DF-VO Documentation

Release latest

Huangying Zhan

Mar 03, 2021

INTRODUCTION

CHAPTER

ONE

INTRODUCTION

This repo implements the system described in the ICRA-2020 paper and the extended report:

Visual Odometry Revisited: What Should Be Learnt?

DF-VO: What Should Be Learnt for Visual Odometry?

Huangying Zhan, Chamara Saroj Weerasekera, Jiawang Bian, Ravi Garg, Ian Reid

The demo video can be found here.

```
@INPROCEEDINGS{zhan2019dfvo,
  author={H. {Zhan} and C. S. {Weerasekera} and J. -W. {Bian} and I. {Reid}},
  booktitle={2020 IEEE International Conference on Robotics and Automation (ICRA)},
  title={Visual Odometry Revisited: What Should Be Learnt?},
  year={2020},
  volume={},
  number={},
  pages={4203-4210},
  doi={10.1109/ICRA40945.2020.9197374}}

@misc{zhan2021dfvo,
      title={DF-VO: What Should Be Learnt for Visual Odometry?},
      author={Huangying Zhan and Chamara Saroj Weerasekera and Jia-Wang Bian and Ravi␣
↪Garg and Ian Reid},
      year={2021},
      eprint={2103.00933},
      archivePrefix={arXiv},
      primaryClass={cs.CV}
}
```

This repo includes

1. the frame-to-frame tracking system **DF-VO**;
2. evaluation scripts for visual odometry;
3. trained models and VO results

1.1 Contents

1.2 Part 1. Requirements

This code was tested with Python 3.6, CUDA 9.0, Ubuntu 16.04, and PyTorch-1.1.

We suggest use Anaconda for installing the prerequisites.

```
cd envs
conda env create -f requirement.yml -p {ANACONDA_DIR/envs/dfvo} # install␣
↪prerequisites
conda activate dfvo  # activate the environment [dfvo]
```

1.3 Part 2. Download dataset and models

The main dataset used in this project is KITTI Driving Dataset. After downloaing the dataset, create a softlink in the current repo.

```
ln -s KITTI_ODOMETRY/sequences dataset/kitti_odom/odom_data
```

For our trained models, please visit here to download the models and save the models into the directory `model_zoo/`.

1.4 Part 3. DF-VO

We have created some examples for running the algorithm.

```
# Example 1: run default kitti setup
python apis/run.py -d options/examples/default_configuration.yml

# Example 2: Run custom kitti setup
python apis/run.py \
-d options/examples/default_configuration.yml \
-c options/examples/kitti_stereo_train_icra.yml \
--no_confirm

# More examples and our experiments can be found in scripts/experiment.sh
```

The result (trajectory pose file) is saved in `result_dir` defined in the configuration file. Please check the `options/examples/default_configuration.yml` or Configuration Documentation for reference.

1.5 Part 4. Result evaluation

The original results, including related works, can be found here.

1.5.1 KITTI

KITTI Odometry benchmark contains 22 stereo sequences, in which 11 sequences are provided with ground truth. The 11 sequences are used for evaluating visual odometry.

```
python tools/evaluation/odometry/eval_odom.py --result result/tmp/0 --align 6dof
```

For more information about the evaluation toolkit, please check the toolbox page or the wiki page.

1.6 Part 5. Run your own dataset

We also provide a guideline to run DF-VO on your own dataset. Please check Run own dataset for more details.

1.7 License

For academic usage, the code is released under the permissive MIT license. Our intension of sharing the project is for research/personal purpose. For any commercial purpose, please contact the authors.

1.8 Acknowledgement

Some of the codes were borrowed from the excellent works of monodepth2, LiteFlowNet and pytorch-liteflownet. The borrowed files are licensed under their original license respectively.

1.8.1 Introduction

DF-VO is a frame-to-frame VO system integrating geometry and deep learning.

Project Structure

This project has the following structure:

```
- DF-VO
    - apis                                  # APIs
    - dataset                               # dataset directories
    - envs                                  # requirement files to create␣
↪anaoncda envs
    - docs                                  # documentation
    - libs                                  # packages
        - dfvo.py                           # core DF-VO program
```

(continues on next page)

(continued from previous page)

```
        - datasets                      # dataset loaders
        - deep_models                   # deep networks
            - depth                     # depth models
            - flow                      # optical flow models
            - pose                      # ego-motion models
        - geometry                      # geometry related operations
        - matching                      # feature matching related packages
        - tracker                       # tracker packages
        - general                       # general functions
    - model_zoo                         # pretrained deep models
    - options                           # configurations
    - scripts                           # scripts for running experiments
    - tools                             # experiment tools, e.g. evaluation
```

1.8.2 Installation

This code was tested with Python 3.6, CUDA 9.0, Ubuntu 16.04, and PyTorch.

