bbbinaryDocumentation

Release 0.0.1

Benjamin Wild, Leon Sixt

Jun 14, 2018

Contents

CHAPTER 1

Common

These objects are providing access to our *Schema* in Python's Cap'n Proto implementation.

FrameContainer

FrameContainer are basically the root of our data container. They represent a video that in beesbook context usually consist of 5 minutes. Each FrameContainer only has the frames of **one** camera.

```
    filename @1 :Text;                # filename of the data source
    videoPreviewFilename @2 :Text;    # (optional) filename of the preview video
}

# Corresponds to a video
struct FrameContainer {
  id @0 :UInt64;                      # global unique id of the frame container
  dataSources @1:List(DataSource);    # list of data sources (videos / images)
  fromTimestamp @2 :Float64;          # unix timestamp of the first frame
  toTimestamp @3 :Float64;            # unix timestamp of the last frame
  frames @4 :List(Frame);             # frames must be sorted by in the order they␣
↪where recorded.
  camId @5 :UInt16;                   # the cam number
  hiveId @6 :UInt16;                  # the id of the hive
  transformationMatrix @7 :List(Float32);
                                      # the transformation matrix from image␣
↪coordinates to hive coordinates.
                                      # The matrix is of dimension 4x4 and stored␣
↪this way
                                      #     1 | 2 | 3 | 4
```

Frame

A Frame holds all the information about a single image in a video.

```
  }
}
```

(continues on next page)

(continued from previous page)

```
# Corresponds to an image in the video.
struct Frame {
  id @0 :UInt64;                  # global unique id of the frame
  dataSourceIdx @1:UInt32;        # the frame is from this data source
  frameIdx @6 :UInt32;            # sequential increasing index for every data␣
↪source.
  timestamp @2 :Float64;          # unix time stamp of the frame
  timedelta @3 :UInt32;           # time difference between this frame and the␣
↪frame before in microseconds
  detectionsUnion : union {
    detectionsCVP   @4 :List(DetectionCVP);    # detections format of the old␣
↪computer vision pipeline
```

DataSource

This is a part of a *FrameContainer* and references the original video file.

```
    detectionsTruth @7 :List(DetectionTruth); # detections format of ground truth␣
↪data
  }
}

struct DataSource {
```

DetectionCVP

This is the format of a detection in the old Computer Vision Pipeline format. It got replaced with *DetectionDP* in the Summer 2016.

```
struct DetectionCVP {
  idx @0 :UInt16;                 # sequential index of the detection, counted␣
↪from 0 for every frame
                                  # the combination (idx, Frame.id) is a global key
  candidateIdx @1 :UInt16;        # sequential index of the candidate per tag
  gridIdx @2 :UInt16;             # sequential index of the grid/decoding per␣
↪candidate
  xpos @3 :UInt16;                # x coordinate of the grid center
  ypos @4 :UInt16;                # y coordinate of the grid center
  xposHive @5 :UInt16;            # x coordinate of the grid center wrt. the hive
  yposHive @6 :UInt16;            # y coordinate of the grid center wrt. the hive
  zRotation @7 :Float32;          # rotation of the grid in z plane
  yRotation @8 :Float32;          # rotation of the grid in y plane
  xRotation @9 :Float32;          # rotation of the grid in x plane
  lScore @10 :Float32;            # roi score (Localizer)
  eScore @11 :UInt16;             # ellipse score (EllipseFitter)
  gScore @12 :Float32;            # grid score (GridFitter)
  decodedId @13 :UInt32;          # decoded id
}
```

DetectionDP

This is the new format for a detection that replaced *DetectionCVP*.

```
struct DetectionDP {
  idx @0 :UInt16;                 # sequential index of the detection, counted␣
↪from 0 for every frame
                                  # the combination (idx, Frame.id) is a global key
  ypos @1 :UInt16;                # y coordinate of the grid center wrt. the image
  xpos @2 :UInt16;                # x coordinate of the grid center wrt. the image
```

(continues on next page)

(continued from previous page)

```
  yposHive @3 :UInt16;            # y coordinate of the grid center wrt. the hive
  xposHive @4 :UInt16;            # x coordinate of the grid center wrt. the hive
  zRotation @5 :Float32;          # rotation of the grid in z plane
  yRotation @6 :Float32;          # rotation of the grid in y plane
  xRotation @7 :Float32;          # rotation of the grid in x plane
  radius @8 :Float32;             # radius of the tag
  localizerSaliency @9 :Float32; # saliency of the localizer network
  decodedId @10 :List(UInt8);     # the decoded id, the bit probabilities are␣
↪discretised to 0-255.
                                  # p(first bit == 1) = decodedId[0] / 255. bits␣
↪are in most significant
                                  # bit first order starting at the 1 o'clock␣
↪position on the tag in
                                  # clockwise orientation.
                                  # see https://arxiv.org/pdf/1611.01331.pdf␣
↪Figure 1(a) for a graphical
```

