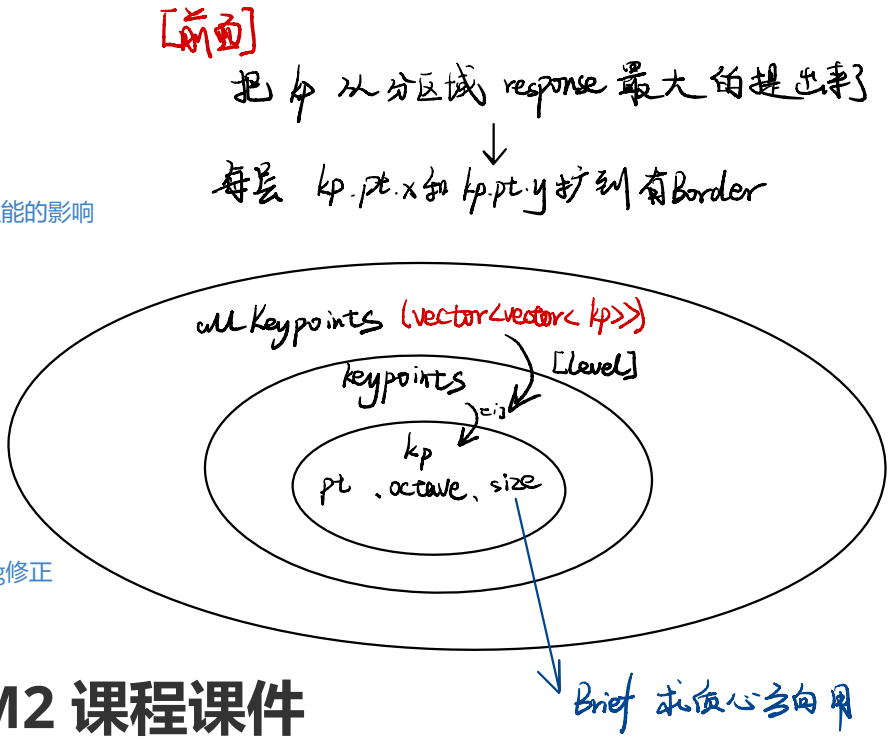


第2周：ORB_SLAM2 课程课件

本课件是公众号 计算机视觉life 旗下课程《全网最详细的ORB-SLAM2精讲：原理推导+逐行代码分析》（点击可跳转课程详情）的课程课件。谢谢各位学员的支持！

本课程对应的注释代码：https://github.com/electech6/ORBSLAM2_detailed_comments

由于源码注释和课件在持续更新，所以：

如视频课程中注释与上述GitHub中有不同，**以GitHub上最新源码为准。**

如视频课程中课件与本课件不同，**以本课件为准。**

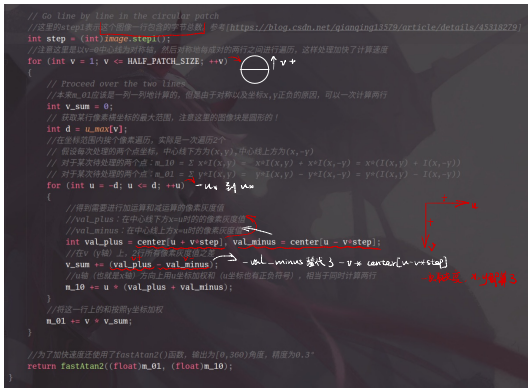
本周课程重点：

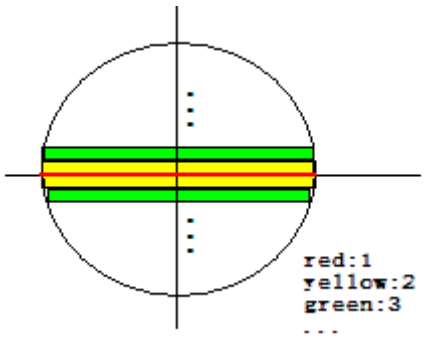
1. 理解ORB特征点旋转不变性的原理及代码高效实现方法。
2. 理解ORB-SLAM2里的ORB特征点提取相对OpenCV里ORB特征的优势。
3. 了解高斯模糊、去畸变。
4. 理解双目稀疏立体匹配的原理及代码实现。
5. **掌握单目初始化里特征匹配的原理及代码**（重要）。

1. ORB 特征点

1.1 函数IC_Angle 加速计算技巧

在一个圆域中算出m10（x坐标）和m01（y坐标），计算步骤是先算出中间红线的m10，然后在平行于x轴算出m10和m01，一次计算相当于图像中的同个颜色的两个line。

