Technical Report

MVL Lab5: Multi-modal Indoor Person Localization Dataset

Rok Mandeljc, Stanislav Kovačič, Matej Kristan, Janez Perš

MVL Technical Report Series **FE–LSV–02/12**

Machine Vision Laboratory **December 2012**

University of Ljubljana
Faculty of Electrical Engineering
Tržaška 25
SI-1000 Ljubljana
Slovenia

Technical Report FE–LSV–02/12

MVL Lab5: Multi-modal Indoor Person Localization Dataset

Rok Mandeljc, Stanislav Kovačič, Matej Kristan, Janez Perš

rok.mandeljc@fe.uni-lj.si

December 27, 2012

Abstract: This technical report describes *MVL Lab5*, a multi-modal indoor person localization dataset. The dataset contains a sequence of video frames obtained from four calibrated and time-synchronized video cameras and location event data stream from a commercially-available radio-based localization system. The scenario involves five individuals walking around a realistically cluttered room. Provided calibration data and ground truth annotations enable evaluation of person detection, localization and identification approaches. These can be either purely computer-vision based, or based on fusion of video and radio information. This document is intended as the primary documentation source for the dataset, presenting its availability, acquisition procedure, and organization. The structure and format of data is described in detail, along with documentation for bundled Matlab code and examples of its use.

Contents

1 Introduction

This technical report describes *MVL Lab5*, a multi-modal indoor person localization dataset, which was captured as part of our work on multi-modal person detection, localization and identification [1, 2]. The dataset was captured using four calibrated and time-synchronized video cameras and Ubisense localization system [3], which is based on Ultra-Wideband radio technology. A 6.5-minute sequence involving five individuals walking around a realistically cluttered room was captured and manually annotated. The main part of dataset are undistorted down-scaled (512×384) video frames and location events data from radio-based system, along with calibration data and ground-truth annotations. In addition, full-sized (2048×1536) undistorted and original video frames are also provided. At the time of writing, this is the first publicly-available dataset that comprises both video frames from multiple calibrated views and location events from a radio-based localization system. We hope it will prove useful to other researchers in their work, primarily in the field of person detection, identification and tracking, but also possibly in other fields.

The remainder of this report is structured as follows. In Section 2, we provide information on the dataset's availability, such as terms of use and download location. The dataset's acquisition procedure, scenario and camera calibration procedure are detailed in Section 3. Section 4 provides description of data and its format, required for cases when one does not wish to use bundled Matlab code. The latter is described in Section 5 in form of use examples, which should point the reader to the most relevant functions. Finally, we briefly provide some details regarding use of dataset in our papers (Section 6) and conclude the report in Section 7.

2 Availability

The dataset is provided *free of charge for academic and non-commercial use*, and the bundled Matlab code is provided under GPLv2 license[1]. If you use any part of the dataset, please *cite the paper [1].*

The *primary download address* for the dataset is the authors' web site [4]. Due to its size, we broke the dataset into several archives, so that users can download the ones

[1] `http://www.gnu.org/licenses/gpl-2.0.html`

are interested in. All archives are compressed using 7-Zip[2], and larger archives were further split into 500 MB parts.

Table 1: Archives that make up the dataset, their compressed and uncompressed size, and their content. Large archives have been split into 500 MB parts. *Note that down-scaled video frames are considered to be the main part of this dataset, at least as far as task of person detection, localization and identification is concerned.*

File(s)	Compressed	Uncompressed	Contents
data.7z	47.1 MB	55.8 MB	calibration data, ground truth annotations, timestamps and data from radio-based system (**main**)
frames4.7z.XYZ	1.1 GB	1.6 GB	undistorted frames, down-scaled to resolution of 512×384 (**main**)
frames.7z.XYZ	9.2 GB	12.7 GB	undistorted frames at original resolution of 2048×1536 (**optional**)
source.7z.XYZ	6.8 GB	10.6 GB	original frames at resolution of 2048×1536 (**optional**)
code.7z	22.3 kB	114.3 kB	bundled Matlab code (**optional**)

The list of archives and their contents is given in Table 1. One would likely want to download at least undistorted down-scaled frames, which are considered the main part of this dataset, the archive containing calibration, ground truth and data stream from radio-based system and, optionally, bundled Matlab code for quick start and reference. For description of data and its format, see Section 4.

