

Scallop Detector (v1.2) - Quick User Manual

Please send any other questions not answered here or bugs to:

matthew.d.dawkins@gmail.com

M-F I'm usually available 10-6pm

Package Folder Organization

-- Binaries	Various Pre-Built Binaries (Executables)
-- Code	All code (Matlab and C++) for the project
-----+ data	Repository for Classifiers and stored histograms used by the program
-----+ gcc	Makefile for G++ compiler
-----+ src	Source Code
-----+ tools	Training Utilities
-----+ VS2010	Visual Studio 2010 Project
-- Documentation	Misc Documentation

Contents

1. Important Notes
2. Program Usage
3. Compiling Instructions
4. Classifier Training
5. Special Notes to Developers

1. Important Notes

This default algorithm features a general classifier trained on a wide variety of environments with consistent illumination correction contrast settings. Especially for some of the extremely buried cases, a generally trained classifier didn't always perform as well as one trained for a specific environment/image set. It is recommended to first try the general classifier on a data set. If it performs well, great, if not you might want to train a new classifier on the data (or a new general classifier). Additionally, changing the color and illumination correction software settings might invalidate some old classifiers. I don't know how much this software has changed in the last 6 months but the given classifiers may no longer even be valid. Then again, they might also adapt to new changes (see below). In other words they are untested.

The algorithm features a feedback loop which models both the background color and the color of Scallops in a particular environment as we detect Scallops with our more robust measures. Because of this, it is not recommended to use the detection software on an individual image and this also has the side effect that the first few images in a set might have a lower detection and higher false positive rate (as it's still making an initial model of the background/scallop color). Additionally, it is not recommended to mix image sets, and to use the algorithm on back to back images from the same environment.

This program expects the image metadata to be in the same format as all of the images it's seen. If the image metadata is not embedded in this format, the image will be skipped or the program will exit. For JPEGs, this means containing the metadata after the keyword "IMTAKE" within the first 40000 characters of the input file. For TIFFs, it is loaded from the image comment field (commonly 4408 blocks from the end of the file). This is where Jason Rocks' illumination correction software use to embed the data, but I believe this just changed recently and they now lack metadata. If you want the metadata to be loaded in a different format, the source code in the Size Detection/ImageProperties.cpp will have to be changed.

I attached a prebuilt windows executable as well as my source code and instructions on how to compile it. For linux (and maybe even windows) I would recommend downloading OpenCV and building the detector specifically for your system. I never got around to making an installer for Linux, although the supplied makefile for use with G++/GCC should work fine. If it doesn't please contact me with a copy of the error. The supplied binaries for Ubuntu and Windows are both experimental, and were only quickly tested on a few other systems. If either fails, recompiling the code on your system may be necessary.

For additional speed benefits OpenCV can be built with IPP support. I never experimented with this, but a few websites said that it dramatically increased performance in their experiments. Right now the algorithm usually processes images in about 1 sec on a single core, but alas, that speed depends on the altitude of the probe. There is an internal threading system within the scallop detector but it is currently disabled.

2. Program Usage

The Scallop Detector can be used as a command line utility or just a standalone executable. There are now lots of different use cases and potential modes to run it in depending on the intent.

If used on the command line, as is recommended, the program has the following format:

```
ScallopDetectorExecutable [MODE/SYSTEM_CONFIG_FILE] [INPUT FOLDER/FILELIST] [OUTPUT FILE/FOLDER] [MODE-OPTIONAL-SETTINGS]
```

All of the possible modes in the current system (6/9/12) are elaborated in the section “modes” below.

Folder names should not contain spaces, but depending on the Operating System they can (sometimes if you use spaces, the directory is required to be in quotes).

