IPPAI

version 0.1.0

CAMI (Computer Assisted Medical Interventions), DKFZ, Heidelberg

April 03, 2020

Contents

Welcome to IPPAI's documentation!

README

The Image Processing for Photoacoustic Imaging (IPPAI) toolkit.

Internal Install Instructions

These install instructions are made under the assumption that you have access to the phabricator ippai project. When you are reading these instructions there is a 99% chance that is the case (or someone send these instructions to you).

So, for the 1% of you: Please also follow steps 1 - 3:

1. `git clone https://phabricator.mitk.org/source/ippai.git`
2. `git checkout master`
3. `git pull`

Now open a python instance in the 'ippai' folder that you have just downloaded. Make sure that you have your preferred virtual environment activated

1. `cd ippai`
2. `python -m setup.py build install`
3. Test if the installation worked by using `python` followed by `import ippai` then `exit()`

If no error messages arise, you are now setup to use ippai in your project.

Building the documentation

When the installation went fine and you want to make sure that you have the latest documentation you should do the following steps in a command line:

1. Navigate to the `ippai` source directory (same level where the setup.py is in)
2. Execute the command `sphinx-build -b pdf -a documentation/src documentation`
3. Find the PDF file in `documentation/ippai_documantation.pdf`

Overview

The main use case for the ippai framework is the simulation of photoacoustic images. However, it can also be used for image processing.

Simulating photoacoustic images

A basic example on how to use ippai in you project to run an optical forward simulation is given in the samples/minimal_optical_simulation.py file.

Developer Guide

Dear IPPAI developers. This Guide is meant to be used as a collection of How-To's to contribute to the framework.

Coding style

When writing code for IPPAI, please consider to use the following structures in your code in order to make a new developer or someone external always know exactly what to expect.

- Classnames are written in camel-case notation `ClassName`
- Function names are written in small letter with _ as the delimiter `function_name`
- Function parameters are always annotated with their type `arg1: type = default`
- Only use primitive types as defaults. If a non-primitive type is used, then the default should be `None` and the parameter should be initialized in the beginning of a function.
- A single line of code should not be longer than 120 characters.
- Functions should follow the following simple structure:
    1. Input validation (arguments all not `None`, correct type, and acceptable value ranges?)
    2. Processing (clean handling of errors that might occur)
    3. Output generation (sanity checking of the output before handing it off to the caller)

Documenting your code

Only documented code will appear in the sphinx generated documentation.

A class should be documented using the following syntax:

```
class ClassName(Superclass):
    """
    Explain how the class is used and what it does.
    """
```

For functions, a lot of extra attributes can be added to the documentation:

```
def function_name(self, arg1:type = default, arg2:type = default) -> return_type:
    """
    Explain how the function is used and what it does.

    :param arg1: type, value range, Null acceptable?
    :param arg2: type, value range, Null acceptable?
    :returns: type, value range, does it return Null?
    :raises ExceptionType: explain when and why this exception is raised
    """
```

Adding literature absorption spectra

The central point, where absorption spectra are collected and handled is in `ippai.utils.libraries.spectra_library.py`. The file comprises the class `AbsorptionSpectrumLibrary`, in which the new absorption spectra can be added using the following two steps:

1. In the beginning of the class, there is a bunch of constants that define spectra using the `AbsorptionSpectrum` class. Add a new constant here: `NEW_SPECTRUM = AbsorptionSpectrum(absorber_name, wavelengths, absorptions)`. By convention, the naming of the constant should be the same as the `absorber_name` field. The `wavelengths` and `absorptions` arrays must be of the same length and contain corresponding values.
2. In the `__init__` method of the `AbsorptionSpectrumLibrary` class, the class constants are added to an internal list. This has the benefit of enabling the Library class to be iterable. Add your newly added constant field to the list here.
3. Your absorption spectrum is now usable throughout all of ippai and is accessible using the `SPECTRAL_LIBRARY` sngleton that can be imported using `from ippai.utils import SPECTRAL_LIBRARY`.