DetectionTruth

This is the format for manually generated truth data that might be generated via the Editor GUI.

```
  descriptor @11 :List(UInt8);    # visual descriptor of the detection. ordered␣
↪from most
                                  # significant eight bits to least significant␣
↪eight bits.
}

struct DetectionTruth {
  idx @0 :UInt16;                 # sequential index of the detection, counted␣
↪from 0 for every frame
                                  # the combination (idx, Frame.id) is a global key
  ypos @1 :UInt16;                # y coordinate of the grid center wrt. the image
  xpos @2 :UInt16;                # x coordinate of the grid center wrt. the image
  yposHive @3 :UInt16;            # y coordinate of the grid center wrt. the hive
  xposHive @4 :UInt16;            # x coordinate of the grid center wrt. the hive
  decodedId @5 :Int32;            # decoded id by human
  readability @6 :Grade;          # tags might be visible or (partially) obscured
  enum Grade {                    # ranks for evaluation of a tag's readability␣
↪are:
    unknown @0;                   #  - not considered or evaluted
    completely @1;                #  - completely visible **and** human readable
```

CHAPTER 2

Converting

We provide convenience functions to read a *bb_binary* `Repository` to NumPy Arrays and DataFrames. Or to create a `Repository` from existing data via Pandas Dataframes.

You might need them if you want to

- create *bb_binary* Repositories from existing data (like ground truth data)
- take a quick glance at a subset of the data (one that fits into memory)
- experiment with new features

Warning: The convenience functions are **not** designed for performance. When working with huge datasets it is recommended to use the `Repository`.

2.1 Convert *bb_binary* to NumPy Array

To generate NumPy Arrays or Pandas DataFrames we provide a convenience function. Here is an example how to read all the frames and detection data to NumPy:

```
import numpy as np
import pandas as pd
from bb_binary import Repository, convert_frame_to_numpy

repo = Repository("some/path/to/a/repo")

arr = None
for frame, fc in repo.iter_frames():
    tmp = convert_frame_to_numpy(frame)
    arr = tmp if arr is None else np.hstack((arr, tmp))
```

Sometimes we also need fields from the `FrameContainer`. You can add those fields using the *add_cols* argument. This works for every other singular values or lists:

```
arr = None
for frame, fc in repo.iter_frames():
    tmp = convert_frame_to_numpy(frame, add_cols={'camId': fc.camId})
    arr = tmp if arr is None else np.hstack((arr, tmp))
```

It is also possible to restrict the output to a set of fields that should be extracted. When using the *keys* argument you need to specify *detectionsUnion* as `Frame` key when you want to extract detections:

```
arr = None
frame_keys = ('frameId', 'frameIdx', 'timedelta', 'timestamp', 'dataSourceIdx')
detection_keys = ('idx', 'xpos', 'ypos')
keys = frame_keys + detection_keys
for frame, fc in repo.iter_frames():
    tmp = convert_frame_to_numpy(frame, keys=keys + ('detectionsUnion',))
    arr = tmp if arr is None else np.hstack((arr, tmp))
```

2.2 Convert *bb_binary* to Pandas DataFrame

Usually you could directly create a Pandas DataFrame from a NumPy Array:

```
data = pd.DataFrame(arr)
```

Assuming that we have standard pipeline output with `DetectionDP` you have to convert list like fields separately (because Pandas has problems with lists in fields):

```
list_like_fields = set(['decodedId', 'descriptor'])
data = pd.DataFrame(arr[list(set(arr.dtype.fields.keys()) - list_like_fields)])
for field in list_like_fields:
    data[field] = pd.Series([list(list_field) for list_field in arr[field]])
```

2.3 Convert a Pandas DataFrame to *bb_binary*

When you have data from other sources like ground truth data, or you need to generate a `Repository` for testing purposes or feature evaluation you might need this converting function. All the column names in the Pandas DataFrame are matched to field names. You have to specify the *detectionUnion* type and also the camera id, because each `FrameContainer` is specific for a camera.

