Leopold-Franzens-University Innsbruck

Institute of Computer Science
Interactive Graphics and Simulation Group

Quickguide for the programs used in: Automatic Atlas-based Landmark Transfer for Ligaments Identification

Lukas Nocker

June 8, 2018

1 Introduction

This document describes the installation and the usage of the programs that are created for the bachelor thesis 'Automatic Atlas-based Landmark Transfer for Ligaments Identification'. Here it is only described how the programs can be used and configured, for a description of the methods used and how it works have a look at the bachelor thesis itself.

Note: The shown JSON files sometimes contain comments marked by // and /* */ but they are just included to have a better understanding of the files, a real JSON file is not allowed to contain comments.

2 Installation

The programs used in this project are based on Statismo, so it is required to install Statismo and its dependencies to compile the programs of this project. For the installation of Statismo have a look at `https://github.com/statismo/statismo`.
Note: It seems that the installation of Statismo has one of its dependencies missing, so it could lead to a missing 'proj_lib.so' error message. To solve this problem, install proj_lib by yourself before restarting the installation of Statismo.

When Statismo and all its dependencies are installed this project can be compiled. For this download the project from `https://git.uibk.ac.at/csar8121/AutoLandmark`. Currently this project is accessible only for project members, but this can be extended to an internal or even public project in the future.
After downloading the project open its 'CMakeLists.txt' file and adjust the Statismo path to the location where you installed it. Now choose a folder where the project should be installed (typically just a build folder in the project itself), navigate into this folder and execute the following commands:

```
cmake <path to the 'AutoAtlas' folder>
make -j 8
```

Now the folder contains some files and folder created by cmake and the seven programs used in the project. The programs are now ready to use; how this can be done is explained in the next sections program by program.

3 AtlasCreation

This is one of the two main programs of the project and it is used to generate atlases from a set of input data. Like all the programs it is a console program that takes a single argument. This argument is the path to a JSON file which contains all the configuration parameters and the input data. The output of the program is splitted in up to four different folders, as the program distinguishes bones of the left side from bones of the right side and also radii and ulnae will be treated independent. So for each type a own folder is created, which can be specified as parameters or as default './datadir/bonetype/side'. In this folders the aligned meshes will be stored as aligned_?.vtk file, where the ? is replaced with a up counting number; the Gaussian process model as 'gaussModel.h5'; the constrained models as 'constrained_model_?.h5' where the ? matches up with the number of the aligned meshes; the meshes in correspondences as 'correspondent_?.vtk' where the ? again matches up with the number given to the aligned files; the atlas as 'statisticalModel.h5' and if created the enhanced atlas as 'enhancedAtlas.h5'.

```
./AtlasCreation <path to the configuration file>
```

The configuration file for this program consists of two main parts; the part named 'parameters' contains all configuration parameters and the part named 'bones' that contains the input data.To distinguish radii from ulnae the bones part contains two sub parts, one for radii and one for ulnae.

```
{
  /*Section that specifies the parameters*/
  "parameters": {
    //...parameters
  },
  /*Section that specifies the input data set, containing
      two lists, one for radii and one for ulnae*/
  "bones": {
    //List of radii for the input data set, can be empty
    "radius": [
      //...input radii
    ],
    "ulna":  [
      //...input ulnae
    ]
  }
}
```

Listing 1: Base structure of the configuration file.

The 'parameter' part offers the possibility to specify many different parameters of the program. For this program all the parameters are optional because for the required parameters default values are set and other parameters extend the functionality of the program but the main part works without them. Possible parameters are:

"radius_right_folder", "radius_left_folder", "ulna_right_folder" and "ulna_left_folder" specify the output folder for the generated data. If the input data set contains bones of a type the folder that is specified in the corresponding name is created and used as output folder. If the data set contains bones of a type where no output folder is specified, "datadir/bonetype/side" is used as the default folder.

"rigid_iter" specifies the maximal number of iterations for the rigid alignment, if not set the default value 1000 is used.