We suggest use Anaconda for installing the prerequisites.

```
# clone the project
git clone https://github.com/Huangying-Zhan/DF-VO.git

# create conda environment and install prerequisites
cd DF-VO/envs
conda env create -f requirement.yml -p dfvo

# activate the environment [dfvo]
conda activate dfvo
```

1.8.3 Examples

We provide different examples for the DF-VO system.

Run DF-VO

```
# Example 1: run default kitti setup
python apis/run.py -d options/kitti/default_configuration.yml

# Example 2: Run custom kitti setup
# kitti_default_configuration.yml and kitti_stereo_0.yml are merged
python apis/run.py -d options/kitti/default_configuration.yml -c options/
↪kitti/kitti_stereo_0.yml
```

Configuration

We provide the explanation of the configurations in this file. A KITTI experiment is used as an example. We also provide possible choices in the current framework. We have some unused configurations (will be commented) for our Experiment Version but not used in this Release Version.

Notes:

- **None**: when the choice is None, it means leave the option blank. DON'T put 'None', which yaml would recognize as string 'None'

```
# ----------------------------------
# Basic setup
# ----------------------------------
dataset: kitti_odom                     # dataset [kitti_odom, kitti_raw, tum-1/2/3,␣
↪adelaide1/2]
seed: 4869                              # random seed
image:
    height: 192                         # image height
    width: 640                          # image width
    ext: jpg                            # image file extension for data loading
seq: "10"                               # sequence to run
frame_step: 1                           # frame step

# ----------------------------------
# Directories
# ----------------------------------
directory:
    result_dir: {RESULT_DIR}            # directory to save result
    img_seq_dir: {IMAGE_DATA_DIR}       # image data directory
    gt_pose_dir: {GT_POSE_DIR}          # (optional) ground-truth pose data directory
    depth_dir: {DEPTH_DATA_DIR}         # (optional) external depth data, e.g. ground-
↪truth depths

# ----------------------------------
# Depth
# ----------------------------------
depth:                                  # Depth configuration
    depth_src:                          # depth source [None, gt]
                                            # None - depth model predition
                                            # gt - use ground truth depth
    deep_depth:
        network: monodepth2             # depth network
        pretrained_model: {MODEL_DIR}   # directory stores depth.pth and encoder.pth
    max_depth: 50                       # maximum depth
    min_depth: 0                        # minimum depth

# ----------------------------------
# Deep flow
# ----------------------------------
deep_flow:                              # Deep optical flow configuration
    network: liteflow                   # optical flow network, [liteflow]
    flow_net_weight: {FLOW_MODEL}       # optical flow model path
    forward_backward: True              # predict both forward/backward flows and␣
↪compute forward-backward flow consistency

# ----------------------------------
# Deep Pose (Experiment Ver. only)
```

(continues on next page)

(continued from previous page)

```
# -----------------------------------
deep_pose:                              # Deep pose network configuration
    enable: False                       # enable/disable pose network
    pretrained_model: {MODEL_DIR}       # model directory contains pose_encoder.pth␣
↪and pose.pth

# -----------------------------------
# Online Finetuning
# -----------------------------------
online_finetune:                        # online fine-tuning configuration
    enable: False                       # enable/disable flow finetuning
    lr: 0.00001                         # learning rate
    num_frames:                         # number of frames to be fine-tuned, [None,␣
↪int]
    flow:                               # flow fine-tuning configuration
        enable: False                   # enable/disable flow finetuning
        scales: [1, 2, 3, 4, 5]         # scales to be used for training
        loss:                           # flow loss configuration
            flow_consistency: 0.005     # forward-backward flow consistency loss␣
↪weight
            flow_smoothness: 0.1        # flow smoothness loss weight
    depth:                              # depth fine-tuning configuration
        enable: False                   # enable/disable depth finetuning
        scales: [0, 1, 2, 3]            # scales to be used for training
        pose_src: DF-VO                 # pose source for depth-pose finetuning [DF-
↪Vo, deep_pose]
        loss:                           # depth loss configuration
            apperance_loss: 1           # apperance loss weight
            disparity_smoothness: 0.001 # disparity smoothness loss weight
            depth_consistency: 0.001    # depth consistency loss weight
    pose:                               # pose finetuning configuration (with depth)
        enable: False                   # enable/disable pose finetuning

# -----------------------------------
# Preprocessing
# -----------------------------------
crop:                                   # cropping configuration
    depth_crop: [[0.3, 1], [0, 1]]      # depth map cropping, format: [[y0, y1],[x0,␣
↪x1]]
    flow_crop: [[0, 1], [0, 1]]         # optical flow map cropping, format: [[y0,␣
↪y1],[x0, x1]]

# -----------------------------------
# Correspondence (keypoint) selection
# -----------------------------------
kp_selection:                           # correspondence selection configuration
    local_bestN:                        # local best-N configuration
        enable: True                    # enable/disable local best-N selection
        num_bestN: 2000                 # number of keypoints
        num_row: 10                     # number of divided rows
        num_col: 10                     # number of divided columns
        score_method: flow              # selection score, [flow, flow_ratio]
                                            # flow: L2 distance of forward-backward␣
↪flow
                                            # flow_ratio: relative flow difference␣
↪ratio
        thre: 0.1                       # flow consistency masking threshold
```

