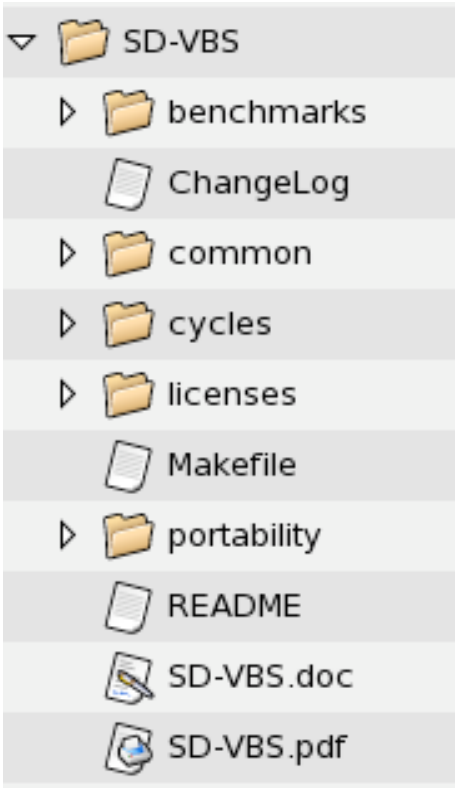
SD-VBS: The San Diego Vision Benchmark Suite

SD-VBS is a comprehensive collection of vision algorithms. The suite includes:

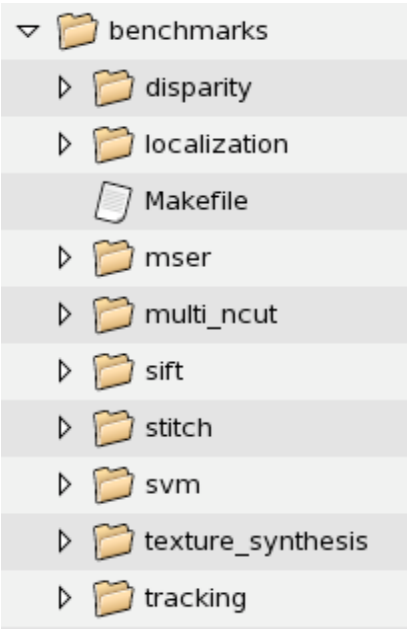
- Disparity
- Tracking
- Scale- Invariant Feature Transform (SIFT)
- Support Vector Machine (SVM)
- Image Stitch
- Texture Synthesis
- Maximally Stable Regions (MSER)
- Image Segmentation (Multi-Ncut)
- Localization

These algorithms have been implemented in both MATLAB and C.
The folder structure of the SD-VBS suite looks like this:



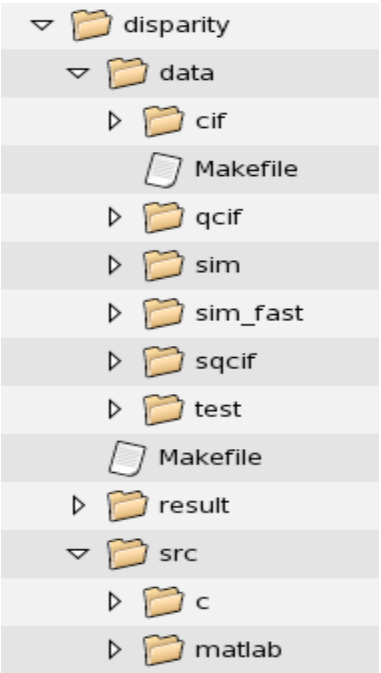
Benchmarks

The SD-VBS → benchmarks folder contains the vision benchmarks listed above.



Algorithm directory structure

The folders listed under SD-VBS → benchmarks correspond to the various algorithms they represent. Every algorithm folder contains source directory (for MATLAB and C source files), data directory and one result directory. Commonly used library functions across different algorithms are placed in SD-VBS -> common directory.



