# CartesianToPolar and PolarToCartesian Image Filters

*Release 1.00*

Bhavya Ajani[1] and Sikander Sanjay Sharda[2]

April 09, 2019

[1]Samsung Research Institute, Bangalore

**Abstract**

In this paper, we describe a set of filters, implemented in the Insight Toolkit www.itk.org, for converting an image from Cartesian co-ordinate space to Polar co-ordinate space and vice-versa. Cartesian to Polar conversion of an image is a useful operation in preprocessing stage of certain image-processing algorithm where feature of interest has simplified representation in the polar space. This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the algorithm described in this paper. This adheres to the fundamental principle that scientific publications must facilitate reproducibility of the reported results.

## Contents

## 1 Implementation of Algorithm

Cartesian to Polar co-ordinate conversion of an image is a useful pre-processing step in many image-processing algorithms. This operation is also termed as unwarping, where an area of interest in an image is un-warped around a given point and the resulting representation in polar space, which may be a simplified representation, is useful in pattern recognition problems. Refer to Fig. 1 for an example of this operation.

A Cartesian to polar conversion of an image is an operation of mapping a point, per pixel, in a polar space to the corresponding point in Cartesian space and interpolating the resulting pixel value through interpolation in the neighborhood of the mapped point. The operation has 4 parameters: 1) "Center (cx,cy)": defines an origin point in the Cartesian space; 2) "Radius ($R$)": defines the physical extent of the polar space along radial direction; 3) "Radial Sampling ($R_n$)": defines the number of equi-spaced samples along an extent [0 , R] in radial direction; 4) "Angular Sampling ($A_n$)": defines the number of equi-spaced angular sampling along [0 , 2π) angular space.

The buffer size of the output polar image is ($A_n$ , $R_n$) with origin at (0 , 0) and spacing of ( 2π/ $A_n$ , R/ $R_n$). In this operation pixel value at each pixel location $(\theta, r)$ in polar image is obtained by mapping point $(\theta, r)$ in polar space to (x, y) in Cartesian space using eq. 1 and interpolating the value.

$$x = cx + r\cos(\theta)\,,\ y = cy + r\sin(\theta) \tag{1}$$

$$r \in [0, R]\ in\ steps\ of\ \frac{R}{\mathrm{Rn}}\ \&\ \theta\ \in [0,2\pi) in\ steps\ of\ 2\pi/\mathrm{An}$$

The Cartesian to polar conversion filter is accompanied by a companion inversion filter, to convert an image in polar space to Cartesian space. This inverse operation first maps a pixel (x, y) in Cartesian space to the corresponding pixel $(\theta, r)$ in polar space using equation (2). The pixel value at the pixel (x, y) is than interpolated from the mapped point in polar space using interpolator.

$$r = \sqrt{(x-cx)^2 + (y-cy)^2}\ ,\ \theta = \begin{cases} \arccos\left(\frac{(x-cx)}{r}\right) & y \geq 0 \\ 2\pi - arccos\left(\frac{(x-cx)}{r}\right) & y < 0 \end{cases} \tag{2}$$

Similarly, a point in Cartesian space (x, y) can be mapped to log-polar space $(\theta, \rho)$ and vice versa, using equations (3) and (4) respectively, where $\rho = \log(r)$

$$x = cx + e^{\rho}\cos(\theta)\,,\ y = cy + e^{\rho}\sin(\theta) \qquad \rho \in [0, \log(\mathrm{R})]\ in\ steps\ of\ \frac{\log(\mathrm{R})}{\mathrm{Rn}}\ \&\ \theta\ \in [0,2\pi) in\ steps\ of\ 2\pi/An \tag{3}$$

$$\rho = \log\left(\sqrt{(x-cx)^2 + (y-cy)^2}\right)\ ,\ \theta = \begin{cases} \arccos\left(\frac{(x-cx)}{e^{\rho}}\right) & y \geq 0 \\ 2\pi - arccos\left(\frac{(x-cx)}{e^{\rho}}\right) & y < 0 \end{cases} \tag{4}$$

## 2 Usage

The image filters are implemented as two classes: itk::CartesianImageToPolarImageFilter for conversion of an image from Cartesian to polar space and itk::PolarImageToCartesianImageFilter for conversion of an image from polar space to Cartesian space respectively. Both classes are templated over the scalar type of image that will be transformed. Currently, both the filters work only for 2D images.