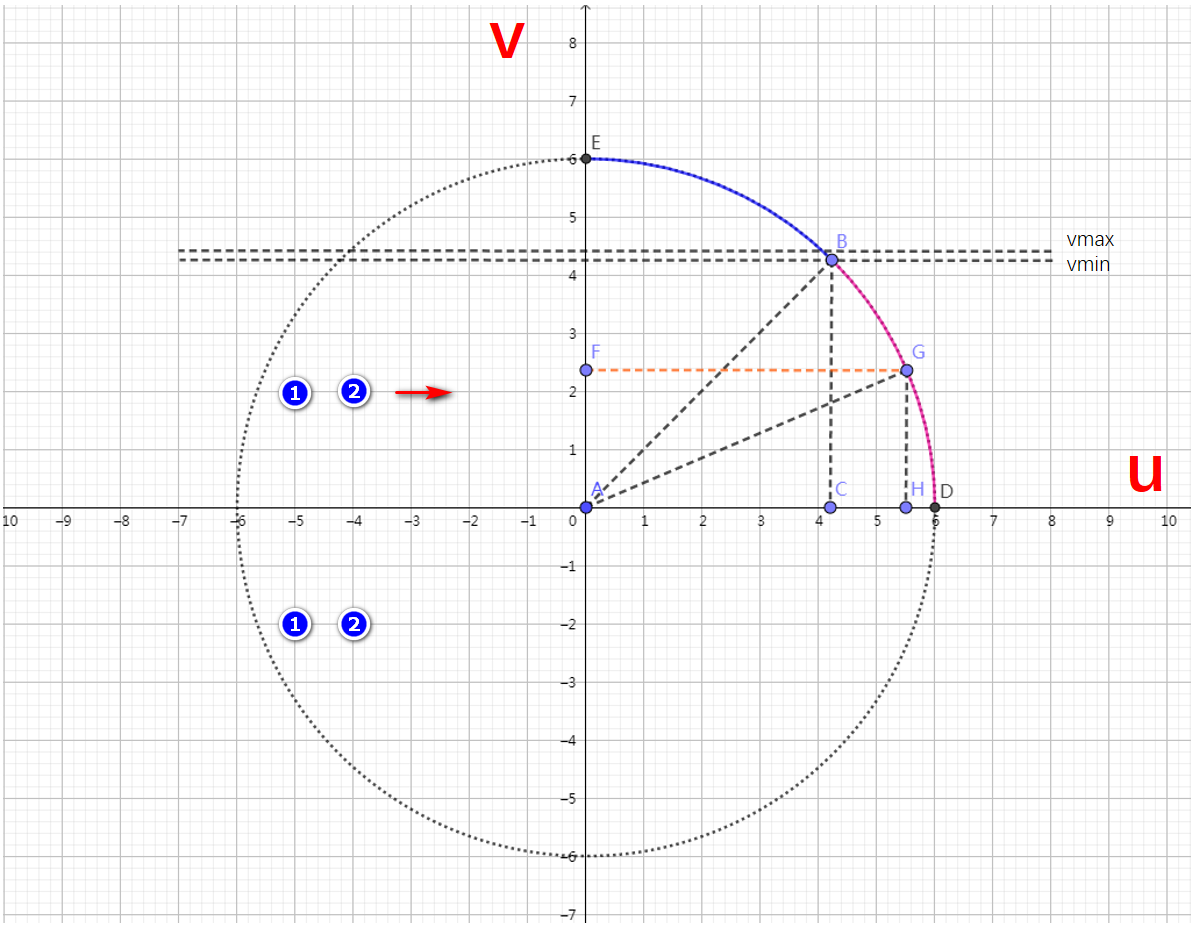




一次算两行

上图参考来源

详细示例图：



1.2 角度的解释

§ fastAtan2()

计算出的是角度

float cv::fastAtan2 (float y,
float x
)

Python:

retval = cv.fastAtan2(y, x)

Calculates the angle of a 2D vector in degrees.

The function fastAtan2 calculates the full-range angle of an input 2D vector. The angle is measured in degrees and varies from 0 to 360 degrees. The accuracy is about 0.3 degrees.

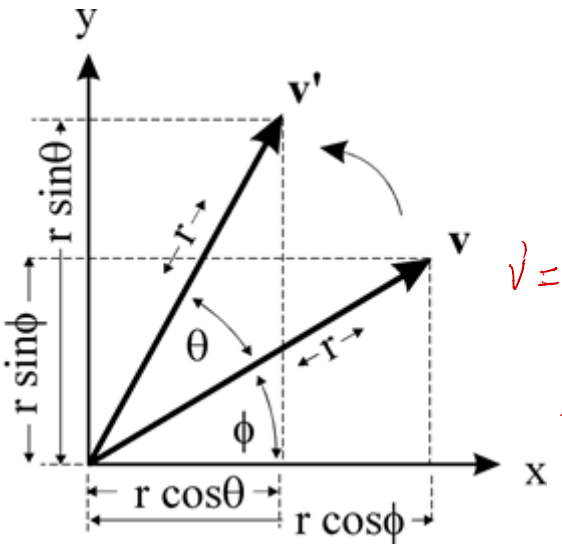
Parameters

x x-coordinate of the vector.

y y-coordinate of the vector.

1.3 Steer BRIEF中如何旋转主方向?

点v 绕 原点旋转θ 角，得到点v'，假设 v点的坐标是(x, y) ，那么可以推导得到 v'点的坐标（x', y')



那么

$$
\begin{aligned}
x &= rcos(\varphi) \\
y &= rsin(\varphi) \\
x' &= rcos(\theta+\varphi) = rcos(\theta)cos(\varphi) - rsin(\theta)sin(\varphi) = xcos(\theta) - ysin(\theta) \\
y' &= rsin(\theta+\varphi) = rsin(\theta)cos(\varphi) + rcos(\theta)sin(\varphi) = xsin(\theta) + ycos(\theta)
\end{aligned}
$$

