An ITK-based Implementation of the Stochastic Rank Correlation (SRC) Metric

Release 1.00

Philipp Steininger[1], Markus Neuner[1], Wolfgang Birkfellner[2], Christelle Gendrin[2], Michaela Mooslechner[1], Christoph Bloch[2], Supyianto Pawiro[2], Felix Sedlmayer[1,3] and Heinrich Deutschmann[1,3]

November 17, 2010

[1]Institute for Research and Development on Advanced Radiation Technologies (radART), Paracelsus Medical University (PMU), Salzburg, AUSTRIA
[2]Center for Medical Physics and Biomedical Engineering, Medical University Vienna, AKH 4L, Vienna, AUSTRIA
[3]University Clinic for Radiotherapy and Radio-Oncology, Salzburger Landeskliniken (SALK), Salzburg, AUSTRIA

Abstract

Recently, Birkfellner et al. proposed a novel image-to-image merit function (stochastic rank correlation, SRC) for robust intensity-based 2D/3D image registration. In this work, we summarize the basic idea of SRC, and present a generic ITK-based implementation of this image-to-image metric including tests for software verification. Moreover, we provide two simple examples that demonstrate the usage of this metric: a) within the native ITK 2D/3D image registration method, and b) within a recently published extended ITK-based 2D/3D registration framework. It is, however, important to note, that this paper neither covers a comprehensive evaluation of SRC, nor a comparison with other metrics. It rather shows that SRC appears to succeed on a femoral and a porcine data set in the course of ITK-based 2D/3D image registration.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3229]

Contents

1 Introduction

Intensity based registration (2D/3D, 3D/3D) is a common task in medical image processing, and requires similarity measures (metrics) to find the "optimal" transformation between two data sets. For example, an intra-interventional and a pre-interventional data set of the same subject are registered. In general, metrics must cope with dissimilarities between the investigated images. Such discrepancies may be due to differing or suboptimally modeled imaging modalities, acquisition times and occlusions emerging from body parts or additional objects (e.g. patient positioning aids).

Consequently, most of the used metrics consider the statistical distribution of the intensity values to some extent. The **stochastic rank correlation (SRC)** metric, which was proposed by Birkfellner et al. [1] in the context of 2D/3D registration, likewise aims at robustly measuring the similarity of two images by reducing the influence of encountered intensity differences.

2 Stochastic Rank Correlation in a Nutshell

2.1 2D/3D Image Registration

For example, in *projection-based 2D/3D image registration [5]*, N 2D reference images $X_{R,i}(x_{R,i}), i = 1 \dots N$ are registered with projections $\mathcal{P}_i(X_M^{\mathcal{T}}(x_M))$ (DRRs) of a transformed 3D volume $X_M^{\mathcal{T}}(x_M)$. At this, the 3D image coordinates of the volume are denoted by x_M defined over domain $\Omega_M \in \mathbb{R}^3$, and the 2D reference image coordinates are denoted by $x_{R,i}, i = 1 \dots N$ defined over domains $\Omega_{R,i} \in \mathbb{R}^2$. Moreover, the considered data sets are assumed being spatially defined in a common reference coordinate system, thereby, requiring an adequate calibration procedure [8]. The projection-based approach aims at finding a spatial transformation $\mathcal{T}$ that aligns the projections $\mathcal{P}_i(X_M^{\mathcal{T}}(x_M))$ best with the reference data $X_{R,i}(x_{R,i})$:

$$\mathcal{T}_{opt} := \underset{\mathcal{T}}{\operatorname{argmin}} \sum_i \mathcal{F}_i(\mathcal{P}_i(X_M^{\mathcal{T}}(x_M)), X_{R,i}(x_{R,i})) \tag{1}$$

where $X_M^{\mathcal{T}}(x_M)$ denotes the transformed volume data set, and $\mathcal{P}_i$ define projection transformations that map the 3D coordinates $x_M \in \Omega_M$ onto points $x_{M,i}$ of the N 2D domains $\Omega_{R,i}$. The implied optimization problem minimizes a set of cost functions $\mathcal{F}_i$ (or metrics) which measure the similarity between the reference images and the respective DRRs.

2.2 Nature of Stochastic Rank Correlation

In practice, even if $\mathcal{T}_{opt}$ was correctly inferred, significant intensity differences between $X_{R,i}(x_{R,i})$ (X-rays) and $\mathcal{P}_i(X_M^{\mathcal{T}}(x_M))$ (DRRs of a CT) can be observed. These discrepancies *mainly* emerge from insufficient radiometric calibration, finite voxel size and simplified DRR algorithms that do not model details of physical interactions in object (e.g. generation of scatter irradiation) or detector (e.g. energy conversion in build-up or photosensitive layer, energy-dependent quantum efficiency). In addition, the observed intensity differences can in general not be described by linear models. Therefore, metrics that strictly rely on a linear relationship between the compared images (e.g. cross correlation - Pearson's product-moment coefficient) usually fail on 2D/3D registration [1]. In this respect, stochastic rank correlation behaves differently. It is the implementation of Spearman's rank correlation coefficient which solely assumes a monotonic relationship between the compared image intensities. Moreover, it is by definition less sensitive to outliers than Pearson's product-moment coefficient.

In order to compute the SRC coefficient, the image intensities must be mapped onto an ordinal scale. Having image intensities $X(x)$ on a metric scale, the according histogram $h_j(X(x)) = h_{X,j}, j = 1 \ldots N_B$ with a specified number of bins N_B can be extracted. Furthermore, the respective cumulative histogram $H_{X,j} = \sum_{k=1}^{j} h_{X,k}$ can be utilized for generating average ranks $R_{X,j}$ which define an ordinal scale. The average ranks $R_{X,j}$ calculate as

$$R_{X,j} := \begin{cases} H_{X,j-1} + \frac{H_{X,j}}{2}, & \text{if } (j > 1) \wedge (h_{X,j} > 0) \\ \frac{H_{X,j}}{2}, & \text{j = 1} \\ 0, & \text{else} \end{cases} . \tag{2}$$

Subsequently, the original image intensities $X(x)$ can be mapped onto the ordinal scale by using the generated average rank map R_X as a lookup table; the resultant image is called *average rank image* and denoted by $\acute{X}(x)$. Instead of considering all pixels N_X of the investigated image $X(x)$, a simple subset $\mathcal{N}_X < N_X$ of randomly sampled pixels may be sufficient for histogram estimation. Besides an enhanced performance in histogram extraction and coefficient calculation (Eq. 3), the sampling may also prevent "overfitting", and possibly introduce some local non-monotonic intensity relationship "tolerance". Birkfellner et al. [1] reported a pixel sample coverage of 5 % being sufficient for 2D/3D registration in practice. Finally, the SRC coefficient calculates as

$$\mathcal{F}_{SRC}(X_A, X_B) := \frac{6 \sum_{k=1}^{\mathcal{N}} \delta_k^2}{\mathcal{N}(\mathcal{N}^2 - 1)} \tag{3}$$

where X_A and X_B are the compared images, $\mathcal{N}$ is the number of samples, and δ_k denotes the difference between the interpolated rank intensities $\acute{X}_A(x_k)$ and $\acute{X}_B(x_k)$ at coordinate $x_k \in \Omega_A \cap \Omega_B$. The $\mathcal{N}$ sampled coordinates should stay constant during continuous optimization in order to guarantee coefficient calculation on a static pixel set.

Eq. 3 gives a minimum of 0 if the sampled intensities of $\acute{X}_A$ and $\acute{X}_B$ correspond to a perfect increasing monotonic relationship (0 square differences), and gives higher values otherwise. Thus, the SRC metric for image similarity measurement implies a *minimization problem*.

It is worth being mentioned, that if one wants to compute the full rank correlation (RC) coefficient $\mathcal{F}_{RC}$ as proposed in [2], simply all available pixel samples must be considered; i.e. $\mathcal{N} = N$.