The “data” folder contains all of the configuration files, settings, stored histograms, and classifiers utilized by the system. **The data folder should be placed in the same folder as the EXECTUABLE, or one folder above it. It is automatically detected. Make sure not to use old data folders with newer algorithms.** For tweaking the system it is important to know how the “data” folder is configured. It has 3 subfolders:

1. ConfigFiles:

There are 2 types of configuration files: classifier system config files and processing-mode config files. This folder contains both. Classifier config files are described in the training section and below. Processing mode config files store settings for each of our processing modes (surprise). Each “mode” has an associated config file that stores information such as specifying what should be outputted from the algorithm. The special case file “SYSTEM_CONFIG” stores settings shared across all modes. This basically just includes the focal length **definition (which should be changed if the new sensor is any different)** and some other misc settings. If you open up the files PROCESS_DIR, DEFAULT, and SYSTEM_CONFIG you’ll get a pretty good idea of what’s in there because they’re well commented.

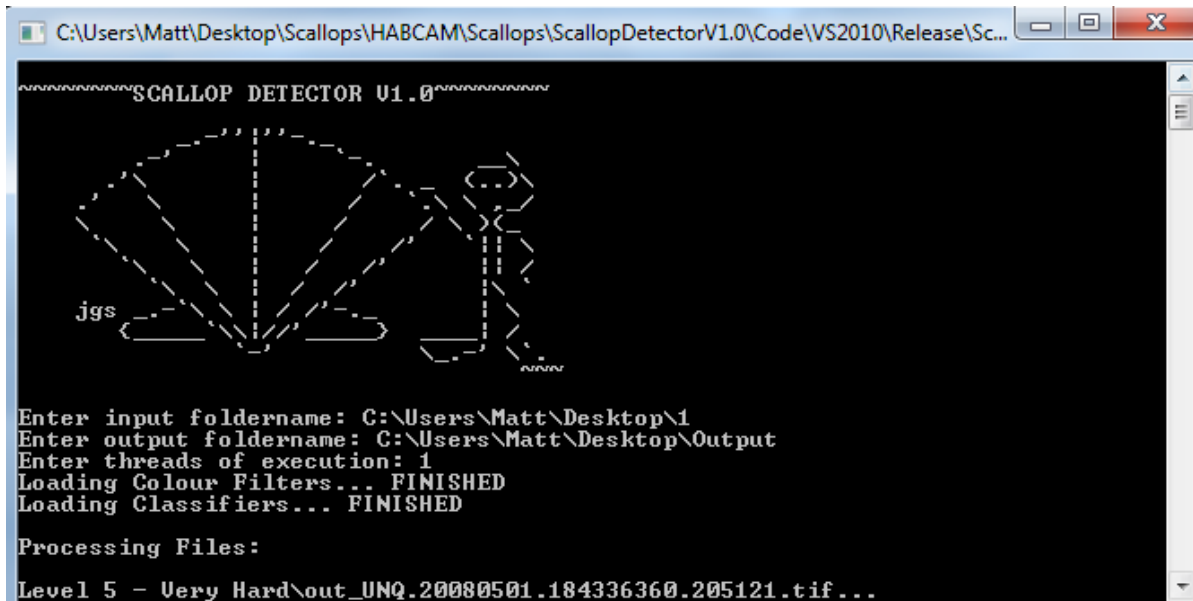
Advanced: New “modes” can be specified simply by making a new configuration file in this directory. For example, if we wanted to make a process_directory_display_only mode that only shows the display and doesn’t create a text output file we could copy the process_dir config file and change the output display option from true to false. “process_directory_display_only” could then be used as the mode string on the command line. Classifier system config files are likewise identified by a single string, they’re name as it appears in this directory. Classifier config files describe classifier systems for an entire substrate/water condition category, or a combination of them.

2. ColorFilterBanks:

Stores initial seed histograms used for detecting special class objects. At this time these can’t be re-learned, but they are modified when running the algorithm over many frames anyways.

3. Classifiers

Stores all pre-trained single-class classifiers. Each classifier should be placed in a subdirectory specifying the substrate the classifier was trained on, ie, sand, gravel, etc. Currently there is just 1, "DEFAULT". Classifier system config files in the ConfigFiles directory may often reference multiple single-class classifiers from the same directory.