得到

$$
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}
$$

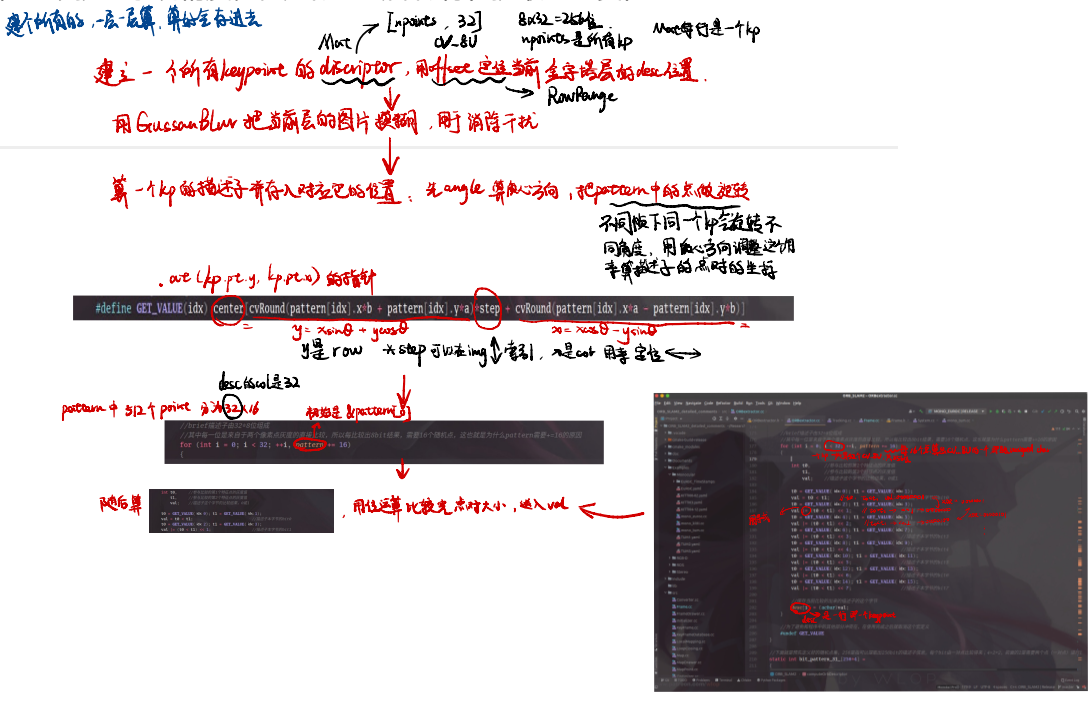
1.4 ORB特征提取策略对ORB-SLAM2性能的影响

ORB-SLAM2中的ORB特征提取方法相对于OpenCV中的方法，提高了ORB-SLAM2的轨迹精度和鲁棒性。增加特征提取的均匀性可以提高系统精度，但是似乎会降低特征提取的重复性。

参考

2. 图像平滑

OpenCV函数高斯模糊



§ GaussianBlur()

```
void cv::GaussianBlur ( InputArray  src,
                        OutputArray dst,
                        Size        ksize,
                        double      sigmaX,
                        double      sigmaY = 0,
                        int         borderType = BORDER_DEFAULT
                      )
```

BORDER_REFLECT_101 边缘插值用的镜像的

Python:

```
dst = cv.GaussianBlur( src, ksize, sigmaX[, dst[, sigmaY[, borderType]]] )
```

Blurs an image using a Gaussian filter.

The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.

Parameters

src	input image; the image can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.
dst	output image of the same size and type as src.
ksize	Gaussian kernel size. ksize.width and ksize.height can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from sigma.
sigmaX	Gaussian kernel standard deviation in X direction.
sigmaY	Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed from ksize.width and ksize.height, respectively (see **getGaussianKernel** for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of ksize, sigmaX, and sigmaY.
borderType	pixel extrapolation method, see **BorderTypes**

常用的高斯模板，中间权重最大

$$\frac{1}{16}\times\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

$$\frac{1}{273}\times\begin{bmatrix}1&4&7&4&1\\4&16&26&16&4\\7&26&41&26&7\\4&16&26&16&4\\1&4&7&4&1\end{bmatrix}$$

能使一些干扰描述子"GG"

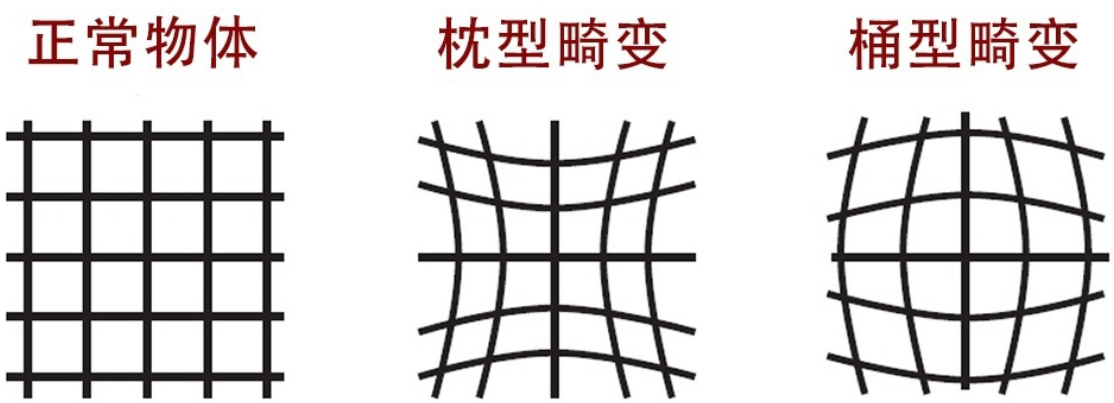
→前一张有个明显的，后一张没有（高斯模糊掉）

高斯公式

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

3. 去畸变

特征点去畸变



OpenCV去畸变函数解释

角发生了变化。通常假设这些畸变呈多项式关系，即：

$$\begin{aligned} x_{\text{distorted}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{\text{distorted}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned}. \tag{5.10}$$

其中 $[x_{\text{distorted}}, y_{\text{distorted}}]^{\mathrm{T}}$ 是畸变后点的**归一化坐标**。另一方面，对于**切向畸变**，可以使用另外的两个参数 p_1, p_2 来进行纠正：

$$\begin{aligned} x_{\text{distorted}} &= x + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_{\text{distorted}} &= y + p_1(r^2 + 2y^2) + 2p_2 xy \end{aligned}. \tag{5.11}$$

因此，联合式(5.10)和式(5.11)，对于相机坐标系中的一点 $\boldsymbol{P}$，我们能够通过 5 个畸变系数找到这个点在像素平面上的正确位置：

§ undistortPoints() [1/2]

```
void cv::undistortPoints ( InputArray  src,
                           OutputArray dst,
                           InputArray  cameraMatrix,
                           InputArray  distCoeffs,
                                       R =
                           InputArray  noArray(),
                           InputArray  P = noArray()
                         )
```

Python:

```
dst = cv.undistortPoints(     src, cameraMatrix, distCoeffs[, dst[, R[, P]]]        )
dst = cv.undistortPointsIter( src, cameraMatrix, distCoeffs, R, P, criteria[, dst] )
```

Computes the ideal point coordinates from the observed point coordinates.