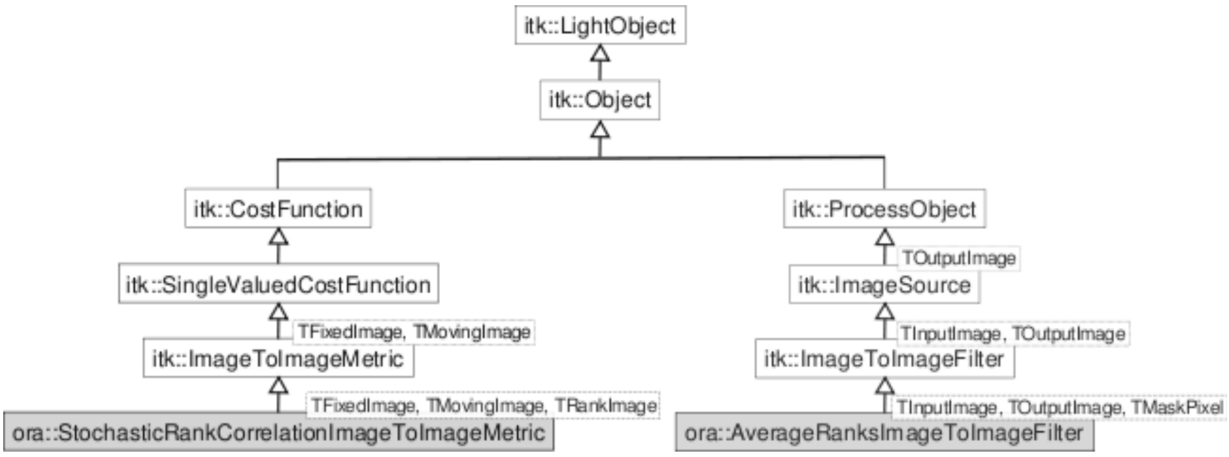

Figure 1: The developed main classes `ora::StochasticRankCorrelationImageToImageMetric` (similarity measure) and `ora::AverageRanksImageToImageFilter` (convenience filter), and the corresponding class hierarchy.

2.3 “Tied Ranks” Correction

In order to correct for the “tied ranks” (samples with equal rank) that are generated during histogram “binning”, *Horn*-correction [4] may be applied. The corrected SRC coefficient $\mathcal{F}^*_{SRC}$ calculates as

$$\mathcal{F}^*_{SRC}(X_A, X_B) := 1 - \frac{2\frac{\mathcal{N}(\mathcal{N}^2-1)}{12} - \Theta_A - \Theta_B - \sum_{k=1}^{\mathcal{N}} \delta_k^2}{2\sqrt{(\frac{\mathcal{N}(\mathcal{N}^2-1)}{12} - \Theta_A)(\frac{\mathcal{N}(\mathcal{N}^2-1)}{12} - \Theta_B)}} \tag{4}$$

where $\Theta_\bullet = \sum_{k=1}^{N_\bullet} \frac{\theta_{\bullet,k}(\theta^2_{\bullet,k}-1)}{12}$ with $N_\bullet$ being the number of distinct ranks (bins) of image $\acute{X}_\bullet$, and $\theta_{\bullet,k}$ being the number of samples having the same rank $R_{\bullet,k}$.

However, first experiments made us believe that the application of this correction method in practice will not have a noticeable influence on registration results.

3 Implemented Classes, Architecture and Tests

NOTE: The presented classes in this section are defined in the namespace `ora` which encapsulates code that is part of the **radART** institution’s open source initiative - **open-radART**.

As shown in Fig. 1, the SRC metric is implemented in the generic template class `ora::StochasticRankCorrelationImageToImageMetric` as typical subclass of `itk::ImageToImageMetric`. In addition to its own code, this class invokes a new convenience image filter `ora::AverageRanksImageToImageFilter` which extracts the *average rank image representation* of an image of arbitrary dimension.

In the following sections the most important characteristics of these classes are highlighted, and explanatory code snippets are provided. More detailed information on the methods and functionality can be found directly in the classes in the form of extensive comments or in a generated **doxygen**-documentation.

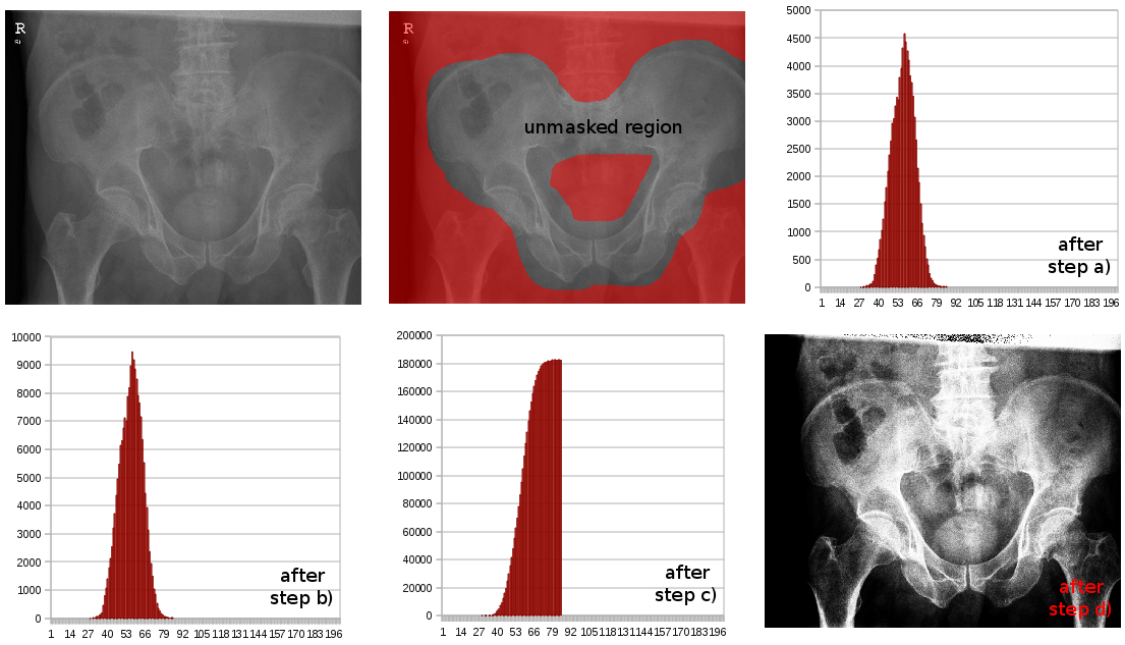

Figure 2: Average ranking example (from left to right and from top to down): an exemplary input image $X(x)$, the overlay of the input and the corresponding mask image $X_M(x)$ (the non-red region represents the pixels that really contribute to histogram estimation), the histogram h_X after step a), the linearly transformed histogram after step b), the rank map R_X after step c), and the generated output average rank image representation $\acute{X}(x)$

.

3.1 ora::AverageRanksImageToImageFilter

This convenience filter involves the following *steps* in order to produce the average rank image $\acute{X}(x)$ of an input image $X(x)$ (see also 2.2):

a) Extract the histogram $h_{X,j}, j = 1 \ldots N_B$ of the image $X(x)$ considering pre-configured minimum and maximum image intensities I_{min} and I_{max}. If a mask image $X_M(x)$ is set, only the pixels that are unmasked ($X_M(x_k) > 0$) will contribute to h_X.

b) This optional step linearly transforms the histogram h_X to its "original" frequency range. This is only relevant if a mask $X_M(x)$ is specified, and the number of unmasked pixels $\mathcal{N}_X$ is smaller than the number of total image pixels N_X. Each frequency $h_{X,j}$ is simply multiplied by the factor $N_X/\mathcal{N}_X$.

c) First generate the cumulative histogram H_X from h_X, and then calculate the rank map R_X (see Eq. 2).

d) Generate the output image $\acute{X}(x)$: iterate through all pixels of $X(x_k)$ and map their intensities to new levels $\acute{X}(x_k)$ using R_X as the central lookup table.

Fig. 2 outlines the input and output of each filter step in the course of an exemplary setup with an overview X-ray of the pelvis.

It is worth being mentioned that this filter is capable of *multi-threading*, actively using the `BeforeThreadedGenerateData()`, `ThreadedGenerateData()` and `AfterThreadedGenerateData()` method interfaces of its superclass. Multi-threading primarily concerns and accelerates step a) where each

thread extracts a partial histogram, and step d) where each thread maps the intensities of a specific pixel subset in the usual ITK fashion (itk::ImageToImageFilter).

The average ranks image filter is *templated* over the input image type (TInputImage), the output image type (TOutputImage) and the mask pixel type (TMaskPixel). It is important that the range of the output image's pixel type is able to cover the average rank range which equals the number of input image pixels N_X or $\mathcal{N}_X$ (mask) in a worst case scenario.

Moreover, the filter fires the following specific *events*:

- ora::AfterHistogramExtraction: Fires after completed histogram extraction. The histogram can be extracted using the GetAveragedRankHistogram() method in the event handler.
- ora::AfterHistogramTransformation: Fires after completed histogram transformation. The transformed histogram can be extracted using the GetAveragedRankHistogram() method in the event handler.
- ora::AfterHistogramRankGeneration: Fires after completed average rank map generation. The rank map can be extracted using the GetAveragedRankHistogram() method in the event handler.