3 Acquisition Procedure

The dataset was acquired using four Ubisense sensors[3] and four Axis P1346 IP cameras[4], which were placed into corners of a 8.0×7.5 *m* room (Figure 1), at height 2.2 *m*. The views from cameras are shown in Figure 2. The room represents a realistically cluttered indoor environment, which is challenging both for radio-based and camera-based person localization. On one hand, difficulties in accurate and reliable camera-based position estimation arise due to occlusions of individuals, both among themselves and by inanimate objects, such as office furniture. On the other hand, the presence of radio-reflective metallic surfaces, in conjunction with obstacles, leads to multipath-related problems in radio-based localization.

Cameras and radio localization system were time-synchronized using a Network Time Protocol (NTP) server. Video from cameras was streamed to two laptops running Windows; we used Axis Media Control[5] for streaming video at resolution 2048×1536 and 20 frames-per-second, using H.264 video codec and ASF container. Location events from the Ubisense system were captured using their On-The-Wire protocol and stored in a file together with the timestamp of their occurrence.

We first converted the obtained video files to a container with a fixed frame rate (AVI), using `ffmpeg`[6]. Then, since their recording did not start simultaneously, we

[2] `http://www.7-zip.org/`

[3] `http://www.ubisense.net/en/resources/factsheets/series-7000-ip-sensors.html`

[4] `http://www.axis.com/products/cam_p1346`

[5] `http://www.axis.com/techsup/software/amc/index.htm`

[6] `http://ffmpeg.org/`

Figure 1: Top view of the room with ground-truth trajectories of the five individuals.

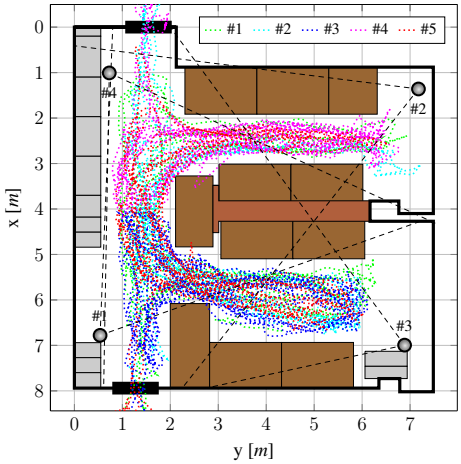


Figure 2: The views from all four cameras. Original images are shown in top row. We use images with lens distortion calibrated and corrected, as shown in the bottom row.

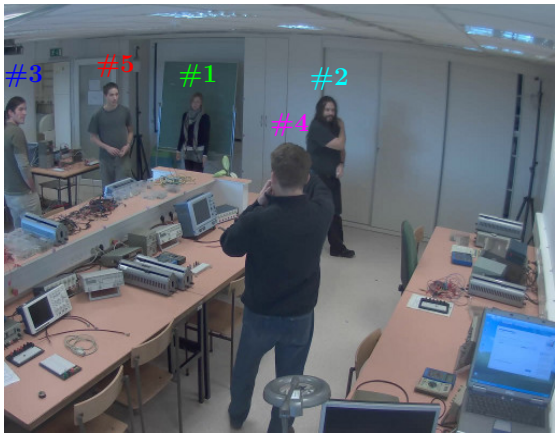
manually synchronized them with the help of timestamps printed on top of the frames. Afterwards, we extracted 7840 video frames from time-aligned videos, again using `ffmpeg`.

To enable association between Ubisense radio events and individual frames, we obtained timestamp values printed on top of the extracted frames. We read the timestamp string using template-based OCR and converted it to Unix epoch with fractional milliseconds. After obtaining timestamps from all four cameras, we computed their median as the timestamp value for a four-view frame.

This way, we also verified the alignment of videos. We computed the deviation of four per-camera timestamps for a frame from their median; we found the mean and standard deviation of the differences across all 7840 four-view frames to be 0.0171 and 0.0182 seconds, respectively. Furthermore, we computed the difference between the maximal and minimal per-view timestamp values for a frame; the mean and standard deviation across all 7840 frames was 0.0544 and 0.0293 seconds, respectively.

Once the frames were undistorted and calibrated (Section 3.2), we manually annotated ground truth positions of individuals in the room, by clicking on their heads in each view and, using calibration information, reconstructing their locations.

Figure 3: Participating individuals.



3.1 Scenario

The scenario of the dataset involves five people (Figure 3), equipped with radio tags, walking around the room. At frame #231, individuals begin to enter the room, and by frame #551, all five of them are inside. After walking around for roughly two minutes, they begin to leave at frame #2861. By frame #3211, everyone has left the room and additional lights have been turned on, changing the illumination. Individuals begin to re-enter the room at frame #3481, and by #3721 everyone is inside again. After walking around for 200 seconds, people begin to remove their radio tags at frame #7761 and then proceed to leave the room. The whole sequence amounts to about 6.5 minutes. The annotated trajectories of individuals are shown in Figure 1.

