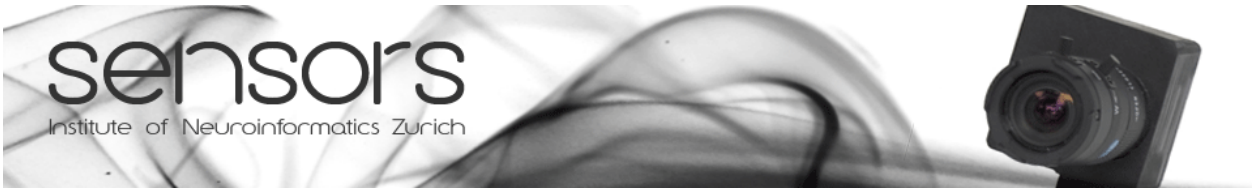


DAVIS Driving Dataset 2017 (DDD17)

Jonathan Binas, Daniel Neil, Shih-Chii Liu, Yuhuang Hu, Tobi Delbruck
June 2017
Web: This document | analytics
Link to source document

DDD17 is superseded by DDD20. Please go to that page for a greatly expanded dataset with improved code.

DDD17 is the first public end-to-end training dataset of automotive driving using a DAVIS event+frame camera. It includes car data such as steering, throttle, GPS, etc. It can be used to evaluate the fusion of frame and event data for automobile driving applications.

See more Inst. of Neuroinformatics Sensors Group datasets here.

Change history

- *June 8, 2018, added info about lens, FOV, and lack of camera calibration*
- *Sept 4, 2017: added wasabi download option, added DAVIS description*
- *Aug 19, 2017, added more tips on recording*
- *June 25, 2017 added more detailed dataset description*
- *June 21, 2017 added initial info on recording new data*
- *June 5, 2017 created*

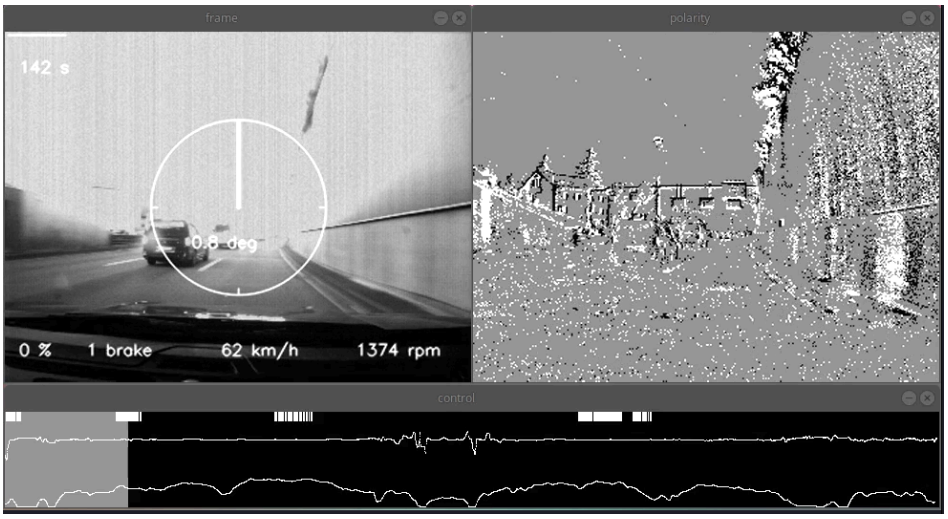
License

Table of contents

This getting-started guide accompanies the data used in the ICML workshop paper below. Publications using this data should cite the following papers

1. The two possible citations for DDD17 are
    a. Binas, J., Neil, D., Liu, S.-C., and Delbruck, T. (2017). DDD17: End-To-End DAVIS Driving Dataset. arXiv:1711.01458 [cs]. Available at: http://arxiv.org/abs/1711.01458 . (available online)
    b. Binas, J., Neil, D., Liu, S.-C., and Delbruck, T. (2017). DDD17: End-To-End DAVIS Driving Dataset. in ICML'17 Workshop on Machine Learning for Autonomous Vehicles (MLAV 2017) (Sydney, Australia). Available at: https://openreview.net/forum?id=HkehpKVG-&noteId=HkehpKVG- .(only available via openreviews)
2. The sensor used for DDD17 is the DAVIS, which is based on the seminal DVS paper
    a. Berner, Raphael, Christian Brandli, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. 2014. "A 240x180 10mW 12us Latency Sparse-Output Vision Sensor for Mobile Applications" In IEEE J. Solid State Circuits. 49(10) p. 2333-2341 10.1109/JSSC.2014.2342715 . Get PDF here.
    b. Lichtsteiner, Patrick, Christoph Posch, and Tobi Delbruck. 2008. "A 128 X 128 120 dB 15 Ms Latency Asynchronous Temporal Contrast Vision Sensor ." *IEEE Journal of Solid-State Circuits* 43 (2): p. 566–576. doi:10.1109/JSSC.2007.914337. Get PDF here.

ICML Workshop page: https://sites.google.com/site/ml4autovehicles2017/home



Link to DDD17 samples.mp4.
Screen shot of view.py output from one DDD17 recording

Contact

For questions related to the use of this dataset or the associated tools, please write jbinas@ini.ethz.ch, daniel.l.neil@gmail.com , yuhuang.hu@ini.uzh.ch or tobi@ini.uzh.ch.

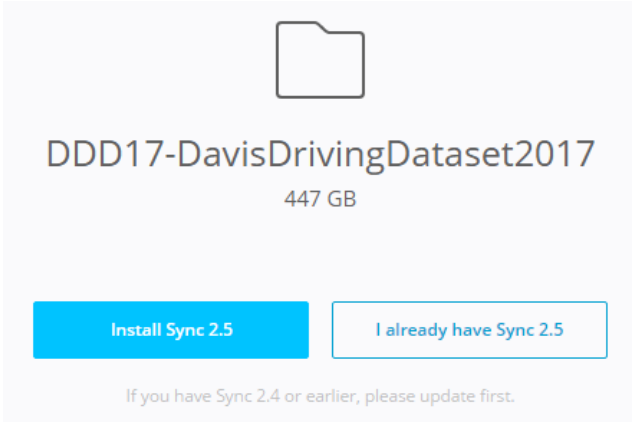
Getting the dataset

There are 2 methods to get the data

1. Use Resilio Sync (a private bittorrent protocol) to get the data. Using this method spreads the bandwidth and doesn't cost us anything. Using Resilio Sync, the more people that download (and continue sharing), the faster the download will be for everyone else.
2. (We have disabled this method since the costs have been too high. Contact us if you cannot use ResilioSync method.)
3. md5 checksums of all the files are available in this file.