Using the prebuilt binaries (Windows)

Run ScallopDetector.exe in the Binaries/Win32/Algorithm/ folder either via the command prompt or by double clicking it

Notes:

-Windows version does not support threading.

Using the prebuilt binaries (Linux)

Run ./ScallopDetector with the below arguments

Notes:

-Prebuilt linux binaries still require OpenCV installed as a prereq

Compile Instructions (Windows)

1. Install OpenCV 2.2 or above using either the windows OpenCV installer, or cmake. Instructions can be found at:

<http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.2/>
<http://opencv.willowgarage.com/wiki/InstallGuide>

2. A provided Visual Studio 2010 solution is given in Code/VS2010. Open Scallop Algorithm.sln and confirm that the project directory options point to the correct OpenCV directories. [Right Click on Project -> Configuration Properties -> VC++ Directories]. See:

http://opencv.willowgarage.com/wiki/VisualC%2B%2B_VS2010

3. Build the project in release mode (select Release option in drop down then [Build->Build Solution])

Note: If making on MinGW or another version of Visual Studio make sure to include all of the source code in the 'src' directory in your project, and to correctly link to OpenCV

Compile Instructions (Linux - Ubuntu and Similar)

1. Install OpenCV using CMAKE for your platform. Install to the default directory. See instruction guide: <http://opencv.willowgarage.com/wiki/InstallGuide> Make sure you also have the gcc/g++ compilers, and a pthreads development package installed.

2. Open a command prompt and use the cd command to go to the directory gcc in Code/gcc. Open up Makefile in a text editor and confirm that LIBPATH and INCPATH correctly link to your OpenCV installation.

3. Type 'make' to use the given makefile to compile the detector in the current directory

4. Run the Scallop Detector by typing ./ScallopDetector

Output List

Output file Info in standard output list

[FILE_NAME_WHERE_DETECTED] [IMG_LOCATION_ROW] [IMG_LOCATION_COL] [MAJOR AXIS] [MINOR AXIS] [AXIS ANGLE] [TYPE=AS SPECIFIED IN CLASSIFIER SYSTEM FILE]
[RELATIVE_MEASURE_OF_SCALLOPINESS][RANK FOR POINT – IF MORE THAN ONE POSSIBLE CLASSIFICATION FOR IT]

Ex output file:

UNQ.20110621.064058703.07158.jpg,320.92,380.674,62.6796,62.6796,0,BROWN,0.015547,1

UNQ.20110621.064137968.07319.jpg,571.894,676.313,51.0535,51.0535,0,BROWN,0.01716,1
UNQ.20110621.064339640.07806.jpg,896.737,473.022,62.6796,62.6796,0,WHITE,0.019734,1

The 2nd to last number of each line is a relative measure of how scallopy an interest point is. The higher the number, the more likely it's actually a scallop. That is not to say that the number means anything. The machine learning algorithms used do not try to make the numbers be "1" or "-1" for positive and negative detections respectively. They simply try to make positive annotations be positive and the negatives negative, the magnitude means little, but some information can be gleaned from their relative magnitudes.

Operating Modes

1. Processing modes:

ScallopDetector PROCESS_DIR [foldername] [outputfilename]

Process a directory of images and create an outputlist at outputfilename. The images must have embedded metadata. Will use the DEFAULT classifier system.

ScallopDetector PROCESS_DIR [foldername] [outputfilename] [CLASSIFIER-ID]

Process a directory of images and create an outputlist at outputfilename. The images must have embedded metadata. CLASSIFIERID specifies the classifier system to use [DEFAULT, SAND, etc.] in a case sensitive manner.

ScallopDetector PROCESS_LIST [input_file_list] [outputfilename]

Input_file_list should be the file path to a text file (relative or absolute) which contains a list to process in the form of a space-delimited list. Images must have metadata in them.