The *frame_offset* is used to generate unique `Frame` ids:

```
from bb_binary import Repository, build_frame_container_from_df
cam_ids = (0, 2)
offset = 0
for cid in cam_ids:
    fc, offset = build_frame_container_from_df(df, 'detectionsTruth', cid, frame_
↪offset=offset)
    repo.add(fc)
```

2.4 Function Documentation

build_frame_container(*from_ts*, *to_ts*, *cam_id*, *hive_id=None*, *transformation_matrix=None*, *data_source_fname=None*, *video_preview_fname=None*)

Builds a *FrameContainer*

Parameters

- **from_ts** (*int or float*) – Timestamp of the first frame
- **to_ts** (*int or float*) – Timestamp of the last frame
- **cam_id** (*int*) – id of camera

Keyword Arguments

- **hive_id** (*Optional int*) – id of the hive
- **transformation_matrix** (*Optional iterable with floats*) – Transformation matrix for coordinates
- **data_source_fname** (*Optional str or list of str*) – Filename(s) of the data source(s).
- **video_preview_fname** (*Optional str or list of str*) – Filename(s) of preview videos. Have to allign to *data_source_fname*!

build_frame_container_from_df(*dfr*, *union_type*, *cam_id*, *frame_offset=0*)

Builds a *FrameContainer* from a Pandas DataFrame.

Operates differently from *build_frame_container()* because it will be used in a different context where we have access to more data.

Column names are matched to *Frame* and *Detection** attributes. Set additional *FrameContainer* attributes like *hiveId* in the return value.

Parameters

- **dfr** (pd.DataFrame) – Pandas dataframe with detection data
- **union_type** (*str*) – the type of detections e.g. *detectionsTruth*
- **cam_id** (*int*) – id of camera, also used as *FrameContainer* id

Keyword Arguments **offset** (*frame*) – offset for unique frame ids

Returns

tuple containing:

- **frame container** (*FrameContainer*): converted data from *dfr*
- **new offset** (int): number of frames (could be used as *frame_offset*)

Return type tuple

convert_frame_to_numpy(*frame*, *keys=None*, *add_cols=None*)

Returns the frame data and detections as a numpy array from the *frame*.

Note: The frame id is identified in the array as *frameId* instead of *id*!

Parameters **frame** (*Frame*) – datastructure with frame data from capnp.

Keyword Arguments

- **keys** (*Optional iterable of str*) – only these keys are converted to the np array.
- **add_cols** (*Optional dict*) – additional columns for the np array, use either a single value or a sequence of correct length.

Returns a structured numpy array with frame and detection data.

Return type numpy array (np.array)

CHAPTER 3

Parsing

This submodule contains a collection of helper to parse filenames and some fields in *bb_binary* repositories to Python datastructures.

There are also some helper to convert some data representations like timeformats or ids.

dt_to_str(*dt*)

get_fname(*camIdx*, *dt*)

get_video_fname(*camIdx*, *begin*, *end*)

get_timezone()

int_id_to_binary(*int_id*, *nb_bits=12*)
Helper to convert an id represented as integer to a bit array.

> **Warning:** This function uses big-endian notation whereas *DetectionDP* uses little-endian notation for *decodedId*.

Parameters **int_id** (*int*) – the integer id to convert to a bit array

Keyword Arguments **nb_bits** (*Optional int*) – number of bits in the bit array

Returns the bit array in big-endian notation

Return type np.array

binary_id_to_int(*decoded_id*, *threshold=0.5*, *endian='big'*)
Helper to convert an id represented as bit array to an integer.

> **Warning:** *DetectionDP* uses little-endian notation for *decodedId*.

Note: This is a generic solution to the problem. If have to decode a lot of ids take a look on numpy.packbits() and implement a vectorized version.

Parameters **decoded_id** (list of int or float) – id as bit array

Keyword Arguments

- **threshold** (*Optional float*) – *decoded_id* values >= threshold are interpreted as 1
- **endian** (*Optional str*) – use either 'big' or 'little', default is 'big'

Returns the decoded id represented as integer

Return type int

parse_cam_id(*fname*)

parse_fname(*fname*)

parse_image_fname_beesbook(*fname*)

parse_image_fname_iso(*fname*)

parse_image_fname(*fname*, *format='auto'*)

parse_video_fname(*fname*, *format='auto'*)

to_datetime(*t*)

to_timestamp(*dt*)

CHAPTER 4

Repository

The Repository class allows to read and write to *bb_binary* data stores.