The following *code snippet* shows how to use ora::AverageRanksImageToImageFilter:

```
typedef itk::Image<unsigned short, 2> ImageType;
typedef itk::Image<float, 2> RankImageType;
typedef unsigned char MaskPixelType;
typedef itk::Image<MaskPixelType, 2> MaskImageType;
typedef ora::AverageRanksImageToImageFilter<ImageType, RankImageType,
    MaskPixelType> FilterType;
typedef itk::CStyleCommand CommandType;

// read input image and mask
ImageType::Pointer image = ReadImage<ImageType>("image.mhd");
MaskImageType::Pointer mask = ReadImage<MaskImageType>("mask.mhd");

// set-up the filter
FilterType::Pointer filter = FilterType::New();
// - set input image and mask
filter->SetInput(image);
filter->SetMaskImage(mask);
// - histogram options (min/max intensities, no. bins, clipping)
filter->SetHistogramClipAtEnds(true);
filter->SetHistogramMinIntensity(0);
filter->SetHistogramMaxIntensity(6000);
filter->SetNumberOfHistogramBins(200);
// - further output options
filter->SetDoNotGenerateOutput(false); // we want the output image
filter->SetGenerateOutputForMaskedPixelsOnly(false); // all pixels
filter->SetUseHistogramTransformation(true); // imitate orig. frequ.
// - attach an event handler
CommandType::Pointer cmd = CommandType::New();
cmd->SetCallback(MyCallback);
cmd->SetClientData(filter);
filter->AddObserver(ora::AfterHistogramExtraction(), cmd);
```

```
filter->AddObserver(ora::AfterHistogramTransformation(), cmd);
filter->AddObserver(ora::AfterHistogramRankGeneration(), cmd);

// generate and write rank image
try
{
  filter->Update();
  WriteImage<RankImageType>("image_rank_rep.mhd", filter->GetOutput());
}
catch (itk::ExceptionObject &e)
{
  std::cerr << "ERROR: " << e << std::endl;
}
```

3.2 ora::StochasticRankCorrelationImageToImageMetric

The principle of operation of `ora::StochasticRankCorrelationImageToImageMetric` can be decomposed into three typical functional components:

- **Initialize()**: Makes sure all components and subcomponents are present, initialized and plugged together correctly.
- **GetValue()**: Computes the value of the SRC cost function corresponding to the specified transformation parameters.
- **GetDerivative()**: Estimates the derivative of the SRC cost function corresponding to the specified transformation parameters.

Initialize() involves the following steps:

a) Call the Initialize() method of the superclass `itk::ImageToImageMetric` in order to ensure that the standard components (transformation, interpolator ...) are present and correctly plugged together.

b) Generate an internal mask image $X_M(x)$ which considers the pre-configured *fixed image region*, *fixed image mask* and *number of spatial pixel samples* $\mathcal{N}_X$ to be extracted. It is important to note that $\mathcal{N}_X$ can be directly set by invoking `SetSampleCoverage()`, or implicitly by specifying an external image mask using `SetStochasticUserMask()` that marks the pixels which should contribute to SRC computation.

c) Set-up an internal instance of `ora::AverageRanksImageToImageFilter` according to the metric pre-configuration, and generate the rank image representation $\acute{X}_F$ of the fixed image.

d) Prepare an ITK histogram (`itk::Statistics::Histogram`) instance according to the metric pre-configuration for the moving image later.

GetValue() involves the following steps:

a) Apply the specified transformation parameters to the connected transform component.

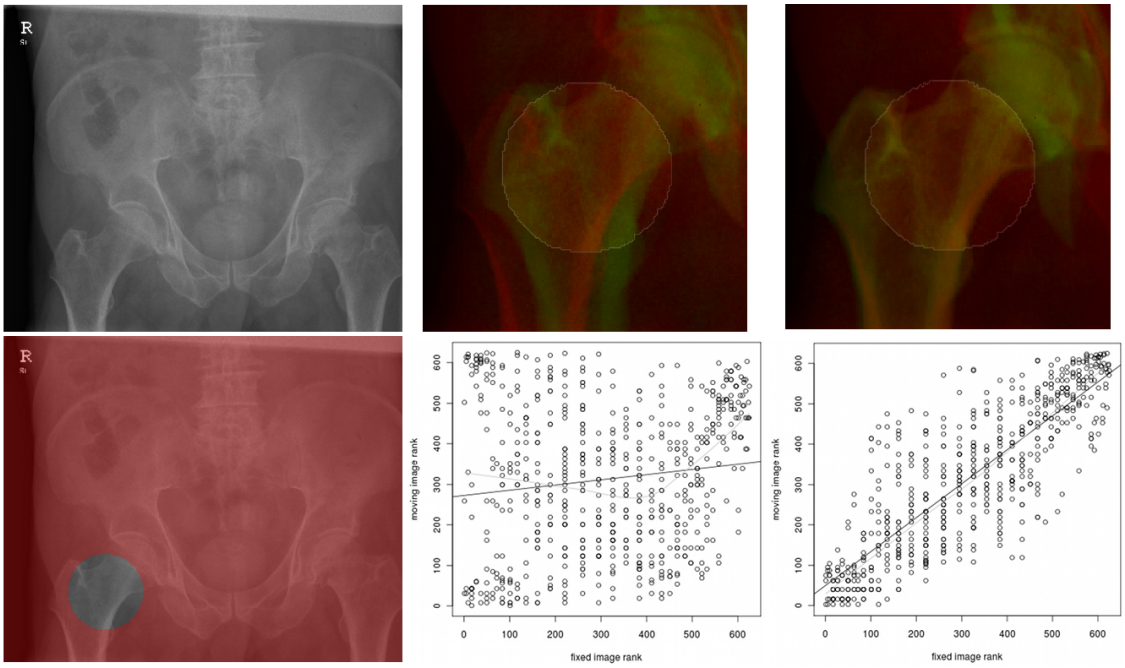

Figure 3: 2D/3D registration example (from top to down): The first column shows a 2D overview X-ray (fixed image) and the applied fixed image mask (the non-red region contributes to SRC computation). The second column shows the red-green-yellow-overlay of the X-ray and the initial DRR (moving image) at the beginning of the process. Furthermore, the fixed image ranks are plotted against the moving image ranks, additionally including a linear regression line (black) and a LOWESS (locally weighted scatterplot smoothing) line (gray). The last column shows the overlay of the fixed image and the final moving image at the end of the registration process. In addition, the according scatterplot is visualized.

b) Extract the rank map $R_{X,M}$ from the moving image. This is achieved by mapping the physical coordinates of all pixels $\acute{X}_F(x_k) > 0$ to the physical space of the moving image, and subsequently interpolating the according moving pixel intensities. If a moving image mask is set, only the pixels within the mask will contribute to $R_{X,M}$. At this, the process of rank map extraction is comparable to the `ora::AverageRanksImageToImageFilter` implementation (3.1), but without multi-threading.

c) Subsequently, the square differences δ_k^2 (see Eq. 3) between the considered moving and fixed image intensities are summed up.

d) Finally, based on $\sum \delta_k^2$, the cost function value is computed according to Eq. 3 or Eq. 4 if Horn-correction is requested. If there is no overlap between the fixed and moving image with respect to the set image masks, GetValue() either returns a constant metric value or throws an exception depending on the configured "NoOverlapReactionMode".

GetDerivative() aims at producing an estimate of the cost function derivative at a specified position in transformation space. The SRC metric implementation uses finite differences for computing this estimation. Using the `SetDerivativeScales()` method one can specify the amount of transformation parameter variation along each dimension.

Fig. 3 shows a 2D/3D registration example where a CT of the right femur is registered with an overview X-ray of the pelvis. Beside of overlay images, scatterplots that visualize the fixed image ranks versus moving

image ranks (image rank plots) of the sampled pixels are shown. The sample distributions shown in the plots are the basic input for SRC coefficient $\mathcal{F}_{SRC}$ computation.

The SRC metric class is *templated* over the fixed image type (`TFixedImage`), the moving image type (`TMovingImage`) and the rank image type (`TRankImage`). Similarly to the average ranks filter, the pixel type of the rank image must be able to cover the rank range of the fixed as well as the moving image.

Moreover, `ora::StochasticRankCorrelationImageToImageMetric` fires the specific `ora::AfterMaskCreation` event. It is invoked directly after internal mask generation (step b) in `Initialize()`, see above). Using the `GetStochasticMask()` method within the event handler, the generated mask can be retrieved.

Furthermore, the SRC metric implementation provides a "debugging"-mode (`SetExtractSampleDistribution()`) which can be used for retrieving the internal sample distributions (`GetSampleDistribution()`) as shown in Fig. 3.

