

Summary: In project two, an interesting problem was addressed: detecting lane markers in images from a forward looking camera. Use insights gained in your project two efforts, plus useful ideas from other approaches to explore an embedded system implementation.

An embedded system needs to support real-time operation. In the most constrained environment, this requires a programming and execution environment that allows task deadlines, priorities, etc. In this project, we will focus on a single aspect of this problem: frame processing time versus frame rate. While the sequences were collected at 60 frames/second (fps), your program can specify the frame rate based on your processing complexity.

While precisely computing a system's computation delay is difficult without the specified platform, we can compute our own estimate using a desktop or laptop running Linux. Embedded processors typically have lower clock rates and larger caches. But computers are generally running multiple processes and a glutinous operating system. The target system would directly decode frames from the imager chip. Our test platform must load and decode jpeg files from a slow disk. To compensate for this different environment, we will evaluate a "null" program (`P3-1-shell.c` with `DEBUG` set to 0) that only loads and decodes a sequence and writes a report shell. The execution time of this process, divided by the number of images, becomes a correction factor that can be used to estimate the actual frame processing time. The null delay on our computer is ~28 mS, but it may be longer or shorter on your machine. The `time` function in Linux is handy for measuring execution time of a command line program.

Once the processing time per frame is determined, it can be specified as the frame period command line argument to select frames that match that processing delay. Fast processing yields shorter delays between frames. Slower processing means longer delays.

New Materials: This project uses the same test sequences used in project two. New for this project is a collection of C source files, header files, and a new makefile in `P3-1.zip`. Unpack these files in a directory and add a link to the sequence directory. Rename `P3-1-shell.c` to `P3-1.c` and try making the package. When the `DEBUG` flag (defined in `utils.h`) is 1, the starting program will create a copy of the original image in the trial directory for the named input sequence. When `DEBUG` is cleared, input frames are retrieved, but no processing occurs. In both cases, a nominal report file is generated. Because lane markers require a variable number of points, they are represented by point structures. These objects are created and managed using routines in the report library.

All of your code should be included in your `P3-1.c` file. New files and changes made to distributed files will not be available when we evaluate your project. Also, you should use the same parameters for all sequences. Please do not assign individual parameters (thresholds, limits, etc.) for each sequence. You may define a vanishing point for each sequence since this would be a constant when the camera is rigidly installed in the automobile.

If you use generic code (e.g., for edge detection), please include a full citation in the accompanying comments and your write up. Please do not share code with other teams in the class or use specialized code for lane marker detection (that's your job).

When evaluating your code performance, it is best to turn off `-g` options and turn on `-O2` or `-O3`. While this has little effect on the starting program (because it is dominated by disk access), it should have a measurable effect on your code's execution time.

Final Submission: The final submission consists of two parts, both of which are prepared as a team:

1. **P3-1.c** (submit one source file per *team*): this file must contain all source code for solving the project task.
2. **P3-2.pdf** (submit one pdf file per *team*): Team Project Write-up, *co-authored by all members of the team*. It should contain the following:
 - a. Team member names and project number.
 - b. **Challenges Addressed:** summary of the technical issues that you focused on and the insights you had in addressing them.
 - c. **Approach:** Brief description of your approach, including workflow diagram and a description of the algorithms used.
 - d. **Results:** Summary of accuracy and performance results, including achieved frames rates, example lane marker annotated frames, and summary assessment.
 - e. **Discussion** of results, including when does your approach work well and what are its limitations.
 - f. **Real-time Embedded Issues:** discuss to what extent your approach is amenable to an embedded implementation in a real-time system (e.g., consider its computational/storage requirements, frame rate, application constraints or features you can take advantage of to realize an efficient implementation).
 - g. **Bibliography:** paper citations and URLs for related work and for algorithms or code you are using that you did not develop.
1. These submissions must be properly uploaded to the submission site before the scheduled due date, **9:00pm on Tuesday, 1 December 2009**.

Demo Day Deliverable: Each team should prepare an *eight-minute* presentation to share with the class during Demo Days, scheduled for the the **3rd and 5th of December 2009** (Note this is Wednesday and Friday). The presentation should summarize your approach, results, and insights.

Project Submission Instructions:

When you are ready to submit a file as your answer to a part of the project, use your web browser to go to the web site: <http://www.ece.gatech.edu/~scotty/8893/projects/index.html>.

Click on FILE UPLOAD which will take you to the ECE 8893 File Upload page. This contains a form for you to enter your GT ID (e.g., gtgeopb) and the name of the file to upload (e.g., C:\My Documents\Framework\GeorgeBurdell.jpg).

Double check that the name of the file uploaded is the one you intended to submit.

If you submit a file as the answer to part of the project and later you would like to submit an improved answer, you may submit the more recent version of the file. Only the most recent one will be graded. However, versions submitted after the due date will not be graded.