Vanderbilt Aerospace Design Laboratory (VADL)
Scale-Invariant Feature Transform (SIFT)
Localization and Imaging System (LIS)
Software Architecture Documentation
(**V**ADL**S**IFT**L**IS**S**AD -> **VSLS**)

by Sebastian Bond

Section 1. Compilation and Development Environment

Section A. Platform support

The VADL implementation of SIFT supports compilation on and for the following two operating systems/platforms. Cross-compilation is not implemented.

1. macOS
    a. Supports all features except the LoRa module (since it requires the pigpio library which requires the Raspberry Pi's GPIO pins)
2. 64-bit Raspberry Pi OS with Nix installed on the Raspberry Pi
    a. A 64-bit/arm64 OS is needed to be compatible with Nix, the package manager chosen by VADL.

Section B. Build systems and IDE support

Nix is used in order to create reproducible builds.

The following build systems are supported.

1. Makefile for macOS and Linux
    a. Before running `make`, execute `nix-shell` with Nix installed.
2. Xcode project for macOS
    a. Before building, ensure Nix is installed, as Xcode will download Nix packages when building.

Section 2. How does SIFT work?

SIFT can be used as a "black box," but here is a great explanation of how it works: https://www.vlfeat.org/api/sift.html#:~:text=A%20SIFT%20keypoint%20is%20a,an%20angle%20expressed%20in%20radians).&text=SIFT%20keypoints%20are%20circular%20image%20regions%20with%20an%20orientation.

The paper used by VADL to implement SIFT: https://www.ipol.im/pub/art/2014/82/ (more specifically, https://www.ipol.im/pub/art/2014/82/article.pdf)

Section 3. Extending SIFT

Section A. How to integrate another SIFT implementation module in 5 easy steps

Another implementation that performs localization can be added, even one that doesn't use SIFT as its general algorithm.

1. -> Define a macro like SIFT*ImplNameGoesHere* in the Makefile under `# Choose a SIFT implementation here:`
    a. -> Add the compilation rules in the corresponding ifeq's for the macro in order to define SIFT_SRC and SIFT_OBJECTS (and append to CFLAGS if needed)
2. -> Implement the following functions for an ifdef SIFT*ImplNameGoesHere*_ in SIFT.hpp and SIFT.cpp, using any return values you want:
    a. -> findKeypoints: keep the signature (parameters) the same
        i. Should be thread-safe (reentrant).
    b. -> findHomography: keep the signature the same
        i. Returns false if not enough keypoints and true otherwise.
        ii. -> After matching, render to the preview window if CMD_CONFIG specifies it.
3. -> Implement the following structs for an ifdef SIFT*ImplNameGoesHere*_ in KeypointsAndMatching.hpp and KeypointsAndMatching.cpp
    a. -> ProcessedImage<SIFT*ImplNameGoesHere*>: use this to store the result of findHomography calls
        i. -> Implement handling of these in siftMain.cpp in the matcher thread (matcherThreadFunc)
    b. -> SIFTParams: use this to store implementation-specific parameters for SIFT.
        i. -> Optionally implement a `--sift-params` command-line argument handler for your ifdef SIFT*ImplNameGoesHere*_ in siftMain.cpp
    c. -> SIFTState: use this to store any implementation-specific state that should persist between findKeypoints and findHomography calls.
        i. The struct's members should be thread-safe as needed, since findKeypoints is called from multiple threads.
4. -> Handle not enough keypoints in siftMain.cpp
5. -> Handle not enough descriptors to match in siftMain.cpp

Section 4. Modular Software Design

- Data sources: can be cameras (CameraDataSource), video files (VideoFileDataSource), image folders (FolderDataSource), or more generally, opencv video captures (OpenCVVideoCaptureDataSource).
- Data outputs: can be video files (FileDataOutput) or preview windows (PreviewWindowDataOutput).