3.2 Camera Calibration

Our cameras were fitted with wide-angle lenses, which introduce quite significant distortion, as seen on Figure 2. Therefore, we first calibrated lens distortion, using Davide Scaramuzza's Omnidirectional Camera Calibration Toolbox for Matlab (OCamCalib) [5][7]. For calibration, we used five images of a calibration checkerboard, which had 20×20 *mm* squares arranged in an 11×7 pattern.

After lens distortion has been calibrated, we undistorted both images of calibration pattern and extracted video frames. For undistortion, we used C/C++ functions provided by OCamCalib toolbox[8] and OpenCV's `remap` function with bilinear interpolation and black border fill. The used values of F_c parameter for undistortion are listed in Table 2.

From the undistorted images of calibration pattern, we estimated the *intrinsic parameters* using Camera Calibration Toolbox for Matlab [6] by Jean-Yves Bouguet. Then, we took an undistorted frame from each camera and manually annotated points with known 3-D coordinates; this way, we obtained points for calibration of *extrinsic parameters*. We again used Camera Calibration Toolbox for Matlab, which provides a function for estimation of extrinsic parameters from known intrinsic parameters and

[7] `https://sites.google.com/site/scarabotix/ocamcalib-toolbox`

[8] In function `create_perspecive_undistortion_LUT()` we changed the values of `Nxc` and `Nyc` from the half of image width and height to the values of `xc` and `yc` provided by the model.

Table 2: F_c values that were used for image undistortion.

Camera	F_c value
1	1.350
2	1.275
3	1.250
4	1.300

calibration points. This resulted in fully-calibrated camera model, which allows projection of 3-D points in the image plane.

For the sake of completeness, we also distorted the image coordinates of annotated points using the lens distortion model, thus effectively projecting them into original images. This enables their use in case when one wishes to use customly-undistorted frames instead of ones provided by the dataset.

After calibration, we took undistorted video frames and scaled them down by factor of four, using `convert` utility from ImageMagick[9]. This resulted in a set of undistorted down-scaled video frames, which are, due to favorable compromise between size and resolution, considered to be the main part of the dataset.

4 Data and Format

This section describes the data and its format. Most of it can be read using Matlab and bundled Matlab code, but this section should enable reading and parsing in other languages, for example C, as well. We use C types (e.g. *double*, *float*, *int32*) to denote the type of variables stored in binary files. Note that in binary files, *little-endian* format is used for storing multi-byte types.

4.1 Video Frames

The dataset provides three set of video frames:

- *original (source) frames* (2048×1536), located in `Data/Source` directory.
- *undistorted frames* (2048×1536), located in `Data/Frames` directory.
- *down-sampled undistorted frames* (512×384), located in `Data/Frames4` directory.

The directory for each of above-mentioned sets contains one directory per camera, each containing JPEG images corresponding to frames. The frames' filename format is therefore `Camera$C/$F.jpg`, with $C = 1 \ldots 4$ being camera number and $F = 1 \ldots 7840$ being frame number.

4.2 Camera Calibration Data

Camera calibration data for the dataset is stored in `Data/Calibration` directory. It contains one directory per camera, each containing the following files that contain *calibration data for undistorted video frames*:

[9] `http://www.imagemagick.org`

- `calibration.mat`: a Matlab file containing calibrated camera model.
- `calibration.xml`: an XML file containing calibrated camera model.
- `calibration.yml`: a YAML file containing calibrated camera model.
- `image.jpg`: an undistorted video frame from the camera.
- `mask.png`: a mask of valid pixels in an undistorted frame.
- `extrinsic-points.mat`: a Matlab file containing points from which extrinsic parameters were estimated.
- `intrinsics-checkerboard`: directory containing undistorted frames with checkerboard pattern, from which intrinsics of camera model were estimated.

Additionally, `original` directory contains calibration data for working with *original (distorted) video frames*.

4.2.1 Calibration data for undistorted video frames

Calibration for undistorted frames is provided as a Matlab file, `calibration.mat`. It contains intrinsic and extrinsic parameters of a camera model used by [6] to 3-D project points to an image plane (for an example, see their `project_points3()` function):

- `f`: a 2×1 vector that contains *focal length* (in pixels).
- `c`: a 2×1 vector that contains *principal point* coordinates.
- `k`: a 5×1 vector that contains values of *image distortion coefficients*[10].
- `alpha`: the value of *skew coefficient*.
- `om`: a 3×1 *rotation vector*, associated with `R` via Rodriguez formula.
- `R`: a 3×3 *rotation matrix*.
- `T`: a 3×1 *translation vector*.

