User Manual
of
Library of Hierarchy of Parts

Intelligent Robotics Laboratory
University of Birmingham

Version 1.0

Contents

1 Introduction

In this User Manual, first software requirements and installation steps of Library of Hierarchy of Parts (LibHOP) are given. Then, the implementation of LibHOP on a benchmark image dataset, namely ETH80 image dataset, is explained using an example.

2 Installation on Linux

In this section, software requirements for compiling Library of Hierarchy of Parts and installation steps are given.

2.1 Software Requirements

2.1.1 Operating System

The LibHOP library has been tested on Ubuntu 12.04.

2.1.2 Compilers

Required compilers:

- CMake,
- gcc-4.6 (or newer) with C++11 support.

2.1.3 Libraries

Required 3rd Party libraries:

- OpenCV 2.3.1 or newer (build from source),
- Intel TBB 3.0 for Open Source.
- gzib (zlib).
- Optional: protobuffer if compiling with java JNI support.

2.2 Installation Steps

1. Update the libraries using

   ```
   sudo apt-get install gcc g++ cmake swig libtbb-dev libprotoc-dev
   protobuf-compiler libprotobuf-java zlib1g-dev
   ```

2. Go to LibHOP root folder and create the installation folder;

   ```
   mkdir build
   ```

3. Go to the installation folder;

   ```
   cd build
   ```

4. Update the cmake file;

   `ccmake ..`

5. During cmake configuration;

   (a) Press **t**, and

   (b) Add **'-std=c++0x'** to **CMAKE_CXX_FLAGS** and **CMAKE_C_FLAGS**.

   (c) Note: If you observer errors about c++ linker,
   please add **'-ldl'** to **CMAKE_EXE_LINKER_FLAGS**.

   (d) Update, save and generate the make file, and exit ccmake.

6. Run cmake;

   `cmake ..`

7. Build the source code;

   `make`

8. Install LibHOP;

   `sudo make install`

3 Examples

In this section, the steps for running LibHOP are described using an example which is given in the folder **Examples/ETH80**. In this example, the samples belonging to the following categories are selected from ETH80 dataset:

- Apple
- Bottle
- Giraffe
- Mug
- Swan

3.1 Requirements

The script for running the example is **run_example.sh**. Before running the script, the following binary files, which are produced in the build folder of LibHOP, should be copied to the example folder **Examples/ETH80**:

- `apisample`: Common files that are used in the library.
- `hop1create`: The file that is used for creating the first layer features and parts in the inference phase.

- `hopncreate`: The file that is used for creating higher layer parts and compositions in the inference phase.
- `hoplearning`: The file that is used for learning parts and compositions.
- `hopserver`: The file that is used as entry points between different files.
- `hopdisplay`: The file that is used for displaying the results, and exporting the learned libraries and realizations.
- `libhop.so`: Shared object file for common libraries.

Otherwise, you can either set the correct location of these files in a shell run-control file under Linux or in the example script.

3.2 Description of the implementation files

In the example file, `run_example_for_category.sh` is called. There are two arguments used with `run_example_for_category.sh`:

- The first argument $1: The name of the folder which contains image files for training and test datasets, such as `apple`.
- The second argument $2: The pattern that defined the names of the groundtruth files, such as `_applelogos.groundtruth`, which are in the folders of training datasets, e.g. `apple`.

For instance, you can examine the LibHOP on Apple dataset using `./run_example.sh` which calls

`./run_example_for_category.sh apple _applelogos.groundtruth`

Note: In order to run LibHOP using other different categories (e.g. bottle) after you have completed an experiment on a category (e.g. apple), you should rename the res folder or remove the contents of the res folder, `eth_learning/res`, and remove the files `lib2.png`, `lib2sc.png`, `lib3.png`, `lib3opt.png`, `lib3sc.png`, `lib4.png`, `lib4sc.png` and `lib4opt.png`. Because, these files are produced using Apple dataset and may not be useful in the experiments on the other datasets.

Descriptions of script files that are called in the file `run_example_for_category.sh` are given in the following subsections.

3.2.1 Training Phase

1. `infer0_train.sh`: Script file for creating the first layer structure, by extracting Gabor features and constructing the first layer parts, in the training phase.

   In the file, `hop1create` is called with the following parameters:

   ```
   ./hop1create ly1-6_inference.cfg $2
   "inference.ly1.from_namespace = inference.ly1_extract; src_dir=$1/train;
   out_dir=$1/layer1;out_prefix=train_;groundtruth_extension=$3;
   part_lib_name = $1/layer1lib.plb;"
   ```

(a) **ly1-6_inference.cfg**: The configuration file which contains the algorithm parameters.

(b) **$1**: The variable which represents the category name, such as apple, and the name of the folders that contain training and test images belonging to that category.

