## Visual Odometry Engineering Notes

Kenneth Laws

10/10/2017

HAD – our product is HD Live map, that’s the only product.

This project is to evaluate performance and resource requirements for a visual, sequential image method odometer.

## Epic Story Planning

Odometer Sensor development

Acceptance Criteria:

Test plan for gathering/analyzing data

Identify/gather hardware

Determine compute requirements for real time integration

Phase I

Optical, lidar, ground radar

1. Identify and gather hardware
   1. Identify design parameters
      1. Precision
      2. Accuracy
      3. Operating conditions
      4. Generate draft test plan target design parameters
   2. Hardware selection (selection should include multiple candidates for evaluation)
      1. Sensor candidate selection (e.g., Frame rate, Resolution)
      2. Mechanical OTS
      3. Electrical OTS
      4. In house component design requirements
      5. Draft hardware selection of sensor prototype design plan
   3. Hardware BOM and procurement
2. Test plan for gathering/analyzing data
   1. Design for preliminary small evaluation data set
      1. Simple rig construction to collect small data set
      2. generate test plan for prelim data analysis
      3. conduct prelim experiment and evaluate results
   2. Design for larger (on vehicle) evaluation data set with ground truth
      1. Mechanical system design
      2. Electrical system design
      3. Outline test plan for data collection and data analysis
3. Determine compute requirements
   1. Algorithm selection
      1. Identify candidate algorithms
      2. Algorithm selection trade study
   2. Evaluate processing speed requirements
   3. Outline Implementation strategy
   4. Generate algorithm section of sensor prototype design plan

OBD System

1. Test plan for gathering/analyzing data
   1. Plan to gather velocity data from obd port
   2. Evaluate methods for obtaining obd data
   3. Define test plan for data analysis
   4. Identify best platform available data check sprint
   5. Identify platform independent available data
   6. Collection plan for gathering matchup data
   7. Outline test plan for data collection and data analysis
2. Hardware component selection
   1. Obd interface selection
   2. Data collection/storage hardware
3. Determine compute requirements
   1. Analysis pseudo code
   2. Identify candidate algorithms

Odometery Sensor Development

Phase II

1. Preliminary data (contrast hardware here)
   1. Basic data set (test plan laid out in phase I)
      1. Simple rig constructed to collect small number of images
      2. Constant height above ground
      3. Use candidate camera
   2. Algorithm validation
      1. Process images to compute delta x,y
      2. Accuracy estimate, find key parameters: angle, range in frame, image noise, low light limits, high/low contrast scene
2. Validation prototype
   1. Prototype data collection system
      1. Mechanical design
      2. Electrical design
      3. Deployment plan
   2. Algorithm validation/characterization (test plan laid out in phase I)
   3. Raw data quality assessment
   4. Data product quality assessment

OBD

Phase II

1. Collect candidate data set
2. Compare with highly accurate GPS matchup set
   1. Use GPS position data to evaluate position accuracy
   2. Use change in GPS velocity to evaluate forcing data (accelerator, brake) if available

## Hardware Selection

Pt Grey looks like a good place to start. We have history with the cameras and they seem to be an industry leader. Web site is well organized with tools for camera selection.

Interface: gig E is slower but allows for longer cable. Usb 3.1 is faster and cable length (10 m) is probably long enough for our needs.

Ptgrey

BFS-U3-13Y3C-C 170 fps 1280 x 1024 global shutter

BFLY-PGE-50A2C-CS 13 fps rolling shutter with global reset (used on localization rig)

FL3-U3-13S2C-CS 120 fps 1328 x 1048

Sony industrial – way too expensive

Imperex

B0610 640 x 480 109 fps

B1020 1024 x 1024 60-74 fps

Basler

acA2040-120um - Basler ace

120 fps 2048 x 1536 mono global shutter IMX252

(requested a quote for this camera from Basler)

The above short search indicates that the pt Grey is similar or better than other options. The Basler is an option if higher resolution is needed.