"kernel_sigma" specifies the sigma of the first level of the Gaussian process model. A lower sigma will result in a more flexible model but with less smoothness and shape stability. If not set 75.0 is used as default value.

"kernel_levels" specifies the number of levels that the multilevel Gaussian process model should have. The levels allow to have a higher sigma for the base level for a smooth deformation but also allows to model small details. Too few levels with a high sigma result in loosing details and too many levels result in loosing the required smoothness. (default 3)

"kernel_scale" specifies the scaling factor that is applied on the Gaussian kernel for the Gaussian process model, so it defines the strength of possible deformations. (default 200)

"gauss_comp" specifies the number of primal components used to approximate the Gaussian process, a higher number will result in a better Gaussian process but also longer run time. If not set 250 is used as default. (For the data on which the program could be tested, values higher than 250 caused exceptions.)

"gauss_fit_iter" specifies the maximal number of iterations that are done to fit the posterior model of the Gaussian process on the bones of the input data set. (default 1500)

"correspond_volume_percentage" specifies the minimal percentage of volume that the fitted mesh has to have in comparison to the given mesh

to be a valid. If not set 0.6 is used as default value. $V(fittedmesh)/V(inputmesh) >$ "*correspond_volume_percentage*"

"landmark_variance" specifies the variance of the landmark inaccuracy, a value greater zero is required for the calculations so if zero or less is specified 0.01 is used instead. (default 0.1)

"atlas_variance" allows the atlas to be a PPCA model with the here specified variance, by setting it to zero it will be a standard PCA model. (default 0)

"atlas_sigma" specifies the sigma of the Gaussian kernel used to enhance the atlas. If not set no enhanced atlas is created.

"atlas_scale" specifies the factor with which the Gaussian kernel for the enhanced atlas is multiplied. If not set no enhanced atlas is created.

"atlas_num_comp" specifies the number of components that the combined model for the enhanced atlas should have. If not set no enhanced atlas is created.

"gauss_path_radius_r", "gauss_path_radius_l", "gauss_path_ulna_r" and "gauss_path_ulna_l" if one ore more if this parameters are set, the step of generating a Gaussian process model for the bones of the group that corresponds to the name of the parameter is skipped and instead the given model is used to generate the posterior models. This allows to reuse a GP model from an earlier execution of this program.

```
"parameters": {
  "rigid_iter": "1000",

  "kernel_sigma": "80",
  "kernel_levels": "4",
  "kernel_scale": "150",
  "gauss_comp": "250",

  "gauss_fit_iter": "1500",
  "correspond_volume_percentage": "0.6",
  "landmark_variance": "0.1",
  "atlas_variance": "1",

  "atlas_sigma": "300",
  "atlas_scale": "100",
  "atlas_num_comp": "100",
```

```
  "radius_right_folder": "radius/right",
  "radius_left_folder": "radius/left",
  "ulna_right_folder": "ulna/right",
  "ulna_left_folder": "ulna/left"
}
```

Listing 2: Example parameters.

The part with the input data consists of two lists, one for radii and one for ulnae. For both lists such an entry contains three elements:

"path" specifies the path to a .vtk or .stl file that is the mesh used as input.

"side" has to be "r" or "l", where "r" means that this input is a bone of the right side and "l" means the left side.

"landmarks" this entry contains a list of landmark coordinates. The name of each entry in this list is the landmark name and contains the three sub entries "x", "y" and "z" that specify the coordinates of the landmark.

```
{
  "path": "path\/Radius right.vtk",
  "side": "r",
  "landmarks": {
    //one entry per landmark, the entry name is the
       landmark name
    "RAF_P": {
      //each landmark contains the x, y and z coordinates
      "x": "30.993999",
      "y": "238.95",
      "z": "113.84"
    },
    //...all required landmarks for the bone type
    "PRUL_R": {
      "x": "-7.8659",
      "y": "254.78",
      "z": "322.38"
    }
  }
}
```

Listing 3: Example of a bone entry.