Examples

Performing a simple opical forward simulation

The file can be found in samples/minimal_optical_simulation.py:

```
from ippai.utils import Tags

from ippai.simulate.simulation import simulate
from ippai.simulate.structures import create_epidermis_layer
from ippai.simulate.structures import create_muscle_background
from ippai.simulate.structures import create_vessel_tube

import numpy as np

# TODO change these paths to the desired executable and save folder
SAVE_PATH = "path/to/save/file"
MCX_BINARY_PATH = "path/to/mcx/binary"

VOLUME_WIDTH_IN_MM = 10
VOLUME_HEIGHT_IN_MM = 10
SPACING = 0.25
RANDOM_SEED = 4711


def create_example_tissue():
    """
    This is a very simple example script of how to create a tissue definition.
    It contains a muscular background, an epidermis layer on top of the muscles
    and a blood vessel.
    """
    tissue_dict = dict()
    tissue_dict["background"] = create_muscle_background()
    tissue_dict["epidermis"] = create_epidermis_layer()
    tissue_dict["vessel"] = create_vessel_tube(x_min=0, x_max=VOLUME_WIDTH_IN_MM,
                                               z_min=0, z_max=VOLUME_HEIGHT_IN_MM,
                                               r_min=1, r_max=3)
    return tissue_dict

# Seed the numpy random configuration prior to creating the settings file in
# order to ensure that the same volume
# is generated with the same random seed every time.

np.random.seed(RANDOM_SEED)

settings = {
    # These parameters set the general propeties of the simulated volume
    Tags.RANDOM_SEED: RANDOM_SEED,
    Tags.VOLUME_NAME: "MyVolumeName_"+str(RANDOM_SEED),
    Tags.SIMULATION_PATH: SAVE_PATH,
    Tags.SPACING_MM: SPACING,
    Tags.DIM_VOLUME_Z_MM: VOLUME_HEIGHT_IN_MM,
    Tags.DIM_VOLUME_X_MM: VOLUME_WIDTH_IN_MM,
    Tags.DIM_VOLUME_Y_MM: VOLUME_WIDTH_IN_MM,
    Tags.AIR_LAYER_HEIGHT_MM: 0,
    Tags.GELPAD_LAYER_HEIGHT_MM: 0,

    # The following parameters set the optical forward model
    Tags.RUN_OPTICAL_MODEL: True,
    Tags.WAVELENGTHS: np.arange(700, 951, 10),
```

```
    Tags.OPTICAL_MODEL_NUMBER_PHOTONS: 1e7,
    Tags.OPTICAL_MODEL_BINARY_PATH: MCX_BINARY_PATH,
    Tags.OPTICAL_MODEL: Tags.MODEL_MCX,
    Tags.ILLUMINATION_TYPE: Tags.ILLUMINATION_TYPE_PENCIL,
    Tags.LASER_PULSE_ENERGY_IN_MILLIJOULE: 50,

    # The following parameters tell the script that we do not want any extra
    # modelling steps
    Tags.RUN_ACOUSTIC_MODEL: False,
    Tags.APPLY_NOISE_MODEL: False,
    Tags.PERFORM_IMAGE_RECONSTRUCTION: False,
    Tags.SIMULATION_EXTRACT_FIELD_OF_VIEW: False,

    # Add the structures to be simulated to the tissue
    Tags.STRUCTURES: create_example_tissue()
}
print("Simulating ", RANDOM_SEED)
simulate(settings)
# TODO settings[Tags.IPPAI_OUTPUT_PATH]
print("Simulating ", RANDOM_SEED, "[Done]")
```

Reading the HDF5 simulation output

The file can be found in samples/access_saved_PAI_data.py:

```
from ippai.io_handling import load_hdf5, save_hdf5
import matplotlib.pylab as plt
import numpy as np
from ippai.simulate import SaveFilePaths

PATH = "/home/janek/test/Vessels_10005/ippai_output.hdf5"
WAVELENGTH = 800 # currently only 800 and 900 are simulated as well

file = load_hdf5(PATH)

print(file['simulations'].keys())

fluence = (file['simulations']['original_data']['optical_forward_model_output']
           [str(WAVELENGTH)]['fluence'])
initial_pressure = (file['simulations']['original_data']
                    ['optical_forward_model_output']
                    [str(WAVELENGTH)]['initial_pressure'])
absorption = (file['simulations']['original_data']['simulation_properties']
              [str(WAVELENGTH)]['mua'])

shape = np.shape(fluence)

if len(shape) > 2:
    plt.figure()
    plt.subplot(231)
    plt.imshow(np.log10(fluence[int(shape[0]/2), :, :]))
    plt.subplot(232)
    plt.imshow(np.log10(absorption[int(shape[0]/2), :, :]))
    plt.subplot(233)
    plt.imshow(np.log10(initial_pressure))
    plt.subplot(234)
    plt.imshow(np.log10(fluence[:, int(shape[1]/2), :]))
    plt.subplot(235)
    plt.imshow(np.log10(absorption[:, int(shape[1]/2), :]))
    plt.subplot(236)
```

```
    plt.imshow(np.log10(initial_pressure))
    plt.show()
else:
    plt.figure()
    plt.subplot(131)
    plt.imshow(np.log10(fluence[1:129, -65:-1]))
    plt.subplot(132)
    plt.imshow(np.log10(absorption[1:129, -65:-1]))
    plt.subplot(133)
    plt.imshow(np.log10(initial_pressure[1:129, -65:-1]))
    plt.show()

save_hdf5(file, _PATH)
```

