Virtual data generation based on a human model for machine learning applications

K. Buys[1], J. Hauquier[2], C. Cagniart[3], T. Tuytelaars[4], and J. De Schutter[1]

[1]Dep. of Mechanical Engineering, KU Leuven, Belgium
[2]eJibe.net, Belgium
[3]Technical University of Munich, Germany
[4]Dep. of Electrical Engineering, KU Leuven, Belgium

Abstract

Most computer vision algorithms for human detection are grounded on a intensively data driven machine learning pipeline. Creating this pipeline is a time and computationally intensive step, so is collecting all the input data for this. Often manually annotated real life images are used as input data, this poses two drawbacks, first only a limited number of datasets are available, secondly this is time intensive or expensive to acquire. This paper presents a work flow to generate input data for human pose detection machine learning algorithms that is grounded on real human motions but is generated in a virtual environment with an accurate sensor model.

Keywords: Data generation, Machine learning, Human Avatar

1 Introduction

Today in the computer vision and gesture recognition field a large number of detection algorithms rely on a data driven machine learning approach to train the detector. This requires training data consisting out of a series of sensor measurements (images or motion capture data or ...) together with the ground truth for these measurements.

Labeling these measurements is often an intensive manual process that can be crowd sourced and distributed with services like Amazons Mechanical Turk [1, 2], however then redundancy is needed to get exact ground truth as you are dependent on the dedication of the worker accepting the task. Patient confidentiality can pose an issue here. A number of services try to provide this redundancy [3, 4] or provide a framework for easier integration [5].

As shown earlier by Shotton et al. [6, 7] for a number of machine learning algorithms this data can also be created in a virtual model as long as the sensor model is well known and the input data shows enough similarities with the real world, thus creating an accurate virtual representation of a real life measurement. However Shotton et al. didn't release their training framework publicly.

This paper presents an open source training framework to generate virtual measurements of humans with a Primesense [8] based RGB-D camera model (like Microsoft Kinect [9] or Asus Xtion Pro Live [10]).

The pipeline consists out of the Makehuman [11] model (which was evaluated in prior work by Buys et al. [12, 13]) that is mapped on human motion capture data in BVH files. BVH files are widely accepted as a standard and are freely available (like the CMU mocap dataset [14]).

First in section 2 the current state of the art

Corresponding author: Koen Buys Email: buys dot koen (at) gmail dot com

will be discussed, after in section 3 we'll discuss the system architecture.

2 Related work

When replacing real life data with virtually created data for machine learning it is important that the created data is still realistic enough to create an accurate representation. For this implementation we fixed ourselves on devices with an camera architecture from Primesense. The manufacturer hasn't released extended specifications for these devices, but they have been studied quantitatively by Bainbridge-Smith and Sumar [15]. This study provides many camera specifications. The RGB-D camera has a built-in RGB-CCD camera and projected IR light triangulation depth sensor. The field of view is 58° horizontal, 45° vertical and 70° diagonal and it runs at a frame rate of 30 frames per second. The output of the RGB camera is a 680x480 pixel image with a 32 bit color value. The depth image is streamed at 30 frames per second at a resolution of 640x480 in a 16 bit image of which only eleven bits are used.

Furthermore, Khoshelham [16] and Kramer et al. [17] did some accuracy analysis of the depth data on a Kinect. They concluded that the density of points in the point cloud decreases with increasing distance to the IR camera. The reason for this is the depth resolution, which is low at large distance. It is seven centimeter at the advised maximum range of five meter with an additional four centimeter noise on the measurement. It is important for our realistic simulation to represent these quantization effects in the data.

Gschwandtner et al. [18, 19] presents a very accurate model of a Microsoft Kinect in their Blensor framework [20] which is an extension to Blender [21] that fixes the disparity quantization on a 1/8 pixel resolution and very accurately captures the parallax effect around objects. However it doesn't consider the correlation window (9x9 or 9x7) of the on-board hardware matching algorithm between the IR projector and the IR camera. And still additional noise needs to be added because the diffraction grating of the projector is very likely to have some minimal distortions from the manufacturing process.

A number of other simulation frameworks exist that try to give a virtual implementation of a real life sensor. These frameworks like MORSE [22, 23, 24] (which is also based on Blender [25]) and Gazebo [26, 27] aim for a multi robot simulation environment and offer a large overhead in communication aspects.