The function is similar to **undistort** and **initUndistortRectifyMap** but it operates on a sparse set of points instead of a raster image. Also the function performs a reverse transformation to projectPoints. In case of a 3D object, it does not reconstruct its 3D coordinates, but for a planar object, it does, up to a translation vector, if the proper R is specified.

For each observed point coordinate (u, v) the function computes:

$$
\begin{aligned}
&x'' \leftarrow (u - c_x)/f_x \\
&y'' \leftarrow (v - c_y)/f_y \\
&(x', y') = undistort(x'', y'', \texttt{distCoeffs}) \\
&[X\,Y\,W]^T \leftarrow R * [x'\,y'\,1]^T \\
&x \leftarrow X/W \\
&y \leftarrow Y/W \\
&\text{only performed if P is specified:} \\
&u' \leftarrow x f'_x + c'_x \\
&v' \leftarrow y f'_y + c'_y
\end{aligned}
$$

where *undistort* is an approximate iterative algorithm that estimates the normalized original point coordinates out of the normalized distorted point coordinates ("normalized" means that the coordinates do not depend on the camera matrix).

The function can be used for both a stereo camera head or a monocular camera (when R is empty).

Parameters

src Observed point coordinates, 1xN or Nx1 2-channel (CV_32FC2 or CV_64FC2).

dst Output ideal point coordinates after undistortion and reverse perspective transformation. If matrix P is identity or omitted, dst will contain normalized point coordinates.

cameraMatrix Camera matrix $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$.

内参矩阵K用来将 u,v 转为 x',y' 归一化坐标

distCoeffs Input vector of distortion coefficients $(k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, \tau_x, \tau_y]]]])$ of 4, 5, 8, 12 or 14 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed.

R Rectification transformation in the object space (3x3 matrix). R1 or R2 computed by **stereoRectify** can be passed here. If the matrix is empty, the identity transformation is used.

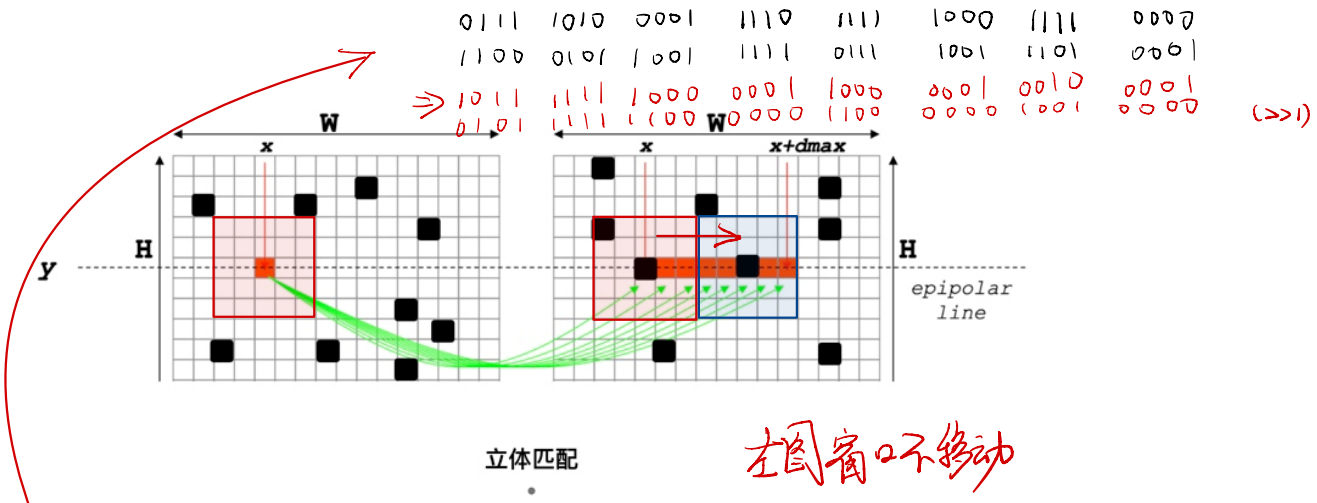
P New camera matrix (3x3) or new projection matrix (3x4) $\begin{bmatrix} f'_x & 0 & c'_x & t_x \\ 0 & f'_y & c'_y & t_y \\ 0 & 0 & 1 & t_z \end{bmatrix}$. P1 or P2 computed by **stereoRectify** can be passed here. If the matrix is empty, the identity new camera matrix is used.

去畸变效果

4 双目稀疏立体匹配

稀疏立体匹配原理

函数ComputeStereoMatches()



立体匹配

左图窗口不移动

* 两帧图像稀疏立体匹配
* 输入：两帧立体矫正后的图像对应的orb特征点集
* 过程：
  1. 行特征点统计.
  2. 粗匹配.
  3. 精确匹配SAD.
  4. 亚像素精度优化.
  5. 最有视差值/深度选择.
  6. 删除离缺点(outliers).
* 输出：稀疏特征点视差图/深度图和匹配结果