It is worth being noted that, although this metric is histogram-based, it is derived from `itk::ImageToImageMetric` and not from `itk::HistogramImageToImageMetric`. This is due to minor performance advantages. Internally, this metric utilizes the multi-threaded average rank filter once in order to extract the fixed image histogram - there is no need for re-computing the fixed image histogram during each call to `GetValue()` as basically implemented in `itk::HistogramImageToImageMetric` (second histogram sample dimension).

The following *code snippet* shows how to use `ora::StochasticRankCorrelationImageToImageMetric`:

```
typedef itk::Image<unsigned short, 2> ImageType;
typedef itk::Image<float, 2> RankImageType;
typedef ora::StochasticRankCorrelationImageToImageMetric<ImageType,
    ImageType, RankImageType> MetricType;
typedef itk::LinearInterpolateImageFunction<ImageType, double>
    InterpolatorType;
typedef itk::Rigid2DTransform<double> TransformType;
typedef TransformType::ParametersType ParametersType;
typedef itk::CStyleCommand CommandType;
typedef itk::ImageMaskSpatialObject<2> MaskSpatialObjectType;
typedef itk::Array<double> DScalesType;

// read input images and masks
ImageType::Pointer fimage = ReadImage<ImageType>("fixed_image.mhd");
ImageType::Pointer mimage = ReadImage<ImageType>("moving_image.mhd");
MaskSpatialObjectType::Pointer fmask = ReadSpatialObject<
    MaskSpatialObjectType>("fixed_mask.mhd");
MaskSpatialObjectType::Pointer mmask = ReadSpatialObject<
    MaskSpatialObjectType>("moving_mask.mhd");

// set-up the metric and components
MetricType::Pointer metric = MetricType::New();
// - images and masks
metric->SetFixedImage(fimage);
metric->SetFixedImageRegion(fimage->GetLargestPossibleRegion());
metric->SetMovingImage(mimage);
metric->SetMovingImageMask(mmask);
metric->SetFixedImageMask(fmask);
// - configure metric
```

```
TransformType::Pointer transform = TransformType::New();
metric->SetTransform(transform);
InterpolatorType::Pointer interpolator = InterpolatorType::New();
metric->SetInterpolator(interpolator);
metric->SetFixedHistogramMinIntensity(0); // fixed image histogram info
metric->SetFixedHistogramMaxIntensity(4096);
metric->SetFixedHistogramClipAtEnds(true);
metric->SetFixedNumberOfHistogramBins(200);
metric->SetMovingHistogramMinIntensity(0); // moving image histogram info
metric->SetMovingHistogramMaxIntensity(4096);
metric->SetMovingHistogramClipAtEnds(true);
metric->SetMovingNumberOfHistogramBins(200);
metric->SetExtractSampleDistribution(false); // no debugging
metric->SetMovingZeroRanksContributeToMeasure(false);
metric->SetStochasticUserMask(NULL); // use sample coverage instead
metric->SetSampleCoverage(10.0); // 10 % of the pixels
metric->SetUseHornTiedRanksCorrection(true); // use Horn-correction
metric->SetNoOverlapReactionMode(0); // throw exception
DScalesType dscales(transform->GetNumberOfParameters());
dscales.Fill(1.0); dscales[0] = 0.017; // 1 degree, 1 mm
metric->SetDerivativeScales(dscales);
CommandType::Pointer cmd = CommandType::New();
cmd->SetCallback(MyCallback);
cmd->SetClientData(metric);
metric->AddObserver(ora::AfterMaskCreation(), cmd);

try
{
  metric->Initialize(); // intialize the metric
  // compute a few sample cost function values / derivatives
  ParametersType tpars(transform->GetNumberOfParameters());
  tpars.Fill(0);
  srand(time(NULL));
  for (int i = 0; i < 10; i++)
  {
    std::cerr << tpars << std::endl;
    MetricType::MeasureType value = metric->GetValue(tpars);
    std::cout << "value" << tpars << ": " << value << std::endl;
    MetricType::DerivativeType derivative;
    metric->GetDerivative(tpars, derivative);
    std::cout << "derivative" << tpars << ": " << derivative << std::endl;
    tpars[0] = 0.1 - (double)(rand() % 100001) / 500000;
    tpars[1] = 10 - (double)(rand() % 100001) / 5000;
    tpars[2] = 10 - (double)(rand() % 100001) / 5000;
  }
}
catch (itk::ExceptionObject &e)
{
  std::cerr << "ERROR: " << e << std::endl;
}
```

3.3 Software Tests

In order to support automatic software and regression testing, two tests were added: `TestAverageRanksFilter` for `ora::AverageRanksImageToImageFilter`, and `TestStochasticRankCorrelationMetric` for `ora::StochasticRankCorrelationImageToImageMetric`. These tests are fully integrated with the **CMake** and **CTest** environment.

Essentially, these tests internally generate random sample images and masks, and test the most important functionality of the provided classes. Both tests offer the following command line options which may be useful for debugging and/or gaining more detailed information:

- `-h or --help` ... get command line help information
- `-v or --verbose` ... message output to std::cout
- `-co or --csv-output` ... CSV-sheet output of some sample data
- `-io or --image-output` ... sample image output to current directory
- `-xo or --extended-output` ... extended output to std::cout

CTest is configured to invoke the `-v` option by default. Running `ctest --verbose` in the project's root folder should produce a similar output on command line:

```
Test project /radART-dev/src-ij-bin
Constructing a list of tests
Done constructing a list of tests
Checking test dependency graph...
test 1
    Start 1: TestAverageRanksFilter

1: Test command: /radART-dev/src-ij-bin/metric/testing/TestAverageRanksFilter -v
1: Test timeout computed to be: 1500
1:
1: Testing average ranks image to image filter.
1:   * Unmasked histogram extraction test ... OK
1:   * Unmasked rank image test ... OK
1:   * Masked histogram extraction test ... OK
1:   * Masked rank image test ... OK
1:   * Masked, transformed histogram extraction test ... OK
1:   * Masked, transformed rank image test ... OK
1:   * Final reference count check ... OK
1: Test result: OK
1:
1/2 Test #1: TestAverageRanksFilter ..............   Passed    3.08 sec
test 2
    Start 2: TestStochasticRankCorrelationMetric

2: Test command: /radART-dev/src-ij-bin/metric/testing/TestStochasticRankCorrelationMetric -v
2: Test timeout computed to be: 1500
2:
2: Testing stochastic rank correlation image to image metric.
2:   * Various sample coverages with simple value / derivative computation ... OK
```

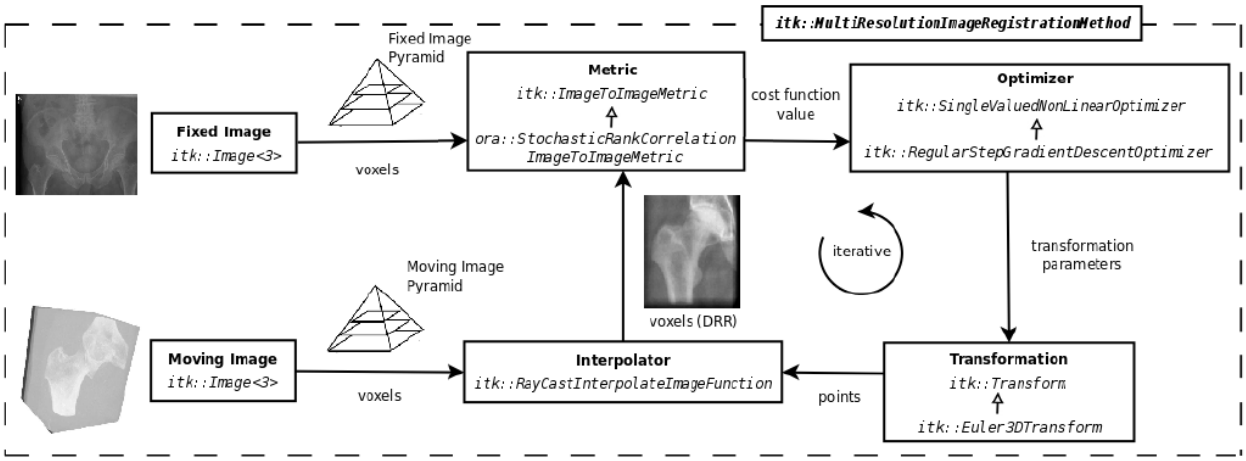

Figure 4: The "native" ITK 2D/3D registration method which is essentially based on itk::RayCastInterpolateImageFunction.

```
2:   * Test deterministic/non-deterministic behavior ... OK
2:   * Check no-overlap behavior ... OK
2:   * Test fixed image region and fixed image mask ... OK
2:   * Fixed + moving image mask, fixed image region test ... OK
2:   * Stress test: sampling around optimum ... OK
2:   * Stress test (with Horn-correction): sampling around optimum ... OK
2:   * 3D stress test: (translational) sampling around optimum ...  OK
2:   * 3D stress test: (translational, with Horn-correction) sampling around optimum ...  OK
2:   * Final reference count check ... OK
2: Test result: OK
2:
2/2 Test #2: TestStochasticRankCorrelationMetric ...   Passed   37.03 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =  40.13 sec
```

4 2D/3D Registration Examples

In order to demonstrate the functionality of the developed classes, a few 2D/3D registration examples are provided based on freely available source code and data sets which are included in this work.