For convenience, camera calibration is also provided in the format used by OpenCV, so that its `projectPoints()` function can be used to project 3-D points to an image plane. Calibration is stored in an XML file, `calibration.xml`, and a YAML file, `calibration.yml`, both of which can be read using OpenCV's `FileStorage`, and contain the following data:

- `cameraMatrix`: *camera matrix*, encompassing intrinsic parameters.
- `distCoeffs`: a vector containing *distortion coefficients*. Since zero distortion coefficients are assumed, an empty vector is stored.
- `rvec`: *rotation vector*.
- `tvec`: *translation vector*.

[10] No distortion is assumed for already-undistorted frames, hence all values are set to 0.

The image, `image.jpg` is a sample undistorted frame from the camera, and `mask.png` is a mask of valid pixels in an undistorted frame. Directory `intrinsics-checkerboard` contains undistorted images of calibration checkerboard, which were used to estimate intrinsic parameters. Matlab file `extrinsic-points.mat` containins a $5 \times P$ array of P calibration points that were used to estimate extrinsic parameters. Each point entry consists of (X, Y, Z, y, x), where the first three are coordinates in the room, whereas the last two are coordinates in the image plane.

Note that calibration data is provided for full-sized undistorted image frames, which should be taken into account when using it with down-scaled undistorted frames.

4.2.2 Calibration data for original video frames

For the sake of completeness, we also provide calibration data for original, undistorted video frames. This allows user to work either with original video frames, or use different undistortion parameters (e.g. resulting in a different level of "zoom" in undistorted images, which is governed by F_c parameter in OCamCalib's undistortion functions). The data is stored in `original` directory.

The image, `image.jpg` is a sample original frame from the camera, and `mask.png` is a mask of valid pixels in an original frame. Directory `intrinsics-checkerboard` contains images of checkerboard pattern that were used to calibrate lens distortion using OCamCalib toolbox. File `ocam_calib.txt` contains the resulting camera model, which can be used to undistort frames. Matlab file `extrinsic-points.mat` containins a $5 \times P$ array of P calibration points that were back-projected from undistorted frames into original frames. Each point entry consists of (X, Y, Z, y, x), where the first three are coordinates in the room, whereas the last two are coordinates in the image plane. Using this data, original frames can be undistorted, and calibration points can be projected into new undistorted frames using appropriate transformation.

4.3 Ground-truth Annotations

For each frame, 3-D coordinates, expressed in the room coordinate system, are provided for each person. For people that are not present in a room at a given moment, their coordinates are set to ∞. The ground truth annotations are stored in a $N \times D \times F$ array, where N is number of people (five), D is number of dimensions (three) and F is number of frames (7840).

The annotations are provided in two forms, both containing the same array:

- a Matlab data file: `Data/GroundTruth/GroundTruth.mat`
- a raw binary file: `Data/GroundTruth/GroundTruth.bin`

In the binary file, N, D and F are stored as three *uint32* values, followed by the array of *float* values. Because the array is directly exported from Matlab, the values are stored in the following order: X-coordinates for all people in 1st frame, Y-coordinates for all people in 1st frame, Z-coordinates for all people in 1st frame, X-coordinates for all people in 2nd frame, and so on.

The INI file (`config.ini`) provides additional metadata that is used by the bundled Matlab code (Section 5.3).

4.4 Timestamps

Timestamps are primarily used for association of Ubisense events to frames. Video cameras were set to print their timestaps on top of the video frames; the printed strings were then read from undistorted frames using template-based OCR, and the median of values from all four cameras were taken. This way, the dataset provides a timestamp value corresponding to each provided video frame.

The timestamp data is located in `Data/Timestamp`. Timestamps are stored as fractional Unix epoch values in a binary file, `timestamp.bin`, in the form of a *double array* that spans the whole file. Number of stored values therefore equals the size of binary file divided by eight, and corresponds to the number of frames in the dataset (7840). The first stored value corresponds to first frame, and so on.

The INI file (`config.ini`) provides additional metadata that is used by the bundled Matlab code (Section 5.2).

