Data Formats Manual

This documents provides a detailed overview on the used data formats in the Bimanual Actions Dataset. If you encounter any issues or problems with this document or with the dataset, please feel free to contact Christian Dreher.

As of 26 Jun 2020. There may be an updated version available at bimanual-actions.humanoids.kit.edu.

Action Label Mapping

Refer to the following table for a mapping of action label IDs and their symbolic name.

#	Action	Description
0	idle	The hand does nothing semantically meaningful
1	approach	The hand approaches an object which is going to be relevant
2	retreat	The hand retreats from an object after interacting with it
3	lift	The hand lifts an object to allow using it
4	place	The hand places an object after using it
5	hold	The hand holds an object to ease using it with the other
6	pour	The hand pours something from the grasped object
7	cut	The hand cuts something with the grasped object
8	hammer	The hand hammers something with the grasped object
9	saw	The hand saws something with the grasped object
10	stir	The hand stirs something with the grasped object
11	screw	The hand screws something with the grasped object
12	drink	The hand is used to drink with the grasped object
13	wipe	The hand wipes something with the grasped object

Object Class Label Mapping

Refer to the following table for a mapping of object class label IDs and their symbolic name.

#	Object	Description
0	bowl	Either a small green bowl or a bigger orange bowl. Used only in the kitchen
1	knife	A black knife. Used only in the kitchen
2	screwdriver	A screwdriver. Used only in the workshop
3	cutting board	A wooden cutting board. Used only in the kitchen
4	whisk	A whisk. Used only in the kitchen
5	hammer	A hammer. Used only in the workshop
6	bottle	Either a white bottle, a smaller black bottle, or a green bottle. Used only in the kitchen
7	cup	Either a yellow, blue or red cup. Used only in the kitchen
8	banana	A banana. Used only in the kitchen
9	cereals	A pack of cereals. Used only in the kitchen
10	sponge	Either a big yellow sponge, or a smaller green one. Used only in the kitchen
11	wood	A piece of wood. Either a long one, or a smaller one. Used only in the workshop

#	Object	Description
12	saw	A saw. Used only in the workshop
13	hard drive	A hard drive. Used only in the workshop
14	left hand	The subject's left hand
15	right hand	The subject's right hand

Object Relations Label Mapping

Refer to the following table for a mapping of object relation IDs to their symbolic name.

Note: The following spatial relations were originally defined by Ziaeetabar *et al.* [1].

#	Relation	Description
0	contact	Spatial relation. Objects are in contact
1	above	Spatial relation (static). One object is above the other
2	below	Spatial relation (static). One object is below the other
3	left of	Spatial relation (static). One object is left of the other
4	right of	Spatial relation (static). One object is right of the other
5	behind of	Spatial relation (static). One object is behind of the other
6	in front of	Spatial relation (static). One object is in front of the other
7	inside	Spatial relation (static). One object is inside of another
8	surround	Spatial relation (static). One object is surrounded by another
9	moving together	Spatial relation (dynamic). Two objects are in contact and move together
10	halting together	Spatial relation (dynamic). Two objects are in contact but do not move
11	fixed moving together	Spatial relation (dynamic). Two objects are in contact and only one moves
12	getting close	Spatial relation (dynamic). Two objects are not in contact and move towards each other
13	moving apart	Spatial relation (dynamic). Two objects are not in contact and move apart from each other
14	stable	Spatial relation (dynamic). Two objects are not in contact and their distance stays the same
15	temporal	Temporal relation. Connects observations of one object instance over consecutive frames

RGB-D Camera Normalisation

The camera angles vary slightly, depending on when the recordings were taken. The file `bimacs_rgbd_data_cam_norm.json` contains the necessary data to perform a normalisation, i.e. to rotate the point clouds to account for the tilted camera, and offsets to center the world frame on the table.

The normalisations are stored in a JSON file, where key indices are stored in an integer array `key_indices`. These key indices denote, at which recording numbers the camera parameters have changed. The corresponding parameters can be obtained by looking for the next biggest key index for any given recording number, to then look up that index in the `map`. For example, for recording number 42, the parameters of key index 90 would be the correct ones. The index 0 is just a dummy to ease automatic processing.

The X-axis of the point cloud is rotated by the negative angle in `angle`. The offsets `offset_rl` (right/left), `offset_h` (height) and `offset_d` (depth) center the world frame on the table. The angle is in degree, all offsets are in millimeters.