[image1] [classifier config for image 1]

[image2] [classifier config for image 2]

...

Ex>

Filename1.png DEFAULT

Filename2.png SAND

Filename3.png DIRT

Etc...

ScallopDetector PROCESS_METALIST [input_file_list] [outputfilename]

Input_file_list should be the file path to a text file (relative or absolute) which contains a list to process in the form of a space-delimited list. In this case we specify metadata in the input file.

[image1] [altitude] [pitch] [roll] [classifier config for image 1]

[image2] [altitude] [pitch] [roll] [classifier config for image 2]

...

Ex>
Filename1.png 1.02 0.01 0.00 DEFAULT
Filename2.png 0.75 0.02 -0.05 SAND
Etc...

2. Data Extraction Modes (see training section)

ScallopDetector GUIEXTRACT_DIR [foldername] [outputfilename]

Foldername should be a folder containing images with metadata. The simple GUI will start.

ScallopDetector MIPEXTRACT_DIR [path_to_annotations_file] [outputfilename]

The given annotations file should lie in a directory containing the images it references. The data extractor will be run on all files in the directory.

ScallopDetector MIPEXTRACT_LIST [list_filename] [outputfilename]

For when we want to supply the metadata externally while training

List_filename should have the following format:

Line 1: link_to_file_containing_annotations.csv
Line 2: filename1 altitude1 pitch1 roll1
Line 3: filename2 altitude2 pitch2 roll2
Etc..

Misc Comments on Building

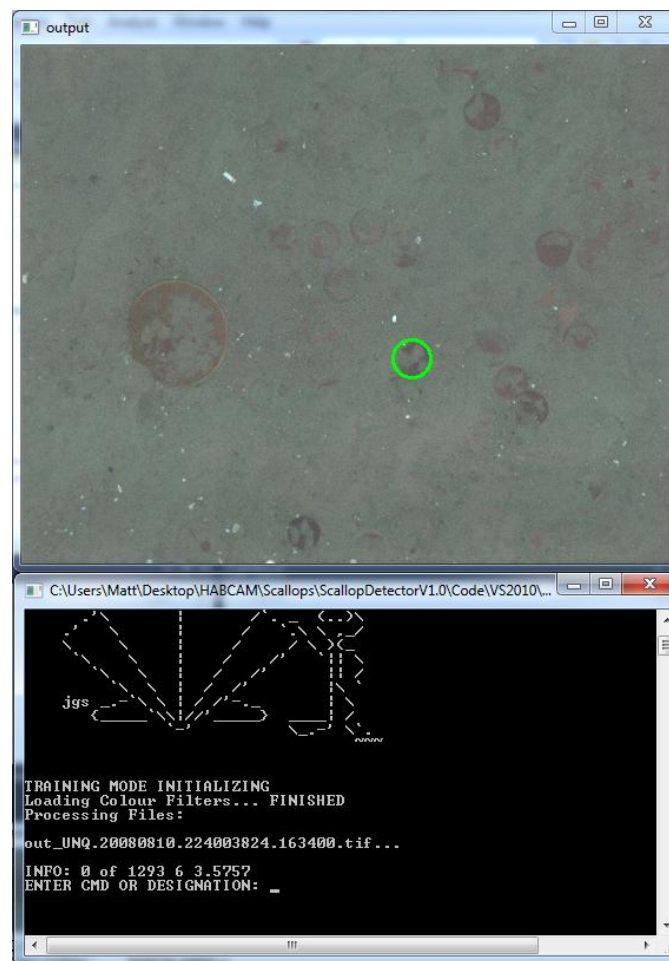
The source file definitions.h in Code/src/Common contains many of the constants and options to be used during compilation, but most of the important ones are now in config files. This still includes things such as Scallop search sizes, resolutions to run different interest point detectors at, and compile time options though. For compiling on the windows platform, WIN32 should be defined here (it should be uncommented, but its usually defined automatically on windows compilers anyway). To compile with POSIX threading support to enable multithreading, USE_PTHREADS should be defined here. For the most part these shouldn't have to be changed although the focal length is also still defined here (but this should really be moved to a new configuration file containing input properties).