SD-VBS -> <algorithm> -> src -> c	C source files for <algorithm>
SD-VBS -> <algorithm> -> src -> matlab	MATLAB source files for <algorithm>
SD-VBS -> <algorithm> -> result	Result folder
SD-VBS -> <algorithm> -> data	Test data input files

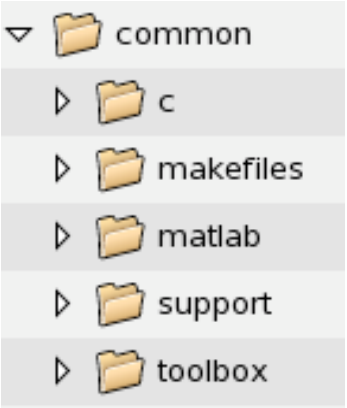
We have 9 different input data sizes for each benchmark. They are called:

WUXGA	2.3M pixels
FULLHD	2.0M pixels (This is the 1080p format)
VGA	300k pixels
CIF	100k pixels
QCIF	25k pixels
SQCIF	12.5k pixels
SIM	Execution time around 6-10 million cycles
SIM_FAST	Execution time around 2-4 million cycles
TEST	Execution time around 10k-100k cycles

The test vectors – sim, sim_fast and test; are cycle-constrained data sets. The cycle count was obtained using C version of the source code, running on 64-bit X86 architecture.

Common directory

The SD-VBS -> common directory contains commonly used library functions, Makefiles and support files.



common -> c	C implementations of the library functions	
common -> matlab	MATLAB implementations of the library functions	
common -> makefiles	Commonly used Makefiles	
	Makefile.recurse	Recursive Makefile to traverse SD-VBS directory structure
	Makefile.common	Top-level Makefile that contains commands to run C/MATLAB/MCC version
	Makefile.include	Common Makefile used to find top directory path
common -> support	Support files	
	buildTable.py	Collates information from Cycles/ directory and

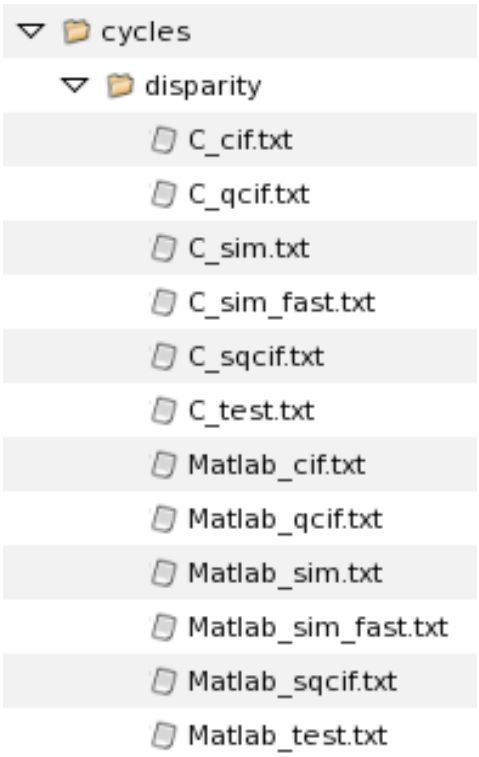
		builds a table with timing information for all benchmarks. Python script
common -> toolbox	This directory has the libraries for various modules of the SD-VBS. These libraries are implemented in MATLAB. The user need not set the path to this Toolbox. Relative paths are taken care of in the Makefile.	

Licenses directory

This folder contains licenses corresponding to each vision benchmark. Benchmark specific license agreements are provided in the files: <Benchmark>.license.txt

Cycles directory

This directory contains execution time files for each benchmark. Every benchmark is run across different input sizes in both MATLAB and C. The corresponding execution time for each run is stored in these files, whose nomenclature is:
<Benchmark> ---> <C or MATLAB>_<input type>.txt



Portability directory

To port SD-VBS to any platform, the user can use/modify the header files contained in this directory. For details, please refer to Portability.txt in SD-VBS directory.