(c) **$2**: The pattern which represents the names of the ground truth files, such as _applelogos.groundtruth.

(d) **inference.ly1.from_namespace**: The namespace to `inference.ly1_extract` in order to extract Gabor features and construct part libraries.

(e) **src_dir**: The directory of the training images.

(f) **out_dir**: The output directory to which the part libraries and the structures produced at the first layer will be exported.

(g) **out_prefix**: The pattern of the names of the files which will contain graph structures and part libraries that are computed using training data.

(h) **groundtruth_extension**: The pattern of the name of the ground truth files.

(i) **part_lib_name**: The name and location of the file to which the part library will be exported.

2. `llearning.sh`: Script file for learning the part libraries at the layers L=1,2,3,4 in the training phase. In the file, `hoplearning` is called with the following parameters:

```
./hoplearning ly1-6_learning.cfg train_*.ly1 "action=optimization;
pattern = train_%%s_0_0.ly1; library = $1/$2;
lib_export_name = $1/lib; src_dir = $1/layer1;
learning.optimization.min_layer = 2;
learning.optimization.max_layer = 4;
learning.optimization.start_layer = 2;
learning.optimization.end_layer = 4;"
```

(a) **ly1-6_learning.cfg**: The configuration file which contains the algorithm parameters.

(b) **train_*.ly1**: The files which contain the graph and part structures computed at the first layer.

(c) **action**: The action that will be invoked. In this case, we invoke optimization.

(d) **pattern**: The patterns of the file names which contain the graph structures computed at the first layer.

(e) **library**: The name of the part library learned at the first layer.

(f) **lib_export_name**: The name of the folder that the learned part library will be exported.

(g) **src_dir**: The name of the folder from which the graph structures constructed at the first layer will be imported.

(h) **learning.optimization.min_layer**: The index of the minimum layer that the optimization will be employed on.

(i) **learning.optimization.max_layer**: The index of the maximum layer that the optimization will be employed on.

(j) **learning.optimization.start_layer**: The index of the layer at which the optimization will be started.

(k) **learning.optimization.end_layer**: The index of the layer upto which the optimization will be employed.

3. `infer1-4_training.sh`: Script file for inference at the layers L=1,2,3,4 in the training phase.

   In the file, `hopncreate` is called with the following parameters:

   ```
   ./hopncreate ly1-6_inference.cfg train_*.ly1 "start_layer = 2;
   end_layer = 4; pattern = train_%%s_*.ly1; part_lib_name = $1/lib4.plb;
   src_dir = $1/layer1; out_dir=$1/layerx;"
   ```

   (a) **ly1-6_inference.cfg**: The configuration file which contains the algorithm parameters.

   (b) **train_*.ly1**: The files which contain the graph and part structures computed at the first layer.

   (c) **start_layer**: The index of the layer from which the inference processes will be started.

   (d) **end_layer**: The index of the layer upto which the inference processes will be implemented.

   (e) **train_%%s_*.ly1**: The pattern of the files which contain the graph and part structures computed during the inference.

   (f) **part_lib_name**: The name of the file which contains part libraries.

   (g) **src_dir**: The directory which contains the files in which the graph and part structures computed at the first layer are stored.

   (h) **out_dir**: The directory which contains the files in which the graph and part structures computed at the second, third and fourth layers will be stored.

4. `olearning.sh`: Script for learning part libraries at the object and category layers (i.e. the layer L=5,6) in the training phase.

   In the file, `hoplearning` is called with the following parameters:

   ```
   ./hoplearning ly1-6_learning.cfg train_*.ly4 "action = learn_objects;
   pattern = train_%%s_*.ly5; category_name = $1; library = $1/lib4.plb;
   out_library = $1/olib.plb; src_dir = $1/layerx;"
   ```

   (a) **ly1-6_learning.cfg**: The configuration file which contains the algorithm parameters.

   (b) **train_*.ly4**: The files which contain the graph and part structures computed at the second, third and fourth layers.

   (c) **action**: The action that will be invoked. In this case, we invoke object learning.

(d) **train_%%s_*.ly5**: The pattern of the files which contain the graph and part structures computed during the inference at the fifth and the sixth layers.

(e) **category_name**: The name of the object category.

(f) **library**: The name of the part library learned at the second, third and fourth layers.

(g) **src_dir**: The name of the folder from which the graph structures constructed the second, third and fourth layers will be imported.

(h) **out_dir**: The directory which contains the files in which the graph and part structures computed at the fifth and sixth layers will be stored.

5. `infer4-6_training.sh`: Script file for inference at the layers L=5,6 in the training phase.

   In the file, `hopncreate` is called with the following parameters:

   ```
   ./hopncreate ly1-6_learning.cfg $2 "start_layer = 4; end_layer = 6;
   part_lib_name = $1/olib.plb; src_dir = $1/layer1/; out_dir = $1/layerx/;
   "
   ```

   (a) **ly1-6_inference.cfg**: The configuration file which contains the algorithm parameters.