RGB-D Video Format

The RGB and depth videos were recorded with a PrimeSense Carmine 1.09 and divided into separate folders, namely `rgb` and `depth`. The recordings are organised in subfolders. The first level of folders contain all recordings of a specific subject (i.e. `subject_x`), the second a specific task (e.g. `task_4_k_wiping` or `task_9_w_sawing`), and the third a specific take or repetition (i.e. `take_x`).

Each recording is a folder which contains a file `metadata.csv` and one or more chunk folders `chunk_x`. These chunks, in turn, contain a certain amount of frames (image files) from the recording. The `metadata.csv` file contains several variables relevant for the recording and may look like this:

```
name,type,value
fps,unsigned int,30
framesPerChunk,unsigned int,100
frameCount,unsigned int,427
frameWidth,unsigned int,640
frameHeight,unsigned int,480
extension,string,.png
```

The first column denotes the variable name, the second the type of the variable, and the third the value of the variable. The variable `fps` denotes the frame rate for the recording, while `frameWidth` and `frameHeight` denote the resolution of the recording, and `frameCount` the amount of frames. The extension of the image files is encoded into `extension`. Because the image frames are chunked, it is also important to know the amount of frames per chunk. This is stored in `framesPerChunk`.

For this dataset, the variables `extension`, `framesPerChunk`, `frameWidth` and `frameHeight` can be assumed constant, as the frame dimensions did not change and the other parameters were not changed during recording.

Action Ground Truth

This data format is used in **Action ground truth**. For each recording, there is a JSON file containing the action segmentation for both hands. Let's consider a simple example segmentation for a recording with 1015 frames:

```
{
  "right_hand": [0, 0, 1015],
  "left_hand": [0, 0, 1015]
}
```

For each hand, there is an array of elements, and the length of the array is always odd. All even elements are always integer values and depict key frames. All odd elements are either integer or `null`, and depict the action label ID (if integer), or that no action is associated (if `null`).

Note: If you are looking for the **action label ID mapping**, please refer to the corresponding section.

The example above therefore translates to: "For both hands, there is an action segment with the ID '0' beginning from frame 0 and ending before frame 1015" ('0' is the action label ID of 'idle'). A graphical representation with action label IDs substituted with the actual action labels would look like this:

```
Frame number:   0        ...       1015
                |                    |
  Right hand:   [       idle        )[
   Left hand:   [       idle        )[
```

Now let's consider that the right hand begins holding (the action label ID of 'hold' is '5') an object at frame 500. This would change the previous example to:

```
{
  "right_hand": [0, 0, 500, 5, 1015],
  "left_hand": [0, 0, 1015]
}
```

Or graphically:

```
Frame number:   0    ...    500    ...    1015
                |            |             |
  Right hand:   [   idle    )[    hold    )[
   Left hand:   [           idle          )[
```

2D Human Body Pose Data

The human body pose data is organised in subfolders. The first level of folders contain all pose data of a specific subject (i.e. `subject_x`), the second a specific task (e.g. `task_4_k_wiping` or `task_9_w_sawing`), and the third a specific take or repetition (i.e. `take_x`). The pose data for an individual frame is saved in a corresponding JSON file in the folder `body_pose`.

The root element of each JSON file is a list with one element, which is an object. The properties of this object are again objects, which encode the confidence, the label (e.g. `RAnkle` or `Neck`), and the coordinates in pixels of a given key point of the pose. If the confidence is `0.0`, that pose key point was not found in the image.

2D Human Hand Pose Data

The human hand pose data is organised in subfolders. The first level of folders contain all pose data of a specific subject (i.e. `subject_x`), the second a specific task (e.g. `task_4_k_wiping` or `task_9_w_sawing`), and the third a specific take or repetition (i.e. `take_x`). The pose data for an individual frame is saved in a corresponding JSON file in the folder `hand_pose`.

The root element of each JSON file is a list with one element, which is an object. The properties of this object are again objects, which encode the confidence, the label (e.g. `LHand_15` or `RHand_4`), and the coordinates relative to the image width and height of a given key point of the pose. If the confidence is `0.0`, that pose key point was not found in the image.