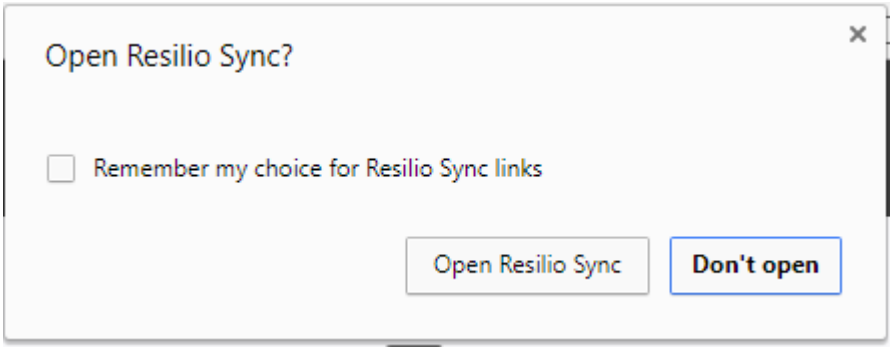
1. Using Resilio Sync

Use Resilio Sync to get the DDD17 dataset, from Resilio Sync DDD17 dataset. The complete dataset download is currently about 447GB. Clicking on the link above will pop up this result:

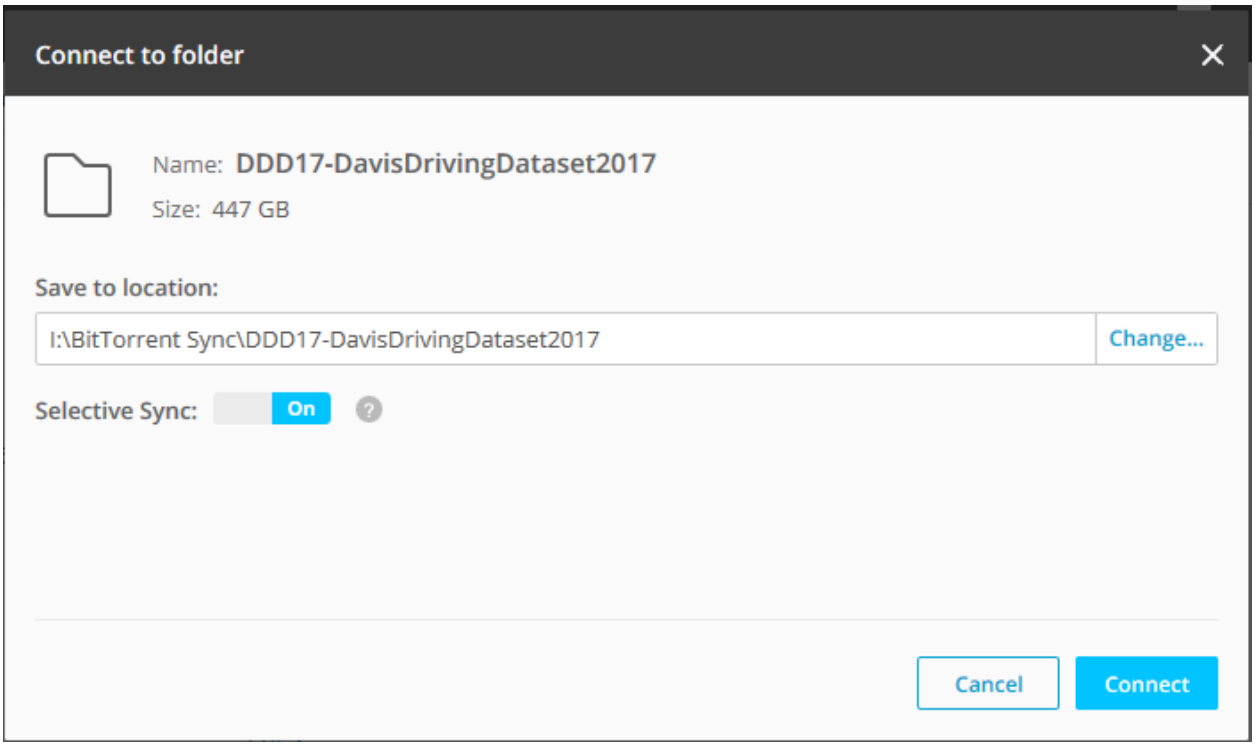


Resilio Sync link result

On Windows, clicking the “I already have Sync” produces this result:

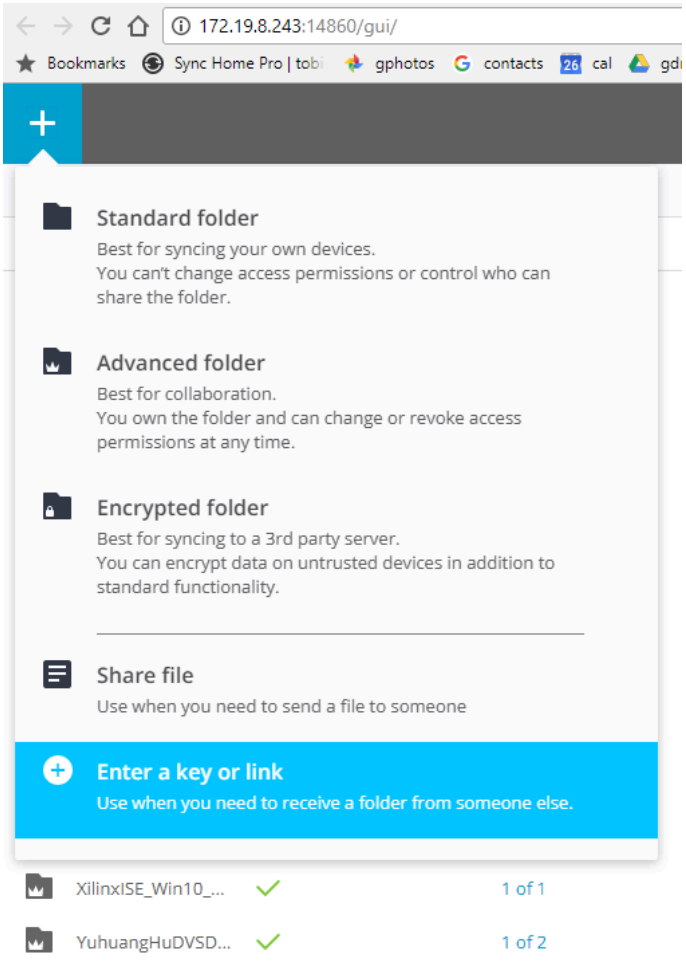


Selecting “Open Resilio Sync” results in this dialog:



Once you install and run Resilio Sync, you can select the option to “Selective Sync” to synchronize only part of the data.