   (b) **$2**: The variable which represents the patterns of the files that store graph and part structures computed at the first layer, e.g. `*.ly1`.

   (c) **start_layer**: The index of the layer from which the inference processes will be started.

   (d) **end_layer**: The index of the layer upto which the inference processes will be implemented.

   (e) **part_lib_name**: The name of the file which contains part libraries computed at the fifth and the sixth layers.

   (f) **src_dir**: The directory which contains the files in which the graph and part structures computed at the first layer are stored.

   (g) **out_dir**: The directory which contains the files in which the graph and part realizations computed at the fifth and the sixth layers will be stored.

3.2.2 Testing Phase

(a) `infer0_test.sh`: Script file for creating the first layer structure, by extracting Gabor features and constructing the first layer graph and part structures, in the testing phase.

In the file, `hop1create` is called with the following parameters:

```
./hop1create ly1-6_inference.cfg $2
"inference.ly1.from_namespace = inference.ly1_normal; src_dir=$1/test;
out_dir=$1/layer1;out_prefix=train_;groundtruth_extension=$3;"
```

i. **ly1-6_inference.cfg**: The configuration file which contains the algorithm parameters.
ii. **$1**: The variable which represents the category name, such as apple, and the name of the folders that contain training and test images belonging to that category.
iii. **$2**: The extension of the names of image files, e.g. `*.jpg`.
iv. **$3**: The pattern which represents the names of the ground truth files, such as _applelogos.groundtruth.
v. **inference.ly1.from_namespace**: The namespace to `inference.ly1_extract` in order to extract Gabor features and compute part realizations.
vi. **src_dir**: The directory of the test images.
vii. **out_dir**: The output directory to which the part libraries and the structures produced at the first layer will be exported.
viii. **out_prefix**: The pattern of the names of the files which will contain graph structures and part libraries that are computed using test data.
ix. **groundtruth_extension**: The pattern of the name of the ground truth files.

(b) `infer1-4_test.sh`: Script file for inference at the layers L=1,2,3,4 in the testing phase. The structure of the file is similar to the structure of `infer1-4_training.sh` given in the previous subsection.

(c) `infer4-6_test.sh`: Script file for inference at the layers L=5,6 in the testing phase. The structure of the file is similar to the structure of `infer4-6_training.sh` given in the previous subsection.

3.2.3 Displaying the results

(a) `savelib.sh`: Script file for displaying the part libraries. In the file, `hopdisplay` is called with the following parameters:

```
./hopdisplay $2 savelib.cfg $1/lib2sc.png "layer=2;show_labels=true"
```

i. **$2**: The variable which represents the file that contains the part libraries.
ii. **savelib.cfg** : The configuration file that contains the algorithm parameters.
iii. **$1**: The variable which represents the folder that an image of the part libraries will be stored.
iv. **layer**: The index of the layer which contains the learned part libraries that will be displayed.
v. **show_labels**: The parameter which determines whether the labels of the part types will be displayed or not.

Sample outputs exported to `lib2sc.eps`, `lib3sc.eps`, `lib4sc.eps` and `lib5sc.eps` are depicted in Figure 1, 2, 3 and 4, respectively. The numbers represent the indexes of parts in the part library.

The details of the algorithm parameters that are used in the scripts and configuration files are given in the document titled **Definitions of Parameters**.

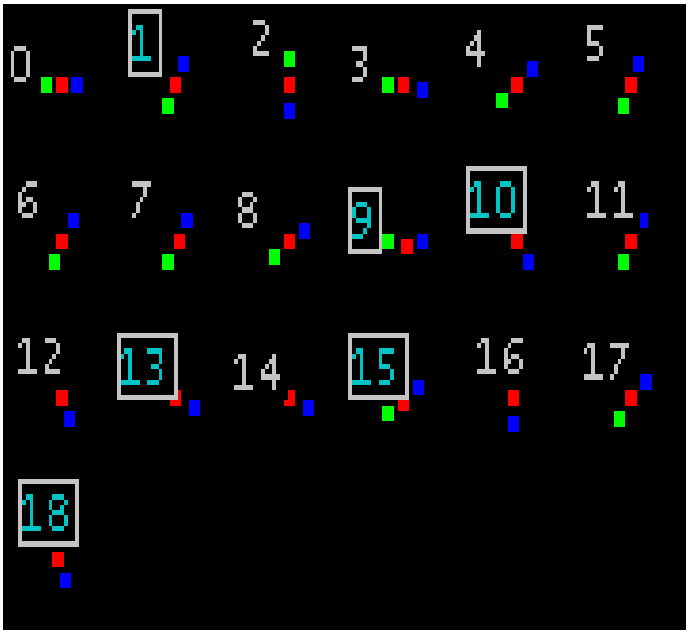

Figure 1: Part libraries learned at the second layer and displayed in `lib2sc.eps`.