The IntensityBased2D3DRegistration2 example application is based on the IntensityBased2D3DRegistration program from InsightApplications. It represents the **"native ITK-way"** of establishing a 2D/3D registration environment, mainly based on itk::RayCastInterpolateImageFunction. Fig. 4 outlines the hierarchy and links between the involved components.

The second example, 2D3DRegistration2, is based on a previously published **extended ITK-based 2D/3D registration framework** [7] which implements multi-threaded filters for DRR computation, and a flexible framework for component configuration. Since the original extended framework did not support SRC or regular step gradient descent optimization, it was upgraded with a set of sub-classes as outlined in Fig. 5.

Table 1: Initial transformation before, and final transformation after registration of the femur examples. The real transformation (ground truth) is unfortunately unknown, but qualitatively the registrations appeared to succeed.

Transformation Parameter	r_x [°]	r_y [°]	r_z [°]	t_x [mm]	t_y [mm]	t_z [mm]
initial transformation	-90.00	15.00	0.00	84.34	77.95	0.00
result EVOL	-91.46	13.82	-5.07	76.21	70.63	0.00
result RSGD	-90.97	17.10	-4.89	76.26	71.07	0.08

For a more detailed overview, the reader should investigate all classes in the example folder which end with `*WithSRC`.

4.1 Configuration of the Extended Framework

Basically, the extended registration framework is set-up with configuration files (`*.cfg`) that contain `key=value` pairs and `option=key value value ...` constructs. A detailed commented configuration file is available in `sampleconfigs/femur/RegistrationExample-Femur-EVOL.cfg`, while all other configuration examples have a shorter documentation.

The framework enables a multi-resolution approach where many components and settings can be configured individually for each level with `key[level]=value`, where `level` indicates the registration level, starting with 0. The metric, optimizer and the interpolator can be re-configured to adapt to the levels' individual needs. For DRR generation an empirical intensity transfer function can be applied. The registration process is logged and documented in a specified sub-folder of the current directory. Moreover, the current registration parameters are logged to console.

The geometric setup of the registration is defined by the position of the DRR/X-ray (which is restricted to lie within the x/y-plane), the volume origin, the initial transformation and the interpolator (projector) configuration (see Fig. 6).

4.2 Femur Examples

The following configurations show an exemplary registration of a 3D CT image of the right proximal femur (0.9 mm isotropic) and a 2D overview X-ray of the pelvis (0.9 mm isotropic). Both images are stored in MetaImage format (`CT.mhd`, `X-ray.mhd`). The image data are located in the data-directory `sampleconfigs/femur/data` along with a `readme.txt` file that contains information on the image files and the data set license. The registration involves rigid transformation (Euler 3D), stochastic rank correlation metric (128 bins, 5% coverage) and ray-casting DRR generation. Here, only a single resolution level is used for registration, and the fixed image is circular-masked with a radius of 40 mm (see Fig. 3).

The first example `RegistrationExample-Femur-EVOL.cfg` uses an evolutionary optimization strategy (EVOL), and the second `RegistrationExample-Femur-RSGD.cfg` a regular step gradient descent optimization (RSGD). The initial setup is visualized in Fig. 7, the results are displayed in Fig. 8 and the transformation parameters are summarized in Tab. 1. The EVOL optimization took 200 iterations (max) and the RSGD 30 iterations. Plots of the rank correlations (scatterplot of the fixed and moving image ranks) can be seen in Fig. 9 and 10.

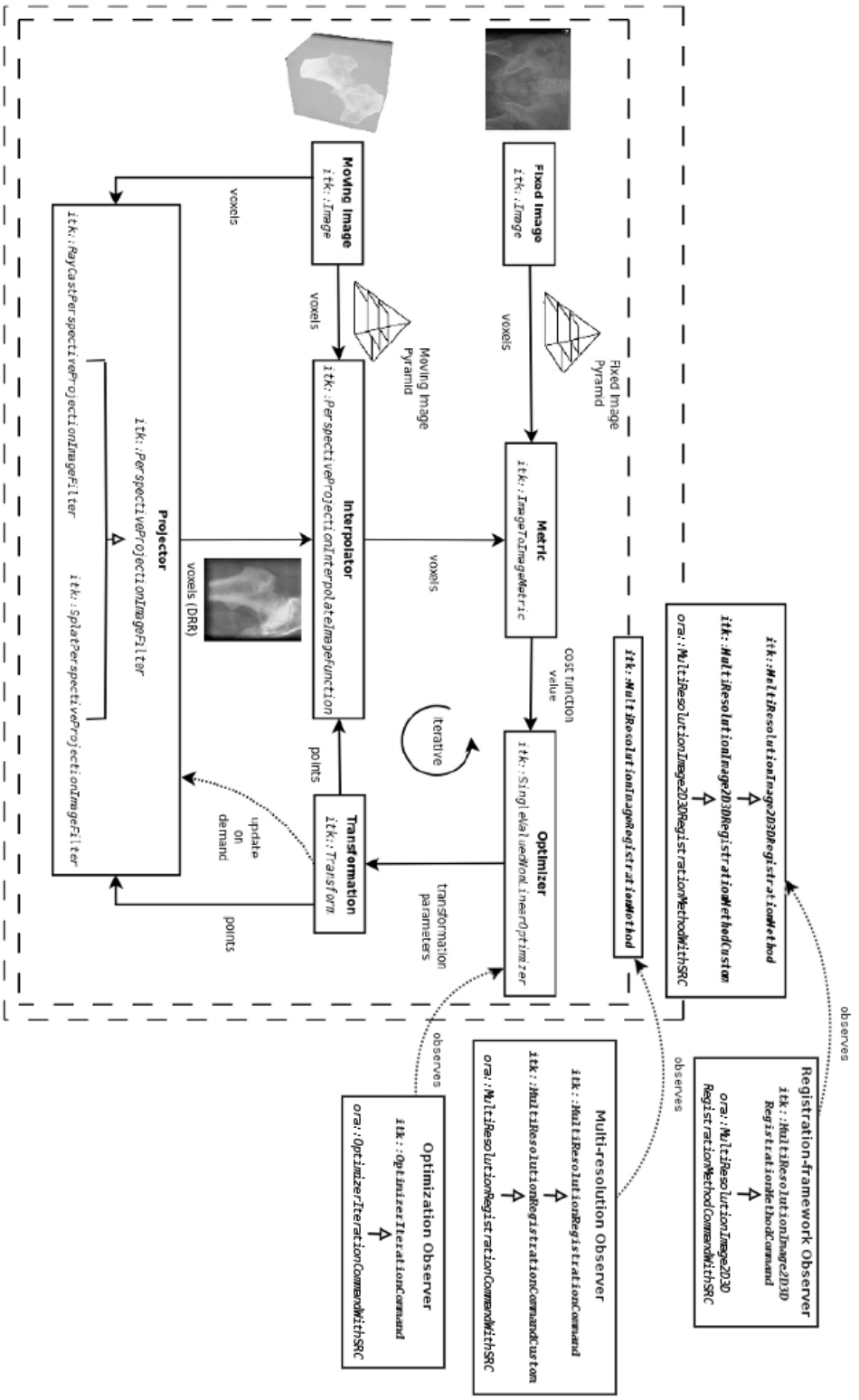

Figure 5: The extended ITK-based 2D/3D registration framework which provides multi-threaded DRR filters, additional support for SRC, and a convenient infrastructure for configuring the registration components.

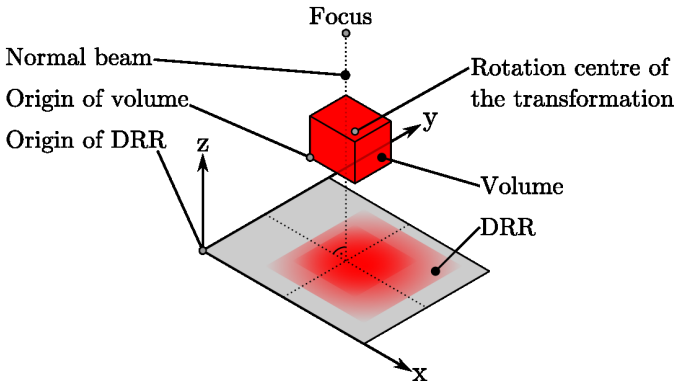

Figure 6: Geometry setup of the extended registration framework.