On linux, you should copy the link above, and paste into the web GUI interface for sync by selecting the + button and selecting the “Enter a key or link” item.



Note: [for linux firefox users, see this post for help to access the data, since firefox will not recognize the btsync protocol].

Site hosting this and other data: http://sensors.ini.uzh.ch/databases.html

2. Download via a set of URLs from wasabi cloud storage

We have disabled these links for now. Please contact us for access if you cannot get ResilioSync method to work.

Dataset contents

Camera

The camera used for these recordings was an advanced 346x260 pixel DAVIS vision sensor with 18.5um pixels. The DAVIS outputs conventional global shutter gray scale image frames and dynamic vision sensor (**DVS**) events. The DAVIS also includes its own inertial measurement unit (**IMU**) that measures camera rotation and acceleration.

Optics

A fixed focal length lens (C-mount, 6mm) was used for all recordings, providing a horizontal field of view of 56◦ . Unfortunately, there is no camera calibration. See sensor alignment for details on camera mounting.

Vehicle data

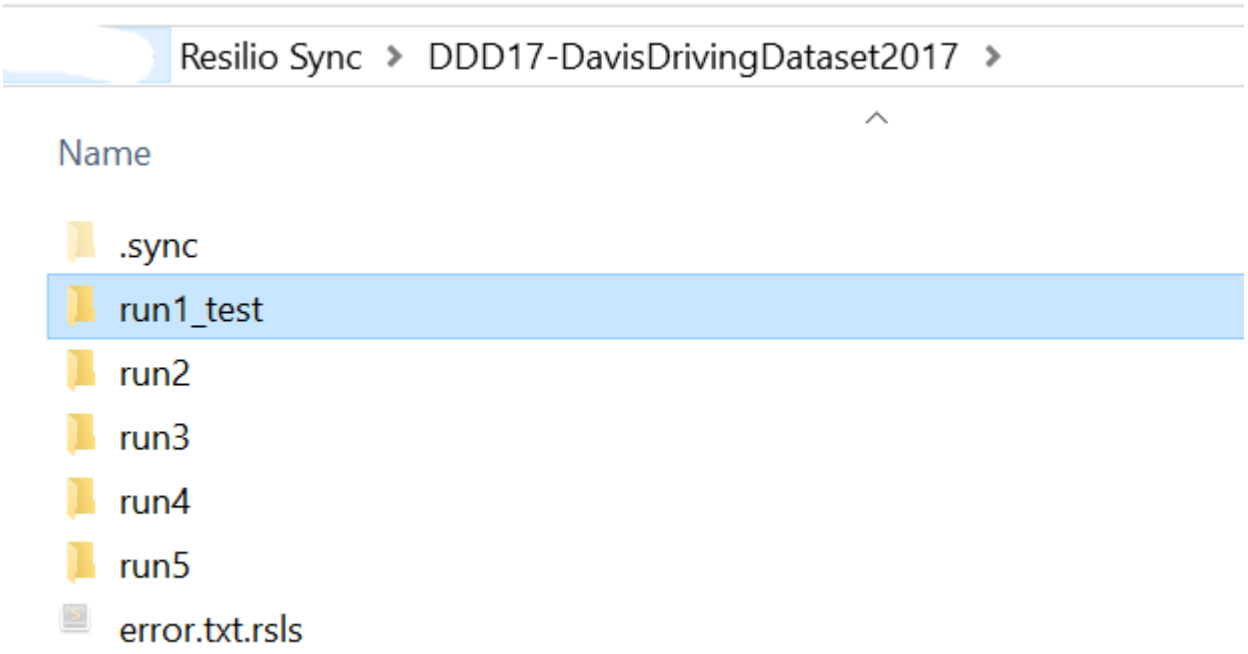
The following vehicle-related variables have been recorded together with the visual DAVIS data.

Variable name	**Range**	**Units**	**Frequency (approx.)**
steering_wheel_angle	-600 to +600	degrees	max 10 Hz
torque_at_transmission	-500 to 1500	Nm	max 10 Hz
engine_speed	0 to 16382	RPM	max 10 Hz
vehicle_speed	0 to 655[1]	km/h	max 10 Hz
accelerator_pedal_position	0 to 100	%	max 10 Hz
parking_brake_status	boolean (1 = engaged)		1Hz and immediately on change
brake_pedal_status	boolean (1 = pressed)		1Hz and immediately on change
transmission_gear_position	States: -1, 0, 1, 2, 3, 4, 5, 6, 7, 8[2]		1Hz and immediately on change
odometer	0 to 16777214	km	max 10 Hz
ignition_status	0, 1, 2, 3[3]		1Hz and immediately on change
fuel_level	0 to 100	%	max 2 Hz
fuel_consumed_since_restart	0 to 4294967295.0	L	max 10 Hz
headlamp_status	boolean (1 = on)		1Hz and immediately on change
high_beam_status	boolean (1 = on)		1Hz and immediately on change
windshield_wiper_status	boolean (1 = on)		1Hz and immediately on change
latitude	-89 to 89	degrees	max 1 Hz
longitude	-179 to 179	degrees	max 1 Hz

The dataset is divided into various recording files, generated under different weather, driving, and street conditions.