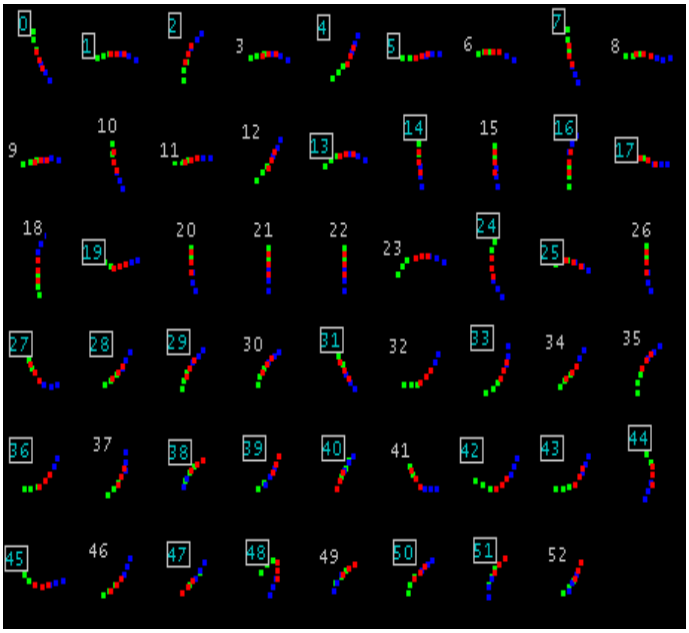

Figure 2: Part libraries learned at the second layer and displayed in `lib3sc.eps`.

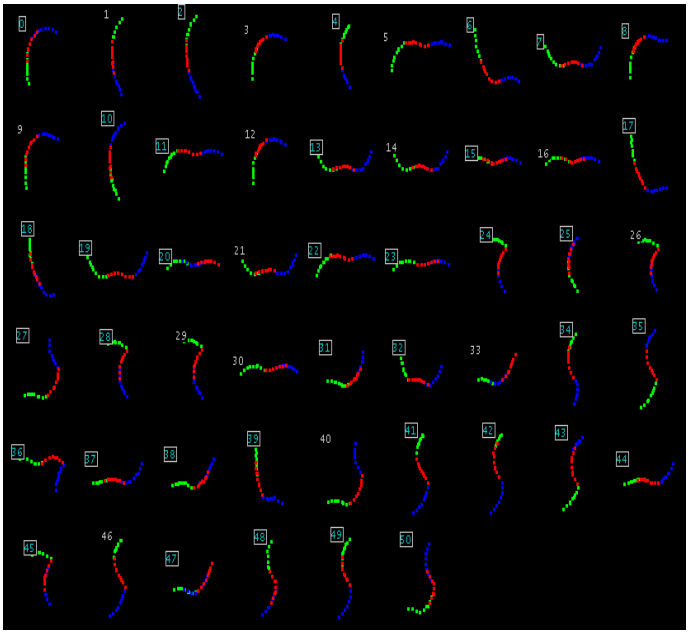

Figure 3: Part libraries learned at the second layer and displayed in `lib4sc.eps`.

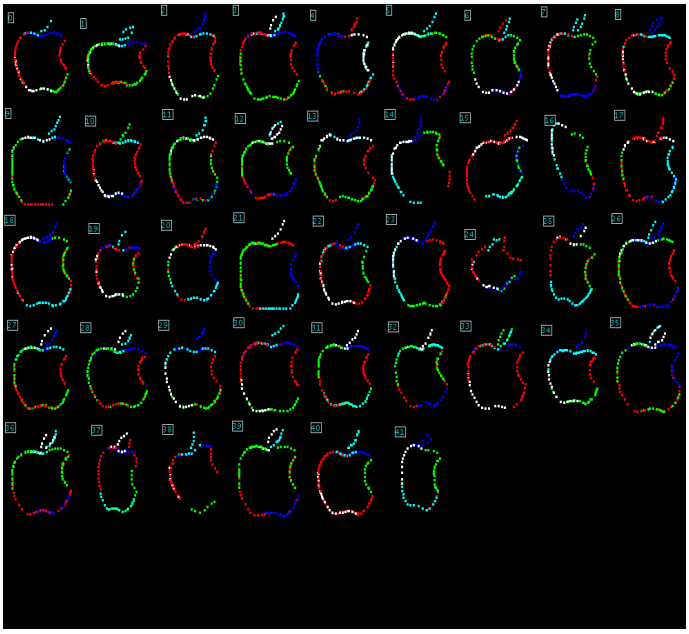

Figure 4: Part libraries learned at the second layer and displayed in `lib5sc.eps`.