Airlab Documentation

Release 0.1

Robin Sandkuehler Christoph Jud

Oct 08, 2020

Building Blocks:

AirLab is an open laboratory for medical image registration. It provides an environment for rapid prototyping and reproduction of registration algorithms. The unique feature of AirLab is, that the analytic gradients of the objective function are computed automatically with fosters rapid prototyping. In addition, the device on which the computations are performed, on a CPU or a GPU, is transparent. AirLab is implemented in Python using PyTorch as tensor and optimization library and SimpleITK for basic image IO. It profits therefore from recent advances made by the machine learning community.

Authors: Robin Sandkuehler and Christoph Jud

CHAPTER 1

registration

class airlab.registration.**PairwiseRegistration**(*verbose=True*)

start(*EarlyStopping=False*, *StopPatience=10*)

class airlab.registration.**DemonsRegistraion**(*verbose=True*)

set_regulariser(*regulariser*)

start()

CHAPTER 2

transformation

2.1 Pairwise Transformation

class airlab.transformation.pairwise.**AffineTransformation**(*moving_image*, *opt_cm=False*)

Affine centred transformation for 2D and 3D.

Parameters

- **moving_image** (Image) – moving image for the registration
- **opt_cm** (*bool*) – using center of as parameter for the optimisation

forward()

set_parameters(*t*, *phi*, *scale*, *shear*, *rotation_center=None*)

Set parameters manually

t (array): 2 or 3 dimensional array specifying the spatial translation phi (array): 1 or 3 dimensional array specifying the rotation angles scale (array): 2 or 3 dimensional array specifying the scale in each dimension shear (array): 2 or 6 dimensional array specifying the shear in each dimension: yx, xy, zx, zy, xz, yz rotation_center (array): 2 or 3 dimensional array specifying the rotation center (default is zeros)

class airlab.transformation.pairwise.**BsplineTransformation**(*image_size*, *sigma*, *diffeomorphic=False*, *order=2*, *dtype=torch.float32*, *device='cpu'*)

class airlab.transformation.pairwise.**NonParametricTransformation**(*image_size*, *diffeomorphic=False*, *dtype=torch.float32*, *device='cpu'*)

None parametric transformation

forward()

set_start_parameter(*parameters*)

class airlab.transformation.pairwise.**RigidTransformation**(*moving_image*, *opt_cm=False*)

Rigid centred transformation for 2D and 3D.

Parameters

- **moving_image** (Image) – moving image for the registration
- **opt_cm** (*bool*) – using center of as parameter for the optimisation

compute_displacement(*transformation_matrix*)

forward()

init_translation(*fixed_image*)

Initialize the translation parameters with the difference between the center of mass of the fixed and the moving image

Parameters **fixed_image** (Image) – Fixed image for the registration

print()

set_parameters(*t*, *phi*, *rotation_center=None*)

Set parameters manually

t (array): 2 or 3 dimensional array specifying the spatial translation phi (array): 1 or 3 dimensional array specifying the rotation angles rotation_center (array): 2 or 3 dimensional array specifying the rotation center (default is zeros)

transformation_matrix

class airlab.transformation.pairwise.**SimilarityTransformation**(*moving_image*, *opt_cm=False*)

Similarity centred transformation for 2D and 3D. :param moving_image: moving image for the registration :type moving_image: Image :param opt_cm: using center of as parameter for the optimisation :type opt_cm: bool

forward()

set_parameters(*t*, *phi*, *scale*, *rotation_center=None*)

Set parameters manually

t (array): 2 or 3 dimensional array specifying the spatial translation phi (array): 1 or 3 dimensional array specifying the rotation angles scale (array): 2 or 3 dimensional array specifying the scale in each dimension rotation_center (array): 2 or 3 dimensional array specifying the rotation center (default is zeros)

class airlab.transformation.pairwise.**WendlandKernelTransformation**(*image_size*, *sigma*, *cp_scale=2*, *diffeomorphic=False*, *ktype='C4'*, *dtype=torch.float32*, *device='cpu'*)

Wendland Kernel Transform:

Implements the kernel transform with the Wendland basis

Parameters

- **sigma** – specifies how many control points are used (each sigma pixels)
- **cp_scale** – specifies the extent of the kernel. how many control points are in the support of the kernel

2.2 Transformation Utils

class airlab.transformation.utils.**Diffeomorphic**(*image_size=None*, *scaling=10*, *dtype=torch.float32*, *device='cpu'*)

Diffeomorphic transformation. This class computes the matrix exponential of a given flow field using the scaling and squaring algorithm according to:

> Unsupervised Learning for Fast Probabilistic Diffeomorphic Registration Adrian V. Dalca, Guha Balakrishnan, John Guttag, Mert R. Sabuncu MICCAI 2018 and Diffeomorphic Demons: Efficient Non-parametric Image Registration Tom Vercauterena et al., 2008

calculate(*displacement*)

static diffeomorphic_2D(*displacement*, *grid*, *scaling=-1*)

static diffeomorphic_3D(*displacement*, *grid*, *scaling=-1*)

set_image_size(*image_szie*)

airlab.transformation.utils.**compute_grid**(*image_size*, *dtype=torch.float32*, *device='cpu'*)

airlab.transformation.utils.**displacement_to_unit_displacement**(*displacement*)

airlab.transformation.utils.**get_displacement_itk**(*displacement*, *refIm*)

airlab.transformation.utils.**rotation_matrix**(*phi_x*, *phi_y*, *phi_z*, *dtype=torch.float32*, *device='cpu'*, *homogene=False*)

airlab.transformation.utils.**unit_displacement_to_displacement**(*displacement*)

airlab.transformation.utils.**upsample_displacement**(*displacement*, *new_size*, *interpolation='linear'*)

Upsample displacement field

airlab.transformation.utils.**warp_image**(*image*, *displacement*)

CHAPTER 3

loss

3.1 Pairwise Loss

class airlab.loss.pairwise.**LCC**(*fixed_image, moving_image, fixed_mask=None, moving_mask=None, sigma=[3], kernel_type='box', size_average=True, reduce=True*)

forward(*displacement*)

class airlab.loss.pairwise.**MI**(*fixed_image*, *moving_image*, *fixed_mask=None*, *moving_mask=None*, *bins=64*, *sigma=3*, *spatial_samples=0.1*, *background=None*, *size_average=True*, *reduce=True*)

Implementation of the Mutual Information image loss.

$$\mathcal{S}_{\text{MI}} := H(F, M) - H(F|M) - H(M|F)$$

Parameters

- **fixed_image** (Image) – Fixed image for the registration
- **moving_image** (Image) – Moving image for the registration
- **bins** (*int*) – Number of bins for the intensity distribution
- **sigma** (*float*) – Kernel sigma for the intensity distribution approximation
- **spatial_samples** (*float*) – Percentage of pixels used for the intensity distribution approximation
- **background** – Method to handle background pixels. None: Set background to the min value of image "mean": Set the background to the mean value of the image float: Set the background value to the input value
- **size_average** (*bool*) – Average loss function
- **reduce** (*bool*) – Reduce loss function to a single value

bins

bins_fixed_image

forward(*displacement*)

sigma

class airlab.loss.pairwise.**MSE**(*fixed_image*, *moving_image*, *fixed_mask=None*, *moving_mask=None*, *size_average=True*, *reduce=True*)

The mean square error loss is a simple and fast to compute point-wise measure which is well suited for monomodal image registration.

$$\mathcal{S}_{\text{MSE}} := \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \Big(I_M\big(x + f(x)\big) - I_F\big(x\big) \Big)^2$$

Parameters

- **fixed_image** (Image) – Fixed image for the registration
- **moving_image** (Image) – Moving image for the registration
- **size_average** (*bool*) – Average loss function
- **reduce** (*bool*) – Reduce loss function to a single value

forward(*displacement*)

class airlab.loss.pairwise.**NCC**(*fixed_image*, *moving_image*, *fixed_mask=None*, *moving_mask=None*)

The normalized cross correlation loss is a measure for image pairs with a linear intensity relation.

$$\mathcal{S}_{\text{NCC}} := \frac{\sum I_F \cdot (I_M \circ f) - \sum \text{E}(I_F)\text{E}(I_M \circ f)}{|\mathcal{X}| \cdot \sum \text{Var}(I_F)\text{Var}(I_M \circ f)}$$

Parameters

- **fixed_image** (Image) – Fixed image for the registration
- **moving_image** (Image) – Moving image for the registration

forward(*displacement*)

class airlab.loss.pairwise.**NGF**(*fixed_image*, *moving_image*, *fixed_mask=None*, *moving_mask=None*, *epsilon=1e-05*, *size_average=True*, *reduce=True*)

Implementation of the Normalized Gradient Fields image loss.

Parameters

- **fixed_image** (Image) – Fixed image for the registration
- **moving_image** (Image) – Moving image for the registration
- **fixed_mask** (*Tensor*) – Mask for the fixed image
- **moving_mask** (*Tensor*) – Mask for the moving image
- **epsilon** (*float*) – Regulariser for the gradient amplitude
- **size_average** (*bool*) – Average loss function
- **reduce** (*bool*) – Reduce loss function to a single value

forward(*displacement*)

class airlab.loss.pairwise.**SSIM**(*fixed_image, moving_image, fixed_mask=None, moving_mask=None, sigma=[3], dim=2, kernel_type='box', alpha=1, beta=1, gamma=1, c1=1e-05, c2=1e-05, c3=1e-05, size_average=True, reduce=True*)

Implementation of the Structual Similarity Image Measure loss.