2D Object Bounding Boxes

The 2D object bounding box data is organised in subfolders. The first level of folders contain all pose data of a specific subject (i.e. `subject_x`), the second a specific task (e.g. `task_4_k_wiping` or `task_9_w_sawing`), and the third a specific take or repetition (i.e. `take_x`). The 2D object bounding box data for an individual frame is saved in a corresponding JSON file in the folder `2d_objects`.

The root element of each JSON file is a list of JSON objects, and each JSON object represents the bounding box of a detected object. Such an object looks like this:

```
{
    "bounding_box": {
        "h": 0.22833597660064697,
        "w": 0.07694989442825317,
        "x": 0.7088799476623535,
        "y": 0.6917555928230286
    },
    "candidates": [
        {
            "certainty": 0.9995384812355042,
            "class_index": 11,
            "class_name": "cereals",
            "colour": [
                0,
                255,
                111
            ]
        }
    ],
    "class_count": 16,
    "object_name": ""
}
```

The property `bounding_box` denotes the bounding box, with `x` and `y` being the coordinates of the center of the bounding box, and `w` and `h` its width and height, respectively. These values are relative to the input image's height and width.

Sometimes there are several object class candidates for a detected object, but most of the times it is only one. The candidates are listed in the `candidates` property. Here, the property `class_index` and `class_name` store the possible object class candidate for that bounding box, and the property `certainty` the certainty as estimated by Yolo. The property `colour` encodes an RGB colour unique to the given class ID for visualisation purposes.

Note: If you are looking for the **object class label ID mapping**, please refer to the corresponding section.

The total number of classes is stored in the `class_count` property.

3D Object Bounding Boxes

The 3D object bounding box data is organised in subfolders. The first level of folders contain all pose data of a specific subject (i.e. `subject_x`), the second a specific task (e.g. `task_4_k_wiping` or `task_9_w_sawing`), and the third a specific take or repetition (i.e. `take_x`). The 3D object bounding box data for an individual frame is saved in a corresponding JSON file in the folder `3d_objects`.

The root element of each JSON file is a list of JSON objects, and each JSON object represents the bounding box of a detected object. Such an object looks like this:

```
{
    "bounding_box": {
        "x0": -78.30904388427734,
        "x1": 36.15547561645508,
        "y0": -789.953125,
        "y1": -749.12744140625,
        "z0": -1120.7742919921875,
        "z1": -976.8253784179688
    },
    "certainty": 0.9998389482498169,
    "class_index": 10,
    "class_name": "banana",
    "colour": [
        0,
        255,
        31
    ],
    "instance_name": "banana_2",
    "past_bounding_box": {
        "x0": -78.34809112548828,
        "x1": 36.506187438964844,
        "y0": -789.953125,
        "y1": -749.7930908203125,
        "z0": -1120.77294921875,
        "z1": -976.9805297851563
    }
}
```

The extents of the 3D bounding box are defined in the `bounding_box` property, where `x0`, `x1`, `y0`, `y1`, `z0`, and `z1` denote the minimum and maximum extents for the x, y, and z axis respectively. Similarly, the property `past_bounding_box` contains the bounding box of the same object 333 ms in the past, which allows to compute dynamic spatial relations (cf. [1]).

The properties `certainty`, `class_index`, `class_name`, and `colour` were assumed from the 2D bounding boxes.

The property `instance_name` holds an unique identifier for the recording to make sure that the same object can be tracked across several frames.

Note: It is possible that the instance name suggests a different class name than denoted by the `class_id` or `class_name` properties. This is not a bug, but a result of allowing candidates for each bounding box in the 2D object detection.

The reason for this is that, whenever a new object is detected, it will be assigned a globally unique identifier consisting of the most probable class and a unique number. If if later turns out that the actual class is most likely another one, the assigned class may change, however, to ensure that objects remain trackable, an assigned identifier will never change.

3D Spatial Relations

The spatial relations data is organised in subfolders. The first level of folders contain all pose data of a specific subject (i.e. `subject_x`), the second a specific task (e.g. `task_4_k_wiping` or `task_9_w_sawing`), and the third a specific take or repetition (i.e. `take_x`). The spatial relations data for an individual frame is saved in a corresponding JSON file in the folder `spatial_relations`.

The root element of each JSON file is a list of JSON objects, and each JSON object represents one spatial relation between a pair of objects. Such an object looks like this:

```
{
    "object_index": 0,
    "relation_name": "behind of",
    "subject_index": 1
}
```

The properties `object_index` and `subject_index` are the respective object and subject (from a grammatical point of view) of a relation, for example: "The *bowl (object)* is *behind of* the *cup (subject)*".