## Identify hardware & design parameters

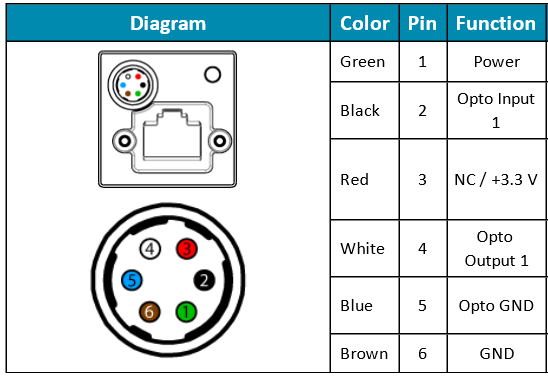
See: Visual Odometry Hardware and Design Parameters.docx

Downloaded

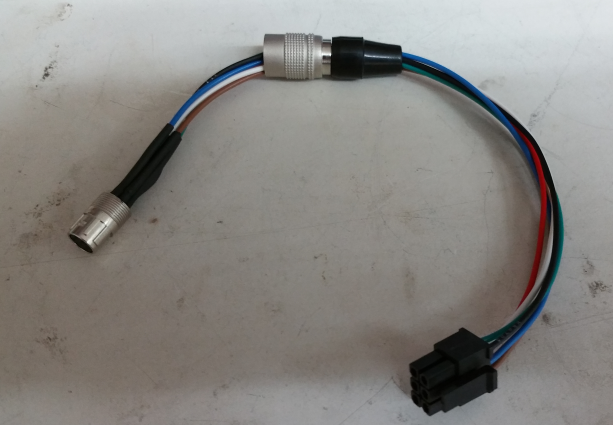
Installing FlyCap2viewer – done

Flycap viewer is versatile and user friendly. Can easily capture images and save to disk, set frame rate and other parameters.

## Driveway Tests



FLIR camera connector, bulkhead, face view.



In house cable corresponding to above diagram.

## Algorithm Evaluation

Design or select algorithm to reduce raw data to desired data products.  
Test and evaluate different methods,

Determine best data analysis algorithm/strategy for prototype experiments selected

See: Visual Odometry Algorithm Evaluation.docx

## Test Plan Design

11/1/2017

Developed plan for first phase prototype testing.

Visual Odometry Test Plan

11/15/2017

* Purchased, or sent out orders for all hardware, see test plan for table of hardware
* Started a folder for invoices
* Created a git repository for code development
* Got git shell working in windows by changing the default shell to cmd
* Created the git repo on the remote server and cloned it locally
* Modified matlab scripts to improve structure and speed

Time to process one image pair is about 0.29 seconds on my small laptop. This time does not include time to read image data from disk, but does include the time to perform fft and find position shift.

The time measured, using 5 repetitions and averaging, was 0.265 seconds to process one image pair, 1600 x 1200 image with 400 x 200 subframe

Changing subframe and image matrix size to be factors of 2^n helps a little

imageRes = [1600, 1200];

w = 256; % width of subframe

h = 128; % height of subframe

xPix = 1024; % matrix dimensions for image processing factor of 2^n

yPix = 1024;

reduces processing time to 0.20 seconds. Snr improves as the subframe gets larger. No improvement in speed was found making the subframe smaller than 256 x 128.

11/16

* Experimented with frame and image size and found best configuration, processing time about 0.2 sec
* Installed matlab on a T4 PC
* Created a git repo on the T4 PC and cloned the optical odometry repo
* Set up public key access for the repo
* Moved 2 images to a sample image folder and added to the repo
* Tested performance on the T4 PC and found only about a factor of 2 improvement over the laptop, 0.1 sec to process on image file pair

11/20/2017

Looked into speed improvement going from matlab to C. One writeup I found reports speed gains of a factor of 500 to 550. This example included plotting and handling large volumes of data. Another source stated an average improvement of about a factor of 50. Coincidentally, a factor of 50 is about what we need to process images at a rate of 150 fps.

11/22/2017

Evaluating frame rate with Bassler camera

11/28/2017

beginning to work on setting up processing in linux.

With Bassler camera, storage options are bitmap or tiff. Tiff has larger file size so going with bitmap.

Set up git repo on linux laptop

branch is

algo\_speed\_eval

Have captured a couple of sample images from Bassler camera (bmp format)

As of now, lens is 1/1.8 not matched to camera. I was expecting a focused image that just did not fill the frame. What I see is rounded corners without exposure, which is expected, but image is not in focus and not possible to focus with 1/1.8 lens. Hopefully this will be resolved when 1/1.2 lenses arrive.

Working on c-code development

able to compile hello world

want to read in a bitmap file.

I found this info from the web useful: <http://www.instesre.org/howto/BW_image/ReadingBitmaps.htm>

Bitmap files are separated into three or four sections, as shown in the table below.

|  |  |
| --- | --- |
| Section | Description |
| Header | Basic file information, 14 bytes |
| Image Information Header | Information about the image, 40 bytes |
| Color Information (optional) | Information about how the image encodes colors, a variable number of bytes if it's present |
| Image data | The actual image, a variable number of bytes |

14 bytes of the file header

|  |  |
| --- | --- |
| The uppercase characters BM, ASCII codes 66 and 77 expressed as a base-10 integer | 2 bytes |
| File size, bytes | 4 bytes |
| Two "reserved values" that are not needed | 2 bytes each |
| Offset to beginning of image data | 4 bytes |

In principle, it is possible to read these integer values directly by choosing appropriately declared data types that will automatically select the appropriate number of bytes. However, this may give unpredictable results with different C compilers, so I chose a more simple-minded, although probably more tedious solution: read the header one character at a time (characters occupy a single byte) and then use explicit typecasting to force C to interpret each character as an integer.

