Project no. IST-34107
Project acronym: ARTTS
Project title: Action Recognition and Tracking based on Time-of-flight Sensors

Action Recognition Toolbox

Duration of the project: October 2006 – September 2009

Partners of the project consortium:

INB, University of Lübeck, Germany
IMM, Technical University of Denmark, Denmark
LAPI, University Politehnica Bucuresti, Romania
Swiss Center for Electronics and Microtechnology, Switzerland
SensoMotoric Instruments GmbH, Germany

Project website:

www.artts.eu

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

1 Introduction

The ARTTS Action Recognition toolbox is a library of MATLAB routines that provide functionality for action recognition in TOF image sequences and for labelling ground truth data. Here is an overview of the functionality included in the toolbox:

- Labelling actions in TOF image sequences
- Range flow computation for action recognition
- Intrusion detection based on histogram measures
- Action recognition from trajectories of key points
- Action recognition using motion history images

2 Licence

The ARTTS Action Recognition toolbox (code and accompanying documentation, including this document) is provided under the following licence:

3 Installation

Since some components in the Activity Recognition toolbox require routines from the Signal Processing and Object Tracking toolboxes, first make sure that the toolboxes have been installed and added to the MATLAB search path. Then, copy the directory `ar_toolbox` to a suitable location, and add this directory to MATLAB's search path. The best way to do this is to include the corresponding `addpath` calls in your `startup.m` file.

Some components of the toolbox require supporting libraries to be present:

- If you want to use the routines for action recognition from trajectories of key points (Section 4.4), you need to obtain the “Better Skeletonization” library by Nicholas Howe from MATLAB Central. As of this writing, this library was available under the following link:

http://www.mathworks.com/matlabcentral/fileexchange/11123

Install the library and compile the MEX files contained in it.

4 Components

This section gives a more detailed overview of the components included in the toolbox. For each of the major components, we will give a description of the functionality, including links to relevant publications, a list of the routines that implement this functionality, and an example of how to use the routines.

For a detailed description of the routines, refer to the documentation that is included directly in the source code of the routines; this can be accessed from MATLAB by typing

`help` <routine name>

For example, the documentation for the routine read_artts_range can be displayed by typing

```
help read_artts_range
```

4.1 Labelling

This component provides a utility for labelling actions in TOF image sequences. These labelled actions can then be used to train detectors and compare the detection results against a known ground truth. Each action is associated with a bounding box that contains the action as well as a starting and end frame.

The following routines are provided:

`label_actions`
Label bounding box and starting / end frame of actions

`parse_actions`
Parse action parameters in labelled action database

Here is an example of how to label actions:

```
label_actions('data/action_db/action_db.arttsidx', ...
    'labelled.arttsidx');
```

Press the space bar to move to the next image; 'b' to move to the previous image; 'f' to mark the first frame of an action; and 'l' to mark the last frame of an action. Drag the blue lines to set the bounding box for an action. Press 's' to save the results to `labelled.arttsidx`. (See the documentation in `label_actions.m` for more details.) The sample file `action_db.arttsidx` already contains a labelled action.

Here is an example of how to read the labelled actions from a database index file:

```
% Read database index file
[names, params]=read_arttsidx('data/action_db/action_db.arttsidx');

% Parse action descriptors
actions=parse_actions(params);
```

The actions are returned as a cell array `actions`; see the documentation in `parse_actions.m` for more details.

4.2 Range flow computation for action recognition

This component provides a routine for the estimation of range flow on a sequence of TOF images. For further details on the implemented algorithm refer to:

> Hagen Spies, Bernd Jähne, and John L. Barron. Dense Range Flow from Depth and Intensity Data. Proceedings of International Conference on Pattern Recognition (ICPR'00), volume 1, pages 131–134, 2000.

The routine estimates the 3D motion vector for each pixel of the sequence if sufficient information for a robust estimation is available. Otherwise, the routine returns zero for all components of the 3D motion vector.

The range flow estimation algorithm is implemented in the following routine:

`range_flow`
: Takes a cell array containing a sequence of TOF images and estimates the range flow for each frame. The routine returns three cell arrays representing the estimated components of the 3D motion vector [u, v, w]' for each frame.