(continues on next page)

(continued from previous page)

```
    bestN:
        enable: False                   # enable/disable best-N selection
        num_bestN: 2000                 # number of keypoints
    sampled_kp:                         # random/uniform keypoint sampling
        enable: False                   # enable/disable random/uniform keypoint␣
↪sampling
        num_kp: 2000                    # number of keypoints
    rigid_flow_kp:                      # keypoint selection from optical-rigid flow␣
↪consistency (for scale recovery)
        enable: False                   # enable/disable rigid-flow based keypoint␣
↪selection
        num_bestN: 2000                 # number of keypoints
        num_row: 10                     # number of divided rows
        num_col: 10                     # number of divided columns
        score_method: flow              # selection score, [flow]
        rigid_flow_thre: 3              # masking threshold for rigid-optical flow␣
↪consistency
        optical_flow_thre: 0.1          # masking threshold for forward-backward flow␣
↪consistency
    depth_consistency:                  # (Experiement Ver. only) depth consistency␣
↪configuration
        enable: False                   # enable/disable depth consistency
        thre: 0.05                      # masking threshold

# -------------------------------------
# Tracking
# -------------------------------------
tracking_method: hybrid                 # tracking method [hybrid, PnP, deep_pose]
                                            # hybrid - E-tracker + PnP-tracker;
                                            # PnP - PnP-tracker
                                            # deep_pose - pose_cnn-tracker


e_tracker:                              # E-tracker configuration
    ransac:                             # Ransac configuration
        reproj_thre: 0.2                # inlier threshold value
        repeat: 5                       # number of repeated Ransac
    validity:                           # model selection condition
        method: GRIC                    # method of validating E-tracker, [flow, GRIC]
        thre:                           # threshold value for model selection, only␣
↪used in [flow]
    kp_src: kp_best                     # type of correspondences to be used [kp_list,
↪ kp_best]
                                            # kp_list - uniformaly sampled keypoints
                                            # kp_best - keypoints sampled from best-N␣
↪/ local best method

scale_recovery:                         # scale recovery configuration
    method: simple                      # scale recovery method [simple, iterative]
    ransac:                             # Ransac configuration
        method: depth_ratio             # fitting target [depth_ratio, abs_diff]
                                            # depth_ratio: find a scale s.t. most␣
↪triangulated_depth/cnn_depth close to 1
                                            # abs_diff: find a scale s.t.␣
↪abs(triangulated_depth - cnn_depth) close to 0
        min_samples: 3                  # minimum number of min_samples
        max_trials: 100                 # maximum number of trials
        stop_prob: 0.99                 # The probability that the algorithm produces␣
↪a useful result
```

(continues on next page)

(continued from previous page)

```
        thre: 0.1                           # inlier threshold value
    kp_src: kp_best                         # type of correspondences to be used [kp_list,
↪ kp_best, kp_depth]
                                                # kp_list - uniformaly sampled keypoints
                                                # kp_best - keypoints sampled from best-N␣
↪/ local best method
                                                # kp_depth - keypoints sampled after␣
↪optical-rigid flow consistency masking

pnp_tracker:                                # PnP-tracker configuration
    ransac:                                 # Ransac configuration
        iter: 100                           # number of iteration
        reproj_thre: 1                      # inlier threshold value
        repeat: 5                           # number of repeated Ransac
    kp_src: kp_best                         # type of correspondences to be used [kp_list,
↪ kp_best, kp_depth]
                                                # kp_list - uniformaly sampled keypoints
                                                # kp_best - keypoints sampled from best-N␣
↪/ local best method
                                                # kp_depth - keypoints sampled after␣
↪optical-rigid flow consistency masking

# ------------------------------------
# Visualization
# ------------------------------------
visualization:                              # visualization configuration
    enable: True                            # enable/disable frame drawer
    save_img: True                          # enable/disable save frames
    window_h: 900                           # frame window height
    window_w: 1500                          # frame window width
    kp_src: kp_best                         # type of correspondences to be drawn
    flow:                                   # optical flow visualization configuration
        vis_forward_flow: True              # enable/disable forward flow visualization
        vis_backward_flow: True             # enable/disable backward flow visualization
        vis_flow_diff: True                 # enable/disable forward-backward flow␣
↪consistency visualization
        vis_rigid_diff: False               # enable/disable optical-rigid flow␣
↪consistency visualization
    kp_match:                               # keypoint matching visualization
        kp_num: 100                         # number of selected keypoints to be␣
↪visualized
        vis_temp:                           # keypoint matching in temporal
            enable: True                    # enable/disable visualization
        vis_side:                           # keypoint matching side-by-side
            enable: True                    # enable/disable visualization
            inlier_plot: False              # enable/disable inlier plot
    trajectory:                             # trajectory visualization configuration
        vis_traj: True                      # enable/disable predicted trajectory␣
↪visualization
        vis_gt_traj: False                  # enable/disable ground truth trajectory␣
↪visualization
        mono_scale: 1                       # scaling factor to align with gt (if gt is␣
↪available)
    depth:                                  # depth visualization configuration
        use_tracking_depth: False           # enable/disable visualizing depth map used␣
↪for tracking (preprocessed, e.g. range capping)
        depth_disp: disp                    # visualize depth or disparity map [depth,␣
↪disp]
```