Figure 7: From left to right: X-ray image (fixed image), DRR with initial transformation, overlay of both images.

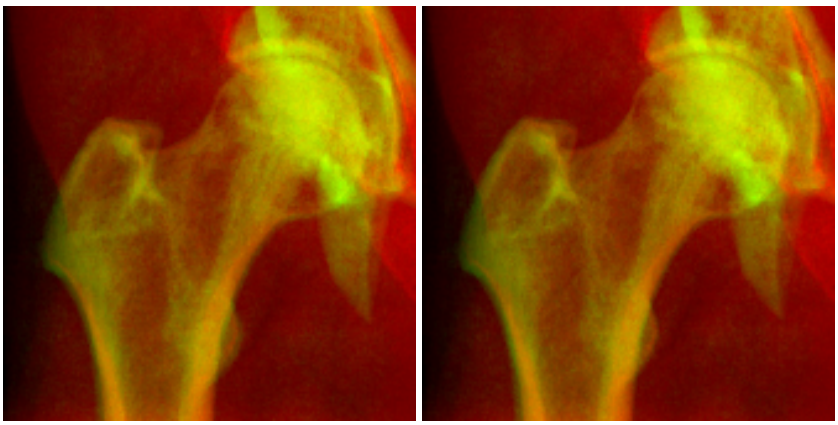

Figure 8: Overlay images of the X-ray with the resulting DRRs of the femur EVOL registration (left) and RSGD registration (right).

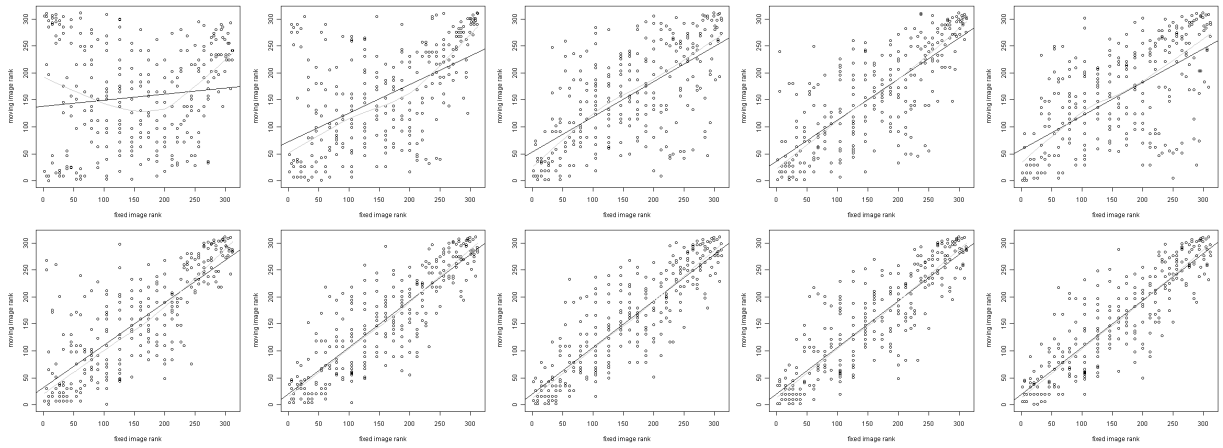

Figure 9: EVOL image rank plots of the fixed and moving image for iterations 1, 10, 20, 30, 40 (top), 50, 100, 130, 160, 200 (bottom).

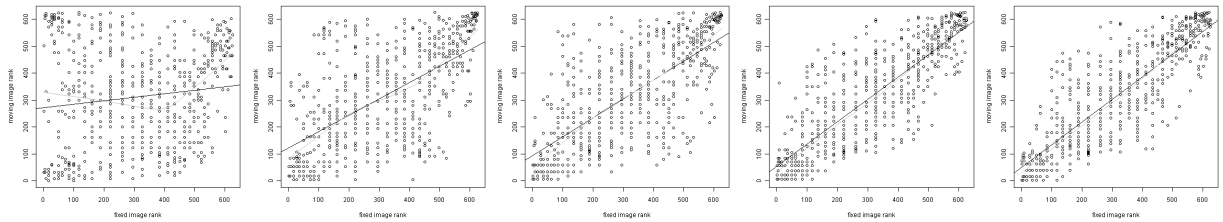

Figure 10: RSGD image rank plots of the fixed and moving image for iterations 1, 5, 10, 20, 30.

Femur example using 'IntensityBased2D3DRegistration2'

This example performs 2D/3D-registration with a modified version of `IntensityBased2D3DRegistration` from `InsightApplications` (see `execute-NativeRegistrationExample-Femur.bat`). First a fixed image (DRR) for registration is generated from the sample CT data set with a threshold of 130. Subsequently, registration is performed with moving images (DRRs) that have a different threshold of 160 (in order to demonstrate the SRC metric's robustness against intensity differences). Furthermore, a centered fixed image mask with a radius of 100 mm is applied. The initial transformation (fixed image) is translated by 8.509 mm in each dimension and rotated by about 9.2 ° around each axis. SRC is configured with 128 bins and 10 % sample coverage. The initial setup is displayed in Fig. 11, the registration result in Fig. 12, and the transformation parameters are summarized in Tab. 2.

Table 2: Initial transformation before, and final transformation after registration of the synthetic femur example based on `IntensityBased2D3DRegistration2`.

Transformation Parameter	r_x [°]	r_y [°]	r_z [°]	t_x [mm]	t_y [mm]	t_z [mm]
real transformation	90.00	0.00	0.00	-60.00	-50.00	-70.00
initial transformation	99.12	9.17	9.17	-68.509	-58.509	-78.509
result	89.76	0.24	-0.11	-60.05	-49.99	-70.12

Figure 11: From left to right: DRR of simulated fixed image, DRR that simulates a moving image with the initial transformation, overlay of both images.

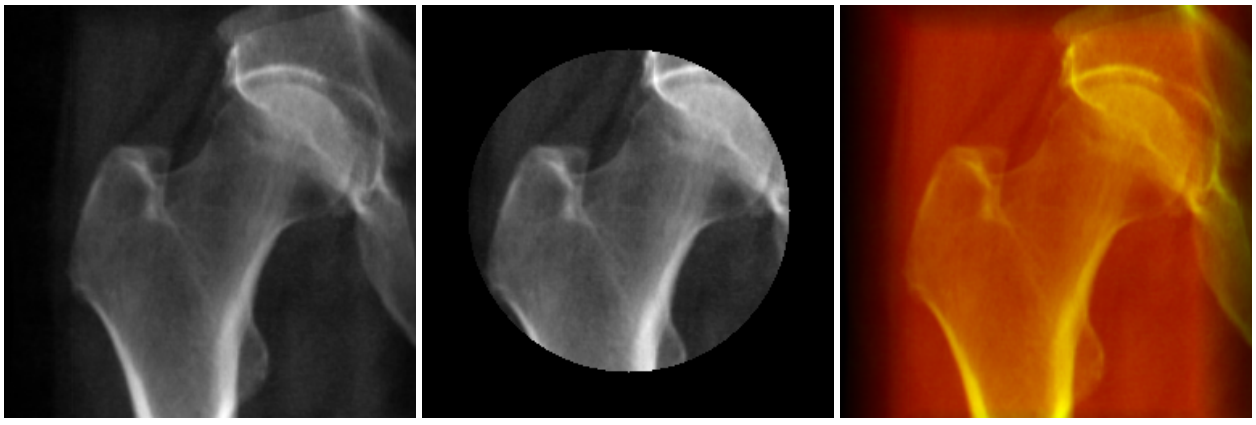

Figure 12: From left to right: Moving image with the resulting transformation of the native registration, moving image masked with the used circular fixed image mask, overlay of the moving image with the final fixed image.

Table 3: Initial transformation before, and final transformation after registration of the pork examples. The real transformation is the gold-standard from [6].