Parameters

- **fixed_image** (*Image*) – Fixed image for the registration
- **moving_image** (*Image*) – Moving image for the registration
- **fixed_mask** (*Tensor*) – Mask for the fixed image
- **moving_mask** (*Tensor*) – Mask for the moving image
- **sigma** (*float*) – Sigma for the kernel
- **kernel_type** (*string*) – Type of kernel i.e. gaussian, box
- **alpha** (*float*) – Controls the influence of the luminance value
- **beta** (*float*) – Controls the influence of the contrast value
- **gamma** (*float*) – Controls the influence of the structure value
- **c1** (*float*) – Numerical constant for the luminance value
- **c2** (*float*) – Numerical constant for the contrast value
- **c3** (*float*) – Numerical constant for the structure value
- **size_average** (*bool*) – Average loss function
- **reduce** (*bool*) – Reduce loss function to a single value

forward(*displacement*)

CHAPTER 4

regulariser

4.1 Demons Regulariser

class airlab.regulariser.demons.**GaussianRegulariser**(*pixel_spacing*, *sigma*, *dtype=torch.float32*, *device='cpu'*)

regularise(*data*)

4.1.1 Graph Diffusion Regulariser

class airlab.regulariser.demons.**GraphDiffusionRegulariser**(*image_size*, *pixel_spacing*, *edge_updater*, *phi=1*, *dtype=torch.float32*, *device='cpu'*)

get_edge_image()

regularise(*data*)

set_krylov_dim(*krylov_dim*)

class airlab.regulariser.demons.**EdgeUpdaterDisplacementIntensities**(*pixel_spacing*, *image*, *edge_window=0.9*, *edge_mean=False*)

update(*data*)

class airlab.regulariser.demons.**EdgeUpdaterIntensities**(*pixel_spacing*, *image*, *scale=1*, *edge_window=0.9*, *edge_mean=False*)

set_scale(*sale*)

update(*data*)

4.2 Displacement Regulariser

class airlab.regulariser.displacement.**DiffusionRegulariser**(*pixel_spacing*, *size_average=True*, *reduce=True*)

forward(*displacement*)

class airlab.regulariser.displacement.**IsotropicTVRegulariser**(*pixel_spacing*, *size_average=True*, *reduce=True*)

forward(*displacement*)

class airlab.regulariser.displacement.**SparsityRegulariser**(*size_average=True*, *reduce=True*)

forward(*displacement*)

class airlab.regulariser.displacement.**TVRegulariser**(*pixel_spacing*, *size_average=True*, *reduce=True*)

forward(*displacement*)

4.3 Parameter Regulariser

class airlab.regulariser.parameter.**DiffusionRegulariser**(*pixel_spacing*, *size_average=True*, *reduce=True*)

forward(*displacement*)

class airlab.regulariser.parameter.**IsotropicTVRegulariser**(*parameter_name*, *scaling=[1]*, *size_average=True*, *reduce=True*)

forward(*parameters*)

class airlab.regulariser.parameter.**SparsityRegulariser**(*parameter_name*, *size_average=True*, *reduce=True*)

forward(*parameters*)

class airlab.regulariser.parameter.**TVRegulariser**(*parameter_name*, *scaling=[1]*, *size_average=True*, *reduce=True*)

forward(*parameters*)

CHAPTER 5

utils

5.1 Image

class airlab.utils.image.**Displacement**(**args*, ***kwargs*)

itk()
Returns a SimpleITK image

Note: the order of axis is flipped back to the convention of SimpleITK

magnitude()

numpy()
Returns a numpy array

static read(*filename*, *dtype=torch.float32*, *device='cpu'*)
Static method to directly read a displacement field through the Image class

filename (str): filename of the displacement field dtype: specific dtype for representing the tensor device: on which device the displacement field has to be allocated return (Displacement): an airlab displacement field

class airlab.utils.image.**Image**(**args*, ***kwargs*)
Class representing an image in airlab

initializeForImages(*sitk_image*, *dtype=None*, *device='cpu'*)
Constructor for SimpleITK image

Note: the order of axis are flipped in order to follow the convention of numpy and torch

sitk_image (sitk.SimpleITK.Image): SimpleITK image dtype: pixel type device ('cpu'|'cuda'): on which device the image should be allocated return (Image): an airlab image object

initializeForTensors(*tensor_image*, *image_size*, *image_spacing*, *image_origin*)
Constructor for torch tensors and numpy ndarrays

