```
//Countstars for Autowow                                                    Oct 2007
//-----------------------------------------------------------------------------------------------------------------------------------------------
/*The image processing is performed through the opensource imaging library OpenCv in C++. Some library and header
files are OS specific and omitted . Here are the core functions to find and categorize stars by size and brightness and to map
the results onto a display.*/
//-----------------------------------------------------------------------------------------------------------------------------------------------
int countstars(capture_info newcapture, bool show)
{
int numstars = 0;
size.width = IMAGE_WIDTH;size.height = IMAGE_HEIGHT;
CvRect BR; CvPoint p;
double minval, maxval;
CvSeq* contour = 0;
string imagename;
string sp = "_";
short radius = 20;
double area = 0.0;
color ncolor;
//zero out the images
cvZero(singlech); cvZero(diff); cvZero(stack);
//load background
dark = cvLoadImage(darkframe.c_str(), 1);
// subtract background and stack images (if they exist)
for (int i = 0; i< newcapture.numgoodimages; i++)
{
if(cvMean(stack) < TH_stack)
{
cout << "mean of ith stack: " << cvMean(stack) << ", " << i << endl;
cvZero(image);
imagename = imagepath + newcapture.place + sp + newcapture.timestamp4name + sp + toString(i) + ".jpg";
if((image = cvLoadImage(imagename.c_str(), 1)) == 0)
{
cout << "ERROR loading image.." << "at # " << i << " of " <<
newcapture.numgoodimages << endl;
exit(1);
}
cvSub(image, dark, diff, NULL);
cvAdd(stack, diff, stack, NULL);
}//end ifcvMean <stack
}
//convert to gray scale
cvConvertImage(stack, singlech, NULL);
//normalize (convert scale based on maxvalue..)
cvMinMaxLoc(singlech, &minval, &maxval, NULL, NULL, 0);
//cout << minval << ", " << maxval << endl;
cvConvertScale(singlech, singlech, (255.0/maxval), 0);
//low pass filter (fat)
cvSmooth(singlech, singlech, CV_BLUR, 3, 3);
//binary threshold
int effective_threshold = cvMean(singlech) + (cvMean(singlech) * TH_factor *
newcapture.numgoodimages);
if(effective_threshold > TH_limit)
effective_threshold = TH_limit;
cout << "using this as the threshold: " << effective_threshold << endl << endl;
cvThreshold(singlech, final, effective_threshold, 255.0, CV_THRESH_BINARY);
//find the centers of the clusters and the number of stars after stacking
CvMemStorage* storage = cvCreateMemStorage(0);
cvFindContours(final, storage, &contour, sizeof(CvContour), CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE );
```

```
numstars = 0;
for( ; contour != 0; contour = contour->h_next )
{
if(show)
{
area = fabs(cvContourArea(contour, CV_WHOLE_SEQ));
if (area < TH_area)
{
BR = cvBoundingRect(contour,0);
p.x = BR.x + int(BR.width/2);
p.y = BR.y + int(BR.height/2);
cvCircle(final, p, radius, CV_RGB(255,255,255));
ncolor = get_color(area);
cvCircle(image, p, radius, CV_RGB(ncolor.r,ncolor.g,ncolor.b));
}
else
cout << "too big to be a star" << endl;
}
numstars++;
}
//release mem storage
cvReleaseMemStorage(&storage);
return (numstars);
}
//-----------------------------------------------------------------------------------------------------------------------------------------
short create_darkimage()
{
//create a site (including temperature) and date specific darkframe for a reference of the (shot) noise generated by the CCD
camera
CvScalar mean, std;
short retval; char result;
char* windowname = "darkimages";
string sp = "_"; int key = -1;
size.width = 640; size.height = 480;
short i = 0; short num_images = 25;
cout << "DARK IMAGE. Cover lens of telescope. and hit ENTER.." << endl;
//wait for key
while(!_kbhit()) {Sleep(50);}
cout << "preparing for dark image.." << endl;
result = PrepareCamera(cameratype, format);
//create windows for output images
cvNamedWindow(windowname,1);
//create an image object
dark = cvCreateImage(size, IPL_DEPTH_8U, 3);
//grabe a few dummy frames first..
cout << "taking 25 images to warm up.." << endl;
for(i=0;i<num_images;i++)
{ cvZero(dark);
//capture an image
if(theCamera.CaptureImage() == CAM_SUCCESS)
{
theCamera.getRGB((unsigned char*) dark->imageData);
cvShowImage(windowname,dark);
cvWaitKey(1);
}
}
//get the mean and standard deviation from the last dark image
cvAvgSdv(dark, &mean, &std, NULL);
```

```
cout << "hit y to accept" << endl;
cin >> result;
if(result == 'y')
{
cvSaveImage(darkframe.c_str(), dark);
cout << "darkimage saved" << endl;
retval = 1;
}
else
{
cout << "NO dark stored - repeat..." << endl;
retval = 0;
}
//stop capture
theCamera.StopImageCapture();
//release header and image data
cvReleaseImage(&dark);
//kill window
cvDestroyWindow(windowname);
return retval;
}
//-----------------------------------------------------------------------------------------------------------------------------------------------
color get_color(double area)
{
color ncolor;
short color_multiplier = 20;
short color_add = 100;
short color_low = 5;
ncolor.r = area*color_multiplier;
ncolor.g = int(area*color_multiplier/2);
ncolor.b = int(area*color_multiplier/2);
if(ncolor.r < color_low)
{
ncolor.g = int(area + color_add);
ncolor.b = int(area + color_add);
}
if(ncolor.r > TH_8bitcolor)
{
ncolor.r = TH_8bitcolor;
ncolor.g = 0;
ncolor.b = 0;
}
cout << ncolor.b << "," << ncolor.g << endl;
return(ncolor);
}
//-----------------------------------------------------------------------------------------------------------------------------------------------
```