用描述子求汉明距离，找出右图该行中dist最小的点

滑动窗口，比较两周围图像块与左图中图像块的相似度

视差公式：z 深度，d （disparity）视差，f 焦距，b （baseline）基线

$u_L>0 \quad u_R<0$

$$z = \frac{fb}{d}, \quad d = u_L - u_R.$$

视差就是左右眼在成像平面下像素的距离

亚像素插值

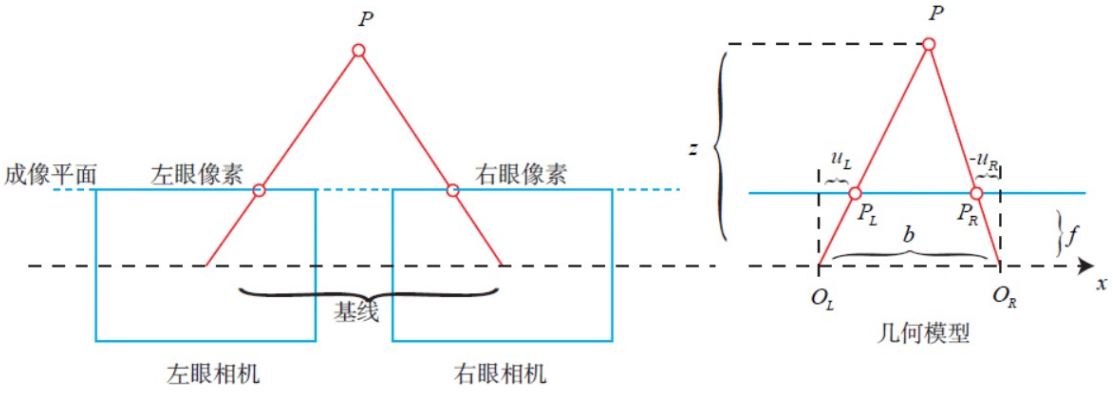
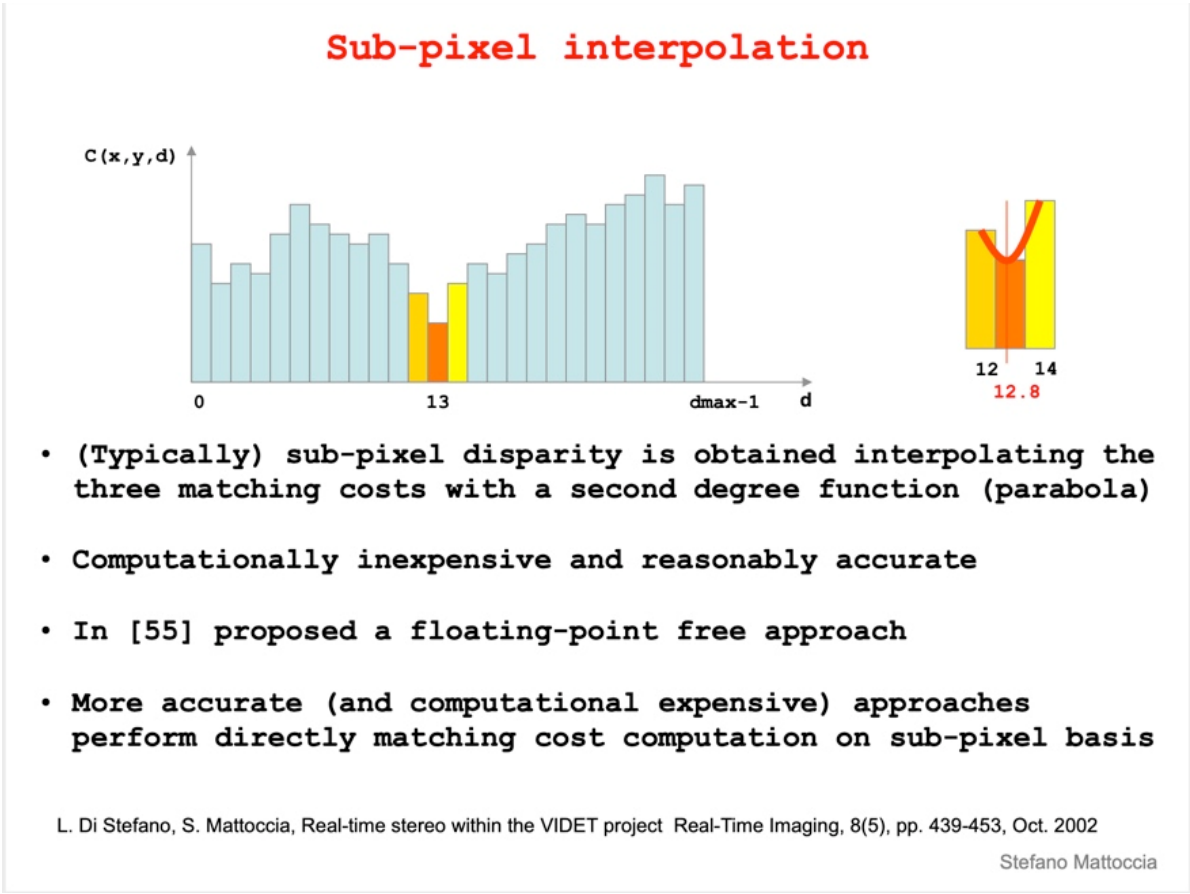


图 5-6　双目相机的成像模型。O_L, O_R 为左右光圈中心，方框为成像平面，f 为焦距。u_L 和 u_R 为成像平面的坐标。请注意，按照图中坐标定义，u_R 应该是负数，所以图中标出的距离为 $-u_R$。

$$\frac{z}{b} = \frac{z-f}{b-u_L+u_R} = \frac{z-f}{b-d}$$

$$\Rightarrow \quad zb - zd = zb - fb$$

$$\Rightarrow \quad d = \frac{fb}{z}$$



Sub-pixel Enhancement: Finally, a sub-pixel enhancement process based on quadratic polynomial interpolation is performed to reduce the errors caused by discrete disparity levels [20]. For pixel $\mathbf{p}$, its interpolated disparity d^* is computed as follows:

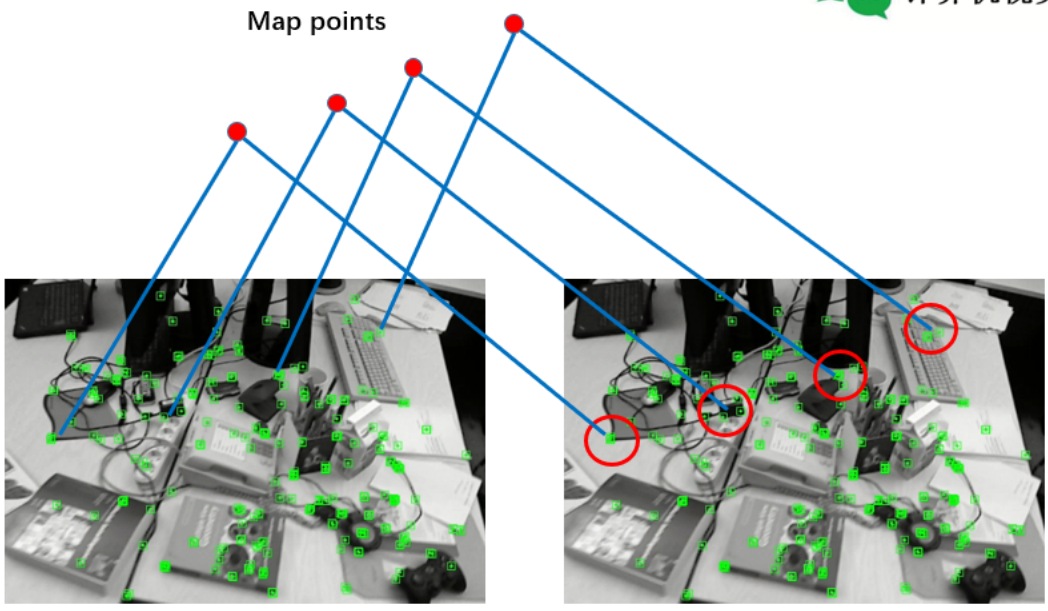
$$d^* = d - \frac{C_2(\mathbf{p}, d_+) - C_2(\mathbf{p}, d_-)}{2(C_2(\mathbf{p}, d_+) + C_2(\mathbf{p}, d_-) - 2C_2(\mathbf{p}, d))} \tag{7}$$

where $d = D_L(\mathbf{p}), d_+ = d + 1, d_- = d - 1$. The final disparity results are obtained by smoothing the interpolated disparity results with a 3×3 median filter.

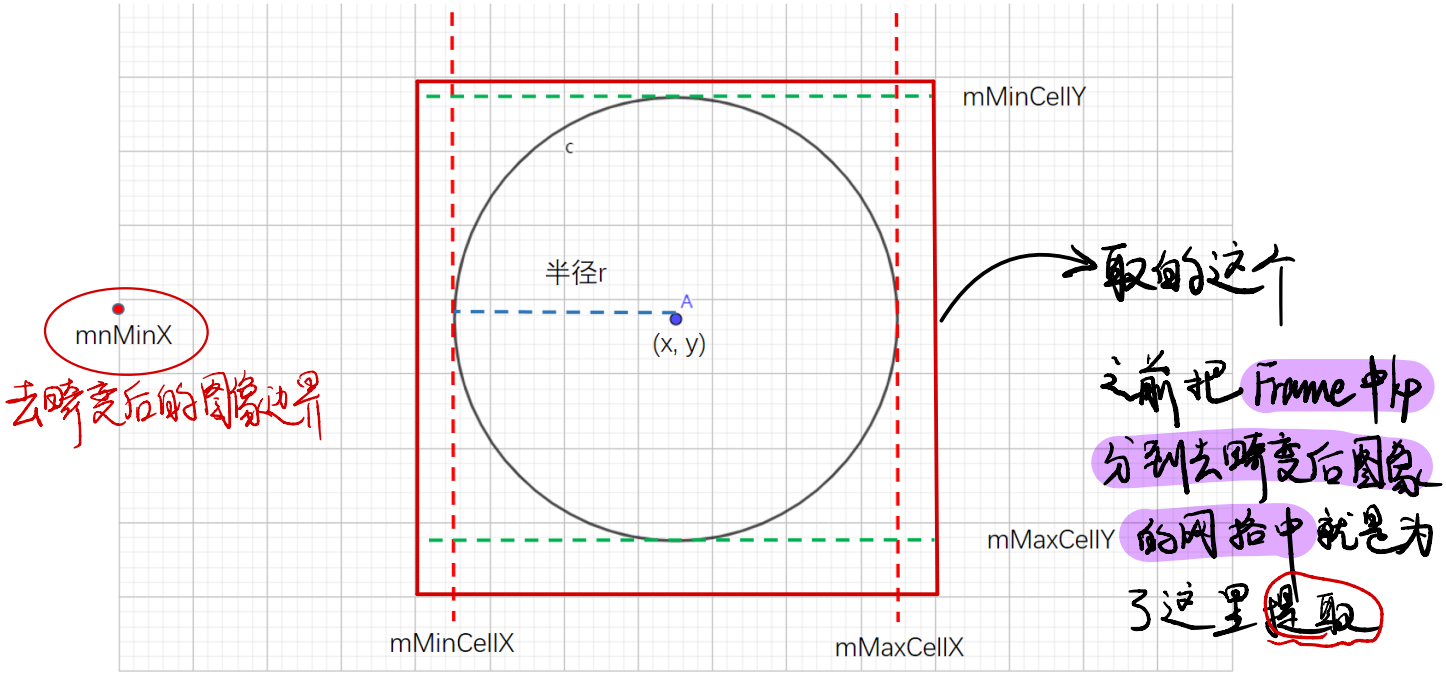
5. 单目初始化中特征点搜索匹配

5.1 函数SearchForInitialization

(vbPrevMatched 存放上帧的特征点,
vnMatches12 存放匹配关系
index是img1特征点的位置
存放的值是img2特征点的位置)



5.2 函数GetFeaturesInArea



5.3 通过角度一致性过滤外点原理及bug修正

初始化采用了金字塔第0层的keypoints

找img1中kp周围 R=100的圆的外接正方形，取最相近的网格.