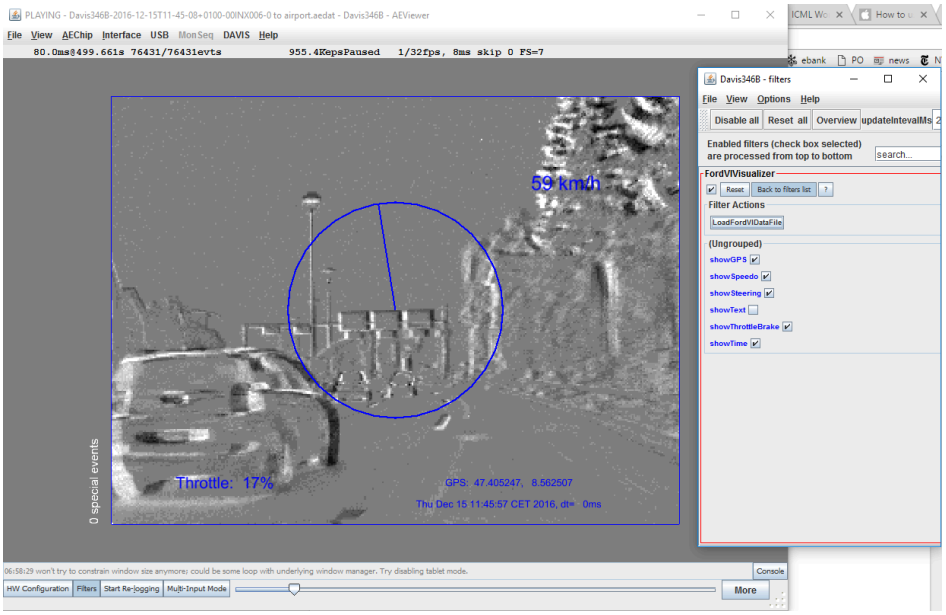
The spreadsheet DVS Driving Dataset 2017 (DDD17) description describes each recording. The overall folder structure is shown below.

[1] Speed is always positive (even in reverse), use transmission gear position to figure out direction.
[2] -1 corresponds to reverse, 0 to neutral, all other numbers to the respective gears
[3] 1: off; 2: accessory; 2: run; 3: start.



DDD17 folder structure

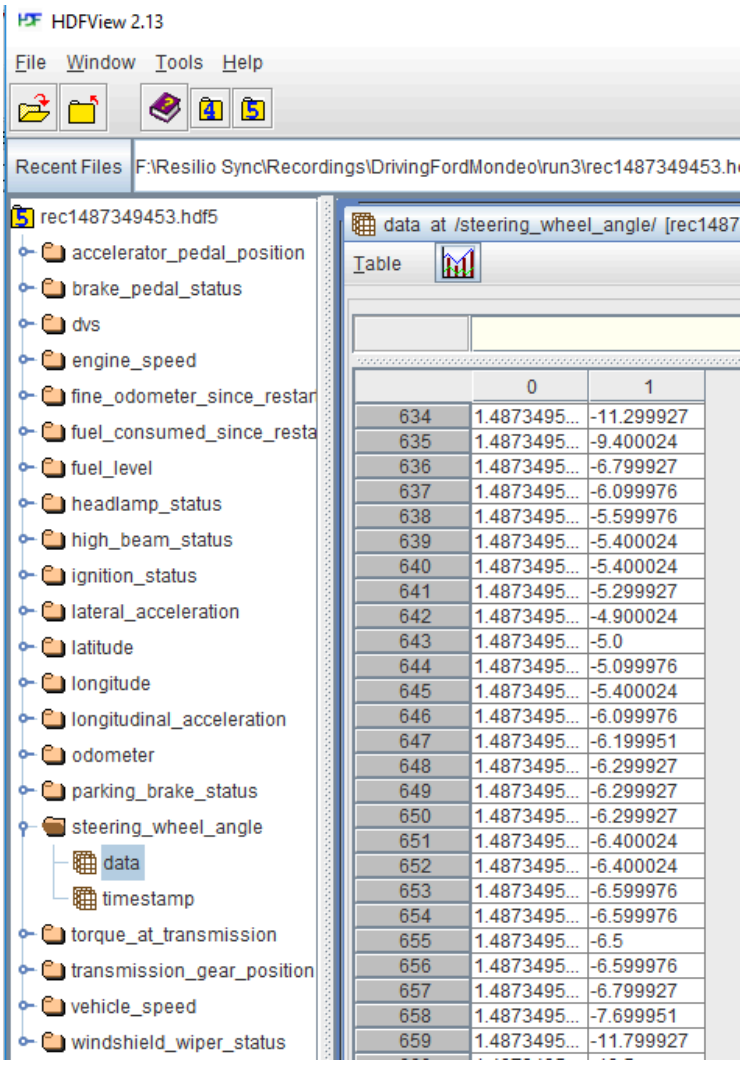
Several preliminary recordings in *run1_test* were done using jAER and these recordings are supplied in AER-DAT2.0 format as aedat files. The associated OpenXC data are supplied as .dat files. These preliminary recordings can be played in jAER's AEViewer using the AEChip class *eu.seebetter.ini.chips.davis.Davis346B* and the EventFilter *ch.unizh.ini.jaer.projects.e2edriving.FordVIVisualizer.* A screen shot is shown below. The *aedat* file should be opened **after** the *.dat* file is loaded using the *LoadFordVIDataFile* button.



Screen shot showing 80ms DVS frame and associated OpenXC data from the recording "Davis346B-2016-12-15T11-45-08+0100-00INX006-0 to airport.aedat" with OpenXC data file "FordVI-2016-12-15-rec01-to-airport-and-back.dat"

HDF5 file contents

We recommend HDFView for exploring the file container structure. Here is the steering wheel data for one file