4.5 Events from the Ubisense System

The data for Ubisense localization system is located in `Data/Ubisense`. The captured data stream is stored in a binary file, `ubisense.bin`, which contains sequentially-stored records about Ubisense location events. Each record consists of the following fields:

- timestamp (*double*): event's timestamp
- tag ID (*int32*): tag identifier
- timeslot (*int32*): time slot at which the event occurred (system's property)
- x (*double*): x coordinate (in meters)
- y (*double*): y coordinate (in meters)
- z (*double*): z coordinate (in meters)
- gdop (*double*): geometrical dilution of precision
- error (*double*): estimated localization error (as given by the system)

The most important fields are *timestamp*, which can be used to associate the event with video frame (e.g. by searching for frame with the closest timestamp), *tag identifier*, which is used to identify tags, and *coordinates*. The mapping between tag identifiers and ground truth annotations is given in Table 3.

The INI file (`config.ini`) provides additional metadata that is used by the bundled Matlab code (Section 5.4).

Table 3: Mapping between Ubisense tag identifiers and ground truth annotations (person numbers).

Person	Name	Tag ID (integer value)	Tag ID (human-readable string)
1	Dana	335574327	020-000-117-055
2	Janez	335574368	020-000-117-096
3	Mitja	335574326	020-000-117-054
4	Marko	335574403	020-000-117-131
5	Rok	335574329	020-000-117-057

5 Matlab Code

The dataset comes with bundled Matlab code, which is provided under GPLv2 license. As it is part of the first author's PhD work, its functions are unimaginatively prefixed with his name in order to avoid potential namespace clashes. The functions and classes are self-documented, therefore this section does not provide a comprehensive reference. Instead, it aims to direct the reader to the most relevant functions by giving examples of their use.

The code encompasses reader classes for provided timestamp data (Section 5.2), ground truth annotations (Section 5.3) and radio location events data (Section 5.4); in combination with Section 4, they can also serve as reference parser implementations. Additionally, code for localization system evaluation framework is provided (Section 5.5), and its use is demonstrated on the case of radio-based system. Finally, we also included functions for generating configuration for occupancy-map-like algorithms (Section 5.6), which demonstrate the use of provided calibration data, and convenience functions for distorting/undistorting points with the calibrated lens model (Section 5.7).

5.1 External Dependencies

The code presented in the following sections requires the following publicly-available functions to be in path:

- `ini2struct.m`[11]: INI file parser
- `isabsolute.m`[12]: verifies whether a given path is absolute

Additional dependencies are listed in sections where they are applicable.

5.2 Timestamp Database Reader

The `TimestampDatabase` class enables retrieval of stored timestamp values and frame number to timestamp conversion. It is primarily intended to be used by radio-event reader class (Section 5.4) to associate the radio-based location events with individual frames, but can also be used on its own, as shown by the following code snippet:

```
% Create TimestampDatabase object, using the supplied metadata
timestamp = TimestampDatabase('Data/Timestamp/config.ini');

% Query for the frame range
[ startFrame, endFrame ] = timestamp.get_frame_range()

% Get timestamp for frame 5000:
ts = timestamp.get_timestamp(5000)

% Get timestamps for a range of frames:
ts = timestamp.get_timestamp(5000:5010)

% Apply date and time offset correction. For technical reasons,
% the clocks of computers that were used for recording data were
% not set to the actual data. The difference can be corrected
```

[11] http://www.mathworks.com/matlabcentral/fileexchange/17177-ini2struct
[12] http://en.verysource.com/code/1904001_1/isabsolute.m.html

```
% using supplied metadata. Note that is for display purposes only;
% for data retrieval, raw timestamps should be used
ts = timestamp.get_timestamp(5000, true)

% Get time string for a frame, using specified format:
ts = timestamp.get_timestring(5000, 'yyyy-mm-dd HH:MM:SS.FFF')

% Time string with corrected time offset. The actual recording date
% was 2011/11/11, but computers' clocks were set to 2010/04/01. Also,
% raw timestamps are in UTC, whereas corrected time stamps are in
% (dataset's) local time
ts = timestamp.get_timestring(5000, 'yyyy-mm-dd HH:MM:SS.FFF', true)
```

5.3 Ground Truth Database Reader

The `GroundTruthDatabase` class enables retrieval of stored ground-truth positions. These can be obtained for one or more individuals in one or more frames, as shown by the following code snippet:

```
% Create GroundTruthDatabase object, using the supplied metadata
gt = GroundTruthDatabase('Data/GroundTruth/config.ini');

% Query for frame range
[ startFrame, endFrame ] = gt.get_frame_range()

% Get number of people; can also be obtained by looking at
% public property 'people'
numPeople = gt.get_num_people()

% Get ground truth position for Person #4 in Frame #5000:
data = gt.get_ground_truth(5000, 4)

% Get ground truth positions for Persons 1 and 3 in Frames #5000
% and #5001:
data = gt.get_ground_truth([5000,5001], [1, 3])
```