FRAME_GRID_ROWS = 48
COLS = 64

默认resolution
=640×480

原因：错误把长度为30的直方图（单个bin角度范围12°）变成了长度为12（单个bin角度范围30°），

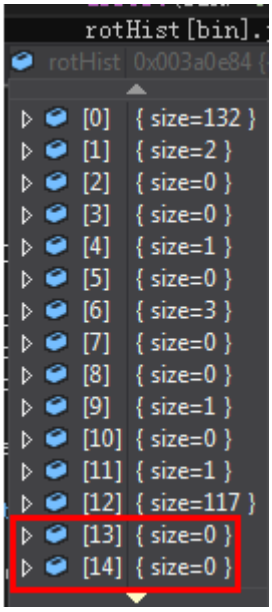
影响：弱化了对角度不一致性的错误剔除，导致只有角度差超过30°（本来是12°）才会被去除。

rotHist 预分配了30个bin，实际第13-30个都没有用到

修改前打印输出

```
rot = 345.953; bin = 12
rot = 347.258; bin = 12
rot = 2.77868; bin = 0
rot = 0.149423; bin = 0
rot = 359.748; bin = 12
rot = 0.0733032; bin = 0
rot = 359.469; bin = 12
rot = 0.248901; bin = 0
rot = 359.873; bin = 12
rot = 0.470245; bin = 0
rot = 359.901; bin = 12
rot = 359.913; bin = 12
rot = 3.95618; bin = 0
rot = 4.16946; bin = 0
rot = 1.0623; bin = 0
rot = 1.03557; bin = 0
rot = 359.286; bin = 12
rot = 359.286; bin = 12
rot = 1.8367; bin = 0
rot = 0.888199; bin = 0
```

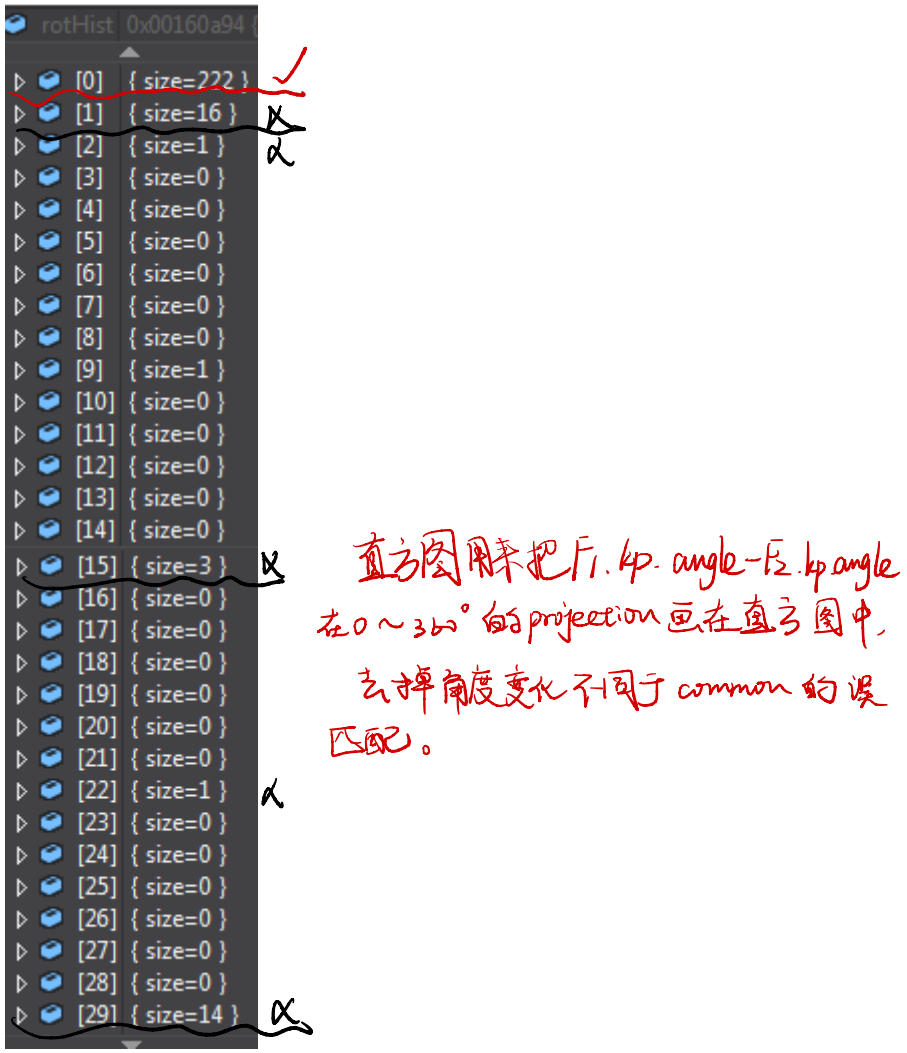
直方图用来判断orientation
用来去些角度不一致的匹配


rotHist[bin].
rotHist 0x003a0e84 {
[0] { size=132 }
[1] { size=2 }
[2] { size=0 }
[3] { size=0 }
[4] { size=1 }
[5] { size=0 }
[6] { size=3 }
[7] { size=0 }
[8] { size=0 }
[9] { size=1 }
[10] { size=0 }
[11] { size=1 }
[12] { size=117 }
[13] { size=0 }
[14] { size=0 }