(continues on next page)

(continued from previous page)

Run own dataset

Run your own dataset with DF-VO is not complicated. Basically, you need to add a dataset loader and update the configuration file. Here are the steps to run your own dataset.

- Add dataset loader

  Refer to the example `libs/datasets/adelaide.py`, there are some least functions you need to provide for your dataset loader. Some functions are optional where you would find "NotImplemented". Basically you need to have a fuction for loading camera intrinsics and image data. There are intrustions in the *libs/datasets/adelaide.py* as well.

- Add the loader to Dataset

  After creating the dataset loader, add the dataset loader to `libs/dataset/__init__.py`. You need to import the loader and put it in the dictionary *datasets* in the same file.

- Update configuration file

  Update at least the following configurations in the config file.

  - dataset
  - img_seq_dir

Tools usage

Evaluation

To evaluate the odometry result on KITTI dataset, here we provide an example. For details, please refer to the eval_odom wiki page.

```
# Evaluate Odometry Split
python tools/evaluation/eval_odom.py \
--result {RESULT_DIR} \
--gt dataset/kitti_odom/gt_poses/ \
--align 7dof \
--seqs "09" "10"
```

General tools

```
# Generate ground truth poses from KITTI Raw dataset
python tools/generate_kitti_raw_pose.py \
--data_dir dataset/kitti_raw \
--result_dir dataset/kitti_raw_pose \
--seqs 2011_09_26_drive_0005_sync 2011_09_26_drive_0009_sync

# Generate KITTI Flow 2012/2015 prediction
python tools/generate_flow_prediction.py \
--dataset kitti2012 \
--model {FLOW_MODEL_PATH} \
--result {RESULT_DIR}
```

1.8.4 apis

Submodules

apis.run

1.8.5 libs

Subpackages

libs.datasets

Submodules

libs.datasets.adelaide

libs.datasets.dataset

libs.datasets.kinect

libs.datasets.kitti

libs.datasets.tum

libs.deep_models

Subpackages

libs.deep_models.depth

Subpackages

libs.deep_models.depth.monodepth2

Submodules

libs.deep_models.depth.monodepth2.depth_decoder

libs.deep_models.depth.monodepth2.layers

libs.deep_models.depth.monodepth2.monodepth2

libs.deep_models.depth.monodepth2.resnet_encoder

Submodules

libs.deep_models.depth.deep_depth

libs.deep_models.flow

Subpackages

libs.deep_models.flow.lite_flow_net

Submodules

libs.deep_models.flow.lite_flow_net.correlation

libs.deep_models.flow.lite_flow_net.lite_flow

libs.deep_models.flow.lite_flow_net.lite_flow_net

Submodules

libs.deep_models.flow.deep_flow

libs.deep_models.pose

Subpackages

libs.deep_models.pose.monodepth2

Submodules

libs.deep_models.pose.monodepth2.monodepth2

libs.deep_models.pose.monodepth2.pose_decoder

Submodules

libs.deep_models.pose.deep_pose

Submodules

libs.deep_models.deep_models

libs.flowlib

Submodules

libs.flowlib.flowlib

libs.flowlib.png

class Image(*rows*, *info*)
Bases: object

A PNG image. You can create an *Image* object from an array of pixels by calling png.from_array(). It can be saved to disk with the *save()* method.

__init__(*rows*, *info*)

> **Note:** The constructor is not public. Please do not call it.

save(*file*)
Save the image to *file*. If *file* looks like an open file descriptor then it is used, otherwise it is treated as a filename and a fresh file is opened.

In general, you can only call this method once; after it has been called the first time and the PNG image has been saved, the source data will have been streamed, and cannot be streamed again.

class Reader(*_guess=None*, ***kw*)
Bases: object

PNG decoder in pure Python.