4. How to Train a New Classifier System

Classifiers are trained via 2 parts: a C++ executable and a Matlab training script. The C++ executable extracts data around interest points within HabCam images by asking the user which designation it should give the interest points it generates or by using pre-annotated MIP csv annotations. The Matlab training script then takes this data and builds classifiers around the annotated data.

Stage 1: Data Extraction - GUI

The data extractor looks exactly like the algorithm in use, only that for every interest point in the image it will ask the user what designation it should give the interest point. Interest points that are more interesting to the algorithm will be shown first, however, at this stage of the algorithm, scallops may still be towards the back of the list of what the algorithm finds “interesting”.



Designations MUST be positive integers (0+). The user can choose whatever designations he/she likes to represent different objects and categories, however designation '0' should be uninteresting interest points no matter what (background points, objects we're not classifying, etc.).

For example, I used '1' to be brown scallops, '2' to be white scallops, '3' to be buried scallops where the interest points were relatively well localized over the potential scallop. I then selected '4' to be interest

points which were poorly localized over scallops (too big, too small, off center) and '0' to be everything else. During training I then choose to ignore points with designation '4' and trained with designations '1', '2', and '3' to be my positive examples (ie what we're searching for). Lots of configurations are possible.

To allow the rapid collection of data from images, I added multiple commands to speed up the process:

ALLZERO - Marks all interest points in an image to have the designation '0'

SKIP – Do nothing with the rest of the interest points in the image and jump to the next image in input

S – Will also skip to the next image

SKIPN – Skip the next N interest points

SKIPSMALL – Will skip arbitrarily small interest points

SKIPLARGE – Will skip arbitrarily large interest points

EXIT – Will exit training and output any remaining data not outputted yet

SKIPCUSTOM – Will skip a custom size of interest points, requires two further parameters in the [0->1] range (the lower and upper bound). 0 corresponds to the smallest scallop size (2cm) and 1 to the largest (24cm)

SKIPAREA – useful if you only want to look at part of an image, will ask for 4 params specifying a rectangular region in the area to scan

Recommendations:

Typically, to train decent classifiers, they require a lot of input cases. Having 100+ annotated scallops per classifier and more total annotations than features (default=3800 with all feature modules turned on) is best. While this seems like a daunting task, there are 2 ways around this and when I trained classifiers for my final paper it only took me around 20 min/classifier to collect data from images. On one hand, the given training program has some prior examples attached with it so if you only collect a little bit of data with the extractor, you can fill in the rest with prior examples (see below). On the other, the extraction utility lets you quickly annotate a lot of false examples by finding images with no scallops in them and using the ALLZERO command. It is more important to focus on collecting good cases of scallops and objects that look like scallops (and attempting to capture those cases in the extractor).

Stage 1 – MIP Extraction

Using MIP annotations to extract features is still experiment and is complicated by a few issues including that the annotations may sometimes lack entries near the image border (which still may get picked up as interest points). **When using MIP annotations it is imperative that all scallops get annotated as any ones that are not may get picked up as false examples if not.** There are settings in SYSTEM_CONFIG that detail if we should ignore candidates near borders in case they are not annotated, and for detailing how many candidates we want to output. If using one of the above extractor tools, a data file will be generated containing a line by line breakdown of a candidates classification followed by all 4000 features for that candidate. This is fed into the Matlab training script.