5.4 Ubisense Database Reader

The `UbisenseDatabase` class provides access to stored location events from the Ubisense radio-based localization system. It allows retrieval of all stored events for a single person (tag ID), or retrieval of an event for a single person at the specified time (or frame). Because a location event might not be available exactly at the specified time instant, the closest event is returned; depending on the parameters, this can be the closest previous event (zero-order-hold behavior), closest next event, or closest event in the next direction. Therefore, same location event might be returned for several frames. The typical use of the `UbisenseDatabase` class is illustrated by the following code snippet:

```
% Create UbisenseDatabase object, using the supplied metadata
ubisense = UbisenseDatabase('Data/Ubisense/config.ini');

% The object has three public properties:
%  - timestamp = TimestampDatabase object
%  - people = cell array of individuals' names
%  - tags = cell array of ids for tags that individuals wear
```

```
% Convert tag id for Person 1 from string to integer
id = ubisense.get_tag_id_from_string(ubisense.tags{1})

% Get all location events for Person 1, using obtained integer tag id
data = ubisense.get_events(id)

% We can also use string tag id directly
data = ubisense.get_events(ubisense.tags{1})

% Get the location event for person 4 at frame 5000, using
% zero-order-hold behavior (get closest previous event).
% Getting event by frame number is equivalent to translating
% frame number to timestamp using TimestampDatabase and using
% get_event_by_timestamp() function.
data = ubisense.get_event_by_frame(ubisense.tags{4}, 5000)

% Twelve frames later, we still get the same event, due to
% slower refresh frequency of the radio-based system.
data = ubisense.get_event_by_frame(ubisense.tags{4}, 5012)

% Change of event
data = ubisense.get_event_by_frame(ubisense.tags{4}, 5013)

% Compare with the behavior when using 'closest' parameter
% instead of (default) 'prev':
data = ubisense.get_event_by_frame(ubisense.tags{4}, 5000, 'closest')
data = ubisense.get_event_by_frame(ubisense.tags{4}, 5012, 'closest')
data = ubisense.get_event_by_frame(ubisense.tags{4}, 5013, 'closest')
```

5.5 Localization System Evaluation Framework

The bundled Matlab code also provides the framework for localization system evaluation, which is based on methodology and metrics from [1, Section 5]. The framework consists of two sets of functions, prefixed with `rok_evaluation_anonymous_` and `rok_evaluation_full_`; the former provides framework for evaluation of anonymous detection and localization, whereas the latter provides framework for evaluation of identification, detection and localization.

Framework operates on the premise that at each time instant, a structure describing ground-truth annotations and a structure describing detection hypotheses are provided. In the examples below, we illustrate the use of the framework for evaluation of radio-based localization; this essentially replicates results for radio-based system, found in [1, Section 6.2 and 6.4].

The framework relies on external implementation of Munkres-Kuhn algorithm (Hungarian method) [7], `Hungarian.m`[13], which should be made available in the path.

The ground-truth annotations wrapper function, `rok_ground_truth_get_points()`, which we use in the code example below, requires metadata to be provided by a configuration file. Therefore, create a file called `room-config.ini` with the following content:

```
# room-config.ini: Room configuration file
```

[13] `http://www.mathworks.com/matlabcentral/fileexchange/11609-hungarian-algorithm`

```
# Room dimensions (x1, y1, x2, y2)
area = [ 0.00, 0.00, 8.00, 7.50 ]

# Cell size and stride
cell_size = [ 0.50, 0.50 ]
cell_stride = [ 0.25, 0.25 ]
```

The code below evaluates *anonymous detection and localization* performance of radio-based system, using *Metric A* and *Metric B* from [1, Section 5.1]. The obtained results should be the same as those reported in [1, Section 6.2].

```
% Create room configuration, which is needed by wrapper function
% for ground truth points
room = rok_room_create('room-config.ini');

% Load ground truth points
groundTruth = GroundTruthDatabase('Data/GroundTruth/config.ini');
numIds = groundTruth.get_num_people();

% Load Ubisense data
ubisense = UbisenseDatabase('Data/Ubisense/config.ini');

% Initialize evaluation structures
metricA = rok_evaluation_anonymous_initialize(numIds);
metricB = rok_evaluation_anonymous_initialize(numIds);

for frame = 551:2861
    % Get detections; this wrapper function returns structured
    % required by evaluation framework's functions
    detections = rok_ubisense_get_detections(ubisense, frame);

    % Get ground truth points; this wrapper function returns structure
    % required by the evaluation framework's functions
    ground = rok_ground_truth_get_points( groundTruth, room, frame );

    % Metric A: without distance threshold
    metricA = rok_evaluation_anonymous_update(metricA, detections, ...
        ground, Inf);

    % Metric B: with distance threshold (0.5 meter)
    metricB = rok_evaluation_anonymous_update(metricB, detections, ...
        ground, 0.5);
end

% Print evaluation results
fprintf('Evaluation using Metric A:\n');
rok_evaluation_anonymous_display(metricA);
fprintf('\n');

fprintf('Evaluation using Metric B:\n');
rok_evaluation_anonymous_display(metricB);
fprintf('\n');
```