4 FitAtlas

This is the second main program of the project, it is used to fit an atlas on a patient mesh and identify the ligaments on the bone. In addition this program performs error calculations if control landmarks are given in the configuration. Like the 'AtlasCreation' program it is a console program with one argument, which is the path to the configuration file.

```
./FitAtlas <path to the configuration file>
```

This configuration file can contain the following parameters:

"atlas_path" the path to the atlas that should be used for this fitting, this parameter is required and the program will give an error message if the parameter is not set or the specified file cannot be found.

"patient_path" the path to the mesh on which the atlas should be fitted, this can be a .stl or .vtk file. This is a required parameter and the program will print an error message if the path is not set or the specified file does not exist.

"type" has to be "radius" or "ulna". This parameter is required so that the program knows what landmarks are contained in the model. Without this parameter the program will exit with an error message.

"align_iter" specifies the maximal number of iterations during the alignment of the patient mesh to the atlas. If not set the default value of 1000 is used.

"fit_iter" specifies the maximal number of iterations for the fitting of the atlas to the patient mesh. If not set the default value of 2000 is used.

"vtk_output_path" deters the output path for the two generated meshes. If not set the current working directory is used as output folder.

"json_output_path" specifies the path and file name of a JSON file that can be generated as additional output. If not set no JSON file is created as output.

"scalismo_gui_path" specifies the path and file name of an additional output file. This file is a text file containing scala code that can be executed in the program Scalismo-ui to get a graphical visualization of the output. If not set the file will not be generated.

”landmarks” this parameter allows to evaluate the fitting by performing the fitting on a mesh where the landmarks are known, by passing the landmark coordinates to the program over this parameter error calculations are done and stored in the JSON file, if the path for the JSON file is set. The structure of this parameter is the same as the 'landmark' parameter in the configuration file for the atlas creation.

```
{
  "atlas_path": "path\/enhancedStatisticalModel.h5",
  "patient_path": "path\/Radius right.vtk",
  "type": "radius",
  "align_iter": "1000",
  "fit_iter": "1500",
  "vtk_output_path": ".",
  "json_output_path": ".\/output.json",
  "scalismo_gui_path": ".\/output.txt",
  //optional control landmarks for the evaluation
  "landmarks": {
    "RAF_P": {
      "x": "30.993999",
      "y": "238.95",
      "z": "113.84"
    },
    //...all landmarks
    "PRUL_R": {
      "x": "-7.8659",
      "y": "254.78",
      "z": "322.38"
    }
  }
}
```

Listing 4: Example AtlasFit configuration.

The main output of this program consists of two .vtk files, the first is called 'fit_aligned.vtk' which is the input mesh but aligned with the model and the second file is called 'output.vtk' which is the same mesh as the other one but this time with additional points that represent the landmarks. Which landmark is located on which index depends on the bone type and the total number of points in the mesh and can be taken from the following table:

Landmarks		
Index	Radius	Ulna
n-1	PRUL_R	PRULd_U
n-2	DRUL_R	PRULs_U
n-3	DOAC_R	DRULd_U
n-4	POC_R	DRULs_U
n-5	DOB_R	DOAC_U
n-6	AB_R	POC_U
n-7	CBP_R	DOB_U
n-8	CBD_R	AB_U
n-9	RAF_P	CBP_U
n-10		CBD_U
n-11		RAF_D

Table 1: Indices of the landmarks depending on the bone type and $n :=$ the number of points in the mesh.

With the parameter "scalismo_gui_path" set a text file is generated that contains scala code that can be loaded in the Scalismo-ui program or Scalismolab which can be downloaded from:
`https://github.com/unibas-gravis/scalismo/wiki/scalismoLab`.
This code displays the mesh and the landmarks on it, if control landmarks are given they are also displayed to have a graphical comparison between the identified landmarks and the real one.