Stage 2: Matlab Classifier Training Utility using the GML AdaBoosting Toolkit

After running the above data extractor, a data file will be generated in whatever output folder you specified, called "tdata.dat" or whatever you specified. This file contains your designations for each annotated interest point, and the features extracted around each IP. In Code/tools/trainer is the Matlab function CreateClassifier which will convert this data file into a classifier.

```
function CreateClassifier(data_input_fn, class_output_fn, target_ids, skip_ids, mode, max_iter)
```

data_input_fn - Is the path to the extracted data file generated by the extractor

class_output_fn - Whatever you want to call your generated classifier, it will be generated in your current directory

targets_ids – a vector [x x x] containing the target_ids to classify as positive points (ie what we're searching for) This would be [1 2 3] in my example case. If using MIP files, these should be the object IDs in the annotations 18*** or the like.

skip_ids – a vector of designations to skip [4] and ignore. **For example, we may want to skip dead scallops when making a level 1 classifier because they are so close to scallops that we don't want to confuse the classifier, the same thing for interest points we are unsure about.**

mode – can either be 0, 1, 2, 3 or 4, indicating whether or not you want to append any prior data collected by MD. **The data in them are old, so it's probably recommended to just train on the newer data.**

mode 0 – Append no additional examples

mode 1 – Append all false examples (ie no example scallops, only examples of what is not a scallop)

mode 2 – Appends all prior examples (including buried, white and brown scallops with other false examples)

mode 3 – Will append just a little bit of the false examples on top of your data, until we reach 4000 IP

mode 4 – Will append just a little bit of the mixed examples on top of your data, until we reach 4000 IP

max_iter = Maximum number of iterations during training. Recommended value = approx 200-250, but it depends on how many points you supply, and how fast you want the algorithm to run. Too large and you run the risk of becoming over-constrained, too small and vice-versa.

Training time varies based on the machine you're running and the amount of annotations inputted. This can vary between 15 minutes and 20 hours (← when I trained on 25k interest points).

Stage 3: Creating a new classifier system comprised of multiple classifiers that can be called from the command line, or in a process list

A classifier system contains multiple single- stage classifiers (see thesis). First, all “level 1” classifiers are applied to all candidates. Any candidates which have a positive result from any of the level 1 classifiers moves onto the level 2 suppressors. Suppression classifiers should be trained to identify things that look like our Objects of Interest, those that may make it past the level 1 classifiers. If a level 2 classifier has a value higher than a level 1, then the candidate is suppressed.

For a good sample of a classifier config file, see the “DEFAULT” file in data/ConfigFiles/

1. To make a new classifier system, you can simply copy this default file to a new file in ConfigFiles and rename it accordingly. For example say we want a SANDY classifier system. We would first copy the DEFAULT classifier to have a new name, “SANDY”. We can now call the SANDY classifier when processing a list or a directory, for example ScallopDetector PROCESS_DIR [foldername] [output_list_name] SANDY
2. Now we just have to differentiate our “SANDY” classifier from our DEFAULT classifier by filling out the settings in the SANDY config file. For organizational purposes, in the Data/Classifiers folder we should add a new folder, SANDY, where we put all of the single classifiers we trained for this setting. We would then change the topmost string in the SANDY config file from DEFAULT to SANDY to point to this new directory.
3. You would then follow the comments in the SANDY config file on how to add whatever level 1 and level 2 classifiers you want to use trained in the previous section.

5. Notes to Developers

In this algorithm there is a wide range of code from a wide range of sources. Some functions are commented very nicely, broken up into many smaller subsections, and are highly modular. Others are clunky, long poorly commented scripts. If you have any questions about anything feel free to email me.

The overall architecture of the algorithm closely follows the provided paper.

ScallopDetector.cpp contains the main algorithm

Function ProcessImage takes an image and turns it into what the algorithm perceives as scallops. It is called once for every image in the input folder. First the size range of scallops is detected, and then various IP detection methods are applied to the base and derived images.

A candidate struct is created for every detected interest point in the image. As we process more information about each candidate, we add more information (features) to the struct until eventually we can classify it. Deallocation of all candidates occurs in two spots: during IP consolidation and at the end of ProcessImage.

Changing some settings in definitions.h may require the retraining of certain classifiers.