An Optimized N-Dimensional Hough Filter for Detecting Spherical Image Objects

Release 0.00

Kishore Mosaliganti, Arnaud Gelas, Paul Cowgill and Sean Megason

September 30, 2009

Department of Systems Biology, Harvard Medical School, Boston, MA-02139, USA

Abstract

An Insight Toolkit (ITK) algorithm for detection of spherical objects using Hough methods with voting is presented in this paper. Currently, the usage of Hough methods for detecting linear and circular elements exists for 2*D* images in ITK. The current work extends those filters in several ways. Firstly, the new filters operate on *N*-dimensional images. Secondly, they work in physical coordinates which is quite essential in medical imaging modalities. Thirdly, they are optimized (multi-threaded execution, stratified sampling etc.) for usage on large datasets and show a significant speedup even in 2*D* and on small images. Our implementation follows the same underlying mathematics of Hough transforms (as implemented by the 2*D* filters) but with some minor variations. The main variation lies in the pattern of voting that involves selecting voting regions easily and efficiently accessible to region iterators rather than cones that are difficult to generalize in higher dimensions. We include 2D example code, parameter settings and show the results generated on embryonic images of the zebrafish from optical microscopy.

Latest version available at the Insight Journal [http://hdl.handle.net/1926/1]

Contents

1 Introduction

In image analysis, we are often interested in segmenting spherical objects in images. Examples of spherical objects include cells, nuclei, embryos or virus particles, astronomical objects like stars or other point processes etc. Therefore, there is considerable interest in segmenting using geometric attributes. Often, these objects are occluded by overlapping objects but fit a spherical geometry on the non-occluded boundaries. Other common problem with fitting a parametric geometric shape is the variations in the object shape and noise in the images. The Hough transform provides a mathematical framework for easily detecting partially occluded but simple geometry by looking at the parameter space of the geometry. Robustness from image noise is achieved by using a voting framework where individual voxels vote on the likelihood of parameters based on their location and intensity information.

In the current ITK framework, there are two filters utilizing the Hough methods namelly, `itkHoughTransform2DLinesImageFilter` and `itkHoughTransform2DCirclesImageFilter`. Both these filters are inadequate to our needs and need to be fundamentally re-worked. Firstly, most images acquired are high-dimensional and these filters do not fully make use of ITK's templated mechanism in their coding style and hence, restricted to $2D$ processing alone. They assume unit spacing among pixels which is not justified in current modalities where the image spacing depends on the modality and can be anisotropic. Finally, they are poorly optimized with multiple iterations over the same image domain. The bulk of the computational resources is spent in voting on image regions that are shaped irregularly.

In our current submission, we introduce a new filter `itkHoughTransformRadialVotingImageFilter` that solves the above problems. We essentially re-implemented the Hough transform in N-dimensions for detecting spherical objects. Our coding allows us to work in physical coordinates and be efficient in the computation of the voting regions. Our voting regions are regularly-shaped that allow region iterators to quickly run through them. We also save on costly iterations on the image domain by using already multithreaded ITK filters.

2 Implementation

This filter derives from the base class `itkImageToImageFilter`. The filter identified appropriate voxels, computes the local gradient at this point and votes on a small region defined using the minimum and maximum radius given by the user, and fill in the array of radii. The input is an image, and the output consists of the accumulator image that shows the voting result on the image domain. This shows the probability of the centers. The other output consists of a radius image which has the average radius of the circle. We also output a list of spherical objects with center and radii using the `itkSphericalObject` class. We now describe each of the parameters, their range and typical values. There is no typical limit that can be set on most parameters but depends on experimentation. Note that except for the first three, the remaining constitute weights to the different energy terms. Depending on their contribution to the overall energy, these weights need to be modified so that all the terms have an influence.

- `m_Threshold` - Pixels above some conservative intensity threshold are considered during the process. Usually set conservatively in the range $[0, 255]$ for a 8-bit image. Typical value depends on the intensity of the foreground object that can be decided from a histogram of intensity values. The default setting is 0.
- `m_GradientThreshold` - Boundary pixels above some conservative gradient magnitude threshold are considered during the process. Usually set conservatively in the range $[0, 255]$ for a 8-bit image.

Typical, its value can be decided from a histogram of gradient magnitude values that has a bimodal distribution. One can also inspect the boundaries of foreground objects and note the change in intensity values and set a conservative estimate. This greatly improves the processing time. The default setting is 0.

