UNIVERSITY OF COPENHAGEN

**Københavns Universitet**

# HUMIM software for articulated tracking

Hauberg, Søren; Pedersen, Kim Steenstrup

*Publication date:*
2012

*Document Version*
Publisher's PDF, also known as Version of record

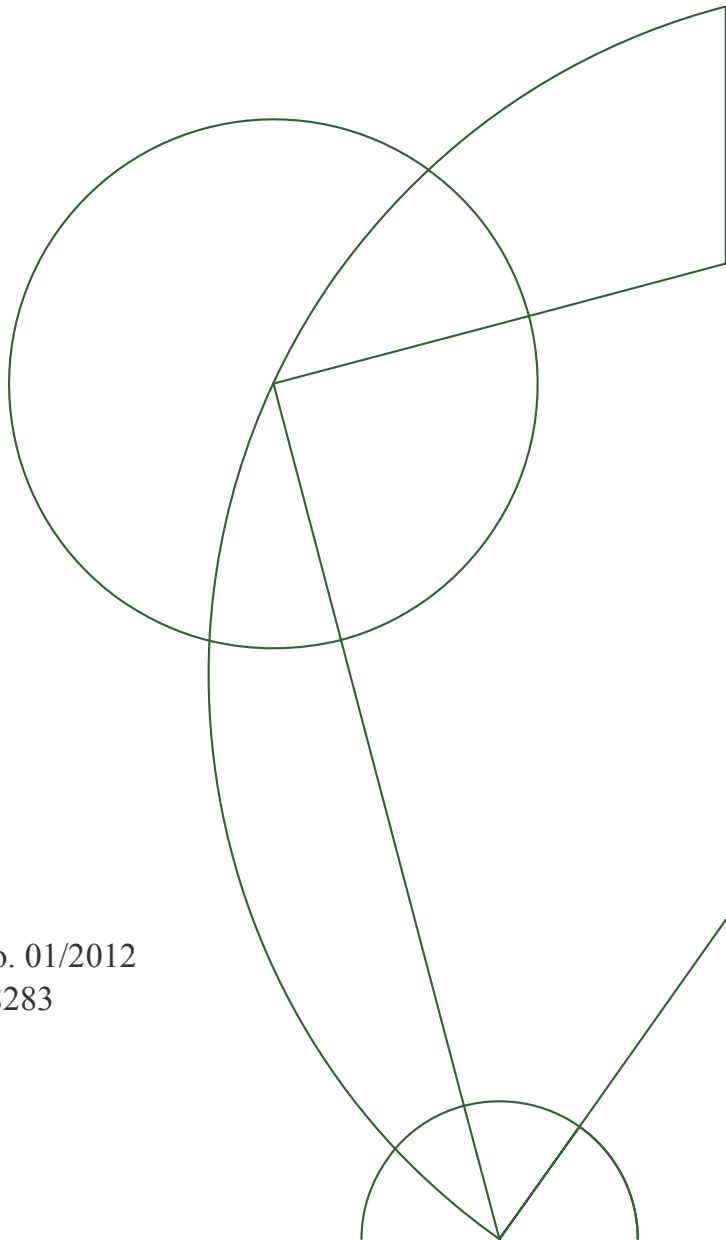
*Citation for published version (APA):*
Hauberg, S., & Pedersen, K. S. (2012). HUMIM software for articulated tracking. Datalogisk Institut, Københavns Universitet: Faculty of Science, University of Copenhagen. Koebenhavns Universitet. Datalogisk Institut. Rapport

Download date: 21. maj. 2019

FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN


# HUMIM Software for Articulated Tracking

Søren Hauberg and Kim Steenstrup Pedersen



Technical Report no. 01/2012
ISSN: 0107-8283

# Contents

# 1 Introduction

This technical report documents the software developed for articulated tracking as part of the *Human Motion Imitation (HUMIM)* project[1] at the Dept. of Computer Science, University of Copenhagen. The purpose of this document is to show how to install and run the software as well as provide some insights into how the code can be modified.

The software has been developed with academic research in mind. As such, the software has been designed to make it easy to modify and adapt to new theoretical models. It has, however, not been designed with speed in mind, nor to provide a black box for articulated tracking.

## 1.1 How to Cite this Software

If you use this software in academic work, we would appreciate if you say so in your papers. If you just use the basic particle filter classes you can cite this report. If you use specific priors, we would appreciate if you cite the paper that introduces the prior in question. The details of the specific papers are given in the Doxygen documentation for the individual classes.