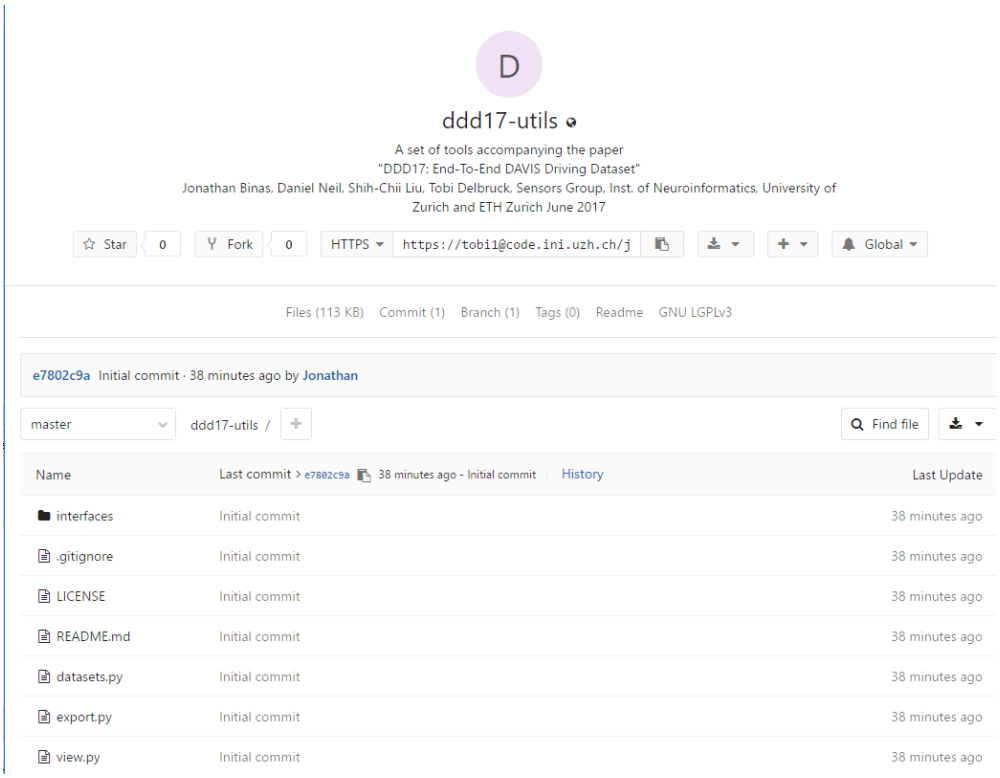
HDFView inspecting steering wheel angle data for one recording

Note: attempting to explore the *dvs* data throws an exception in HDFView, but this is expected from the variable length cAER data that is contained in the dvs container.

Getting the software

We provide python software for viewing and exporting the data. Code and further instructions are available at https://github.com/SensorsINI/ddd20-utils (DDD17 version: https://code.ini.uzh.ch/jbinas/ddd17-utils). Clone it with

```
git clone https://github.com/SensorsINI/ddd20-utils
```



See the README.md for python library dependencies (opencv-python and h5py).
Useful command line

```
pip install opencv-python h5py
```

Branches

- *master* branch is the main one.
- *cleanexit* branch is the current working copy for collecting new data. It has substantial modifications to improve stability under error conditions to prevent corrupted HDF5 files. It also adds a (hardcoded at present) option to view and record to rotate the displays by 180 degrees.

Usage

The following examples are from a terminal/console command prompt.

Note for python newbies regarding PATH

ddd17-utils should be on the PATH to enable python to find the scripts like *view.py*. It should also be on PYTHONPATH so that the scripts can import the libraries. Examples from .bashrc

```
export PATH="~/git/ddd17-utils:~/bin:~/jdk1.8.0_131/jre/bin:$PATH"
export PYTHONPATH=$PYTHONPATH:~/git/ddd17-utils
```

If the script is on the PATH, and it is executable, then a leading python is not needed to launch the script.

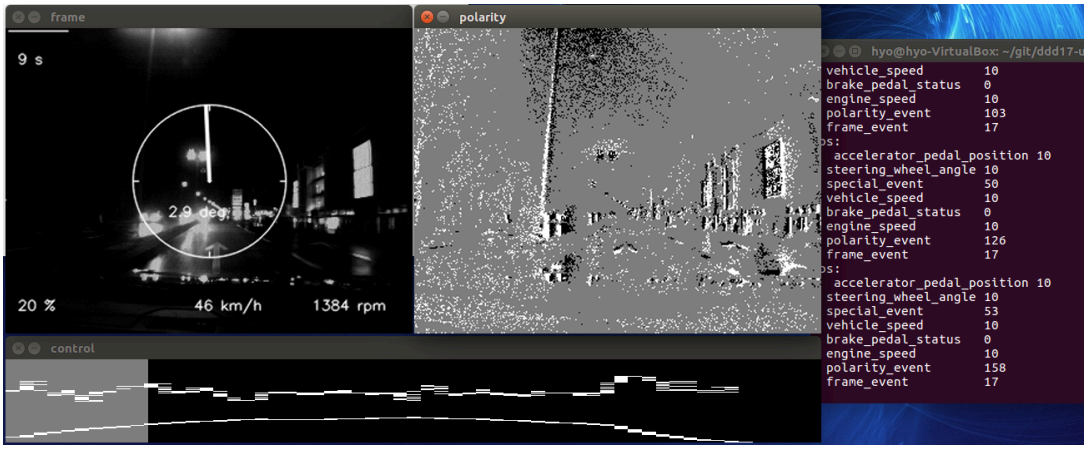
Play a file from the beginning

$ view.py <recorded_file.hdf5>

Example

Assumes data is stored in /media/driving

$ **view.py /media/driving/run3/rec1487355025.hdf5**



Play a file, starting at X percent

$ view.py <recorded_file.hdf5> X%

Play a file starting at second X

$ view.py <recorded_file.hdf5> Xs

Rotating 180 degrees during playback/recording

There is an option rotate180 in view.py that applies a 180 degree rotation to the APS and DVS displays. It must currently be set by editing line 515 of view.py to set variable r180=True.

Notes

1. If you get a gtk error when trying to run view.py, then you may need to use a different opencv or download/install opencv yourself, since pip install opencv-python may not install a working opencv. One instance of downloading/building/installing is

```
wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.1.0.zip
unzip opencv.zip
cd opencv-3.1.0/
mkdir build
cd build/
cmake -D INSTALL_PYTHON_EXAMPLES=ON  -D BUILD_EXAMPLES=ON ..
make -j 4
make install
```

Export data for training a network (reduce data to frame-based representation):

export.py creates another hdf5 file, containing the frame-based data.