修改后打印输出

```
rot = 359.936; bin = 30
rot = 9.02561; bin = 1
rot = 359.747; bin = 30
rot = 0.2117; bin = 0
rot = 0.0107851; bin = 0
rot = 1.3223; bin = 0
rot = 4.2951; bin = 0
rot = 359.575; bin = 30
rot = 12.9874; bin = 1
rot = 6.37431; bin = 1
rot = 359.403; bin = 30
rot = 1.29114; bin = 0
rot = 354.754; bin = 30
rot = 106.491; bin = 9
rot = 353.854; bin = 29
rot = 0.651245; bin = 0
rot = 0.505882; bin = 0
rot = 0.146748; bin = 0
rot = 358.641; bin = 30
rot = 357.253; bin = 30
rot = 359.265; bin = 30
rot = 353.544; bin = 29
rot = 0.168167; bin = 0
rot = 0.027504; bin = 0
rot = 359.822; bin = 30
rot = 359.716; bin = 30
rot = 0.00915527; bin = 0
rot = 357.79; bin = 30
rot = 354.483; bin = 30
rot = 359.195; bin = 30
rot = 1.03874; bin = 0
rot = 359.977; bin = 30
rot = 359.696; bin = 30
rot = 359.852; bin = 30
rot = 176.608; bin = 15
rot = 12.4402; bin = 1
rot = 1.01569; bin = 0
rot = 357.3; bin = 30
rot = 6.14644; bin = 1
rot = 358.133; bin = 30
rot = 358.249; bin = 30
rot = 5.80826; bin = 0
rot = 4.07965; bin = 0
rot = 0.151054; bin = 0
rot = 344.193; bin = 29
rot = 359.936; bin = 30
rot = 0.247917; bin = 0
rot = 3.9053; bin = 0
```

直方图分布



总结，修改后代码

能够更有效的滤掉方向不一致的匹配对，更精确了。

举个例子，假如方向差别6°，

原来的代码：factor = 1/30，假设rot =6°，

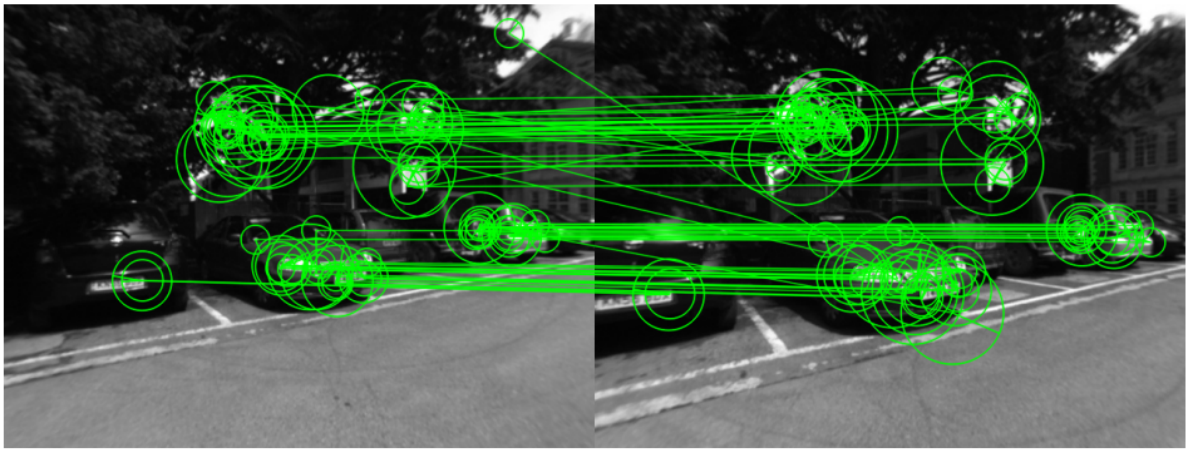
计算得到 所在 bin =0; 和正常的角度±1°（bin=0）混在一起，不会被滤掉

改进后代码 factor = 1/12，假设rot =6°，

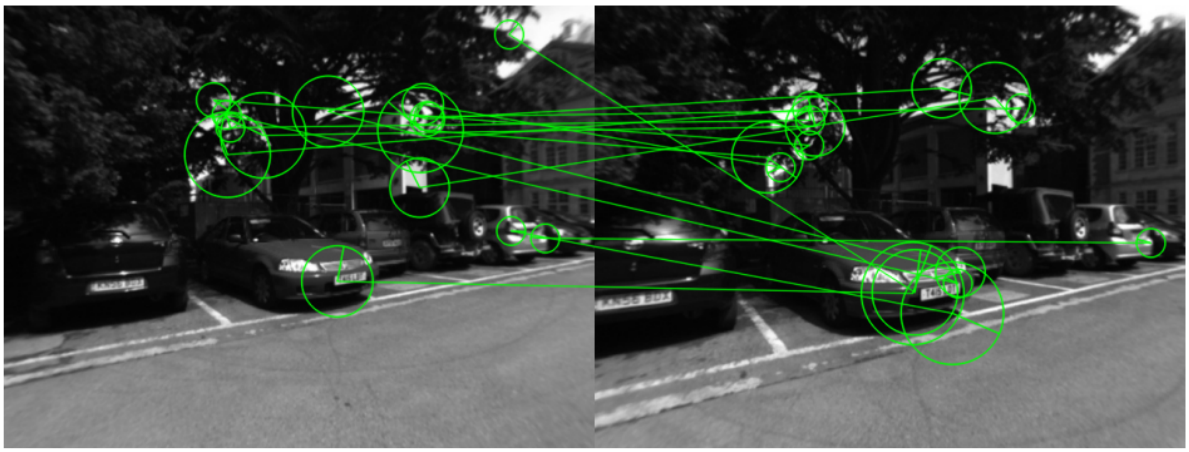
计算得到 所在 bin =1

会被当做异常值滤掉

下图（a）是初始假定的匹配结果，（b）是经过方向一致性检测后删除的错误的匹配对



(a) Putative matches.



(b) Discarded matches by orientation consistency test.