Figure 2: Ground truth output image, every color matches a label

3 System overview

Two versions where made of the system, first a component oriented system in the ROS framework [28] that is connected as seen in the system diagram in figure 1. In the second version the system is fully integrated into MakeHuman and can be loaded as a plugin.

The first version can be used in a command line interface on a server with the advantage to facilitate distributed calculation on a cluster, the second can only be run in a GUI with the advantage of providing more feedback to the user.

The system takes two inputs, the human avatar (mesh with kinematics) and human motion capture files and output two files for each pose, a depth image and a ground truth image, this can be seen in figure 1.

3.1 BVH files

The input data obtained from motion capture sources will be provided in the Biovision hierarchical data file format (BVH). BVH files can be found bought from specialized motion cap-

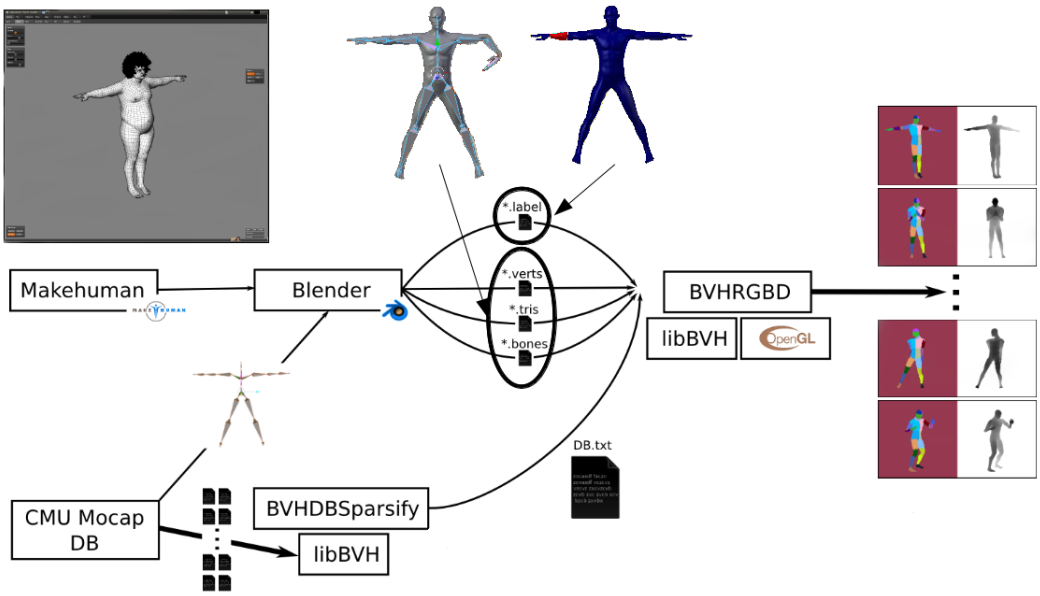

Figure 1: Generation pipeline of the component oriented implementation. A mesh in generated in MakeHuman and mapped onto motion capture data from BVH files in Blender. These meshes are then rendered in an OpenGL environment.

ture providers [29] or extracted from existing datasets [14, 30, 31]. It is a commonly used textual format that is relatively easy to understand and parse, it declares in the first part the rest pose and in a second part the motion data . It is a joint-based format (in contrast with bone-based formats) that describes connected joints in a hierarchic manner. Bones are defined implicitly as the relation between each two joints, where the relative offset from the parent joint to the first child is the bone length and rest-pose direction. Joint offsets and rotations are defined in a local manner, relative to the parent joint, so in order to calculate the full transformation of a joint in world-space coordinates one needs to propagate the matrix calculations from a joint up till the root joint. To finish the bones at the leaf nodes, special end_connector joints are used. Since mostly the rotations matter for applying a pose (except for the root bone that can be translated to make a character move), a rotation on a joint or a bone is essentially the same.

Porting this file format poses two problems, first the rest pose can be different, secondly the kinematics (rig) can be different (fig. 3). The default rig is the one used in makehuman and is called the MHX rig.

