MIT Splash

Fall 2013

Machine Learning and Audio Analysis with Python

Course Notes

Daryl Sew

Contents

1 Course Description

Machine learning is a field of computer science that concerns writing programs that can make and improve predictions or behaviors based on some data. The applications of machine learning are very diverse - they range from self driving cars to spam filters to autocorrect algorithms and much more. Using scikits-learn, an open source machine learning library for Python, we'll cover reinforcement learning (the kind used to create artificial intelligence for games like chess), supervised learning (the kind used in handwriting recognition), and unsupervised learning (the kind eBay uses to group its products). We'll then cover audio analysis through Fourier transforms with numpy, an open source general purpose computational library for Python, and we'll use our newfound audio analysis and machine learning skills to write very basic speech recognition software.

2 Implementation

2.1 `vision/modules/`

`process_image(Image *img)`
`Image *img` is actually a set of two images. The following code splits img into the left and right components:

```
img->lockImage();
Mat left_mat = img->getImage(BGR, 0).clone();
Mat right_mat = img->getImage(BGR, 1).clone();
img->unlockImage();
```

Checks to see whether `cv::findChessboardCorners` can find chessboard corners in the images; if they are found in either image, it calls `cv::drawChessboardCorners` and posts the corners to the GUI. If they are found in both images, it saves the corners to `imagePoints`. Once `count` frames have been found, it calls `calibrate()`, which will use the saved corners to calculate camera parameters.

`calibrate()`
Uses `cv::stereoCalibrate()` to generate intrinsic and extrinsic camera parameters and save them to intrinsics.yml and extrinsics.yml.

2.2 `vision/modules/stereo/intrinsics.yml`

Stores the intrinsic camera calibration parameters. These do not change with different scenes (depend on the cameras themselves). These are used as inputs to `cv::stereoRectify` and `cv::initUndistortRectifyMap()` in `generateDepthMap()` from `Stereo_Vision.cpp`.
M1 - 3x3 left camera matrix
M2 - 3x3 right camera matrix
D1 - 1x5 left camera distortion coefficients
D2 - 1x5 right camera distortion coefficients

2.3 `vision/modules/stereo/extrinsics.yml`

Stores the extrinsic camera calibration parameters. These are used as inputs to `cv::stereoRectify()` in `generateDepthMap()` from `Stereo_Vision.cpp`.
R - 3x3 rotation matrix between the coordinate systems of the first and second cameras
T - 3x1 translation vector between coordinate systems of the first and second cameras
R1 - 3x3 rectification transform (rotation matrix) for the first camera
R2 - 3x3 rectification transform (rotation matrix) for the second camera
P1 - 3x4 projection matrix in the new rectified coordinate systems for the first camera
P2 - 3x4 projection matrix in the new rectified coordinate systems for the second camera
Q - 4x4 disparity to depth mapping matrix

2.4 `vision/modules/stereo/Stereo_Vision.cpp`

`generateDepthMap()`
Given a left and right image, camera calibration parameters, and vision tuning parameters, calculates a depth map using a semi-global block matching algorithm. The main functions in use here are `cv::stereoRectify()`, `cv::initUndistortRectifyMap()`, and `cv::StereoSGBM::SGBM()`. StereoSGBM must be initialized and given the vision tuning parameters in order to use `cv::StereoSGBM::SGBM`.

2.5 vision/modules/stereo/stereo.cpp

Architectural code to fit `generateDepthMap()` from `Stereo_Vision.cpp` into the module `StereoModule`. Nothing interesting going on here.

2.6 vision/modules/stereo/samplecalib.cpp

Sample stereo calibration code straight from an OpenCV book that might be helpful while experimenting; note that this file does NOT interact with any other file.

2.7 vision/modules/stereo/selectimages.py

Short script that assists in handpicking images from two video streams for stereo calibration. Uses ffmpeg to sample images from left and right video streams, then facilitates deletion of images not needed once the key (best for calibration) images have been picked.