$ export.py [-h] [--tstart TSTART] [--tstop TSTOP] [--binsize BINSIZE]
[--update_prog_every UPDATE_PROG_EVERY]
[--export_aps EXPORT_APS] [--export_dvs EXPORT_DVS]
[--out_file OUT_FILE]
filename

1. All times are in seconds.
2. If BINSIZE is positive, events will be binned into time-slices of BINSIZE seconds (typical values are tens of milliseconds, e.g. 0.050 for 50 ms). Default binsize is 0.1 seconds.
3. If BINSIZE is a negative integer, frames will be generated every abs(BINSIZE) events (e.g. BINSIZE = -5000 for a constant event count of 5000).
4. EXPORT_APS and EXPORT_DVS are integer values, 1 for true (default is 1). They determine if DVS and/or APS frames are generated.
5. UPDATE_PROG_EVERY is the progress bar update interval (in percentage points), default is 0.01.
6. If -out_file is not supplied, then it takes the original filename and adds _export (i.e. rec01.hdf5 -> rec01_export.hdf5)

Example: default export

Export a file with default settings

$ **cd /media/sf_ddd17/run3**
$ **export.py rec1487417411.hdf5**
accelerator_pedal_position final block: 161
parking_brake_status final block: 32
…..
[DEBUG] sys_ts/tstop 1487419509.07 1487419509.07

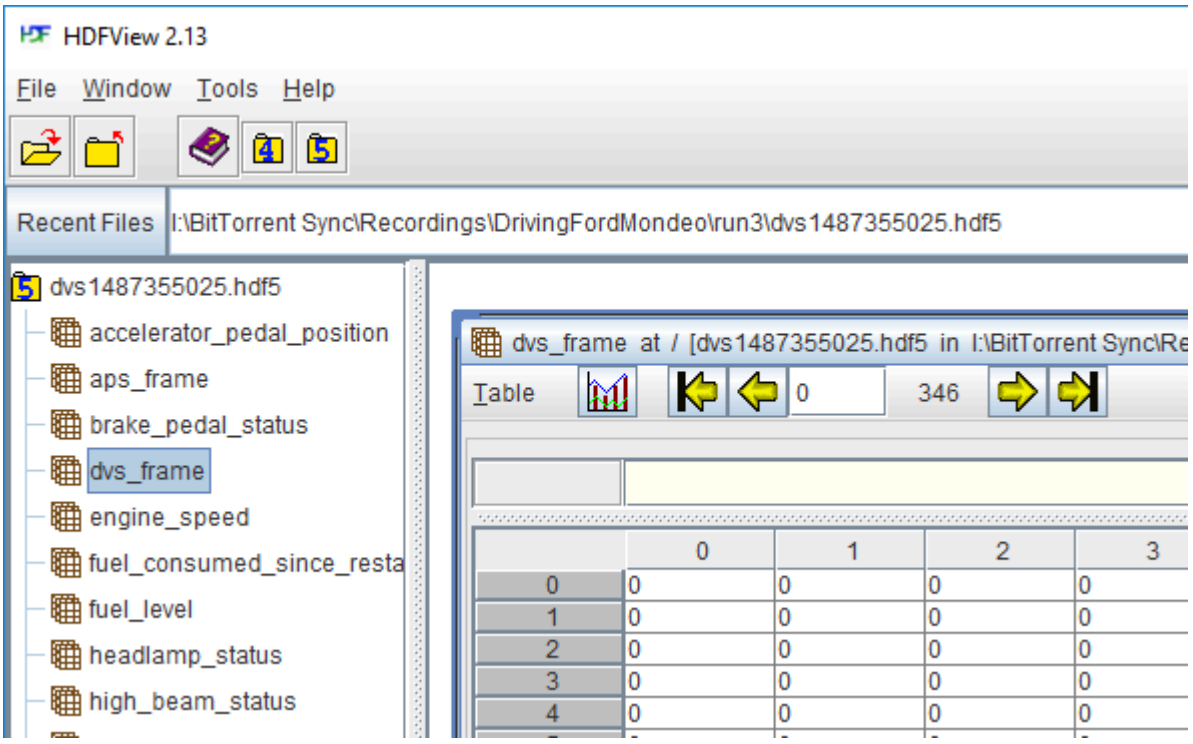
closed output file
[DEBUG] output done
[DEBUG] merger done
[DEBUG] stream joined
[DEBUG] merger joined
Finished. Wrote 1443.0MiB to rec1487417411_export.hdf5.

Example: make HDF5 file with 10k event frames

$ **export.py --binsize -10000 --update_prog_every 10 --export_aps 0 --out_file**
/media/driving/run3/dvs1487355025.hdf5 /media/driving/run3/rec1487355025.hdf5

accelerator_pedal_position final block: 3
parking_brake_status final block: 0
...
searching parking_brake_status 1487355026840441
 ...
closed input file
 ...
Finished. Wrote 168.0MiB to /media/driving/run3/dvs1487355025.hdf5.

The output *dvs_frame* is a 3-tensor, with number_of_frames x width x height elements.



Example networks and tools

You should be able to access ***dvs_frames*** with h5py, and you can call .dtype on it to determine datatype or .size to get its multidimensional shape.
Before you call export, however, there won't be a dvs_frames entry.

We recommend looking at the ddd17-utils ipython notebooks Data Visualization.ipynb and Model Evaluation and Movie Render.ipynb. These have an example of how to to view the data and render the movies.

To start the notebook, type “ipython notebook” to a terminal. You can pass in the notebook name as the third argument.

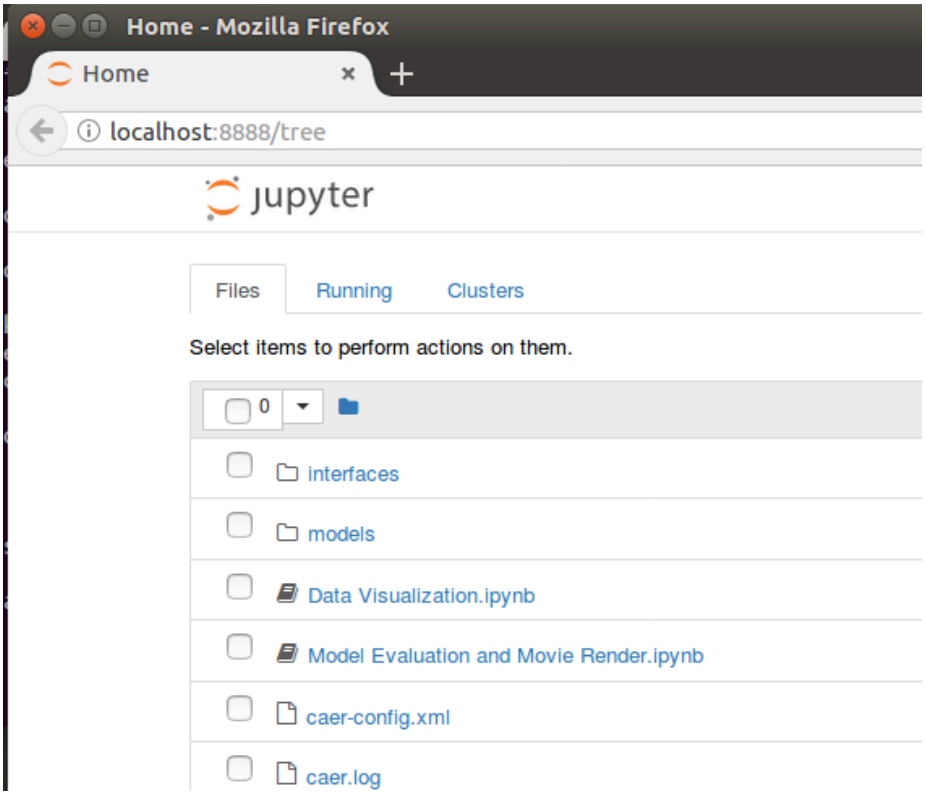
1. ***Data Visualization.ipynb*** walks the user through loading the data and visualizing it
2. ***Model Evaluation and Movie Creation*** loads a trained network, plots the results, and renders one of those driving movies using whatever network is supplied
3. if you want to see how it’s done, check **process_data.sh**, which calls export, then preprocesses the data to resize it to something reasonable, then trains and tests a network on whatever datafiles are listed in the work list
4. The latest version supports multiple hdf5 input files, Check ***multiprocess_data.sh*** to see the script for running all the data from day 5.

