Optical Flow
AI Programming for Games
COMP09041

Paul Keir

March 18, 2025

Date Performed:	March 18, 2025
Instructor:	Paul Keir

Laboratory Objectives

Plug in your USB camera. If your camera is built into the lid of your laptop, there is nothing else to do. You may then be able to see it via the Windows Control Panel - Hardware and Sound - Devices and Printers. To help manage app permissions for your camera in Windows, visit the Microsoft site here.

1 Dense Optical Flow

Gunnar Farneback proposed an optical flow algorithm for use in dense optical flow tracking. The original paper from 2003 can be read here. We will look at two programs which use optical flow to detect motion; for example of a large object in front of the camera; e.g. your head.

- Open the `optflow-opencv` directory
- Use CMake to create a 64-bit Visual Studio 2022 solution within a new subdirectory called "build" as usual
- Open the project in Visual Studio
- Compile and run `fback`

  - ...observe how the size of some of the vectors change with your movements. Does every vector change with your movement?

- Compile and run the simplified variant simple_fback
  - This version displays (in the console window) the average flow vector from the uflow variable updated by calcOpticalFlowFarneback
  - Find in simple_fback.cpp where the total variable is calculated
  - Modify simple_fback.cpp so that with a significant left/right (head) movement. You can use the cv::norm function to obtain the *magnitude* of the cv::Point2f vector v. Store it in a new double variable called magnitude. You can either:
    a. Draw a circle using cv::circle (API is here). Inflate it like a balloon with each head movement; using magnitude and an accumulator variable; or
    b. Blur the video image using cv::GaussianBlur (API is here). The ksize parameter is usually a pair of numbers (e.g. {3,3} or {5,5}); and they **must** each be odd. Perhaps use (5*magnitude) for the ksize. The sigma parameter can be zero, but can perhaps also be (5*magnitude).
- Thought experiment: could the camera be held, and used as an analogue controller?

2 Sparse Optical Flow

Improve the Lukas-Kanade OpenCV example program by allowing it to *automatically* detect features.

- Compile and run the (lkdemo) project
- This program uses the Lukas-Kanade *sparse* optical flow method
- Click with the mouse to add features for the algorithm to track
- Look in the code at how the array of two std::vectors (named points) is used to hold feature points from both the current and previous frame
  - ...observe how the second std::vector is used in the initialisation call to goodFeaturesToTrack
  - Note also how the onMouse callback function sets the addRemovePt boolean, which is then used in main to trigger a call to push_back on points[1].

- The C++ `switch` statement in `main` shows that:
  - Pressing 'r' will add a fresh new set of feature points
  - Pressing 'c' will clear all of the feature points

Third Objective

Experiment with a set of OpenCV programs demonstrating a range of detector algorithms and descriptor matchers.

- Change directory into `opencv_essentials_book`
- Use CMake (again 64-bit) to create opencv_essentials.sln
- Compile & run `FASTDetector`. Which of the two images has more keypoints circled?
- Try with different threshold values in `FASTDetector.cpp`
  - n.b. The two SURF programs will not compile/run, as the version of OpenCV we are using was built with `OPENCV_ENABLE_NONFREE` undefined.