Defining custom tissue structures and properties

The file can be found in samples/create_custom_tissues.py:

```
from ippai.utils import TissueSettingsGenerator
from ippai.utils import CHROMOPHORE_LIBRARY
from ippai.utils import Chromophore
from ippai.utils import AbsorptionSpectrum
import numpy as np


def create_custom_absorber():
    wavelengths = np.linspace(200, 1500, 100)
    absorber = AbsorptionSpectrum(spectrum_name="random absorber",
                                  wavelengths=wavelengths,
                                  absorption_per_centimeter=np.random.random(
                                      np.shape(wavelengths)))
    return absorber


def create_custom_chromophore(volume_fraction: float = 1.0):
    chromophore = Chromophore(
            spectrum=create_custom_absorber(),
            volume_fraction=volume_fraction,
            musp500=40.0,
            b_mie=1.1,
            f_ray=0.9,
            anisotropy=0.9
        )
    return chromophore


def create_custom_tissue_type():

    # First create an instance of a TissueSettingsGenerator
    tissue_settings_generator = TissueSettingsGenerator()

    water_volume_fraction = 0.4
    bvf = 0.5
    oxy = 0.4

    # Then append chromophores that you want
    tissue_settings_generator.append(key="oxyhemoglobin", value=
                            CHROMOPHORE_LIBRARY.oxyhemoglobin(oxy*bvf))
    tissue_settings_generator.append(key="deoxyhemoglobin", value=
                            CHROMOPHORE_LIBRARY.deoxyhemoglobin(oxy * bvf))
```

```
    tissue_settings_generator.append(key="water", value=
                            CHROMOPHORE_LIBRARY.water(water_volume_fraction))
    tissue_settings_generator.append(key="custom", value=
                            create_custom_chromophore(0.1))

    return tissue_settings_generator.get_settings()
```

Class references

Module: utils

class ippai.utils.libraries.literature_values.**MorphologicalTissueProperties**
This class contains a listing of morphological tissue parameters as reported in literature. The listing is not the result of a meta analysis, but rather uses the best fitting paper at the time pf implementation. Each of the fields is annotated with a literature reference or a descriptions of how the particular values were derived for tissue modelling.

class ippai.utils.libraries.literature_values.**OpticalTissueProperties**
This class contains a listing of optical tissue parameters as reported in literature. The listing is not the result of a meta analysis, but rather uses the best fitting paper at the time pf implementation. Each of the fields is annotated with a literature reference or a descriptions of how the particular values were derived for tissue modelling.

class ippai.utils.libraries.literature_values.**StandardProperties**
This class contains a listing of default parameters that can be used. These values are sensible default values but are generally not backed up by proper scientific references, or are rather specific for internal use cases.

class ippai.utils.libraries.spectra_library.**AbsorptionSpectrum** (spectrum_name: str, wavelengths: numpy.ndarray, absorption_per_centimeter: numpy.ndarray)
An instance of this class represents the absorption spectrum over wavelength for a particular

get_absorption_for_wavelength (wavelength: float) → float

Parameters: **wavelength** – the wavelength to retrieve a optical absorption value for [cm^{-1}].

Returns: the best matching linearly interpolated absorption value for the given wavelength.

get_absorption_over_wavelength ()

Returns: numpy array with the available wavelengths and the corresponding absorption properties

ippai.utils.libraries.spectra_library.**view_absorption_spectra** (save_path=None)
Opens a matplotlib plot and visualizes the available absorption spectra.