Briefly put, you can use the following papers for the following classes:

- `projection_state` and `tangent_state`: [4]
- `pik_state` and `pik_stick_state`: [3]
- `angle_state`: [7]
- `KBK_stick_state`: [6]
- `stick_state` and `stick2d_state`: [1]
- `brownian_state` and `brownian_stick_state`: [5]
- `hull_state`, `bgsub_state` and `angle_bgsub_state`: [2]

## 1.2 Software License

The software is made available under the *de facto* Free and Open Source software license, the *GNU General Public License, version 3 (GPLv3) or later*. This allows you to study and modify the code anyway you please as long as you give other users the same access to your modifications. For details of the license, please read the `LICENSE` file distributed with the source code.

If you make useful changes to the software, we would appreciate if you send us your changes such that we can make them available to others.

# 2 Directory Layout and Dependencies

The source code for the articulated tracker is placed in the top-level directory. This contains two sub-directories: `ntk` and `OpenTissue`. Each of these directories contain a third-party library that we distribute along with the code. These libraries are not available under the GPLv3 license, but they are compatible with this license.

[1] `http://humim.org`

### 2.1 Included Libraries

We currently include two libraries: *NTK*[2] and *OpenTissue*[3]. The first is used for interacting with the Microsoft Kinect sensor and the latter is used for basic data structures for kinematic skeletons, forward and inverse kinematics and for basic mathematical routines.

## 3 Compiling the Code

We use *CMake*[4] for compiling the software as this is used by the included libraries. In principle this allows the code to be easily compiled in different operating systems, but it has only been verified to compile on Linux and Mac OS X 10.6 with Xcode 3.2.6.

Please consult the CMake documentation on how to compile the code as this depends on your operating system. On Linux this can be done with

```
cmake .
make
```

assuming you have the necessary dependencies in place.

### 3.1 Dependencies

The code uses several different standard libraries for image processing, mathematical routines, visualisation, etc. Specifically, we depend on

- OpenCV 2.2 (or later)
- ANN
- Boost
- Boost Bindings
- QHull
- Atlas
- DeVIL
- GLEW
- GLUT

You need to install these and ensure that CMake is able to find them. Furthermore, the software requires GCC 4.2.1 or later to compile.

[2] `http://nicolas.burrus.name/index.php/Research/KinectRgbDemoV6`
[3] `http://www.opentissue.org`
[4] `http://www.cmake.org`

# 4 Running the Code

When compiled, you will have the following executables

- `gui_tracker` – A tracker that visualises the results in 3D using OpenGL.
- `gui_projecter` – A tracker that visualises the results in the image plane.
- `headless_tracker` – A tracker that does not visualise results at all. This can be useful on clusters etc.
- `poser` – A program that allows you to change the pose of a skeleton to fit the data.
- `skeleton_sizer` – A program that allows you to change the size of limbs in a skeleton.
- `view_data` – A program for viewing 3D point clouds.

To run a program you need a configuration file that sets parameters, data paths, output paths, etc. The name of this file should be given as first input to all programs. The tracking programs also support an optional second argument –playback that replays previous tracking data rather than perform the tracking again. An example configuration file can be seen here

```
observation_type          point_cloud
state_type                angle_state
data_template             hands2_060410/image_%.3d.points
left_image_template       hands2_060410/image_2_%.3d.png
background_image          hands2_060410/Background_2.png
intrinsic_matrix          838.08544922 838.90100098 496.17398071 392.37945557
data_start_idx            1
output_directory          output
num_particles             150
initial_pose              config/hands2_060410_1.pose
pred_noise                0.15
convex_lambda             0.05
external_view             config/hands2_060410.view
bone_appearance           config/soren_thin_appearance_milimeters_cylinder.txt
skeleton_xsf              config/soren_milimeters_cylinders.xsf
measure_threshold         2
measure_variance          0.1
mocap_filename            hands2_060410/hands2_060410.mocap
```

If this configuration has been saved in the file `config_file` then, e.g. the `gui_tracker` program can be executed using

```
gui_tracker config_file
```

In general you should not expect this code to be something you just run and get a result. It is designed for development of new tracking techniques and not for developing applications of tracking. So some modifications is to be expected to get you what you want.

