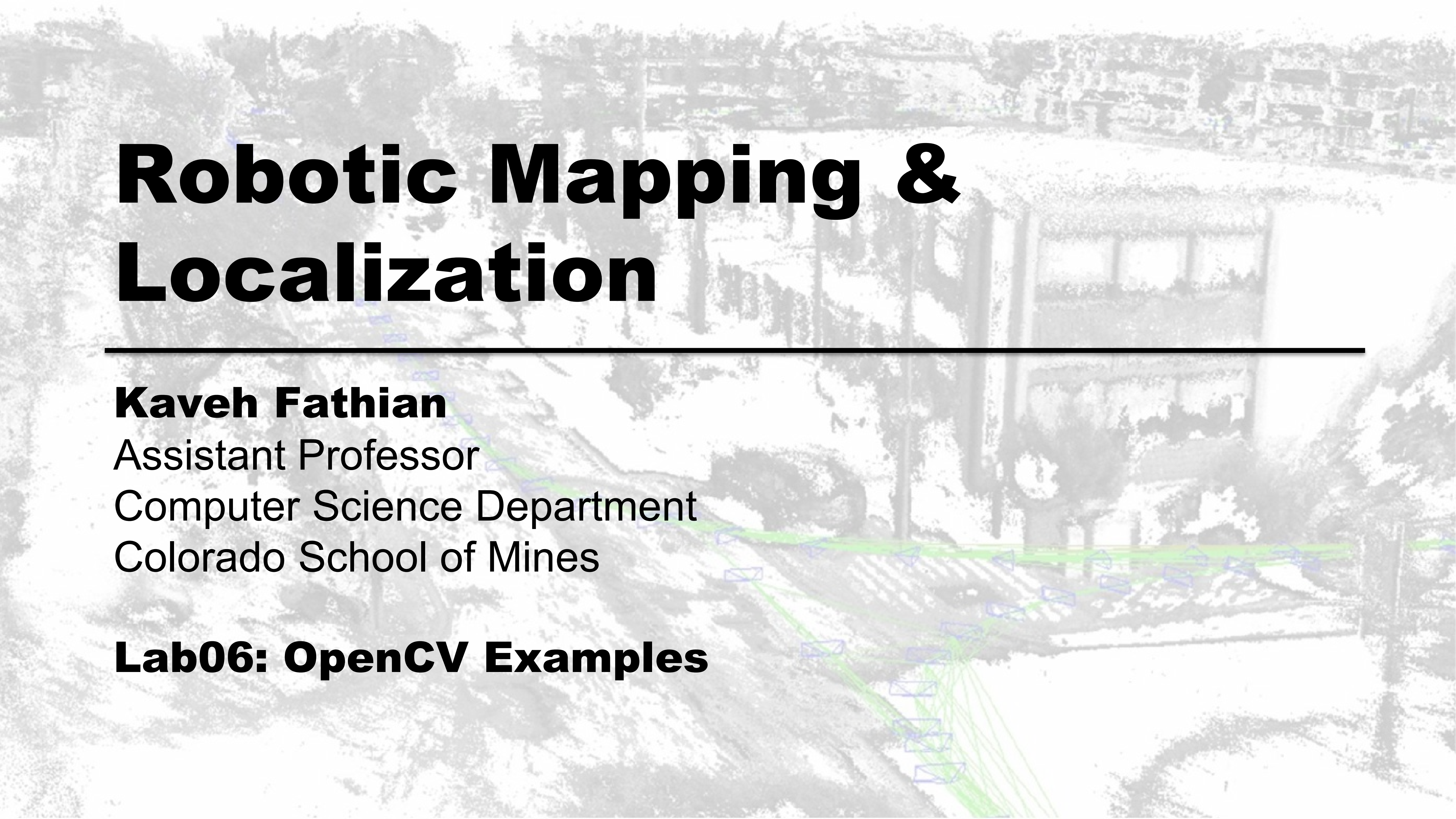


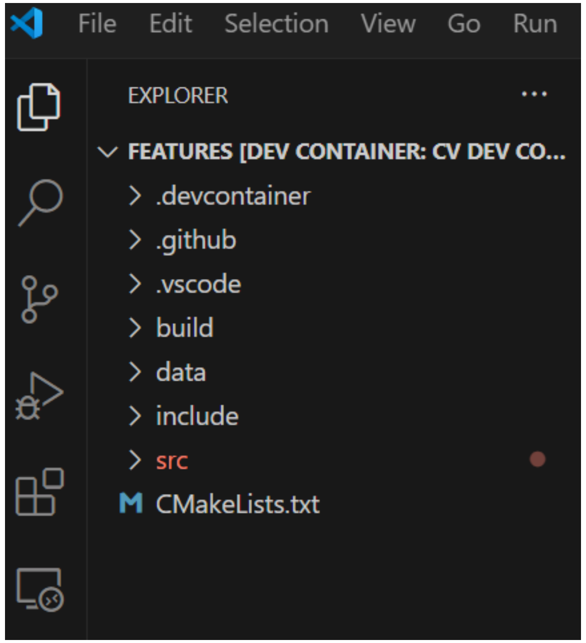
Robotic Mapping & Localization