- **itk::CartesianImageToPolarImageFilter :**

This filter requires setting an input image (Cartesian space) through `SetInput`() and a center (physical point) through `SetCenter`(). To set the radial extent 'R', use `SetMaxRadius`(). Output image size can be set using the `SetAngularSamples`() and `SetRadialSamples`(). The interpolator can also be explicitly set by using the `SetInterpolator`(). The interpolator defaults to `itk::LinearInterpolatorImageFunction`. To convert the Cartesian image to log-polar space, use `SetUseLogPolar`(true). Example usage is shown below.

```
using CartesianToPolarFilterType = itk::CartesianImageToPolarImageFilter<ImageType>;
CartesianToPolarFilterType::Pointer cartToPolarFilter = CartesianToPolarFilterType::New();

using InterPolatorType = itk::NearestNeighborInterpolateImageFunction < ImageType, double > ;

InterPolatorType::Pointer interpolator = InterPolatorType::New();

cartToPolarFilter->SetInput(inCartImage);
cartToPolarFilter->SetCenter(centroidPhysicalPoint);
cartToPolarFilter->SetInterpolator(interpolator);
cartToPolarFilter->UpdateLargestPossibleRegion();
```

This filter can also transform Cartesian image to log-polar space :

```
cartToPolarFilter->SetUseLogPolar(true);
```

- **itk::PolarImageToCartesianImageFilter :**

This filter requires setting an input image (polar space) through `SetPolarImage`(), reference image `SetReferenceImage`() (region in cartesian space) and a physical point SetCenter(). Output image is created using the spacing, size, direction cosines and origin of the reference image. The interpolator can also be explicitly set by using the `SetInterpolator`(). The interpolator defaults to `itk::LinearInterpolatorImageFunction`. To convert an image in log-polar space to Cartesian space, use `SetUseLogPolar`(true). Example usage is shown below.

```
using PolarToCartesianFilterType = itk:: PolarImageToCartesianImageFilter<ImageType>;
PolarToCartesianFilterType::Pointer polarToCartFilter = PolarToCartesianFilterType::New();

using InterPolatorType = itk::NearestNeighborInterpolateImageFunction < ImageType, double > ;

InterPolatorType::Pointer interpolator = InterPolatorType::New();

polarToCartFilter ->SetPolarImage(inPolarImage);
polarToCartFilter ->SetCenter(centroidPhysicalPoint);
polarToCartFilter->SetReferenceImage(refCartImage);
polarToCartFilter ->SetInterpolator(interpolator);
polarToCartFilter ->UpdateLargestPossibleRegion();
```

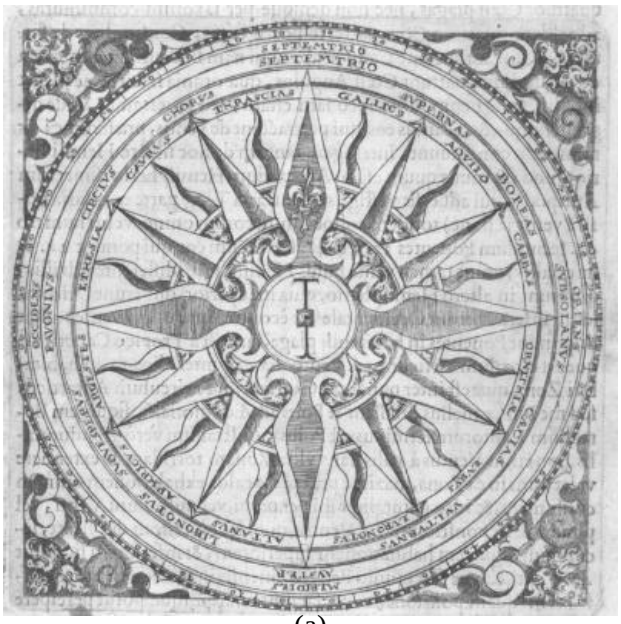
This filter can also transform from log Polar space to Cartesian space :

```
polarToCartFilter ->SetUseLogPolar(true);
```