Parameters: **save_path** – If not None, then the figure will be saved as a png file to the destination.

class ippai.utils.libraries.tissue_library.**TissueLibrary**
TODO

blood_arterial ()

Returns: a settings dictionary containing all min and max parameters fitting for full blood.

blood_generic ()

Returns: a settings dictionary containing all min and max parameters fitting for full blood.

blood_venous ()

Returns: a settings dictionary containing all min and max parameters fitting for full blood.

bone ()

Returns: a settings dictionary containing all min and max parameters fitting for full blood.

constant (mua**,** mus**,** g**)**
TODO

dermis (background_oxy=0.5**)**

Returns: a settings dictionary containing all min and max parameters fitting for dermis tissue.

epidermis ()

Returns: a settings dictionary containing all min and max parameters fitting for epidermis tissue.

get_blood_volume_fractions (total_blood_volume_fraction**,** oxygenation**)**
TODO

muscle (background_oxy=0.5**)**

Returns: a settings dictionary containing all min and max parameters fitting for generic background tissue.

subcutaneous_fat (background_oxy=0.5**)**

Returns: a settings dictionary containing all min and max parameters fitting for subcutaneous fat tissue.

class ippai.utils.libraries.tissue_library.**TissueSettingsGenerator**
TODO

Module: io_handling

ippai.io_handling.io_hdf5.**load_hdf5** (file_path, file_dictionary_path='/')
Loads a dictionary from an hdf5 file.

Parameters:
- **file_path** – Path of the file to load the dictionary from.
- **file_dictionary_path** – Path in dictionary structure of hdf5 file to lo the dictionary in.

Returns: Dictionary

ippai.io_handling.io_hdf5.**save_hdf5** (dictionary: dict, file_path: str, file_dictionary_path: str = '/')
Saves a dictionary with arbitrary content to an hdf5-file with given filepath.

Parameters:
- **dictionary** – Dictionary to save.
- **file_path** – Path of the file to save the dictionary in.
- **file_dictionary_path** – Path in dictionary structure of existing hdf5 file to store the dictionary in.

Returns: Null

Module: simulate

ippai.simulate.simulation.**simulate** (settings)

Parameters: **settings** –

Returns:

class ippai.simulate.tissue_properties.**TissueProperties** (settings: dict)

ensure_valid_settings (`settings: dict`)

Parameters: **settings** – a dictionary containing at least one key - value pair of a string and a corresponding Chromophore instance.

get (`wavelength`)
TODO

`ippai.simulate.volume_creator.`**create_acoustic_properties** (`volumes`, `settings`)
Creates maps of density, speed of sound and acoustic attenuation based on the segmented mask in volumes.

Parameters:
- **volumes** – The volumes to append the acoustic parameters to
- **settings** – The settings to extract if the medium is homogeneous of if heterogeneous parameters should be set.

Returns: The volumes with appended maps of acoustic parameters of size volumes[0].size

`ippai.simulate.volume_creator.`**create_gruneisen_map** (`volumes`, `settings`)
Creates a map the gruneisenparameter based on the temperature given in Tags.MEDIUM_TEMPERATURE_CELCIUS. If no medium temperature is specified, then a standard body temperature of 36°C is assumed.

Parameters:
- **volumes** – The volumes to append the gruneisen parameter to
- **settings** – The settings to extract the temperature from

Returns: the volumes with an appended map of gruneisen parameters of size volumes[0].size

`ippai.simulate.volume_creator.`**create_simulation_volume** (`settings`)
This method creates a in silico respresentation of a tissue as described in the settings file that is given. :param settings: a dictionary containing all relevant Tags for the simulation to be able to instantiate a tissue. :return: a path to a npz file containing characteristics of the simulated volume:

absorption, scattering, anisotropy, oxygenation, and a segmentation mask. All of these are given as 3d numpy arrays.

`ippai.simulate.volume_creator.`**fnc_straight_ellipse** (`x`, `y`, `z`, `r_x`, `r_z`, `X1`, `X2`)
cartesian representation of a straight tube that goes from position X1 to position X2 with radius r. :param x: :param y: :param z: :param r: :param X1: :param X2: :return:

`ippai.simulate.volume_creator.`**fnc_straight_tube** (`x`, `y`, `z`, `r`, `X1`, `X2`)
cartesian representation of a straight tube that goes from position X1 to position X2 with radius r. :param x: :param y: :param z: :param r: :param X1: :param X2: :return:

`ippai.simulate.volume_creator.`**merge_voxel** (`volumes`, `x_idx`, `y_idx`, `z_idx`, `mua`, `mus`, `g`, `oxy`, `seg`, `fraction`)
Updates a voxel position in the volumes by merging the given physical properties with the properties already stored in the volumes. The merging is done in a relative manner using the given fraction.