Transformation Parameter	r_x [°]	r_y [°]	r_z [°]	t_x [mm]	t_y [mm]	t_z [mm]
anterior-posterior (AP)						
gold-standard transformation	91.45	1.35	1.21	1.00	-2.89	1.04
initial transformation	98.00	5.00	-5.00	6.00	-7.00	1.04
result	91.53	1.30	1.66	1.78	-4.12	1.04
lateral (LAT)						
gold-standard transformation	229.94	88.19	138.28	1.59	-2.93	2.50
initial transformation	233.00	85.00	141.00	7.00	-8.00	2.50
result	231.58	88.05	139.39	1.40	-3.38	2.50

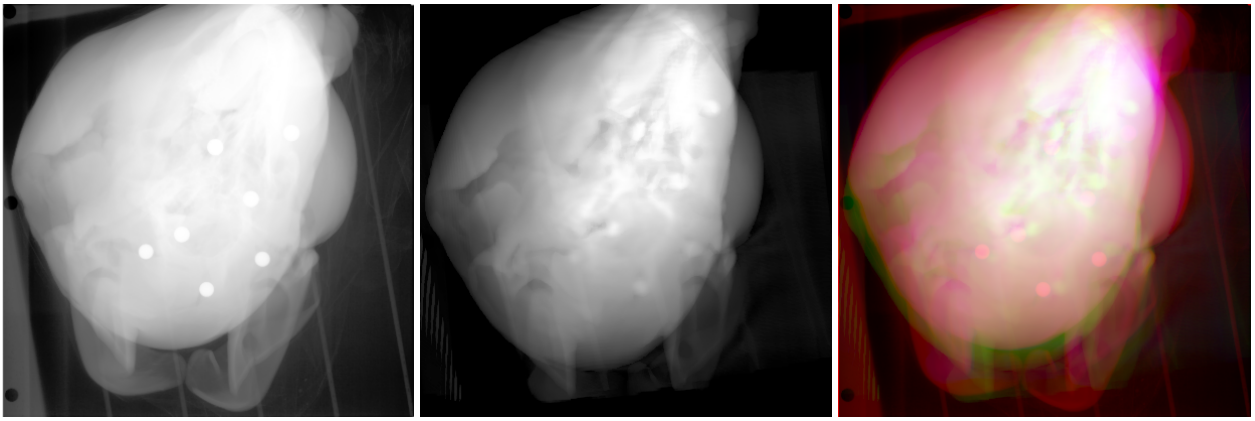

Figure 13: From left to right: AP X-ray (fixed image), DRR (moving image) with the initial transformation, overlay of both images. Note that the images were normalized and equalized for a better contrast.

4.3 Pork Data Set Examples

The following configurations show an exemplary registration of a big field of view 3D cone beam CT (CBCT) image and an anterior-posterior (AP, 0 ° gantry, `RegistrationExample-Pork-EVOL-AP.cfg`), and a lateral (LAT, 90 ° gantry, `RegistrationExample-Pork-EVOL-LAT.cfg`) 2D X-ray image, respectively. The images were taken from [6], resampled, and converted into MetaImage format (`CBCT_BigFOV_1000mu_plus1024_2x2.mhd`, `kV_AP_1x1mm_410x410px.mhd`, `kV_LAT_1x1mm_410x410px.mhd`). The image data are located in the data-directory `sampleconfigs/pork/data` along with a `readme.txt` file that contains information on the image files and the according license. The geometric setup is equal to the one described in [6]. The center of the CBCT data set is shifted to the isocenter, the film-to-focus-distance is 1536 mm, and the film-to-isocenter-distance is 536 mm. The normal-beam is positioned in the middle of the DRR/X-ray. The registration involves rigid transformation (Euler 3D), stochastic rank correlation metric (256 bins, 100% coverage), evolutionary optimization strategy and ray-casting DRR generation. Here, only a single resolution level is used for registration, and the fixed image is circular-masked with a radius of 100 mm (see Fig. 14 and 16). **NOTE**: The extrinsic markers in the data sets are not masked!

The initial setup for AP is visualized in Fig. 13 and the results in Fig. 14, and for LAT in Fig. 15 and 16. The resulting transformations are summarized in Tab. 3. The AP optimization took 200 iterations (max) and the LAT 197 iterations. The image rank plots can be seen in Fig. 17 and 18.

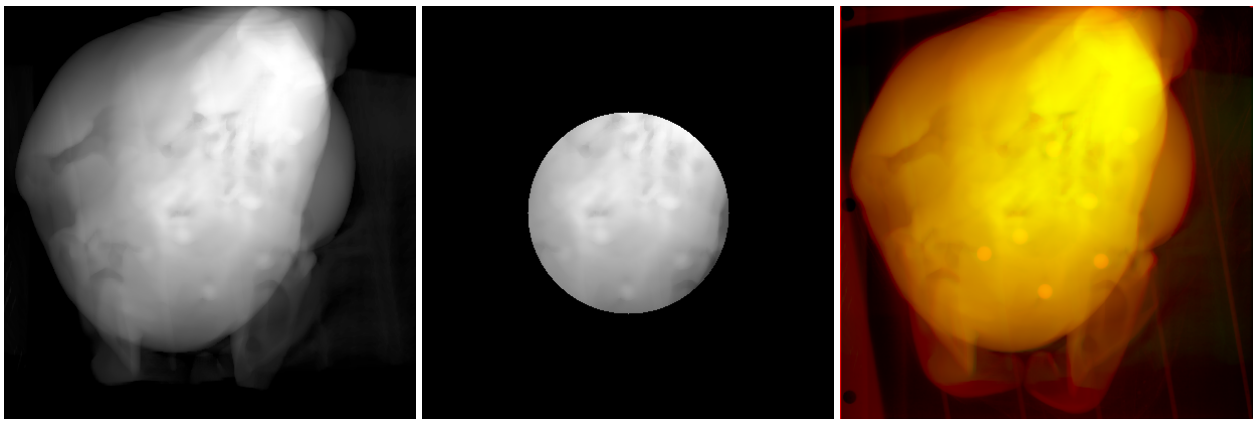

Figure 14: From left to right: AP moving image with the resulting transformation after registration, masked moving image with the applied circular fixed image mask, overlay of the moving image with the final fixed image.

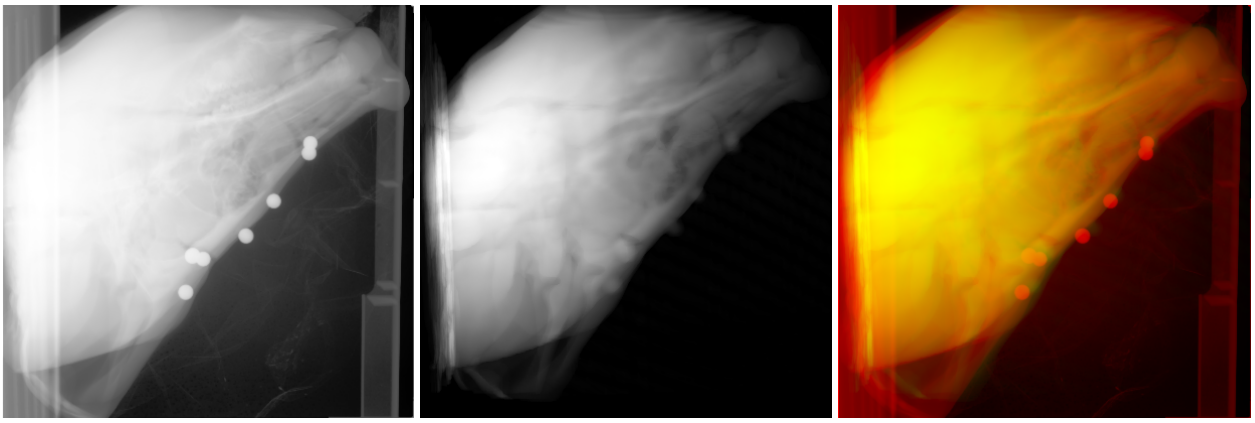

Figure 15: From left to right: LAT X-ray (fixed image), DRR (moving image) with the initial transformation, overlay of both images. Note that the images were normalized and equalized for a better contrast.

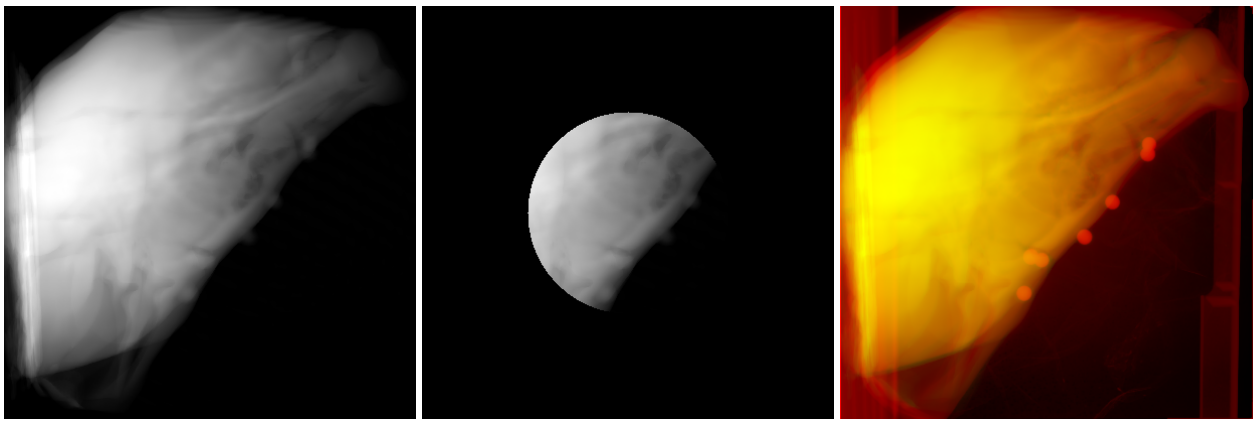

Figure 16: From left to right: LAT moving image with the resulting transformation after registration, masked moving image with the applied circular fixed image mask, overlay of the moving image with the final fixed image.