The code below evaluates *detection, localization and identification* performance of radio-based system, using *Metric A*, *Metric B* and *Metric C* from [1, Section 5.2]. This example prints out the confusion matrix like the one reported in [1, Section 6.4] and the LaTeX code for it.

```
% Create room configuration, which is needed by wrapper function
```

```
% for ground truth points
room = rok_room_create('room-config.ini');

% Load ground truth points
groundTruth = GroundTruthDatabase('Data/GroundTruth/config.ini');
numIds = groundTruth.get_num_people();

% Load Ubisense data
ubisense = UbisenseDatabase('Data/Ubisense/config.ini');

% Initialize evaluation structures
metricA = rok_evaluation_full_initialize(numIds);
metricB = rok_evaluation_full_initialize(numIds);
metricC = rok_evaluation_full_initialize(numIds);

for frame = 3721:7761,
    % Get detections; this wrapper function returns structured
    % required by evaluation framework's functions
    detections = rok_ubisense_get_detections(ubisense, frame);

    % Get ground truth points; this wrapper function returns structure
    % required by the evaluation framework's functions
    ground = rok_ground_truth_get_points( groundTruth, room, frame );

    % Metric A: no identity information, no thresholding
    metricA = rok_evaluation_full_update(metricA, detections, ...
        ground, Inf, false);

    % Metric 2: no identity information, thresholding (0.5 meter)
    metricB = rok_evaluation_full_update(metricB, detections, ...
        ground, 0.5, false);

    % Metric 3: identity information, no thresholding
    metricC = rok_evaluation_full_update(metricC, detections, ...
        ground, Inf, true);
end

% Print results for metric 2
fprintf('Full evaluation using Metric B:\n');
rok_evaluation_full_display(metricB);
fprintf('\n\n');

% Print source for LaTeX table
rok_evaluation_full_latex_table(metricB, false);
```

5.6 Room Configuration Code

For illustration of how the provided calibration data can be used to project 3-D points to the images, we provide two functions that generate configuration for occupancy-map-like algorithms. Both functions discretize the room into a grid, and project the cells into images, using rectangle-based and convex-hull-based appearance model.

The following example assumes that `room-config.ini` from Section 5.5 has already been created. This configuration file describes room dimensions and discretization. Another configuration file, which provides paths to camera calibration data and describes the parameters for appearance model (e.g. rectangle height), is required. Therefore, create a file called `rectangles-config.ini`, with the following content:

```
# rectangles-config.ini: Rectangles configuration file

# Cameras
cameras = [ 1, 2, 3, 4 ]

# Rectangles bottom and top height
height_bottom = 0.0
height_top = 1.75

# Calibration
calibration_data_format = Data/Calibration/Camera%c/calibration.mat
calibration_image_format = Data/Calibration/Camera%c/image.jpg
mask_format = Data/Calibration/Camera%c/mask.png

# Image scaling (with regards to calibration)
scaling = 0.25

# Rectangle filtering parameters
min_area = 15000
only_full_rectangles = false
```

Using those configuration files, it is possible to create the room configuration:

```
% Generate room configuration for rectangle-based visual model and
% visualize results. Corners of generated rectangles can be,
% along with other data, found in config.rects
config = rok_room_generate_rectangles('room-config.ini', ...
    'rectangles-config.ini', true)

% Generate room configuration for convex-hulls-based visual model and
% visualize results. Vertices of generated rectangles can be, along
% with other data, found in config.hulls
config = rok_room_generate_convex_hulls('room-config.ini', ...
    'rectangles-config.ini', true)
```