Parameters:
- **volumes** – list of numpy arrays with len(volumes) >= 3
- **x_idx** – integer
- **y_idx** – integer
- **z_idx** – integer
- **mua** – scalar, the optical absorption coefficient in 1/cm
- **mus** – scalar, the optical scattering coefficient in 1/cm
- **g** – scalar, the anisotropy
- **oxy** – scalar, the blood oxygenation in [0, 1]
- **seg** – integer, the tissue segmentation type from SegmentationClasses
- **fraction** – scalar in [0, 1]

Returns: the volumes with the changed properties

ippai.simulate.volume_creator.**set_voxel**(volumes, x_idx, y_idx, z_idx, mua, mus, g, oxy, seg)
Sets a voxel position to a specific value in the volume

Parameters:
- **volumes** – list of numpy arrays with len(volumes) >= 3
- **y_idx** – integer
- **x_idx** – integer
- **z_idx** – integer
- **mua** – scalar
- **mus** – scalar
- **g** – scalar

Returns: the volumes with the changed properties

Conventions

class ippai.simulate.constants.**SaveFilePaths**
The save file paths specify the path of a specific data structure in the dictionary of the ippai output hdf5. All of these paths have to be used like: SaveFilePaths.PATH.format(Tags.UPSAMPLED_DATA or Tags.ORIGINAL_DATA, wavelength)

class ippai.simulate.constants.**SegmentationClasses**
The segmentation classes define which “tissue types” are modelled in the simulation volumes.

Structures

ippai.simulate.structures.**create_subcutaneous_fat_layer**(background_oxy=0.0)
FIXME: DEPRECATED FIXME: first research into fat tissue properties before adding this to simulation (!) :param background_oxy: :return:

Forward models

class ippai.simulate.models.optical_models.**OpticalForwardAdapterBase**
Use this class as a base for implementations of optical forward models.

abstract **forward_model(**absorption_cm**,** scattering_cm**,** anisotropy**,** settings**)**
A deriving class needs to implement this method according to its model.

Parameters:
- **absorption_cm** – Absorption in units of per centimeter
- **scattering_cm** – Scattering in units of per centimeter
- **anisotropy** – Dimensionless scattering anisotropy
- **settings** – Setting dictionary

Returns: Fluence in units of J/cm^2

simulate(optical_properties_path**,** settings**)**
Call this method to invoke the simulation process.
A adapter that implements the forward_model method, will take optical properties of absorption, scattering, and scattering anisotropy as input and return the light fluence as output.

Parameters:

- **optical_properties_path** – path to a *.npz file that contains the following tags: Tags.PROPERTY_ABSORPTION_PER_CM -> contains the optical absorptions in units of one per centimeter Tags.PROPERTY_SCATTERING_PER_CM -> contains the optical scattering in units of one per centimeter Tags.PROPERTY_ANISOTROPY -> contains the dimensionless optical scattering anisotropy
- **settings** –

Returns:

class `ippai.simulate.models.acoustic_models.`**`AcousticForwardAdapterBase`**

Use this class as a base for implementations of optical forward models.

abstract **`forward_model`** **(**`settings`**)**

A deriving class needs to implement this method according to its model.

Parameters: **settings** – Setting dictionary

Returns: Fluence in units of J/cm^2

`simulate` **(**`optical_properties_path`**,** `settings`**)**

Call this method to invoke the simulation process. TODO

A adapter that implements the forward_model method, will take acoustic properties as input and return the time series pressure data as output.

Parameters:

- **optical_properties_path** – path to a *.npz file that contains the following tags: Tags.PROPERTY_ABSORPTION_PER_CM -> contains the optical absorptions in units of one per centimeter Tags.PROPERTY_SCATTERING_PER_CM -> contains the optical scattering in units of one per centimeter Tags.PROPERTY_ANISOTROPY -> contains the dimensionless optical scattering anisotropy
- **settings** –

Returns:

class `ippai.simulate.models.noise_models.`**`GaussianNoiseModel`**

This class is reponsible to apply an additive gaussian noise to the input data.

`apply_noise_model` **(**`data`**,** `settings`**)**

Parameters:

- **data** –
- **settings** –

Returns:

Index

Python Module Index

i