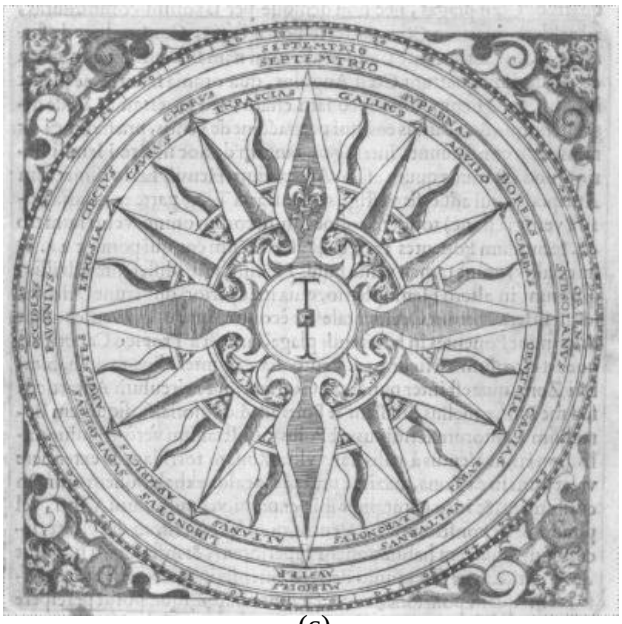
## 3 Testing

The test code included with this article will generate images as in Figure 1 with the following arguments.

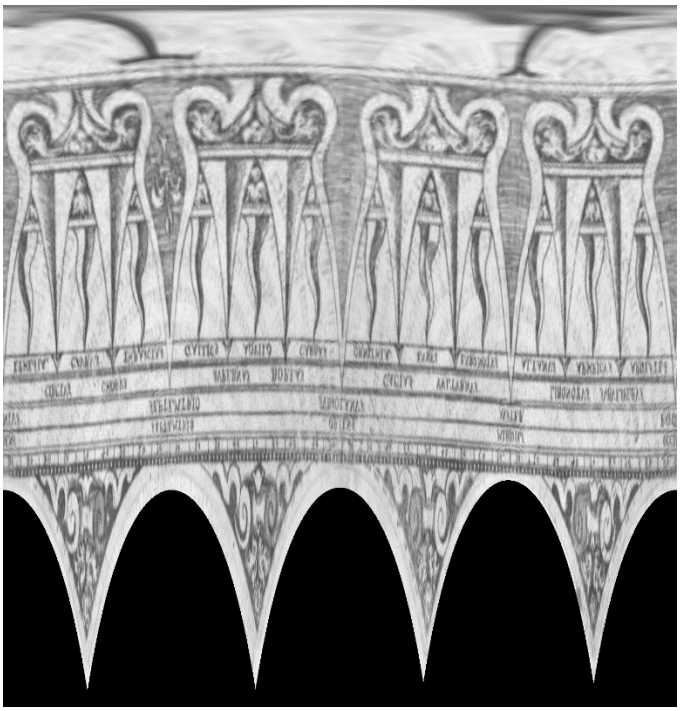
```
CartToPolar.exe cartesianInputImage.mhd

PolarToCart.exe polarInputImage.mha referenceCartesianImage.mhd
```



(a)



(c)



(b)

**Figure 1**: Example of Cartesian to Polar and Polar to Cartesian conversion of an image using defined filters. (a) Is the original image in Cartesian space. (b) Is the unwrapped image in the polar space. (c) Is the recovered image from polar space into original Cartesian space.

The newly generated Cartesian image was compared with the original using `itk::Testing::ComparisonImageFilter.` The average mean pixel value difference between the two images was found to be less than '5'.

## 4 Software Used

The filters were tested on a Windows 10 64-bit computer with ITK version 4.12 and CMake version 3.1