Troubleshooting

$ **ipython notebook**
Could not start notebook. Please install ipython-notebook
$ **pip install ipython-notebook**
Collecting ipython-notebook
  Could not find a version that satisfies the requirement ipython-notebook (from versions:)
No matching distribution found for ipython-notebook
$ **sudo pip install --upgrade ipython[notebook]**
Collecting ipython[notebook]
  Downloading ipython-5.5.0-py2-none-any.whl (758kB)

100% |████████████████████████████████| 768kB 950kB/s
Collecting pickleshare (from ipython[notebook])
…..
Successfully uninstalled ipython-2.4.1
Successfully installed backports.shutil-get-terminal-size-1.0.0 bleach-2.1.1 certifi-2017.7.27.1 decorator-4.1.2 entrypoints-0.2.3 functools32-3.2.3.post2 html5lib-1.0b10 ipykernel-4.6.1 ipython-5.5.0 ipython-genutils-0.2.0 ipywidgets-7.0.3 jsonschema-2.6.0 jupyter-client-5.1.0 jupyter-core-4.3.0 mistune-0.7.4 nbconvert-5.3.1 nbformat-4.4.0 notebook-5.2.0 pandocfilters-1.4.2 pathlib2-2.3.0 pexpect-4.2.1 pickleshare-0.7.4 prompt-toolkit-1.0.15 ptyprocess-0.5.2 python-dateutil-2.6.1 pyzmq-16.0.2 scandir-1.6 setuptools-36.6.0 six-1.11.0 terminado-0.6 testpath-0.3.1 tornado-4.5.2 traitlets-4.3.2 wcwidth-0.1.7 webencodings-0.5.1 widgetsnbextension-3.0.5

$ **ipython notebook**



Data Visualization.ipynb

In [3]:
```
import h5py
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

filename = '/media/sf_ddd17/run3/rec1487417411_export.hdf5'
```

In [4]:
```
dataset = h5py.File(filename, 'r')
print('Dataset contains:')
for key in dataset.keys():
    if np.prod(dataset[key].shape)<1e8:
        data = np.array(dataset[key])
    else:
        data = np.array(dataset[key][:1000])
    print('  {}, Min: {}, Mean: {}, Max: {}, size: {}'.format(
        key, np.min(data), np.mean(data), np.max(data), dataset[key].shape))
```

```
Dataset contains:
  accelerator_pedal_position, Min: 0.0, Mean: 15.0204148682, Max: 37.900002, size: (22528,)
  aps_frame, Min: 0, Mean: 102.289391196, Max: 212, size: (22528, 260, 346)
  brake_pedal_status, Min: 0.0, Mean: 0.0243252840909, Max: 1.0, size: (22528,)
  dvs_frame, Min: -61, Mean: 0.0235965095598, Max: 17, size: (22528, 260, 346)
  engine_speed, Min: 0.0, Mean: 1586.61115057, Max: 2384.0, size: (22528,)
  fuel_consumed_since_restart, Min: 0.0, Mean: 1.27380636328, Max: 2.802054, size: (22528,)
  fuel_level, Min: 0.0, Mean: 72.019624572, Max: 81.522003, size: (22528,)
  headlamp_status, Min: 0.0, Mean: 0.0587269176136, Max: 1.0, size: (22528,)
  high_beam_status, Min: 0.0, Mean: 0.0, Max: 0.0, size: (22528,)
  ignition_status, Min: 0.0, Mean: 1.86052911932, Max: 2.0, size: (22528,)
  latitude, Min: 0.0, Mean: 44.1569197756, Max: 47.532154, size: (22528,)
  longitude, Min: 0.0, Mean: 8.11625923522, Max: 8.997782, size: (22528,)
  odometer, Min: 0.0, Mean: 33588.4205959, Max: 36129.472656, size: (22528,)
  parking_brake_status, Min: 0.0, Mean: 0.0, Max: 0.0, size: (22528,)
  steering_wheel_angle, Min: -111.299927, Mean: -0.122038639338, Max: 237.400024, size: (22528,)
  timestamp, Min: 0.0, Mean: 1974.98317785, Max: 3170.680011, size: (22528,)
  torque_at_transmission, Min: -57.0, Mean: 77.8167169744, Max: 326.0, size: (22528,)
  transmission_gear_position, Min: 0.0, Mean: 5.18829900568, Max: 6.0, size: (22528,)
  vehicle_speed, Min: 0.0, Mean: 82.4259858343, Max: 123.469994, size: (22528,)
  windshield wiper status, Min: 0.0, Mean: 0.0, Max: 0.0, size: (22528,)
```

Recording new data

In addition to software for playback and export of the DDD17 dataset files, we provide tools for recording new data. The recording framework can be modified / extended to record data from arbitrary sources by creating a new ‘interface’ class.

The following instructions focus on recording visual data from a DVS/DAVIS camera and vehicle diagnostic/control data from an OpenXC vehicle interface.