Args: tensor_image (np.ndarray | th.Tensor): n-dimensional tensor, where the last dimensions are the image dimensions while the preceeding dimensions need to empty image_size (array | list | tuple): number of pixels in each space dimension image_spacing (array | list | tuple): pixel size for each space dimension image_origin (array | list | tuple): physical coordinate of the first pixel :return (Image): an airlab image object

itk()
Returns a SimpleITK image

Note: the order of axis is flipped back to the convention of SimpleITK

numpy()
Returns a numpy array

static read(*filename*, *dtype=torch.float32*, *device='cpu'*)
Static method to directly read an image through the Image class

filename (str): filename of the image dtype: specific dtype for representing the tensor device: on which device the image has to be allocated return (Image): an airlab image

to(*dtype=None*, *device='cpu'*)
Converts the image tensor to a specified dtype and moves it to the specified device

write(*filename*)
Write an image to hard drive

Note: order of axis are flipped to have the representation of SimpleITK again

filename (str): filename where the image is written

airlab.utils.image.**create_displacement_image_from_image**(*tensor_displacement*, *image*)

airlab.utils.image.**create_image_from_image**(*tensor_image*, *image*)

airlab.utils.image.**create_image_pyramid**(*image*, *down_sample_factor*)

airlab.utils.image.**create_tensor_image_from_itk_image**(*itk_image*, *dtype=torch.float32*, *device='cpu'*)

airlab.utils.image.**flip**(*x*, *dim*)
Flip order of a specific dimension dim

x (Tensor): input tensor dim (int): axis which should be flipped return (Tensor): returns the tensor with the specified axis flipped

airlab.utils.image.**image_from_numpy**(*image*, *pixel_spacing*, *image_origin*, *dtype=torch.float32*, *device='cpu'*)

airlab.utils.image.**read_image_as_tensor**(*filename*, *dtype=torch.float32*, *device='cpu'*)

5.2 Kernel Function

airlab.utils.kernelFunction.**bspline_kernel**(*sigma*, *order=2*, *dim=1*, *asTensor=False*, *dtype=torch.float32*, *device='cpu'*)

airlab.utils.kernelFunction.**bspline_kernel_1d**(*sigma*, *order=2*, *asTensor=False*, *dtype=torch.float32*, *device='cpu'*)

airlab.utils.kernelFunction.**bspline_kernel_2d**(*sigma=[1, 1]*, *order=2*, *asTensor=False*, *dtype=torch.float32*, *device='cpu'*)

airlab.utils.kernelFunction.**bspline_kernel_3d**(*sigma=[1, 1, 1], order=2, asTensor=False, dtype=torch.float32, device='cpu'*)

airlab.utils.kernelFunction.**gaussian_kernel**(*sigma, dim=1, asTensor=False, dtype=torch.float32, device='cpu'*)

airlab.utils.kernelFunction.**gaussian_kernel_1d**(*sigma, asTensor=False, dtype=torch.float32, device='cpu'*)

airlab.utils.kernelFunction.**gaussian_kernel_2d**(*sigma, asTensor=False, dtype=torch.float32, device='cpu'*)

airlab.utils.kernelFunction.**gaussian_kernel_3d**(*sigma, asTensor=False, dtype=torch.float32, device='cpu'*)

airlab.utils.kernelFunction.**wendland_kernel**(*sigma, dim=1, type='C4', asTensor=False, dtype=torch.float32, device='cpu'*)

airlab.utils.kernelFunction.**wendland_kernel_1d**(*sigma, type='C4', asTensor=False, dtype=torch.float32, device='cpu'*)

airlab.utils.kernelFunction.**wendland_kernel_2d**(*sigma, type='C4', asTensor=False, dtype=torch.float32, device='cpu'*)

airlab.utils.kernelFunction.**wendland_kernel_3d**(*sigma, type='C4', asTensor=False, dtype=torch.float32, device='cpu'*)

5.3 Graph

class airlab.utils.graph.**Graph**(*graph_size, dtype=torch.float32, device='cpu'*)

5.4 Matrix

class airlab.utils.matrix.**LaplaceMatrix**(*number_of_nodes, diag_elements, dtype=torch.float32, device='cpu'*)

full()

update()

class airlab.utils.matrix.**MatrixDiagonalElement**(*edge_index, edge_values, offset, dtype=torch.float32, device='cpu'*)

airlab.utils.matrix.**band_mv**(*A, x*)

airlab.utils.matrix.**expm_eig**(*A*)

airlab.utils.matrix.**expm_krylov**(*A, x, phi=1, krylov_dim=30, inplace=True*)

CHAPTER 6

Indices and tables

- genindex
- modindex

Python Module Index

a

Index

G

I

L

M

N

P

R

S

T

U

W