__init__(*_guess=None*, ***kw*)
Create a PNG decoder object.

The constructor expects exactly one keyword argument. If you supply a positional argument instead, it will guess the input type. You can choose among the following keyword arguments:

filename Name of input file (a PNG file).

file A file-like object (object with a read() method).

bytes array or string with PNG data.

asDirect()
Returns the image data as a direct representation of an x * y * planes array. This method is intended to remove the need for callers to deal with palettes and transparency themselves. Images with a palette (colour type 3) are converted to RGB or RGBA; images with transparency (a tRNS chunk) are converted to LA or RGBA as appropriate. When returned in this format the pixel values represent the colour value directly without needing to refer to palettes or transparency information.

Like the *read()* method this method returns a 4-tuple:

(*width*, *height*, *pixels*, *meta*)

This method normally returns pixel values with the bit depth they have in the source image, but when the source PNG has an sBIT chunk it is inspected and can reduce the bit depth of the result pixels; pixel

values will be reduced according to the bit depth specified in the `sBIT` chunk (PNG nerds should note a single result bit depth is used for all channels; the maximum of the ones specified in the `sBIT` chunk. An RGB565 image will be rescaled to 6-bit RGB666).

The *meta* dictionary that is returned reflects the *direct* format and not the original source image. For example, an RGB source image with a `tRNS` chunk to represent a transparent colour, will have `planes=3` and `alpha=False` for the source image, but the *meta* dictionary returned by this method will have `planes=4` and `alpha=True` because an alpha channel is synthesized and added.

pixels is the pixel data in boxed row flat pixel format (just like the *read()* method).

All the other aspects of the image data are not changed.

asFloat (*maxval=1.0*)
Return image pixels as per *asDirect()* method, but scale all pixel values to be floating point values between 0.0 and *maxval*.

asRGB ()
Return image as RGB pixels. RGB colour images are passed through unchanged; greyscales are expanded into RGB triplets (there is a small speed overhead for doing this).

An alpha channel in the source image will raise an exception.

The return values are as for the *read()* method except that the *metadata* reflect the returned pixels, not the source image. In particular, for this method `metadata['greyscale']` will be `False`.

asRGB8 ()
Return the image data as an RGB pixels with 8-bits per sample. This is like the *asRGB()* method except that this method additionally rescales the values so that they are all between 0 and 255 (8-bit). In the case where the source image has a bit depth < 8 the transformation preserves all the information; where the source image has bit depth > 8, then rescaling to 8-bit values loses precision. No dithering is performed. Like *asRGB()*, an alpha channel in the source image will raise an exception.

This function returns a 4-tuple: (*width*, *height*, *pixels*, *metadata*). *width*, *height*, *metadata* are as per the *read()* method.

pixels is the pixel data in boxed row flat pixel format.

asRGBA ()
Return image as RGBA pixels. Greyscales are expanded into RGB triplets; an alpha channel is synthesized if necessary. The return values are as for the *read()* method except that the *metadata* reflect the returned pixels, not the source image. In particular, for this method `metadata['greyscale']` will be `False`, and `metadata['alpha']` will be `True`.

asRGBA8 ()
Return the image data as RGBA pixels with 8-bits per sample. This method is similar to *asRGB8()* and *asRGBA()*: The result pixels have an alpha channel, *and* values are rescaled to the range 0 to 255. The alpha channel is synthesized if necessary (with a small speed penalty).

chunk (*seek=None*, *lenient=False*)
Read the next PNG chunk from the input file; returns a (*type*, *data*) tuple. *type* is the chunk's type as a byte string (all PNG chunk types are 4 bytes long). *data* is the chunk's data content, as a byte string.

If the optional *seek* argument is specified then it will keep reading chunks until it either runs out of file or finds the type specified by the argument. Note that in general the order of chunks in PNGs is unspecified, so using *seek* can cause you to miss chunks.

If the optional *lenient* argument evaluates to *True*, checksum failures will raise warnings rather than exceptions.

chunklentype()
Reads just enough of the input to determine the next chunk's length and type, returned as a (*length*, *type*) pair where *type* is a string. If there are no more chunks, `None` is returned.

chunks()
Return an iterator that will yield each chunk as a (*chunktype*, *content*) pair.

deinterlace(*raw*)
Read raw pixel data, undo filters, deinterlace, and flatten. Return in flat row flat pixel format.

iterboxed(*rows*)
Iterator that yields each scanline in boxed row flat pixel format. *rows* should be an iterator that yields the bytes of each row in turn.

iterstraight(*raw*)
Iterator that undoes the effect of filtering, and yields each row in serialised format (as a sequence of bytes). Assumes input is straightlaced. *raw* should be an iterable that yields the raw bytes in chunks of arbitrary size.

palette(*alpha='natural'*)
Returns a palette that is a sequence of 3-tuples or 4-tuples, synthesizing it from the `PLTE` and `tRNS` chunks. These chunks should have already been processed (for example, by calling the *preamble()* method). All the tuples are the same size: 3-tuples if there is no `tRNS` chunk, 4-tuples when there is a `tRNS` chunk. Assumes that the image is colour type 3 and therefore a `PLTE` chunk is required.