If the "json_output_path" is set a JSON file is created that contains three entries and three additional if control landmarks are given. This JSON file gives a good alternative to the output.vtk file as this file contains the landmarks as mesh points that are not part of the cell structure, so the mesh is no longer two manifold which can be required for some further steps. For this case and all other cases where the landmark coordinates should be accessible as values the JSON file can be accessed to get them, this coordinates correspond to the landmarks on the surface of the mesh stored in the 'fit_aligned.vtk' file.

"vtk_path" the path to the 'output.vtk' file.

"patient" the path to the mesh on which the atlas was fitted.

"landmarks" contains the list off all landmarks and their coordinates in the generated meshes.

"detailed_errors", only generated when control landmarks are given, contains a list with all landmarks and a value that represents the euclidean distance from the identified landmark position to the control landmark.

"error_sum", only created when control landmarks are given, this value is the sum of all errors in the detailed errors section.

"squared_error_sum", only created when control landmarks are given. This value is the sum of all squared errors and can be used as metric for a good fitting, for example when different parameters are compared.

```
{
  "error_sum": "18.379233082442497",
  "squared_error_sum": "39.969601330300975",
  "detailed_errors": {
    "AB_R": "1.6054380167871276",
    "CBD_R": "1.9649581614145362",
    //... error of the other landmarks
    "PRUL_R": "2.67063645733431",
    "RAF_P": "2.2974567682767266"
  },
  "vtk_path": "output_path\/output.vtk",
  "patient": "input_path\/Radius right.vtk",
  "landmarks": {
    "AB_R": {
      "x": "-2.7932338714599609",
      "y": "240.59327697753906",
      "z": "251.97752380371094"
    },
    //... coordinates of the other landmarks
    "RAF_P": {
      "x": "32.715950012207031",
      "y": "151.43565374894345",
      "z": "34.342556826453234"
    }
  }
}
```

Listing 5: Example JSON output.

Note: The ITK registration framework only uses one thread, so to increase the performance of the 'achieving correspondence' step, a thread for each bone is launched. Which heavily decreases the computation time but increases the memory usage. For more bones this could lead to memory problems depending on available amount, removing the parallelism could

help there or even better a thread pool that keeps track of the available resources.

5 GenerateTests

This program can be used to generate test cases. The program takes one argument which is the path to a JSON file that has the same structure than the configuration file for the 'AtlasCreation' program.

```
./GenerateTests <path to the configuration file>
```

But instead of generating an atlas, this program generates many configuration files that represent all possible test cases for this data set. For this in the specified output folder a series of folders is generated. Each of this folders represents one possible test case, where one mesh is excluded from the input data set and then the fitting is performed on the excluded mesh to evaluate the accuracy of the process. All folders contain three files: the 'build.json' file, which is the configuration file for the 'AtlasCreation' program; the 'fit.json' file, which is the configuration file for the 'FitAtlas' program and a 'run.sh' file which is a simple shell script that calls the atlas creation program and after that the fitting program, so a test can be executed with a single call of the script. To run all tests an additional shell script called 'runAll.sh' is generated in the current working directory, this script executes the run scripts off all the test cases.

6 Statistics

This program can be used to generate statistics out of the JSON output files from the fitting. Important for the program is that the JSON files contain the error information, means that the atlas fitting has to be done with control landmarks.
The program again takes a single argument, which is the path to a JSON configuration file.

```
./Statistics <path to the configuration file>
```

The configuration file needs to contain a list called "paths", this list contains the path to all output files that should be included in the statistics. In addition to this parameter two file paths can be specified, the first parameter is "stat_file_path", which deters the file file where the statistics should be written, if this parameter is not set the output will be written in the console

window. The second parameter is "r_file_path", if this parameter is set an additional file is generated that contains code written in R.

```
{
  "stat_file_path": "./stats.txt",
  "r_file_path": "./rCode.R",
  "paths":
  [
    "./0/output.json",
    "./1/output.json",
    "./2/output.json",
    "./3/output.json",
    "./4/output.json"
  ]
}
```

Listing 6: Example configuration file.