Requirements

Hardware

- A DAVIS event camera with AER output
- An OpenXC-compatible vehicle interface
- A compatible vehicle - see this useful spreadsheet
- Tested on linux Ubuntu 16 LTS. It also runs in a virtualbox linux guest under windows, but graphics slows it down probably too much for recording.

Software

For a virgin Ubuntu 16 LTS system, the script setup.sh can help guide specific steps needed for installation for recording.

In addition to the software required for viewing / exporting (opencv, opencv python bindings, hdf5 libraries, h5py; see Getting the software), a working installation of cAER is required, as well as the openxc tools for readout. Specifically,

- libcaer needs to be installed and functional (see libcaer README for installation instructions)
- cAER should be built with ENABLE_NETWORK_OUTPUT=1 (see cAER README for details and additional software dependencies)
- OpenXC python tools can be installed through "**pip install openxc**" (note that libusb needs to be present on the system for OpenXC to work.)

Firmware

- Firmware needs to be installed to the OpenXC interface for the specific vehicle, or the emulation firmware can be used for testing a setup. Vehicle firmware requires registration with Ford as developer at https://developer.ford.com/register/ . Then firmware is available at https://developer.ford.com/pages/openxc
- Open firmware repo (only for testing, not for actual vehicles) is https://github.com/openxc/vi-firmware . Latest 7.2 Ford release is here. See http://openxcplatform.com/vehicle-interface/firmware.html
- Extract the emulation firmware from zip file; it is vi-emulator-firmware-FORDBOARD-ctv7.2.0.bin
- In case of the Ford reference interface, firmware is installed by holding interface in reset with paperclip in reset button hole while USB is plugged in. The interface appears as a flash drive. The firmware.bin file can be replaced by one specific for vehicle or test emulation.

Recording checklist

1. Is the correct car firmware loaded to interface (and not emulator used for debugging)?
2. In terminal, run caer-bin to start camera
3. In 2nd terminal, cd to target directory for recording files, and ensure it is writeable
4. In 2nd terminal, run record.py to start recording.
    a. If it dies complaining that Ford VI USB interface is not present,
        i. is the car ignition turned on?
        ii. Does some other process still have the interface open?
        iii. Is the cable plugged in? Sometimes it is necessary to unplug both ends of interface, plug in USB to computer, then plug into interface to car.
5. Hit enter to start recording, enter to stop recording.
6. Check recording with view.py rec, where file is the la
7. test recorded HDF5 file with view.py
8.

Usage record.py

python record.py
    - records to a new file recXXX.hdf5
python record.py /media/sf_recordings
    - records to a new file recXXX.hdf5 in the folder /media/sf_recordings

- Run cAER (cd <caer>; ./caer-bin) to capture DVS visual data
- From ddd17-utils, test interface with **python interfaces/caer.py**
- Run ***python record.py***. A preview of the captured data is displayed, this can be used to align the camera etc. Press enter to start writing to a file.
- Terminate the recording with ctrl-c.
- You might prefer to run "**./record.py**", so that in the event of a crash it is not needed to "killall record.py". You may need to "chmod +x record.py"
- A single argument is allowed that specifies the directory to save hdf5 file to

Troubleshooting

You can test DAVIS with
python interfaces/caer.py,
and test OpenXC with
python interfaces/oxc.py
You can also test the interface using
openxc-dashboard.
This utility is installed with openxc.

If OpenXC interface appears in lsusb but cannot be opened, add udev rules:
Make a new file
/etc/udev/rules.d/98-fordvi.rules,
in this file put the line
SUBSYSTEM=="usb", ATTR{idVendor}=="1bc4", ATTR{idProduct}=="0001", MODE="666"
Then replug the interface.

On a virgin install, you may need to install rules for the DAVIS camera as well. See inilabs documentation for thesbe e rules.

Sensor alignment

1. Most files in the run2 directory contain the 'alignment device'.
2. Try rec1487333001.hdf5 or rec1487337800.hdf5, for instance (very dark, auto exposure didn't exist back then...)
3. You can skip to a brighter section by clicking the timeline window of the viewer (third window)

APS exposure control

Exposure control is done in interfaces/caer.py.ExposureCtl. Options should be checked to be correct for camera mounting, i.e. if camera is mounted upside down, then the options

```
  exposure = ExposureCtl(cutoff_bot=80, cutoff_top=0) # set for upside down camera where sky will be at bottom
```

at line 311 of caer.py are appropriate

Detecting corrupted recordings

The utility h5check can be used to detect corrupted hdf5 files. (These can occur any time the HDF5 file is not properly closed). Once h5check is installed, then use this shell command to move corrupted files to a subfolder “corrupted”, first make the folder with “mkdir corrupted”, then

```
 for i in rec*.hdf5; do h5check $i || mv $i corruupted/; done
```

H5check is not installed by default. To obtain it see https://support.hdfgroup.org/products/hdf5_tools/h5check.html

Useful aliases for recording

These aliases and environment variables can be modified and put in your .bashrc file

```
#DATA=/media/tobi/MYLINUXLIVE/data
#DATA='/media/tobi/F0E885B6E8857C1A/Users/Tobi Delbruck/Resilio Sync/DDD17-DavisDrivingDataset2017/fordfocus'
```

```
DATA='/mnt/F0E885B6E8857C1A/Users/Tobi Delbruck/Resilio
Sync/DDD17-DavisDrivingDataset2017/fordfocus'

PATH="/home/tobi/setups/ddd17-utils:/home/tobi/bin:/home/tobi/jdk1.8.0_131/jr
e/bin:$PATH"
#PATH="/home/tobi/setups/ford:$PATH"

alias r="killall record.py;record.py \"$DATA\"" # starts a recording to the
DATA folder
alias cdd='cd "$DATA"' # cd to recordings folder
alias cdc='cd ~/setups/ddd17-utils' # cd to code folder
bind TAB:menu-complete # use uparrow for history completion, allows c^

alias va='for i in rec*.hdf5; do  view.py $i; done' # view all recordings in
current folder
alias v='view.py' # alias for view
alias vl='view.py `ls -t rec* | head -1`' # view last recording in current
folder
alias d='pushd'
alias u='popd'
```

Useful commands

Produce CSV file of recordings

```
find . -name rec* -printf '"%P","%Tc","%TD","%TT","%s",\n' > listing.csv
```