The *bb_binary* data is splitted into several files that are organized in subfolders via date and time. A `Repository` manages these *bb_binary* datafiles and provides methods to **add** new `FrameContainer`, or iterate through `Frame`.

This class provides performant access to a *bb_binary* data store but you will have to parse the data by yourself. You might use some helpers from *Parsing*.

load_frame_container(*fname*)
Loads `FrameContainer` from this filename.

class Repository(*root_dir*, *minute_step=None*)
Bases: `object`

The Repository class manages multiple *bb_binary* files. It creates a directory layout that enables fast access by the timestamp.

add(*frame_container*)
Adds the *frame_container* of type `FrameContainer` to the repository.

open(*timestamp*, *cam_id*)
Finds and load the `FrameContainer` that matches the *timestamp* and *cam_id*.

find(*ts*, *cam=None*)
Returns all files that includes detections to the given timestamp *ts*.

Todo: UTC timestamps! Generall

---

iter_fnames(*begin=None*, *end=None*, *cam=None*, *fname_filter=None*)
Returns a generator that yields filenames in sorted order.

From *begin* to *end*.

Parameters

- **begin** (*Optional timestamp*) – The first filename contains at least one frame with a timestamp greater or equal to *begin*. If *begin* is not set, it will start with the earliest file.

- **end** (*Optional timestamp*) – The last filename contains at least one frame with a timestamp smaller then *end*. If not set, it will continue until the last file.
- **cam** (*Optional int*) – Only yield filenames with this cam id.
- **fname_filter** (*Optional function*) – only yield fnames for which the function returns true

Example:

```
Files:         A     B     C         D     E
Frames:    |-----|-----|-----|   |-----|-----|

               begin       end
```

This should return the files A, B and C. If *begin* and *end* are *None*, then all will be yield.

iter_frames(*begin=None*, *end=None*, *cam=None*, *frame_filter=None*)

Yields frames with their corresponding FrameContainers. The FrameContainers are ordered in time. Beware that individual frames may not be in order if cam is not set.

From *begin* to *end*.

Parameters

- **begin** (*Optional timestamp*) – select frames with begin <= timestamp.
- **with smallest timestamp in repository if not set.** (*Starts*) –
- **end** (*Optional timestamp*) – select frames with timestamp < end.
- **with biggest timestamp in repository if not set.** (*Ends*) –
- **cam** (*Optional int*) – only yield filenames with this cam id.
- **frame_filter** (*Optional function*) – only yield frames for which the function returns true

Returns

tuple containing:

iterator (iterable): iterator with Frames FrameContainer (FrameContainer): the corresponding FrameContainer for each frame.

Return type (tuple)

class TimeInterval(*begin*, *end*)

Bases: object

Helper class to represent time intervals.

in_interval(*dt*)

CHAPTER 5

Schema

We are using the Python Implementation of Cap'n Proto. The original file could be found in our GitHub Repository.

```
@0xa7d4a096084cee0e;

using Java = import "java.capnp";
$Java.package("de.fuberlin.biorobotics");
$Java.outerClassname("BeesBook");


struct DetectionCVP {
  idx @0 :UInt16;                # sequential index of the detection, counted from 0␣
↪for every frame
                                 # the combination (idx, Frame.id) is a global key
  candidateIdx @1 :UInt16;       # sequential index of the candidate per tag
  gridIdx @2 :UInt16;            # sequential index of the grid/decoding per candidate
  xpos @3 :UInt16;               # x coordinate of the grid center
  ypos @4 :UInt16;               # y coordinate of the grid center
  xposHive @5 :UInt16;           # x coordinate of the grid center wrt. the hive
  yposHive @6 :UInt16;           # y coordinate of the grid center wrt. the hive
  zRotation @7 :Float32;         # rotation of the grid in z plane
  yRotation @8 :Float32;         # rotation of the grid in y plane
  xRotation @9 :Float32;         # rotation of the grid in x plane
  lScore @10 :Float32;           # roi score (Localizer)
  eScore @11 :UInt16;            # ellipse score (EllipseFitter)
  gScore @12 :Float32;           # grid score (GridFitter)
  decodedId @13 :UInt32;         # decoded id
}

struct DetectionDP {
  idx @0 :UInt16;                # sequential index of the detection, counted from 0␣
↪for every frame
                                 # the combination (idx, Frame.id) is a global key
  ypos @1 :UInt16;               # y coordinate of the grid center wrt. the image
  xpos @2 :UInt16;               # x coordinate of the grid center wrt. the image
```

