierarchy,0,Point());
        double area=contourArea(contours[i]);
        //Find the biggest blob
        if(area>maxArea){maxArea=area;posMax=i;}
        }//for
        //Save the center of mass from the biggest blob
        CenterMass.push_back(mc[posMax]);
      }//if
  }//for
}//void
```