Software Requirements

1. To execute MATLAB source files, you are required to have MATLAB R2007b or higher.

2. To execute C source files, you need GCC compiler.
3. There is an optional gmake debug command that allows you to debug your code. We use GDB as our choice of debugger. You can also use valgrind or any other debugger. Just remember to modify it in Makefile.common.

Constraints

1. The input filenames for all benchmarks must be named:
    a. 1.bmp, 2.bmp etc for Image inputs
    b. 1.txt, 2.txt etc for text inputs

   Work-around: The user can choose to have his choice of file-names. For this, modify the source file (script_<benchmark.c>) and Makefile of the particular benchmark.

How to run a benchmark, say disparity

1. Go to SD-VBS -> benchmarks -> disparity directory.
   cd disparity
2. Say we want to run "sim" input data size.
   cd data/sim
   gmake c-run (This executes C version of disparity for "sim" input)
   gmake matlab-run (This executes MATLAB version of disparity for "sim" input)
   gmake mcc-run (This executes MCC version of disparity for "sim" input)
3. To run all test cases under disparity benchmark,
   cd disparity
   gmake c-run

Sample C execution output

Benchmark	- disparity
Data set	- sim
Input size	- (100x100)
Verification	- Successful
Cycles elapsed	- 145000

Note:
Input size corresponds to the image size as (rows x columns). If the input to an algorithm were a data file, the Input size field would correspond to the file size and other parameters.
If the generated output from the current C execution matches the expected output file in sim -> expected_C.txt, the Verification stage is marked "Successful".

How to globally run all benchmarks

1. Go to SD-VBS directory
2. Do:
   a. gmake c-run
   b. gmake matlab-run
   c. gmake mcc-run

These commands recursively traverse the VBS structure and will make all benchmarks for every input size.

How to generate timing table, cycles.txt

1. Go to SD-VBS directory.
2. Do: gmake table

This generates cycles -> cycles.txt, which has the collated timing information for all benchmarks.

How to check for correctness

For every benchmark, we have included the expected output files in their respective test case directories. The MATLAB expected output files are named "expected.m" and the C counterparts are named "expected_C.txt". The user can check the correctness of the code (or his modified code) using these expected output files.
To enable correctness check:

- Modify Makefile.common to include a preprocessor command "CHECK". This would turn-on self check mode of the SD-VBS suite. (This is the default mode)
- To turn off the self-check mode, remove –DCHECK from the COMPILE_CC command.

COMPILE_C = gcc -DGCC **-DCHECK** -DGENERATE_OUTPUT -D$(INPUT) -lm -O2 $(INCLUDES)

How to generate new expected output files for new test cases (not in SD-VBS)

In case the user chooses to test the benchmarks with his/her own choice of test data sets (not in SD-VBS), he will need to generate expected output files (for both MATLAB and C) to make use of the self-checking mode.

Let us say, the user wants to create a new test vector, "newData", for disparity.

1. Create the directory "newData", in disparity → data directory.
2. Include your input images (should be named 1.bmp , 2.bmp etc.,) in this directory.
3. Include **–DGENERATE_OUTPUT** pre-processor command in Makefile.common.

   COMPILE_C = gcc -DGCC -DCHECK **-DGENERATE_OUTPUT** -D$(INPUT) -lm -O2 $(INCLUDES)

   The –DGENERATE_OUTPUT command would generate new expected_C.txt and expected.m files for newData images.

4. Copy Makefile from disparity->data->test into disparity->data->newData. Modify INPUT variable to "newData".
5. Do:
   a. gmake matlab-run
   b. gmake c-run

NOTE

Due to the differences in MATLAB and C floating-point semantics, there could be cases where the MATLAB output does not exactly match the C output. Numerical values may differ across MATLAB and C, but visually they may look the same.

Report bugs to sdvbs-users@cs.ucsd.edu