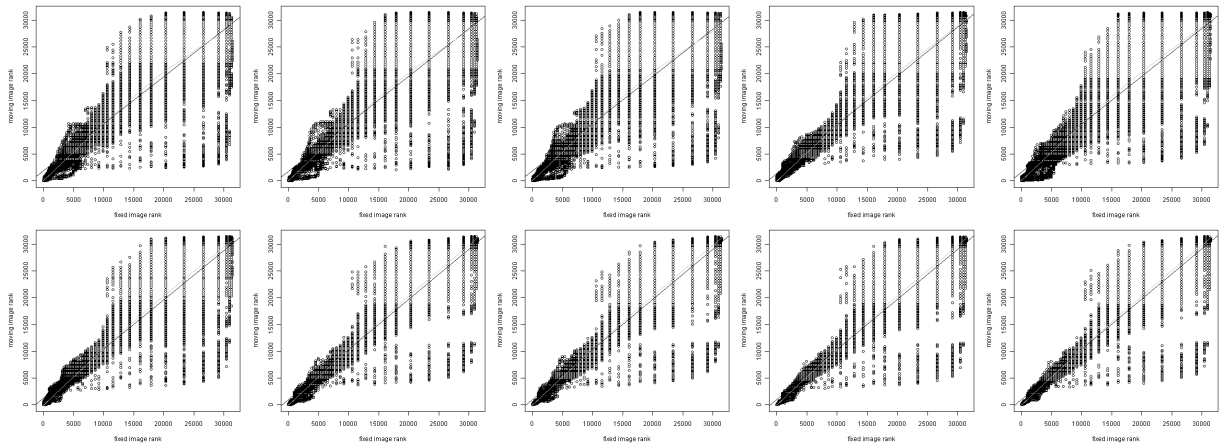

Figure 17: AP image rank plots of the fixed and moving image for iterations 1, 10, 20, 30, 40 (top), 50, 100, 130, 160, 200 (bottom).

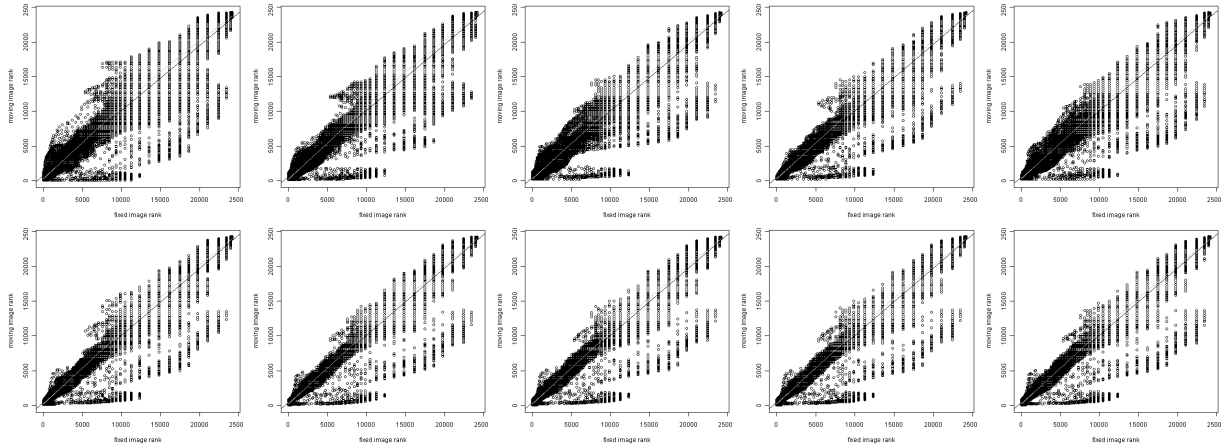

Figure 18: LAT image rank plots of the fixed and moving image for iterations 1, 10, 20, 30, 40 (top), 50, 100, 130, 160, 197 (bottom).

5 Discussion

We presented an implementation of stochastic rank correlation which is well-integrated with the ITK-framework. Furthermore, we provided a set of 2D/3D registration example applications and sample data for testing SRC. The applications showed that the metric was able to successfully assess the similarity of the investigated X-rays and DRRs from a CT of the same subject without specific radiometric calibration or image pre-processing. It is, however, important to note that this paper does not cover a comprehensive SRC validation study. The reader may refer to [1, 2, 3] for comparative SRC studies based on cadaver data sets.

In this paper we concentrated on the usage of SRC with 2D/3D image registration. It would be, however, interesting to investigate the performance of SRC in other image registration scenarios such as deformable constrained B-spline 3D/3D CBCT-to-CT registration.

Finally, we hope that we could encourage researchers to actively investigate SRC, by providing an implementation that is fully compatible with the generic ITK registration framework.

6 Licensing

The software provided was developed by Philipp Steininger and Markus Neuner under contract at the Paracelsus Medical University (PMU), Austria at the Institute for Research and Development on Advanced Radiation Technologies (radART). The software is distributed under the new and simplified BSD license approved by the Open Source Initiative (OSI, www.opensource.org/licenses/bsd-license.php) (see `license.txt`). The software is partially derived from the Insight Segmentation and Registration Toolkit (ITK), and the extended ITK-based 2D/3D registration framework [7] which are both distributed under the new and simplified BSD license. ITK is required for source code compilation.

NOTE: The data set licenses are located in the corresponding directories.

References

[1] Wolfgang Birkfellner, Markus Stock, Michael Figl, Christelle Gendrin, Johann Hummel, Shuo Dong, Joachim Kettenbach, Dietmar Georg, and Helmar Bergmann. Stochastic rank correlation: a robust merit function for 2d/3d registration of image data obtained at different energies. *Med Phys*, 36(8):3420–8, 2009. 1, 2.2, 2.2, 5

[2] M Figl, C Bloch, C Gendrin, C Weber, S A Pawiro, J Hummel, P Markelj, F Pernuš, H Bergmann, and W Birkfellner. Efficient implementation of the rank correlation merit function for 2d/3d registration. *Physics in Medicine and Biology*, 55(19):N465, 2010. 2.2, 5

[3] C. Gendrin, P. Markelj, S. A. Pawiro, J. Spoerk, C. Bloch, C. Weber, M. Figl, H. Bergmann, W. Birkfellner, B. Likar, and F. Pernuš. Validation for 2d/3d registration. part ii : the comparison of intensity and gradient based merit functions using a new gold standard data set. *[submitted to MedPhys]*, 2010. 5

[4] D. Horn. A correction for the effect of tied ranks on the value of the rank difference correlation coefficient. *Journal of Educational Psychology*, 33(9):686 – 690, 1942. 2.3

[5] P. Markelj, D. Tomaževič, B. Likar, and F. Pernuš. A review of 3d/2d registration methods for image-guided interventions. *Medical Image Analysis*, April 2010. 2.1

[6] S. A. Pawiro, P. Markelj, F. Pernuš, C. Gendrin, M. Figl, C. Weber, F. Kainberger, I. Noebauer-Huhmann, H. Bergmeister, M. Stock, D. Georg, H. Bergmann, and W. Birkfellner. Validation for 2d/3d registration. part i: A new gold standard data set. *[submitted to MedPhys]*, 2010. 3, 4.3

[7] Philipp Steininger, Markus Neuner, and Rainer Schubert. An extended itk-based framework for intensity-based 2d/3d-registration. Technical report, radART (PMU) and IBIA (UMIT), http://ibia.umit.at/ResearchGroup/Phil/web/Simple2D3DRegistrationFramework.html, December 2009. 4, 6

[8] Everine B. van de Kraats, Graeme P. Penney, Dejan Tomazevic, Theo van Walsum, and Wiro J. Niessen. Standardized evaluation methodology for 2-d-3-d registration. *IEEE Trans. Med. Imaging*, 24(9):1177–1189, 2005. 2.1