(continues on next page)

(continued from previous page)

```
  yposHive @3 :UInt16;            # y coordinate of the grid center wrt. the hive
  xposHive @4 :UInt16;            # x coordinate of the grid center wrt. the hive
  zRotation @5 :Float32;          # rotation of the grid in z plane
  yRotation @6 :Float32;          # rotation of the grid in y plane
  xRotation @7 :Float32;          # rotation of the grid in x plane
  radius @8 :Float32;             # radius of the tag
  localizerSaliency @9 :Float32; # saliency of the localizer network
  decodedId @10 :List(UInt8);     # the decoded id, the bit probabilities are␣
↪discretised to 0-255.
                                  # p(first bit == 1) = decodedId[0] / 255. bits are␣
↪in most significant
                                  # bit first order starting at the 1 o'clock position␣
↪on the tag in
                                  # clockwise orientation.
                                  # see https://arxiv.org/pdf/1611.01331.pdf Figure␣
↪1(a) for a graphical
                                  # representation
  descriptor @11 :List(UInt8);    # visual descriptor of the detection. ordered from␣
↪most
                                  # significant eight bits to least significant eight␣
↪bits.
}

struct DetectionTruth {
  idx @0 :UInt16;                 # sequential index of the detection, counted from 0␣
↪for every frame
                                  # the combination (idx, Frame.id) is a global key
  ypos @1 :UInt16;                # y coordinate of the grid center wrt. the image
  xpos @2 :UInt16;                # x coordinate of the grid center wrt. the image
  yposHive @3 :UInt16;            # y coordinate of the grid center wrt. the hive
  xposHive @4 :UInt16;            # x coordinate of the grid center wrt. the hive
  decodedId @5 :Int32;            # decoded id by human
  readability @6 :Grade;          # tags might be visible or (partially) obscured
  enum Grade {                    # ranks for evaluation of a tag's readability are:
    unknown @0;                   #  - not considered or evaluted
    completely @1;                #  - completely visible **and** human readable
    partially @2;                 #  - only partially visible and therefore **not**␣
↪human readable
    none @3;                      #  - **not** visible at all
  }
}


# Corresponds to an image in the video.
struct Frame {
  id @0 :UInt64;                  # global unique id of the frame
  dataSourceIdx @1:UInt32;        # the frame is from this data source
  frameIdx @6 :UInt32;            # sequential increasing index for every data source.
  timestamp @2 :Float64;          # unix time stamp of the frame
  timedelta @3 :UInt32;           # time difference between this frame and the frame␣
↪before in microseconds
  detectionsUnion : union {
    detectionsCVP   @4 :List(DetectionCVP);   # detections format of the old computer␣
↪vision pipeline
    detectionsDP    @5 :List(DetectionDP);    # detections format of the new␣
↪deeppipeline
    detectionsTruth @7 :List(DetectionTruth); # detections format of ground truth data
```

(continues on next page)

(continued from previous page)

```
  }
}

struct DataSource {
    idx @0 :UInt32;                  # the index of the data source
    filename @1 :Text;               # filename of the data source
    videoPreviewFilename @2 :Text;   # (optional) filename of the preview video
}

# Corresponds to a video
struct FrameContainer {
  id @0 :UInt64;                     # global unique id of the frame container
  dataSources @1:List(DataSource);   # list of data sources (videos / images)
  fromTimestamp @2 :Float64;         # unix timestamp of the first frame
  toTimestamp @3 :Float64;           # unix timestamp of the last frame
  frames @4 :List(Frame);            # frames must be sorted by in the order they␣
↪where recorded.
  camId @5 :UInt16;                  # the cam number
  hiveId @6 :UInt16;                 # the id of the hive
  transformationMatrix @7 :List(Float32);
                                     # the transformation matrix from image␣
↪coordinates to hive coordinates.
                                     # The matrix is of dimension 4x4 and stored this␣
↪way
                                     #     1 | 2 | 3 | 4
                                     #     5 | 6
                                     #           ...
                                     #              15| 16
}
```

We switched to Cap'n Proto as data interchange format to tackle three of our main problems:

1. IO-Operations have been a huge bottleneck in our Pipeline
2. The format has an inherent data Schema
3. There are implementations for all the major programming languages

As most of our applications are written in Python this is the recommended interface. It is also **possible** to use *bb_binary* from other languages like C, C++ and Java, but they are not supported right now.

The Python Module is divided into the three submoduls *Converting*, *Parsing* and the *Repository* Class.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index