- `m_OutputThreshold` - In the output accumulator image, only select spherical objects that have been voted beyond a threshold number of times. Once again, this is set on a trial-error basis. It depends on the number of pixels lying on the boundary of objects and helps detecting large objects compared to smaller ones that may result from noise. A lower value may help in the detection of partially occluded objects. The default setting is 0.
- `m_MinimumRadius` - Defines the minimum radius of the objects being detected. It is specified as a fraction of the maximum value found in the accumulator.
- `m_MaximumRadius` - Defines the maximum radius of the objects being detected.
- `m_SigmaGradient` - The standard deviation in physical distances of the derivative of Gaussian (doG) filter used in computing image gradients.
- `m_Variance` - The variance in physical distances for smoothing the accumulator image prior to identifying the center locations. This is important for robustly smoothing out local maxima points.
- `m_VotingRadiusRatio` - The small rectangular region near the center around which voting takes place. The dimensions of the region are specified as a fractional ratio of the `m_MinimumRadius` value. Its range is $[0, 1]$. Typically, it is between 0.1-0.5.
- `m_SphereRadiusRatio` - After identifying the centers, a region of the accumulator is filled with 0s to prevent multiple detections of local maxima regions. The dimensions of the region are specified as a fractional ratio of the `m_MinimumRadius` value. Its range is $[0, 1]$. Typically, it is between 0.4-0.7.
- `m_SamplingRatio` - A sampling is applied on the boundary pixels to speed up the calculations. This ratio is a value in the range of $[0, 1]$.
- `m_NbOfThreads` - The voting procedure is multithreaded. The number of threads can be set by the user depending on the available processing units available.

Our implementation makes use of CMake 2.6 version for compilation and has been tested using ITK 3.16 release.

3 Usage

We begin by including the appropriate header files for the filter.

```
#include "itkHoughTransformRadialVotingImageFilter.h"

...
int main(int argc, char *argv[])
{
  ...
  const     unsigned int    Dimension = 2;
```

```
  typedef   unsigned char   InputPixelType;
  typedef   float           InternalPixelType;

  typedef itk::Image< InputPixelType, Dimension >  FeatureImageType;
  InputImageType::IndexType localIndex;
  InputImageType::SpacingType spacing;
  typedef itk::Image< InternalPixelType, Dimension >  InternalImageType;
```

The following typedef for the filter is required.

```
  typedef itk::HoughTransformRadialVotingImageFilter< InputImageType,
              InternalImageType > HoughTransformFilterType;
```

After initialization, it is imperative that the user specify the number of circles and minimum and maximum radii for the spherical objects. All other parameters have default settings that can be adjusted optionally.

```
HoughTransformFilterType::Pointer houghFilter = HoughTransformFilterType::New();
  houghFilter->SetInput( reader->GetOutput() );
  houghFilter->SetNumberOfSpheres( atoi(argv[3]) );
  houghFilter->SetMinimumRadius(   atof(argv[4]) );
  houghFilter->SetMaximumRadius(   atof(argv[5]) );

  if( argc > 7 )
    {
    houghFilter->SetSigmaGradient( atof(argv[7]) );
    }
  if( argc > 8 )
    {
    houghFilter->SetVariance( atof(argv[8]) );
    }
  if( argc > 9 )
    {
    houghFilter->SetSphereRadiusRatio( atof(argv[9]) );
    }
  if( argc > 10 )
    {
  houghFilter->SetVotingRadiusRatio( atof(argv[10]) );
    }
  if( argc > 11 )
    {
    houghFilter->SetThreshold( atof(argv[11]) );
    }
  if( argc > 12 )
    {
    houghFilter->SetOutputThreshold( atof(argv[12]) );
    }
  if( argc > 13 )
    {
    houghFilter->SetGradientThreshold( atof(argv[13]) );
    }
  if( argc > 14 )
```

```
  {
houghFilter->SetNbOfThreads( atoi(argv[14]) );
  }
if( argc > 15 )
  {
houghFilter->SetSamplingRatio( atof(argv[15]) );
  }
```

The output consists of the accumulator image that probabilistically stores the center information and the radius image. The centers are detected as local maxima regions in the accumulator. The radius is obtained at the corresponding voxel location in the radius image.

```
houghFilter->Update();
InternalImageType::Pointer localAccumulator = houghFilter->GetOutput();
InternalImageType::Pointer radiusImage = houghFilter->GetOutput();
```

The detected circles are identified as follows:

```
HoughTransformFilterType::SpheresListType circles;
circles = houghFilter->GetSpheres( );
std::cout << "Found " << circles.size() << " circle(s)." << std::endl;
```

Note that although a user specifies the number of spherical objects to be identified, it only serves as an upper-limit. Depending on the maxima in the accumulator image, the code outputs a list that may contain fewer circles.