If the *alpha* argument is `'force'` then an alpha channel is always added, forcing the result to be a sequence of 4-tuples.

preamble(*lenient=False*)
Extract the image metadata by reading the initial part of the PNG file up to the start of the `IDAT` chunk. All the chunks that precede the `IDAT` chunk are read and either processed for metadata or discarded.

If the optional *lenient* argument evaluates to *True*, checksum failures will raise warnings rather than exceptions.

process_chunk(*lenient=False*)
Process the next chunk and its data. This only processes the following chunk types, all others are ignored: `IHDR`, `PLTE`, `bKGD`, `tRNS`, `gAMA`, `sBIT`, `pHYs`.

If the optional *lenient* argument evaluates to *True*, checksum failures will raise warnings rather than exceptions.

read(*lenient=False*)
Read the PNG file and decode it. Returns (*width*, *height*, *pixels*, *metadata*).

May use excessive memory.

pixels are returned in boxed row flat pixel format.

If the optional *lenient* argument evaluates to True, checksum failures will raise warnings rather than exceptions.

read_flat()
Read a PNG file and decode it into flat row flat pixel format. Returns (*width*, *height*, *pixels*, *metadata*).

May use excessive memory.

pixels are returned in flat row flat pixel format.

See also the *read()* method which returns pixels in the more stream-friendly boxed row flat pixel format.

serialtoflat(*bytes*, *width=None*)
Convert serial format (byte stream) pixel data to flat row flat pixel.

undo_filter(*filter_type*, *scanline*, *previous*)
Undo the filter for a scanline. *scanline* is a sequence of bytes that does not include the initial filter type byte. *previous* is decoded previous scanline (for straightlaced images this is the previous pixel row, but for interlaced images, it is the previous scanline in the reduced image, which in general is not the previous pixel row in the final image). When there is no previous scanline (the first row of a straightlaced image, or the first row in one of the passes in an interlaced image), then this argument should be `None`.

The scanline will have the effects of filtering removed, and the result will be returned as a fresh sequence of bytes.

validate_signature()
If signature (header) has not been read then read and validate it; otherwise do nothing.

class Writer(*width=None*, *height=None*, *size=None*, *greyscale=False*, *alpha=False*, *bitdepth=8*, *palette=None*, *transparent=None*, *background=None*, *gamma=None*, *compression=None*, *interlace=False*, *bytes_per_sample=None*, *planes=None*, *colormap=None*, *maxval=None*, *chunk_limit=1048576*, *x_pixels_per_unit=None*, *y_pixels_per_unit=None*, *unit_is_meter=False*)
Bases: `object`

PNG encoder in pure Python.

__init__(*width=None*, *height=None*, *size=None*, *greyscale=False*, *alpha=False*, *bitdepth=8*, *palette=None*, *transparent=None*, *background=None*, *gamma=None*, *compression=None*, *interlace=False*, *bytes_per_sample=None*, *planes=None*, *colormap=None*, *maxval=None*, *chunk_limit=1048576*, *x_pixels_per_unit=None*, *y_pixels_per_unit=None*, *unit_is_meter=False*)
Create a PNG encoder object.

Arguments:

width, height Image size in pixels, as two separate arguments.

size Image size (w,h) in pixels, as single argument.

greyscale Input data is greyscale, not RGB.

alpha Input data has alpha channel (RGBA or LA).

bitdepth Bit depth: from 1 to 16.

palette Create a palette for a colour mapped image (colour type 3).

transparent Specify a transparent colour (create a `tRNS` chunk).

background Specify a default background colour (create a `bKGD` chunk).

gamma Specify a gamma value (create a `gAMA` chunk).

compression zlib compression level: 0 (none) to 9 (more compressed); default: -1 or None.

interlace Create an interlaced image.

chunk_limit Write multiple `IDAT` chunks to save memory.

x_pixels_per_unit Number of pixels a unit along the x axis (write a *pHYs* chunk).

y_pixels_per_unit Number of pixels a unit along the y axis (write a *pHYs* chunk). Along with *x_pixel_unit*, this gives the pixel size ratio.

unit_is_meter *True* to indicate that the unit (for the *pHYs* chunk) is metre.

The image size (in pixels) can be specified either by using the *width* and *height* arguments, or with the single *size* argument. If *size* is used it should be a pair (*width*, *height*).

greyscale and *alpha* are booleans that specify whether an image is greyscale (or colour), and whether it has an alpha channel (or not).

bitdepth specifies the bit depth of the source pixel values. Each source pixel value must be an integer between 0 and `2**bitdepth-1`. For example, 8-bit images have values between 0 and 255. PNG only stores images with bit depths of 1,2,4,8, or 16. When *bitdepth* is not one of these values, the next highest valid bit depth is selected, and an `sBIT` (significant bits) chunk is generated that specifies the original precision of the source image. In this case the supplied pixel values will be rescaled to fit the range of the selected bit depth.