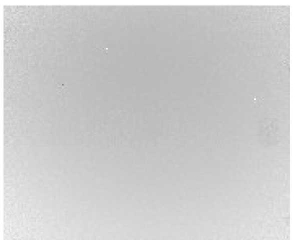
Here is an example of how to use the range flow estimation routine:

```
% Read database index file
images=read_artts_db('data/action_db/action_db.arttsidx');

% Compute range flow
[U V W]=range_flow(images);

% Visualize
[M N]=size(images{1}.range);
[X Y]=meshgrid(1:N, 1:M);
for t=1:length(images)
    imshow(images{t}.amplitude,[]);
    title(sprintf('frame %d', t));

    hold on;
    u=U(:,:,t); v=V(:,:,t); w=W(:,:,t);
    idx=find(u~=0 & v~=0 & w>0.02);
    quiver(X(idx), Y(idx), u(idx), v(idx), 'g');
    idx=find(u~=0 & v~=0 & w<-0.02);
    quiver(X(idx), Y(idx), u(idx), v(idx), 'r');
    hold off;
    pause(0.1)
end
```

4.3 Intrusion detection based on histogram measures

This component implements an intrusion detection algorithm for video surveillance using the TOF camera, exploiting the special ability of the device to measure distances. Two different detection methods (based on histogram and motion estimation) are presented and compared. The algorithm is described in detail in the paper referenced below.

The basic idea is illustrated by Figures 1 and 2. The histogram of a scene containing a person (Figure 2b) differs from the histogram of the empty scene exactly in the pixels corresponding to the person in the scene. Hence, through a simple histogram difference operation, one can detect the event of a person entering or leaving the scene.


a)

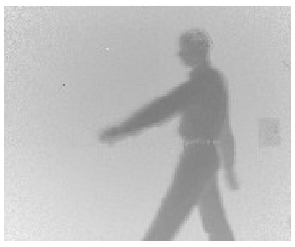

b)

Figure 1: a) Empty scene. b) Person entering the scene.

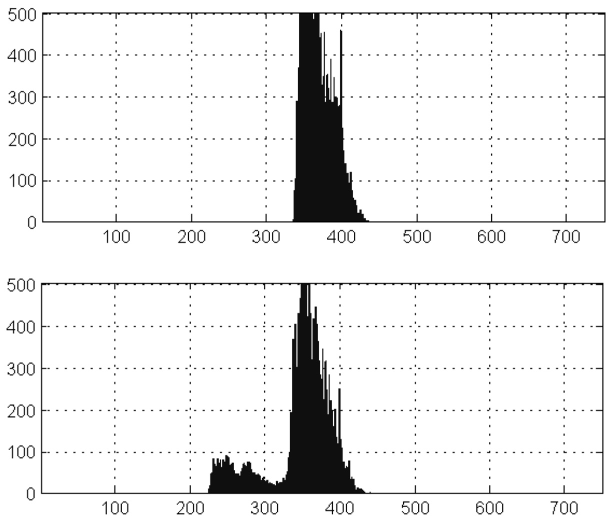


Figure 2: Histograms corresponding to Figures 1 a) and b)

The algorithm uses two histogram techniques to detect an intrusion:

a) Histogram comparison technique:

   The overall difference between the *i*th frame and its successor can be evaluated by the expression

$$SD_i = \sum_{j=1}^{G} |H_i(j) - H_{i+1}(j)|$$

   When the overall difference SD_i is larger than a given threshold T, a segment boundary (corresponding to a person entering or leaving the scene) is declared.

b) Presence detection by counting histogram peaks:

   This step begins with a low-pass filtering of the histogram for smoothing, and then a peak-finder method is applied. Finally, the number of histogram peaks is counted and compared along the video.

The second method for intrusion detection is based on motion estimation. We use an improved version of the block-matching algorithm, with zero-motion detection.

For more details, see the following paper:

Şerban Oprişescu, Mihai Ciuc, Vasile Buzuloiu. Histogram and motion based intrusion detection and tracking algorithms for ToF cameras. Proceedings of the IEEE International Symposium on Signals, Circuits & Systems (ISSCS), Iasi, Romania, 2009.

The intrusion detection algorithms can be accessed through a graphical user interface (GUI), see Figure 3.

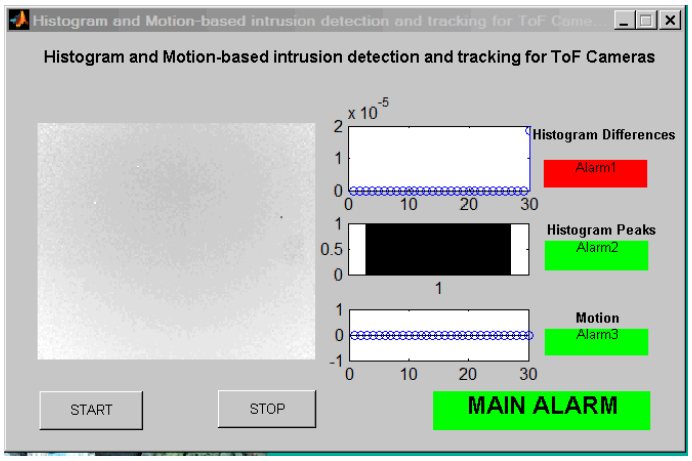

Histogram and Motion-based intrusion detection and tracking for ToF Came...
Histogram and Motion-based intrusion detection and tracking for ToF Cameras
Histogram Differences
Alarm1
Histogram Peaks
Alarm2
Motion
Alarm3
START
STOP
MAIN ALARM

Figure 3: Graphical user interface (GUI) for intrusion detection.