To calculate the statistics the program groups the input data in two different ways: by landmark name and by bone, means by input file. For each group then the minimum, maximum, mean, variance and standard derivation is calculated and written in the output file or console. If the "r_file_path" is set, a file is created that contains code written in R. This code can be executed to receive boxplot diagrams, which is an easy way to get a graphical representation of the statistics.

7 ConfigCreate

A small console program that takes one parameter, which is the file name of the file that should be created from this program.

```
./ConfigCreate <file name of the output>
```

The program will ask many user inputs and generates a JSON file out of the requested input that can be used as configuration file for the 'AtlasCreation' program. To generate a configuration file with this program takes a lot of time but allows users to create a configuration file even when they do not know how the JSON file is structured.

8 ConfigFit

Like 'ConfigCreate' this program is also used to generate a configuration file from user inputs. The output file name is again specified as the program parameter.

```
./ConfigFit <file name of the output>
```

The file generated from this program can be used as configuration file for the 'AtlasFit' program.

9 Tune

This is a small program that runs the atlas creation and fitting process multiple times with different parameters to identify the parameter combination that works best for this specific case. To keep it simple only the three most significant parameters vary during the performed tests. The search of the best parameters is within a specified range with a defined step length. As the program performs an exhaustive search a really high amount of tests is done and the program has a very high run time, which makes the program unusable for a real parameter specification and should only gives an example how the parameters could be tested automatically.
Like all the other programs it is a console program that takes a single parameter which is the path of a configuration file in JSON format.

```
./Tune <path to the configuration file>
```

The configuration file has to contain the following 13 entries:

"build_config_path" the path to a configuration file for the 'AtlasCreation' program.

"fit_config_path" the path to a configuration file for the 'FitAtlas' program, important for this procedure is that this file contains control landmarks to measure the landmark error for the tested parameters.

"type" has to be "ulna" or "radius" to specify the bone type.

"side" has to be "r" or "l", where "r" means that the bones are from the right side and "l" means that the bones are from the left side.

"gp_sigma_min" specifies the lower limit of the search for the best sigma used in the Gaussian process model.

"gp_sigma_max" specifies the upper limit of the search for the best sigma used in the Gaussian process model.

"gp_sigma_step" specifies the step size of the search for the best sigma used in the Gaussian process model.

"at_sigma_min" specifies the lower limit of the search for the best sigma used in the Gaussian kernel for the enhanced model.

"at_sigma_max" specifies the upper limit of the search for the best sigma used in the Gaussian kernel for the enhanced model.

"at_sigma_step" specifies the step size of the search for the best sigma used in the Gaussian kernel for the enhanced model.

"levels_min" specifies the lower limit of the search for the best number of levels for the multilevel Gaussian process model.

"levels_max" specifies the upper limit of the search for the best number of levels for the multilevel Gaussian process model.

"levels_step" specifies the step size of the search for the best number of levels for the multilevel Gaussian process model.

The output of the program is a folder called 'atlas_tune' in the current working directory that contains many pairs of JSON files called 'build_?.json' and 'fit_?.json' where the ? are replaced with matching up counting numbers. This files are the generated configuration files with different parameters. For each pair of configuration files a folder called 'atlas_tune?', where the ? is replaced with the corresponding number, is created in the output folder specified in the original given configuration file. This folders will then contain all files generated during the atlas generation and fitting.

```
{
  "build_config_path": "path\/build.json",
  "fit_config_path": "path\/fit.json",
  "type": "ulna",
  "side": "r",
  "gp_sigma_min": "20",
  "gp_sigma_max": "200",
  "gp_sigma_step": "20",
  "at_sigma_min": "200",
  "at_sigma_max": "500",
  "at_sigma_step": "50",
  "levels_min": "3",
  "levels_max": "6",
  "levels_step": "1"
}
```

Listing 7: Example tune configuration.