The corresponding circle image can be filled up by first allocating an image whose pixels are initialized to 0 and then setting the boundary pixels to 1.

```
OutputImageType::RegionType region;
region.SetSize( localImage->GetLargestPossibleRegion().GetSize() );
region.SetIndex( localImage->GetLargestPossibleRegion().GetIndex() );
localOutputImage->SetRegions( region );
localOutputImage->SetOrigin(localImage->GetOrigin());
localOutputImage->SetSpacing(localImage->GetSpacing());
localOutputImage->Allocate();
localOutputImage->FillBuffer(0);

typedef HoughTransformFilterType::SpheresListType SpheresListType;
SpheresListType::const_iterator itSpheres = circles.begin();

while( itSpheres != circles.end() )
  {
  std::cout << "Center: ";
  std::cout << (*itSpheres)->GetObjectToParentTransform()->GetOffset()
            << std::endl;
  std::cout << "Radius: " << (*itSpheres)->GetRadius()[0] << std::endl;

  for(double angle = 0;angle <= 2*vnl_math::pi; angle += vnl_math::pi/60.0 )
    {
    localIndex[0] =
```

```
        (long int)((*itSpheres)->GetObjectToParentTransform()->GetOffset()[0]
            + ( (*itSpheres)->GetRadius()[0]*vcl_cos(angle) )/spacing[0] );
      localIndex[1] =
        (long int)((*itSpheres)->GetObjectToParentTransform()->GetOffset()[1]
            + ( (*itSpheres)->GetRadius()[1]*vcl_sin(angle) )/spacing[1] );
      OutputImageType::RegionType outputRegion =
                                  localOutputImage->GetLargestPossibleRegion();

      if( outputRegion.IsInside( localIndex ) )
        {
        localOutputImage->SetPixel( localIndex, 255 );
        }
      }
    itSpheres++;
    }
```

4 Results

The results in this example can be obtained by using `HoughTransformRadialVotingImageFilter2D.cxx` on the input image `input.png`. In this example, spherical embryos of the zebrafish are detected in the optical microscopy image. The image several circular regions with a lot of inhomogenous intensities inside. Hence, this image serves as a good example of the robustness of our filter. The output consisting of circles is written out to the image `output.png`. The parameters to the filter are set at the command line to facilitate easy modification and exploration by the user. The command to run this particular executable is as follows:

```
Usage: ./hough2D
 inputImage
 outputImage
 accumulatorImage
 numberOfSpheres
 radius Min
 radius Max
 SigmaGradient (default = 1)
 variance of the accumulator blurring (default = 1)
 radius ratio of the disk to remove from the accumulator (default = 1)
 voting radius ratio (default = 0.5)
 input threshold
 output threshold
 gradient threshold
 number of threads
 sampling ratio


./hough2D input.tif output.png accumulator.mha 50 6 7.5 1 1 0.5 0.15 100 0.5 1 1 1
```

We also compare the time performance of our implementation with respect to `itkHoughTransform2DCirclesImageFilter` in ITK for the image example shown in Figure 1. The figure has dimensions $1K \times 1K$. The following execution times were obtained:

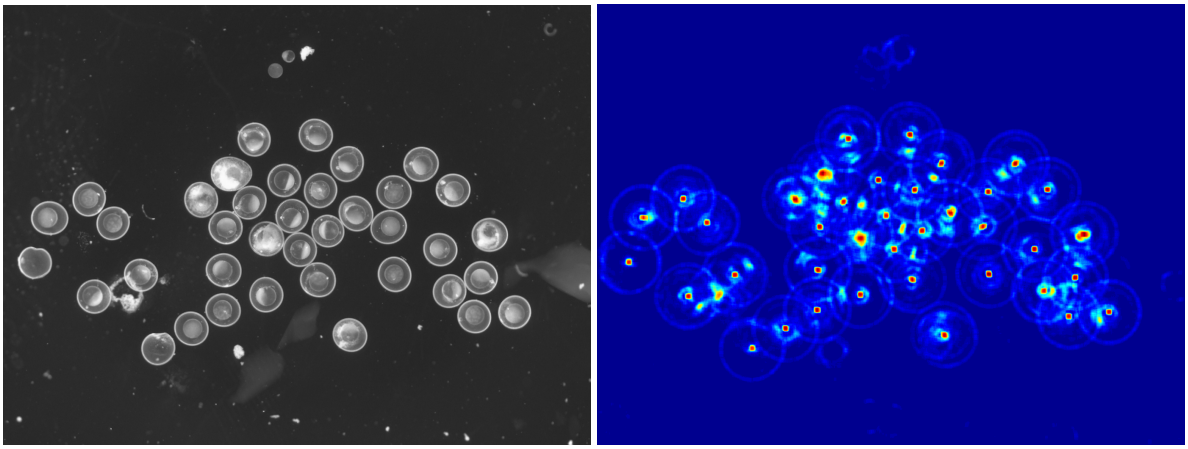
- ITK: 5.7s

- Single-thread execution: 3.3s

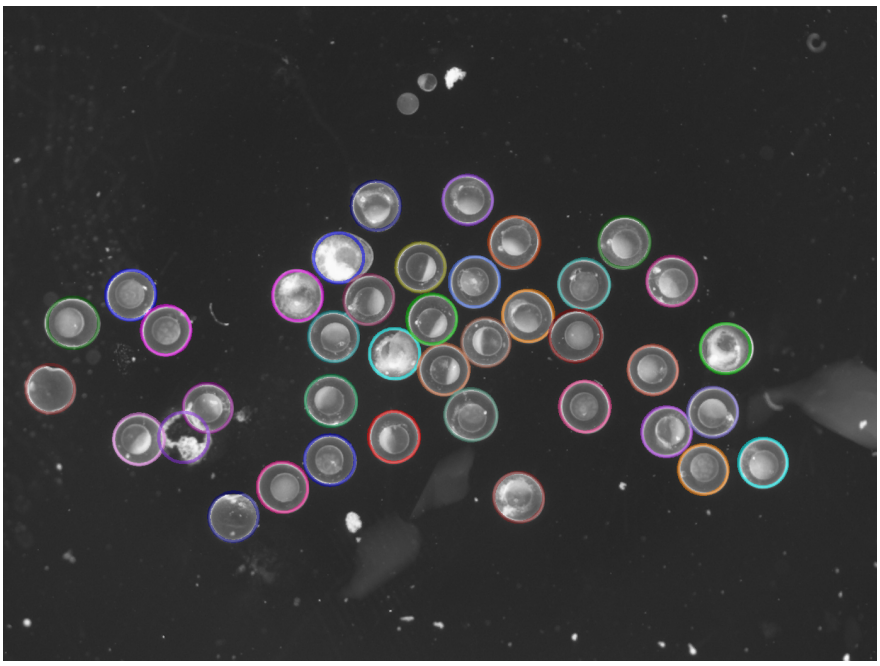
To be fair, we used only one thread and similar parameters in both the implementations yet obtained an almost 2X speedup on a small image. However, note that this implementation is multi-threaded. For larger images and circles, the running time of the Hough transform scales up in a non-linear manner and the performance advantages of the newer implementation will be quite significant. On a $3D$ image with dimensions $1K \times 1K \times 58$, with anisotropic pixel spacing and containing 1000 circles, the following running times were obtained:

- Single-thread execution: 286s
- Dual-thread execution: 164s

The results of the input and the voting are shown graphically in Figure 2.

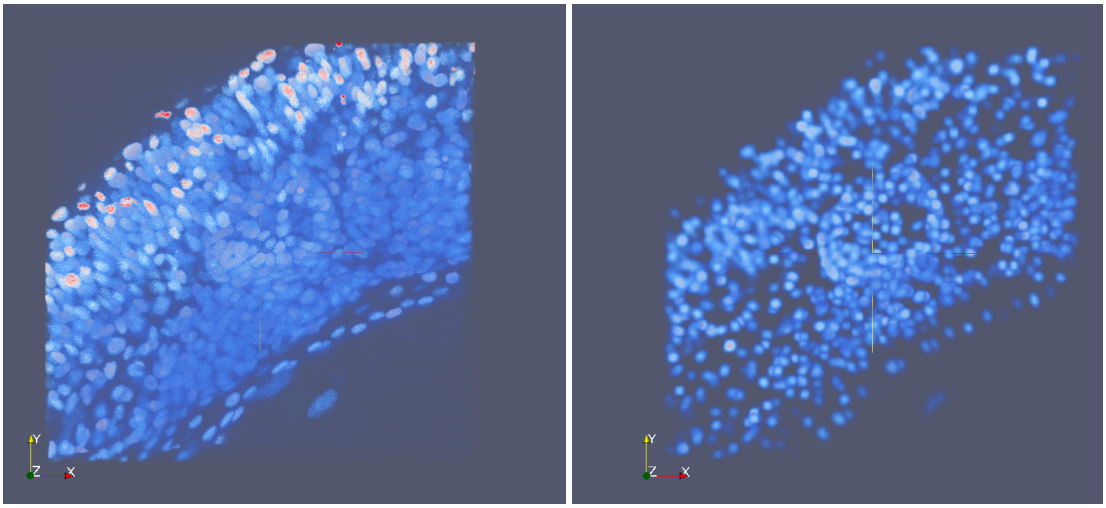


(a) (b)



(c)

Figure 1: Hough segmentation example on a zebrafish embryo image ($1K \times 1K$): (a) Input image, (b) Accumulator and (c) Circles image



(a) (b)

Figure 2: Hough segmentation example on a zebrafish nuclei image ($1K \times 1K \times 58$): (a) Input image (b) Voting accumulator image for detecting centers.