This GUI includes charts showing the result of the three intrusion detection indicators at each frame. The algorithm runs in almost real time.

This component provides the following routines:

`intrusion_det`
: This routine takes the name of a TOF image sequence file and performs intrusion detection on it. The routine returns the results of the three intrusion detectors described above.

`intrusion_det_gui`
: Call this routine to start the intrusion detection GUI.

Here is an example of how to use the intrusion detection routine:

```
% Process sample image
intrusion=intrusion_det('data/pres_det_movie.mat');

% Plot intrusions detected by the three types of intrusion detector
for j=1:3
    subplot(3, 1, j);
    stem(intrusion(j,:));
end
```

4.4 *Action recognition from trajectories of key points*

This component recognizes actions based on trajectories of key points. The first step is the segmentation (silhouette extraction) of the moving person from the scene; this is straightforward due to the distance information delivered by the TOF camera. Then, we extract a number of key points from the silhouette: The centroid, the head, and the extremities of the limbs. Apart from the centroid, all key points are extracted from the skeleton of the silhouette, obtained using morphological operations and a median filter. Once the key points are extracted, the next step is to track them through the image sequence and to record their trajectories. The trajectories can be absolute or referenced to the centroid. On these key point trajectories, we perform action recognition by computing several features, such as mean and absolute speed, total variation etc. A discrimination table (Table 1) is used to discriminate between the six different actions shown in Figure 4.

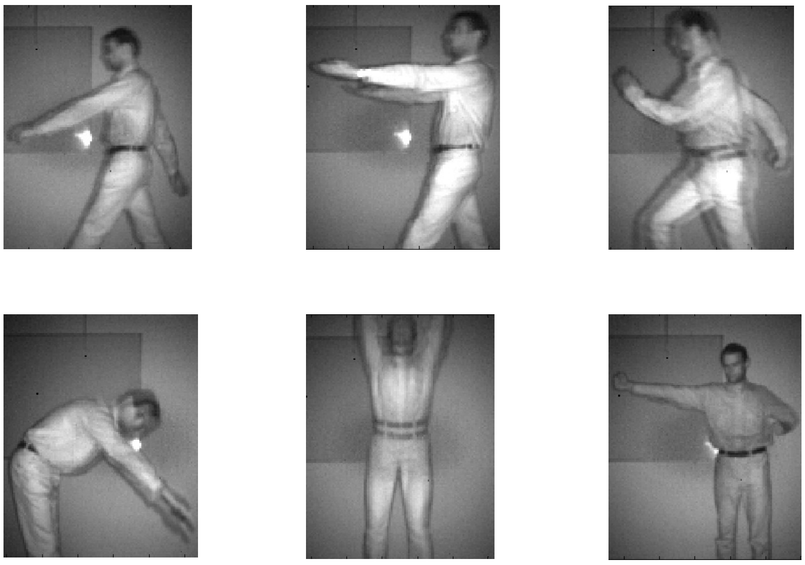


Figure 4: Sample frames from the six actions to be recognized. From left to right, top to bottom: Walk, carry, run, bend, jump, and box.

Action	Conditions			
Walk	V_x > thr	$V_{Hx} >> \Delta_{Hx}$	v_x avr	any y small
Carry	V_x > thr	$V_{Hx} \cong 0$	v_x avr	
Run	V_x > thr	$V_{Hx} >> \Delta_{Hx}$	v_x big	
Box	$V_x >> \Delta_x$	$V_{Hx} >> \Delta_{Hx}$	V_{vHx} big	
Jump	V_y > thr	V_{Hy} > thr	v_y big	any x small
Bent	V_y > thr	V_{Hy} > thr	v_y avr	

V_x and V_y are the total variations of $x(k)$ and $y(k)$ respectively; V_{Hx} , V_{Hy} are total variations of one key point on hand coordinates; v_x, v_y are the speeds on corresponding coordinates; V_{vHx} is the total variation of the speed of hand key point on x.

Table 1: Conditions defining the six actions.