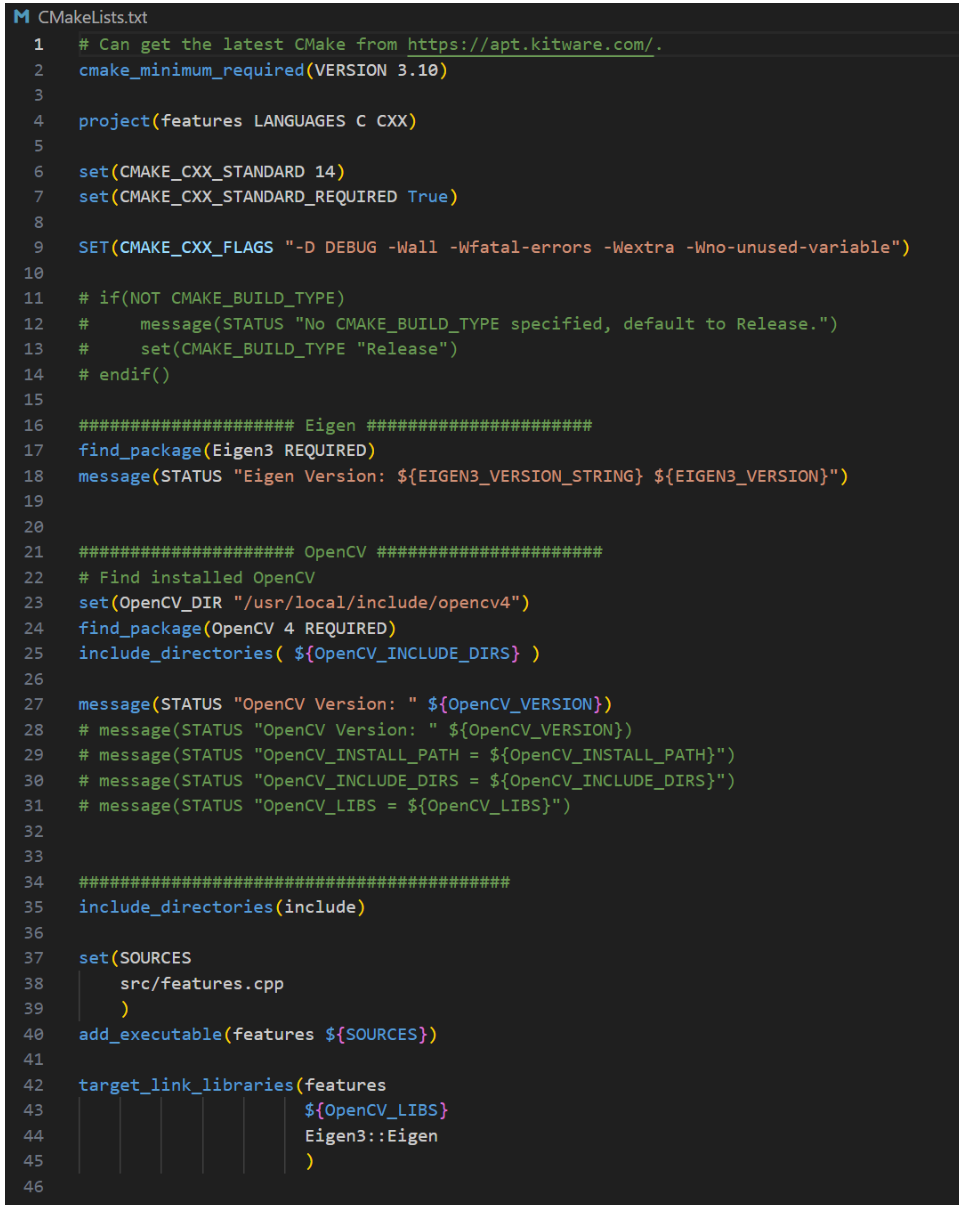
Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