The property `relation_name` is the label of the relation, in plain text.

Note: If you are looking for the **relation label ID mapping**, please refer to the corresponding section.

The list of relations is explicit. That is, if there are any implicit relations, they will be in that list as well.

Object Detection Data

The file `bimacs_object_detection_data.zip` is structured like this:

```
bimacs_object_detection_data/
 |- images/
 |   `- *.jpg
 |
 |- labels/
 |   `- *.txt
 |
 |- images_index.txt
 |- labels_index.txt
 `- object_class_names.txt
```

The folder `images/` contains all training images in the dataset, and the corresponding ground truth can be found in the `labels/` folder. The ground truth files have the same filename as the image in the dataset, just the file extension differs. The files `images_index.txt` and `labels_index.txt` contain a list of all files inside the `images/` and `labels/` folders respectively.

The file format for the ground truth is the format Darknet uses. Each ground truth bounding box is denoted in a line as a 5-tuple separated by white spaces:

```
<object class ID> <box center X> <box center Y> <box width> <box height>
```

The object class name corresponding to the ID can be found in `object_class_names.txt`. Please note that this file contains two unsued object classes (`unused1` and `unused2`). You can safely ignore them. The extents of the bounding box, given by the coordinates of the center of the bounding box and its width and height are relative to the image width and height. That is, these values should range from 0 to 1.

Note: If you just want to **train** Yolo, download the training environment instead (Yolo environment for training). The dataset is already included there.

Yolo Training Environment

The file `bimacs_yolo_train_setup.zip` contains the training environment to be used with Yolo together with the object detection dataset and is structured like this:

```
bimacs_object_detection_data/
 |- images/
 |   `- *.jpg
 |
 |- labels/
 |   `- *.txt
 |
 |- weightfiles/
 |   `- darknet53.conv.74
 |
 |- images_index.txt
 |- labels_index.txt
 |- object_class_names.txt
 |- net.cfg
 |- train_start.sh
 `- train_resume.sh
```

The folder `images/` contains all training images in the dataset, and the corresponding ground truth can be found in the `labels/` folder. The ground truth files have the same filename as the image in the dataset, just the file extension differs. The files `images_index.txt` and `labels_index.txt` contain a list of all files inside the `images/` and `labels/` folders respectively.

The file `weightfiles/darknet53.conv.74` is a pre-trained weights file supplied by pjreddie.com (Joseph Redmon's homepage).

The file `train_setup.cfg` contains the setup for the training for Yolo, while the file `net.cfg` is a configuration of the used network architecture within Darknet.

There are also 2 script files provided, namely `train_start.sh` to easily start the training, and `train_resume.sh` to resume the training (loading a backup).

The object class name corresponding to the ID can be found in `object_class_names.txt`. Please note that this file contains two unsued object classes (`unused1` and `unused2`). You can safely ignore them.

To use this training setup, extract the zip file and move the folder `bimacs_object_detection_data` into the root folder of your Darknet installation. Then, in the root of your Darknet installation, run `./bimacs_object_detection_data/train_start.sh` to begin training. Backup and milestone weight files of the training will be written into the folder `./bimacs_object_detection_data/backup/`. To resume the training, run `./bimacs_object_detection_data/train_resume.sh`

Note: If you just want to **run** Yolo, download the execution environment instead (Yolo environment for execution).

Yolo Execution Environment

The file `bimacs_yolo_run_setup.zip` contains the runtime environment to be used with Yolo and is structured like this:

```
bimacs_object_detection_data/
 |- weightfiles/
 |   `- net_120000.weights
 |
 |- object_class_names.txt
 `- net.cfg
```

The file `weightfiles/net_120000.weights` is the weight file pre-trained on the objects used in the dataset.

The file `net.cfg` is the configuration of the used network architecture within Darknet.

The object class name corresponding to the ID can be found in `object_class_names.txt`. Please note that this file contains two unsued object classes (`unused1` and `unused2`). You can safely ignore them.

References

[1] F. Ziaeetabar, T. Kulvicius, M. Tamosiunaite, and F. Wörgötter, “Recognition and Prediction of Manipulation Actions Using Enriched Semantic Event Chains,” Robotics and Autonomous Systems (RAS), vol. 110, pp. 173–188, Dec. 2018.