2.8 vision/tests/calib_stereo

If calibrating with images, set the type of the `capture_sources` forward camera to ``ImageDirectory'', `directory_0` to the directory with images for the left camera, and `directory_1` to the directory with images for the right camera. For example:

```
capture_sources: {
    forward: {
        n = 2
        type = "ImageDirectory";
        directory_0 = "/home/daryl/stereocalib/left";
        directory_1 = "/home/daryl/stereocalib/right";
        loop_0 = true;
        loop_1 = true;
    };
};
```

If calibrating with video, set the type of the `capture_sources` forward camera to ``Video'', `video_0` to the filepath for the left video, and `video_1` to the filepath for the right video. For example:

```
capture_sources: {
    forward: {
        n = 2
```

```
        type = "Video";
        video_0 = "/home/daryl/stereocalib/left.avi";
        video_1 = "/home/daryl/stereocalib/right.avi";
        shmframelock_forward_0 = true;
        shmframelock_forward_1 = true;
        undistort_0 = true;
        undistort_1 = true;
    };
};
```

`auv-visiond -g calib_stereo` will start running stereo calibration. If you are calibrating with video, you will need to change the frame number of the forward camera manually using `auv-shm-editor` or in a program by importing shm.

2.9 `vision/tests/stereo_test`

Set `video_0` to the filepath for the left video, and `video_1` to the filepath for the right video. For example:

```
capture_sources: {

    forward: {
        n = 2
        type = "Video";
        video_0 = "/home/daryl/stereologs/left.avi";
        video_1 = "/home/daryl/stereologs/right.avi";
        shmframelock_forward_0 = true;
        shmframelock_forward_1 = true;
        undistort_0 = true;
        undistort_1 = true;
    };
};
```

`auv-visiond -g stereo_test` will start running the stereo module.

3 Usage

3.1 Calibration

The first step is to gather images of a chessboard from many different angles; the chessboard must be visible in both the left and right camera in its

entirety, and big enough so that the calibration module recognizes the corners. Once this is done, run `selectimages.py`. Hand pick (preferably) 15+ images that look particularly good for calibration, then enter their numbers (i.e. 024, 080, 130, 320) into the script. Edit `calib_stereo` to include the directories of your images, then run `auv-visiond -g calib_stereo` to calibrate. If you would like to calibrate with a video, see section 2.8. Copy the output yml files into `vision/modules/stereo` and you've completed calibration

3.2 Vision Tuning

Try to set the vision parameters to anything that results in a better depth map. The following are the meaningful vision parameters.
P1_scale - The first parameter that controls disparity smoothness. Larger is smoother. `P1_scale` is the penalty on the disparity change by plus or minus 1 between neighbor pixels. A suggested value is `8*number_of_image_channels*SADWindowSize*SADWindowSize`.
P2_scale - The second parameter that controls disparity smoothness. Larger is smoother. `P2_scale` is the penalty on the disparity change by more than 1 between neighbor pixels. Note that `P2_scale` must be greater than `P1_scale`. A suggested value is `32*number_of_image_channels*SADWindowSize*SADWindowSize`.
SADWindowSize - Matched block size. Must be an odd number greater than or equal to 1. Normally should be between 3 and 11.
disp12MaxDiff - Maximum allowed difference in integer pixel units in the left-right disparity check. Set it to a negative value to disable it.
fullDP - Set it to `true` to run the full-scale two-pass dynamic programming stereo algorithm, which is very memory intensive.
minDisparity - Minimum possible disparity value. Normally zero but rectification algorithms can shift images so this parameter needs to be adjusted accordingly.
speckleRange - Maximum disparity variation within each connected component. If you do speckle filtering, (see below), set it to a positive value. Usually 1 or 2 will be fine - internally this number is multiplied by 16.
speckleWindowSize - Maximum size of smooth disparity regions to consider their noise speckles and invalidate. Set it to 0 to disable speckle filtering, otherwise it should usually be between 50 and 200.
uniquenessRatio - Margin in percentage by which the best (minimum) computed cost function value should "win" the second best value in order to consider the found match correct. Normally should be between 5 and 15.

4 Future Improvements

With more planning, it might be a more informative course if we wrote everything from scratch rather than used libraries