Lab06: OpenCV Examples

SIFT Features

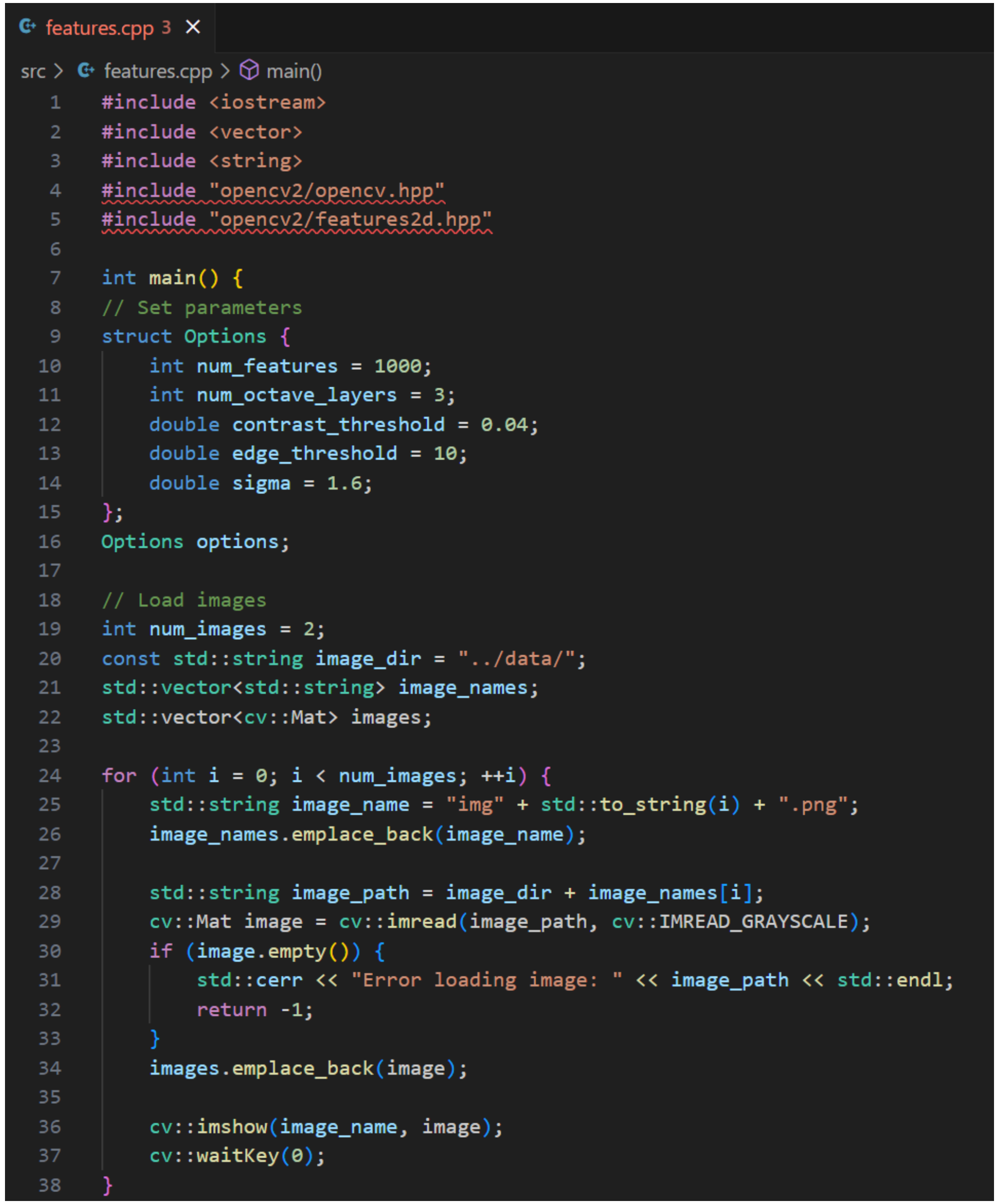
CMake project structure:





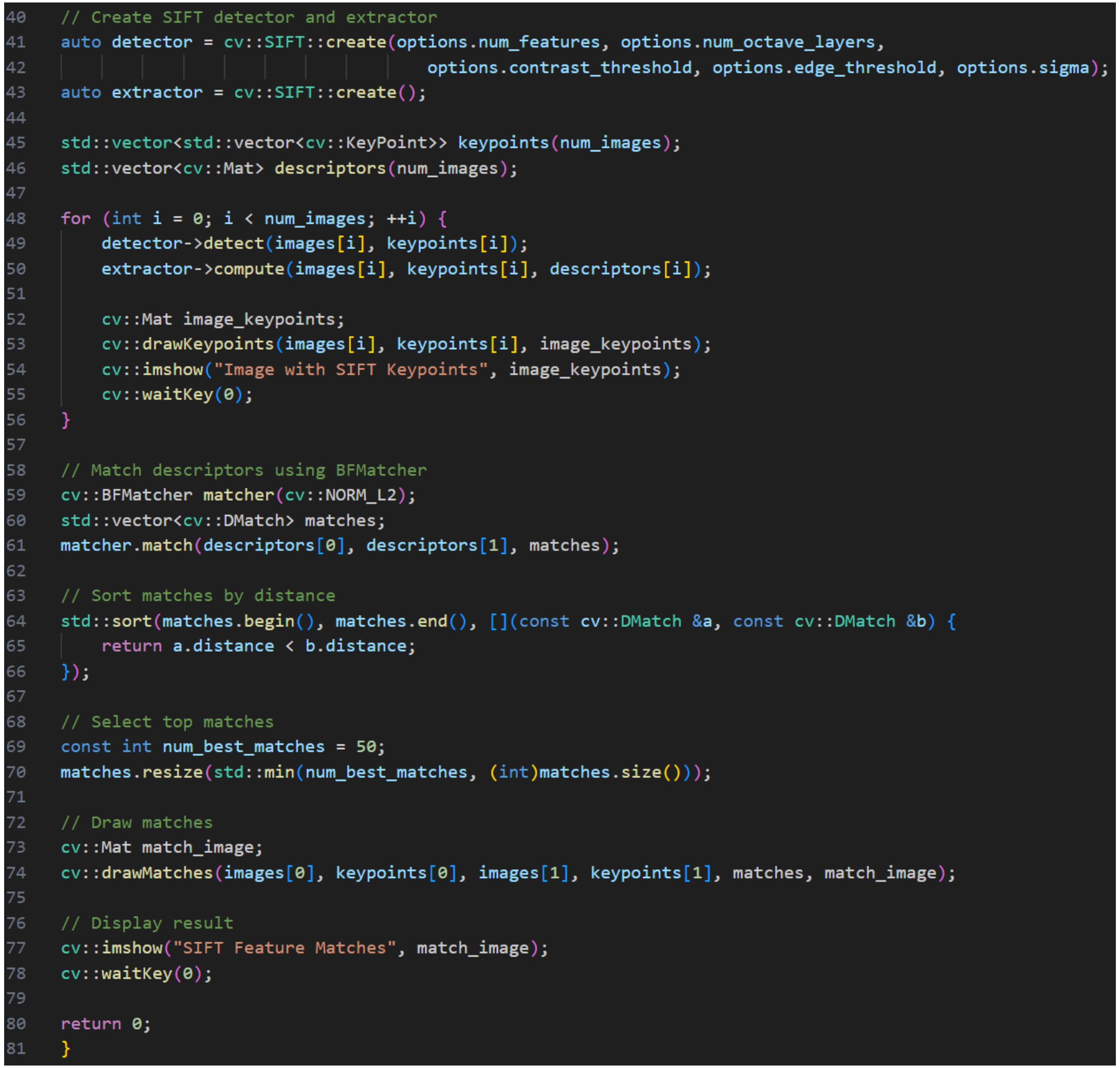
```
# Can get the latest CMake from https://apt.kitware.com/.
cmake_minimum_required(VERSION 3.10)

project(features LANGUAGES C CXX)

set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED True)

SET(CMAKE_CXX_FLAGS "-D DEBUG -Wall -Wfatal-errors -Wextra -Wno-unused-variable")

# if(NOT CMAKE_BUILD_TYPE)
#     message(STATUS "No CMAKE_BUILD_TYPE specified, default to Release.")
#     set(CMAKE_BUILD_TYPE "Release")
# endif()

##################### Eigen #####################
find_package(Eigen3 REQUIRED)
message(STATUS "Eigen Version: ${EIGEN3_VERSION_STRING} ${EIGEN3_VERSION}")


#################### OpenCV #####################
# Find installed OpenCV
set(OpenCV_DIR "/usr/local/include/opencv4")
find_package(OpenCV 4 REQUIRED)
include_directories( ${OpenCV_INCLUDE_DIRS} )

message(STATUS "OpenCV Version: " ${OpenCV_VERSION})
# message(STATUS "OpenCV Version: " ${OpenCV_VERSION})
# message(STATUS "OpenCV_INSTALL_PATH = ${OpenCV_INSTALL_PATH}")
# message(STATUS "OpenCV_INCLUDE_DIRS = ${OpenCV_INCLUDE_DIRS}")
# message(STATUS "OpenCV_LIBS = ${OpenCV_LIBS}")


##########################################
include_directories(include)

set(SOURCES
    src/features.cpp
    )
add_executable(features ${SOURCES})

target_link_libraries(features
                      ${OpenCV_LIBS}
                      Eigen3::Eigen
                      )