The two important values in the header are the file size, in bytes, and the offset from the beginning of the file to the start of the image itself, in bytes. The 14 bytes are stored in a character array. The file size starts at element 2 (in C, the first element in an array is 0, not 1) and is stored from low byte to high byte, left to right. I have assumed that even large .bmp file sizes will need no more than 3 bytes. I also declared many of the integer values as type long because the int data type may not be able to represent large file sizes. For the offset value, two characters should always be enough; this value starts at character 10.

|  |  |
| --- | --- |
| Header size, bytes (should be 40) | 4 bytes |
| Image width, pixels | 4 bytes |
| Image height, pixels | 4 bytes |
| Number of color planes | 2 bytes |
| Bits per pixel, 1 to 24 | 2 bytes |
| Compression, bytes (assumed 0) | 4 bytes |
| Image size, bytes | 4 bytes |
| X-resolution and y-resolution, pixels per meter | 4 bytes each |
| Number of colors and "important colors," bytes | 4 bytes each |

The useful values for this image are the width and height, 73 and 70 pixels, and the image size, 73•70 = 5110 pixels. The first byte in the file is at an offset of 0, so an image offset of 54 means that the image starts at the 55th byte in the file. The image is stored line-by-line. Each pixel requires 3 bytes. Each line has an end-of-line mark. So, the image part of the file equals 5110•3+70=15400. Adding the 54 bytes for the header records gives the file size, 15400+54=15454 byites.

Installing openblas-base on my ubuntu system

$ sudo apt update

$ apt search openblas

$ sudo apt install libopenblas-base

$ sudo update-alternatives --config libblas.so.3

Can’t find my path to the openblas library

I need to add something like this to my makefile:

gcc -o test test.c -I /your\_path/OpenBLAS/include/ -L/your\_path/OpenBLAS/lib -lopenblas

then include

#include <cblas.h>

see [https://github.com/xianyi/OpenBLAS/wiki/User-Manual#link-the-library](https://github.com/xianyi/OpenBLAS/wiki/User-Manual" \l "link-the-library)

12/4/2017

Setting up pbx machine for camera testing.

Installing pylon software

pylon-5.0.11.10914-x86\_64 – done

Run pylon viewer by typing: /opt/pylon5/bin/PylonViewerApp

Runs and powers camera OK.

At first, max frame rate was very low (with basler pylon sofware), about 25 fps. Changed: Auto Function Control, Exposure time upper limit to 800 usec and fps increased to 165 fps.

Now able to read images at a rate of 163 fps using OverlappedGrab.c

Frame rate with simpleGrab is only 44 fps. Clearly need to use multiple buffers to achieve desired frame rate.

12/5/2017

Something changed and max frame rate is 155 fps. Not able to find the set of parameters to get the last 10 fps

Exposure Auto, continuous

gain Auto – continuous

Exposure time upper limit – 6000

Auto function profile – Minimize gain

Letting this go for now and continuing to work on code to read a set of images and write them to disk.

Generating file names: I need to compute the offset between strftime() and local\_time() in usec, probably also should put the time string in the header of the bitmap file.

Without the ultrafast drive, we can write 20,000 files to disk with an effective frame rate of 138.9 fps. With 30000 files, the rate was 138.2 fps, however, a large number of files had zero size. This appears due to running out of disk space rather than a write speed problem.

Now time to install the M2 drive and redo evaluation with a larger number of files.

Installing the Samsung 960 Pro drive was difficult. The drive is of type NVMe SSD and not recognized by the ASUS X99 Extreme4.

Tried updating the bios to latest version but this did not help.

Eventually found an ASROCK forum post related to the problem that revealed that the bios does not see the drive but the operating system can see it anyway. The disk was able to be formatted by running the utility “disk” as admin

sudo gnome-disks. Then it was possible to format the volume.

Using this high speed volume, am able to capture 40,000 frames at an average rate of 156.2 fps. I believe frame rate in this case is limited by the camera parameter settings not interface or storage speed limitations.

Next up, modify my c program to:

include OverlappedGrab functions,

Set camera parameters

Do timed data collection