# 5 Modifying the Code

The articulated tracker uses a particle filter for inference. As such, the most interesting parts to modify is the *likelihood* and the *prior* of the model. Other parts of the code can also easily be modified, but we shall only consider these in this document.

In general, the code makes heavy use of `C++` templates, though it only uses basic template functionality. As such, the code can often be extended simply by building a class with a given interface and telling the application core that a new class exists.

## 5.1 The Prior

In order to extend the tracker with a new prior distribution, you need to implement a class that implements the `state` interface. This is practically always done by inheriting from the `skeleton_state` class, which provides basic functionality for manipulating a kinematic skeleton.

When building a new state the most essential function to implement in the derived class is `predict`. This function draws a random pose from the prior distribution and hence encodes the prior in the particle filter. The best way to figure out the details of how to build a new class is to study an example. We recommend looking into the `angle_state` and `projection_state` classes as these are the two most basic examples and they show how to work with both joint angles and joint positions.

Once you have created your new `state` class you need to register it with the tracking framework. This is a two-step procedure, where you first need to ensure that the configuration system knows your new class such that it can be selected at run-time. To do this, edit `options.h` and add a new state name to the `tracker_state_type enum`. Then, edit `options.cc` and add a check for your new state in the

```
if (option_name == "state_type")
```

branch of the constructor. You can just imitate the code for the other `state`'s.

Now, the configuration system knows your new prior class. The second step of the two-step procedure is to register the class with the template system. This is done by editing the macros in `macros.h`. Just add your new class to the list of classes marked with `EXPAND_PLAIN_OBS`. This macro will expand the different combinations of input data with your new class such that all combinations are available at run-time. This part of the code is non-trivial so we recommend to just mimic the other classes in the beginning.

## 5.2 The Observation

Different types of observations are currently supported by the tracking system. At the moment it is possible to use the `TriClops` library along with Point Grey Bumblebee stereo cameras. It is also possible to use the Microsoft Kinect camera as well as reading point clouds from text files where each line contains the $x$ $y$ $z$ $R$ $G$ $B$ coordinates of a point. These are implemented by the `triclops_cloud`, `kinect_cloud` and `point_cloud` respectively. These classes all inherit from the `cloud` class that implements basic point cloud functionality. New point clouds can easily be added in much the same way as `state`'s were added in the previous section.

If you want to be able to work with other types of input data than point clouds you currently have to make compile-time changes to the code. You will basically have to reimplement the `measure` class in `skeleton_cloud_measure.h` file.

# 6 Final Remarks

This document has given some basic knowledge about the HUMIM tracking software. We have, however, only scratched the surface of the software, so you will need to study the code before making proper use of the software. We hope the software is useful to you, and if you extend it in interesting ways we would love to hear about it and distribute your extensions as well (with full credit given to you).

# References


[1] Søren Hauberg and Kim S. Pedersen. Stick It! Articulated Tracking using Spatial Rigid Object Priors. In R. Kimmel, R. Klette, and A. Sugimoto, editors, *ACCV 2010*, volume 6494 of *Lecture Notes in Computer Science*, pages 758–769. Springer, Heidelberg, 2011.

[2] Søren Hauberg and Kim Steenstrup Pedersen. Data-Driven Importance Distributions for Articulated Tracking. In Yuri Boykov et al., editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, Lecture Notes in Computer Science. Springer, 2011.

[3] Søren Hauberg and Kim Steenstrup Pedersen. Predicting Articulated Human Motion from Spatial Processes. *International Journal of Computer Vision*, 94:317–334, 2011.

[4] Søren Hauberg, Stefan Sommer, and Kim Steenstrup Pedersen. Gaussian-like Spatial Priors for Articulated Tracking. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *ECCV*, volume 6311 of *LNCS*, pages 425–437. Springer, 2010.

[5] Søren Hauberg, Stefan Sommer, and Kim Steenstrup Pedersen. Natural Metrics and Least-Committed Priors for Articulated Tracking. *Image and Vision Computing*, 2011.

[6] Hedvig Kjellström, Danica Kragić, and Michael J. Black. Tracking people interacting with objects. In *IEEE CVPR*, 2010.

[7] Hedvig Sidenbladh, Michael J. Black, and David J. Fleet. Stochastic tracking of 3d human figures using 2d image motion. In *ECCV*, volume II of *LNCS 1843*, pages 702–718. Springer, 2000.