```

SIFT Features



```cpp
#include <iostream>
#include <vector>
#include <string>
#include "opencv2/opencv.hpp"
#include "opencv2/features2d.hpp"

int main() {
// Set parameters
struct Options {
    int num_features = 1000;
    int num_octave_layers = 3;
    double contrast_threshold = 0.04;
    double edge_threshold = 10;
    double sigma = 1.6;
};
Options options;

// Load images
int num_images = 2;
const std::string image_dir = "../data/";
std::vector<std::string> image_names;
std::vector<cv::Mat> images;

for (int i = 0; i < num_images; ++i) {
    std::string image_name = "img" + std::to_string(i) + ".png";
    image_names.emplace_back(image_name);

    std::string image_path = image_dir + image_names[i];
    cv::Mat image = cv::imread(image_path, cv::IMREAD_GRAYSCALE);
    if (image.empty()) {
        std::cerr << "Error loading image: " << image_path << std::endl;
        return -1;
    }
    images.emplace_back(image);

    cv::imshow(image_name, image);
    cv::waitKey(0);
}
```

SIFT Features



```cpp
// Create SIFT detector and extractor
auto detector = cv::SIFT::create(options.num_features, options.num_octave_layers,
                                    options.contrast_threshold, options.edge_threshold, options.sigma);
auto extractor = cv::SIFT::create();

std::vector<std::vector<cv::KeyPoint>> keypoints(num_images);
std::vector<cv::Mat> descriptors(num_images);

for (int i = 0; i < num_images; ++i) {
    detector->detect(images[i], keypoints[i]);
    extractor->compute(images[i], keypoints[i], descriptors[i]);

    cv::Mat image_keypoints;
    cv::drawKeypoints(images[i], keypoints[i], image_keypoints);
    cv::imshow("Image with SIFT Keypoints", image_keypoints);
    cv::waitKey(0);
}

// Match descriptors using BFMatcher
cv::BFMatcher matcher(cv::NORM_L2);
std::vector<cv::DMatch> matches;
matcher.match(descriptors[0], descriptors[1], matches);

// Sort matches by distance
std::sort(matches.begin(), matches.end(), [](const cv::DMatch &a, const cv::DMatch &b) {
    return a.distance < b.distance;
});

// Select top matches
const int num_best_matches = 50;
matches.resize(std::min(num_best_matches, (int)matches.size()));

// Draw matches
cv::Mat match_image;
cv::drawMatches(images[0], keypoints[0], images[1], keypoints[1], matches, match_image);

// Display result
cv::imshow("SIFT Feature Matches", match_image);
cv::waitKey(0);

return 0;
}
```