The details of which bit depth / colour model combinations the PNG file format supports directly, are somewhat arcane (refer to the PNG specification for full details). Briefly: "small" bit depths (1,2,4) are only allowed with greyscale and colour mapped images; colour mapped images cannot have bit depth 16.

For colour mapped images (in other words, when the *palette* argument is specified) the *bitdepth* argument must match one of the valid PNG bit depths: 1, 2, 4, or 8. (It is valid to have a PNG image with a palette and an `sBIT` chunk, but the meaning is slightly different; it would be awkward to press the *bitdepth* argument into service for this.)

The *palette* option, when specified, causes a colour mapped image to be created: the PNG colour type is set to 3; *greyscale* must not be set; *alpha* must not be set; *transparent* must not be set; the bit depth must be 1,2,4, or 8. When a colour mapped image is created, the pixel values are palette indexes and the *bitdepth* argument specifies the size of these indexes (not the size of the colour values in the palette).

The palette argument value should be a sequence of 3- or 4-tuples. 3-tuples specify RGB palette entries; 4-tuples specify RGBA palette entries. If both 4-tuples and 3-tuples appear in the sequence then all the 4-tuples must come before all the 3-tuples. A `PLTE` chunk is created; if there are 4-tuples then a `tRNS` chunk is created as well. The `PLTE` chunk will contain all the RGB triples in the same sequence; the `tRNS` chunk will contain the alpha channel for all the 4-tuples, in the same sequence. Palette entries are always 8-bit.

If specified, the *transparent* and *background* parameters must be a tuple with three integer values for red, green, blue, or a simple integer (or singleton tuple) for a greyscale image.

If specified, the *gamma* parameter must be a positive number (generally, a *float*). A `gAMA` chunk will be created. Note that this will not change the values of the pixels as they appear in the PNG file, they are assumed to have already been converted appropriately for the gamma specified.

The *compression* argument specifies the compression level to be used by the `zlib` module. Values from 1 to 9 specify compression, with 9 being "more compressed" (usually smaller and slower, but it doesn't always work out that way). 0 means no compression. -1 and `None` both mean that the default level of compession will be picked by the `zlib` module (which is generally acceptable).

If *interlace* is true then an interlaced image is created (using PNG's so far only interace method, *Adam7*). This does not affect how the pixels should be presented to the encoder, rather it changes how they are arranged into the PNG file. On slow connexions interlaced images can be partially decoded by the browser to give a rough view of the image that is successively refined as more image data appears.

Note: Enabling the *interlace* option requires the entire image to be processed in working memory.

---

chunk_limit is used to limit the amount of memory used whilst compressing the image. In order to avoid using large amounts of memory, multiple `IDAT` chunks may be created.

array_scanlines(*pixels*)
Generates boxed rows (flat pixels) from flat rows (flat pixels) in an array.

array_scanlines_interlace(*pixels*)
Generator for interlaced scanlines from an array. *pixels* is the full source image in flat row flat pixel format.

The generator yields each scanline of the reduced passes in turn, in boxed row flat pixel format.

convert_pnm(*infile*, *outfile*)
Convert a PNM file containing raw pixel data into a PNG file with the parameters set in the writer object. Works for (binary) PGM, PPM, and PAM formats.

convert_ppm_and_pgm(*ppmfile*, *pgmfile*, *outfile*)
Convert a PPM and PGM file containing raw pixel data into a PNG outfile with the parameters set in the writer object.

file_scanlines(*infile*)
Generates boxed rows in flat pixel format, from the input file *infile*. It assumes that the input file is in a "Netpbm-like" binary format, and is positioned at the beginning of the first pixel. The number of pixels to read is taken from the image dimensions (*width*, *height*, *planes*) and the number of bytes per value is implied by the image *bitdepth*.

make_palette()
Create the byte sequences for a `PLTE` and if necessary a `tRNS` chunk. Returned as a pair (*p*, *t*). *t* will be `None` if no `tRNS` chunk is necessary.

write(*outfile*, *rows*)
Write a PNG image to the output file. *rows* should be an iterable that yields each row in boxed row flat pixel format. The rows should be the rows of the original image, so there should be `self.height` rows of `self.width * self.planes` values. If *interlace* is specified (when creating the instance), then an interlaced PNG file will be written. Supply the rows in the normal image order; the interlacing is carried out internally.

Note: Interlacing will require the entire image to be in working memory.

---

write_array(*outfile*, *pixels*)
Write an array in flat row flat pixel format as a PNG file on the output file. See also *write()* method.

write_packed(*outfile*, *rows*)
Write PNG file to *outfile*. The pixel data comes from *rows* which should be in boxed row packed format. Each row should be a sequence of packed bytes.