Figure 5 shows an example of trajectories obtained for the centroid.

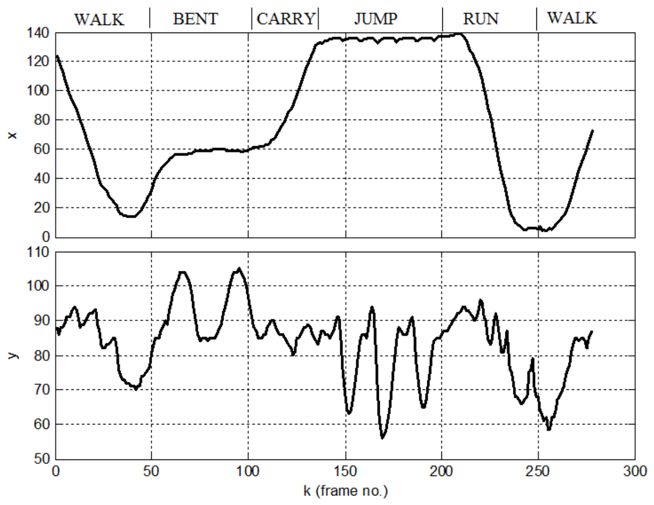


Figure 5: Example of trajectories (x/y) obtained for the centroid for the different actions.

For more details, see the following paper:

> Şerban Oprişescu, Constantin Burlacu, Vasile Buzuloiu, and Mihai Ivanovici. Action Recognition for Simple and Complex Actions using Time of Flight Cameras. International Conference on Image Processing, Computer Vision and Pattern Recognition (IPCV 2009), Las Vegas, USA, 2009.

The algorithm is implemented in the following routine:

`ar_key_points`
: Performs the action recognition on a TOF image sequence from trajectories of key points.

  Before this routine can be used to recognize actions, it needs to learn some parameters of a person walking; this is achived by calling the function with the parameter `learning=1` on an image sequence that shows a person walking without interruption. After the learning phase is completed, the routine can be called with `learning=0` on a different image sequence to recognize the action contained in it; the recognized action is output to the console.

Here is an example of how to use this routine:

```
% Learning phase
[action_lin, action_col, action_lin_rel, action_col_rel, ...
 var_tr, var_tr_Glin, var_tr_Gcol, viteza, viteza_abs, vvhx]= ...
ar_key_points('data/walk_movie', 2, 21, 367, 1);

% Recognize “run” action
[action_lin, action_col, action_lin_rel, action_col_rel, ...
 var_tr, var_tr_Glin, var_tr_Gcol, viteza, viteza_abs, vvhx]= ...
ar_key_points('data/run_movie', 2, 25, 367, 0);

% Recognize “bend” action
[action_lin, action_col, action_lin_rel, action_col_rel, ...
 var_tr, var_tr_Glin, var_tr_Gcol, viteza, viteza_abs, vvhx]= ...
ar_key_points('data/bend_movie', 2, 30, 367, 0);
```

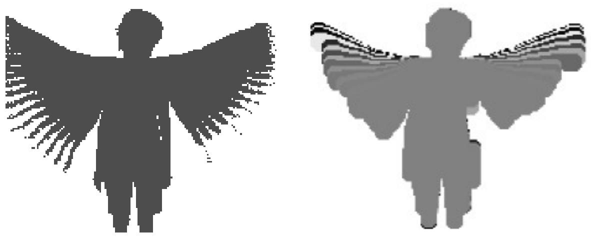
4.5 Action recognition using motion history images (LAPI)

This component performs action recognition using two types of images:

- Binary Motion Energy Image (BMEI), a binary image of the energy of the motion, which shows where motion is present in the movie

- Motion History Image (MHI), a gray-level image which shows motion history, the intensity being inversely proportional to the elapsed time.

Figure 6 shows examples of both types of images for an action of the type “raise both hands”. (The images were inverted for better reproduction in print.)



a) b)

Figure 6: a) Binary Motion Energy Image (BMEI), b) Motion History Image (MHI)

For further details, see

> Diana Rosu. Applications for Action Recognition using ToF Cameras. Diploma thesis, Image Processing and Analysis Laboratory (LAPI), Politehnica University of Bucharest, July 2009.

The algorithm is implemented in the following routine:

`ar_motion_history`
: Performs the action recognition using motion history images. The routine returns the number of actions and a text label containing their type, for instance “the person raised both hands”.

Here is an example of how to use this routine:

```
% Recognize sample action
ar_motion_history('data/raise_both_hands.mat', 1, 21, 300);

% Recognize another action
ar_motion_history('data/lean_left.mat', 1, 25, 300);
```