5.7 Point Reprojection on Customly-undistorted Images

The dataset provides full camera calibration data, which also allows undistortion of original frames with parameters that differ from the ones used with undistorted frames, provided by this dataset. The camera lens calibration data (Section 4.2.2) should be used together with functions from OCamCalib toolbox to undistort the checkerboard pattern images, and use those to estimate new intrinsic parameters. Then, the points used for extrinsic parameters calibration need to be reprojected on newly-undistorted images, which is demonstrated by the code below (the code assumes that OCamCalib Toolbox code is already in path):

```
% This example is for Camera #1
cd './Data/Calibration/Camera1/original';

% Load points on undistorted image
points = load('extrinsic-points.mat');
points_d = points.points(:,4:5)'; % We need only image coordinates

% Read OCamCalib model
model = rok_fisheye_camera_read_model('ocam_calib.txt');

% Read original (distorted) image
```

```
I_d = imread('image.jpg');

% Plot
figure; imshow(I_d); title('Distorted (original)'); hold on;
plot(points_d(1, :), points_d(2, :), 'r+', 'MarkerSize', 14, ...
    'LineWidth', 2);

% Undistort with custom Fc, different from 1.375
fc = 1.0;
I_u = rok_fisheye_camera_undistort_image(model, I_d, fc);
points_u = rok_fisheye_camera_undistort_points(model, points_d, fc);

% Plot
figure; imshow(I_u); title('Undistorted'); hold on;
plot(points_u(1, :), points_u(2, :), 'r+', 'MarkerSize', 14, ...
    'LineWidth', 2);
```

6 Use in Our Papers

At the time of writing, we used the presented dataset in two of our papers [2, 1]. This technical report was written together with [1], whereas [2] was written earlier, therefore it uses a bit different room configuration. In particular, in [2] room dimensions were limited to 0.88–8.0 and 0.55–7.5, in [1], the whole room is used. The training in [2] was done on every 10th frame in range 3721–5141, while for testing, all frames in range 5141–7761 were used. In [1], the anonymous detection evaluation was done on all frames in range 551–2861, whereas tracking was tested on frames 3721–7761.

7 Conclusion

In this technical report, we presented *MVL Lab5*, a multi-modal indoor person localization dataset. The dataset provides calibrated and time-synchronized video frames from four video cameras and location event data from radio-based localization system. Therefore, it is suitable for development and testing of both purely computer-vision-based methods and methods for fusing video with location events from the radio-based system, both for the task of person detection, localization and identification.

This document serves as the primary documentation source for the dataset, describing its availability, acquisition procedure, the data it provides and its format. This information should allow researchers and developers to make full use of the provided data. In addition, we provide some of our Matlab code, which on one hand serves as a reference implementation, but should also ease the use of dataset for new users.

The main part of the dataset comprises 7840 video frames from four calibrated cameras and data stream from the radio-based system, which corresponds to a 6.5-minute sequence. The sequence involves five individuals walking around a realistically cluttered room. The main video data are undistorted frames that were down-scaled to 512×384; however, the dataset also provides both full-sized undistorted and original frames, which should broaden the possibilities of its further use. We make the dataset publicly-available in hopes that it will prove useful to other researchers in their work, primarily in the field of person detection, identification and tracking, but also possibly in other fields.

Document Revisions

- **2012/12/27** — public release
- **2012/10/22** — draft for reviewer-only release of dataset

Acknowledgments

This work was supported by research programs P2-0095 and P2-0098, research project J2-4284 and the research grant 1000-10-310118, all by Slovenian Research Agency.

References

[1] R. Mandeljc, S. Kovačič, M. Kristan, and J. Perš, "Tracking by identification using computer vision and radio," *Sensors*, vol. 13, no. 1, pp. 241–273, 2012.

[2] R. Mandeljc, S. Kovačič, M. Kristan, and J. Perš, "Non-sequential multi-view detection, localization and identification of people using multi-modal feature maps," in *Proceedings of the 11th Asian Conference on Computer Vision (ACCV 2012)*, 2012.

[3] "Research & Development Packages — Ubisense." `http://www.ubisense.net/en/rtls-solutions/research-packages.html`. Accessed on: 07/2012.

[4] "MVL Lab5 Dataset." `http://vision.fe.uni-lj.si/research/mvl_lab5/`.

[5] D. Scaramuzza, A. Martinelli, and R. Siegwart, "A toolbox for easily calibrating omnidirectional cameras," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, pp. 5695–5701, oct 2006.

[6] J. Y. Bouguet, "Camera calibration toolbox for matlab." `http://www.vision.caltech.edu/bouguetj/calib_doc/`, 2010. Accessed on: 07/2012.

[7] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics (NRL)*, vol. 52, no. 1, pp. 7–21, 2005.