Technically, this method does work for interlaced images but it is best avoided. For interlaced images, the rows should be presented in the order that they appear in the file.

This method should not be used when the source image bit depth is not one naturally supported by PNG; the bit depth should be 1, 2, 4, 8, or 16.

write_passes(*outfile*, *rows*, *packed=False*)
Write a PNG image to the output file.

Most users are expected to find the *write()* or *write_array()* method more convenient.

The rows should be given to this method in the order that they appear in the output file. For straightlaced images, this is the usual top to bottom ordering, but for interlaced images the rows should have already been interlaced before passing them to this function.

rows should be an iterable that yields each row. When *packed* is `False` the rows should be in boxed row flat pixel format; when *packed* is `True` each row should be a packed sequence of bytes.

from_array(*a*, *mode=None*, *info={}*)
Create a PNG *Image* object from a 2- or 3-dimensional array. One application of this function is easy PIL-style saving: `png.from_array(pixels, 'L').save('foo.png')`.

Unless they are specified using the *info* parameter, the PNG's height and width are taken from the array size. For a 3 dimensional array the first axis is the height; the second axis is the width; and the third axis is the

channel number. Thus an RGB image that is 16 pixels high and 8 wide will use an array that is 16x8x3. For 2 dimensional arrays the first axis is the height, but the second axis is `width*channels`, so an RGB image that is 16 pixels high and 8 wide will use a 2-dimensional array that is 16x24 (each row will be 8*3 = 24 sample values).

mode is a string that specifies the image colour format in a PIL-style mode. It can be:

`'L'` greyscale (1 channel)

`'LA'` greyscale with alpha (2 channel)

`'RGB'` colour image (3 channel)

`'RGBA'` colour image with alpha (4 channel)

The mode string can also specify the bit depth (overriding how this function normally derives the bit depth, see below). Appending `';16'` to the mode will cause the PNG to be 16 bits per channel; any decimal from 1 to 16 can be used to specify the bit depth.

When a 2-dimensional array is used *mode* determines how many channels the image has, and so allows the width to be derived from the second array dimension.

The array is expected to be a `numpy` array, but it can be any suitable Python sequence. For example, a list of lists can be used: `png.from_array([[0, 255, 0], [255, 0, 255]], 'L')`. The exact rules are: `len(a)` gives the first dimension, height; `len(a[0])` gives the second dimension; `len(a[0][0])` gives the third dimension, unless an exception is raised in which case a 2-dimensional array is assumed. It's slightly more complicated than that because an iterator of rows can be used, and it all still works. Using an iterator allows data to be streamed efficiently.

The bit depth of the PNG is normally taken from the array element's datatype (but if *mode* specifies a bitdepth then that is used instead). The array element's datatype is determined in a way which is supposed to work both for `numpy` arrays and for Python `array.array` objects. A 1 byte datatype will give a bit depth of 8, a 2 byte datatype will give a bit depth of 16. If the datatype does not have an implicit size, for example it is a plain Python list of lists, as above, then a default of 8 is used.

The *info* parameter is a dictionary that can be used to specify metadata (in the same style as the arguments to the `png.Writer` class). For this function the keys that are useful are:

height overrides the height derived from the array dimensions and allows *a* to be an iterable.

width overrides the width derived from the array dimensions.

bitdepth overrides the bit depth derived from the element datatype (but must match *mode* if that also specifies a bit depth).

Generally anything specified in the *info* dictionary will override any implicit choices that this function would otherwise make, but must match any explicit ones. For example, if the *info* dictionary has a `greyscale` key then this must be true when mode is `'L'` or `'LA'` and false when mode is `'RGB'` or `'RGBA'`.

`write_chunks`(*out*, *chunks*)
Create a PNG file by writing out the chunks.

libs.general

Submodules

libs.general.configuration

libs.general.frame_drawer

libs.general.kitti_raw_utils

libs.general.kitti_utils

libs.general.timer

libs.general.utils

libs.geometry

Submodules

libs.geometry.backprojection

libs.geometry.camera_modules

libs.geometry.ops_3d

libs.geometry.projection

libs.geometry.reprojection

libs.geometry.rigid_flow

libs.geometry.transformation3d

libs.matching

Submodules

libs.matching.depth_consistency

libs.matching.keypoint_sampler

libs.matching.kp_selection

libs.tracker

Submodules

libs.tracker.E_tracker

libs.tracker.gric

libs.tracker.pnp_tracker

Submodules

libs.dfvo

1.8.6 tools

Subpackages

tools.evaluation

Subpackages

tools.evaluation.odometry

Submodules

tools.evaluation.odometry.eval_odom

tools.evaluation.odometry.kitti_odometry

Submodules

tools.generate_flow_prediction

tools.generate_kitti_raw_pose

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

l

t

INDEX

Symbols

A

C

D

F

I

L

M

P

R

S

T

U

V

W