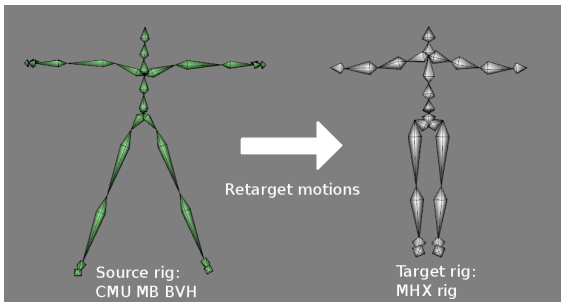

Figure 3: As kinematics can be different a retargetting needs to be applied, here for the Carnegie Mellon University Motion Capture format (CMU MB) to MakeHuman Exchange format (MHX)

3.2 Retargetting

A mapping between a source and a target rig can be as simple as a one-to-one mapping between a joint in the source rig to the target rig, referencing them by name, together with a roll correction value to account for the difference in rest pose (most conventional rest pose variations are the T stance with either legs apart or held together as illustrated in figure 3).

Each joint of the BVH file should be addressed (while some MHX target rig joints can be omitted) but can be mapped to "None", indicating they are not mapped onto a joint on the target rig. Care needs to be taken when doing this, as the rotation of the parent joint actually defines how a bone is rotated. When a BVH joint is mapped to "None", but the children of that joint are mapped onto the target rig, the rotation values need to be accumulated into the parent bone of the targeted MHX rig.

However because of the varying nature of the joint hierarchies defined in BVH files, we need to account for the differences in skeleton when mapping the pose transformations onto the skeleton of our base mesh. The implemented approach is based on the work of Monzani et al. [32] where retargetting works using an indirection in the form of an intermediary skeleton, and the work of Feng et al. [33] and Hecker et al. [34].

Because of the often redundant poses in BVH files we sparsify the original file with a minimal angular difference required on the joint angles. As the vector of joint angles is the input for this sparsify operation, this is a time consuming step as each new vector is iterated over all accepted vectors. However this only needs to be done once for all the poses and can be kept as the base mesh deforms. Failing to do this would bias the machine learning techniques giving an incorrect information gain during the learning.

3.3 Human base mesh

The mesh structure defines the look of the person, as demonstrated in earlier work [12, 13], we can acquire this look in an automated fashion. We use the MakeHuman mesh as the base mesh and use the MakeHuman mesh deformation options in the GUI to define a number of meshes (figure 4 for which to generate the output data. This allows the user to select meshes on age, gender, etnicity, ...

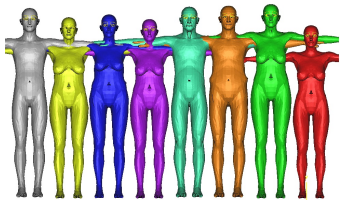

Figure 4: The MakeHuman base mesh allows for natural deformation.

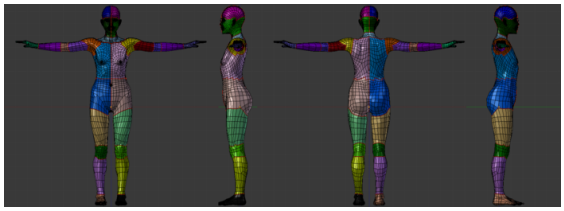

Figure 5: The MakeHuman base mesh showing vertex labels.

The output data needs to be accompanied with a ground truth image, for this the user needs to declare his body part labels. This focuses on a per-vertex labeling of body parts on the MH base mesh. The idea is to assign each vertex of the body to at least one vertex group, named after the user body part declaration (Fig. 5). Each vertex within the same group will be assigned the same vertex color, that can be used in a GLSL fragment shader for rendering out the correct colors for use in the test data.

This rendering approach also allows efficient rendering using only one draw call per generated image (as the model can be rendered in one batch). Individual body parts are marked using edges that were marked as seams, allowing face selection of the faces within one body part. The vertices of each body part were added to vertex groups with a corresponding body part name, and are assigned a diffuse colored material per body part. Future work includes to instance double vertices where vertices are shared among vertex groups, so that they can be given separate vertex colors for rendering.

Additionally this labeling has also been done on the clothes helpers (that are an implicit part of the base mesh, figure 6), which could allow automatic body part tagging on clothed humans.

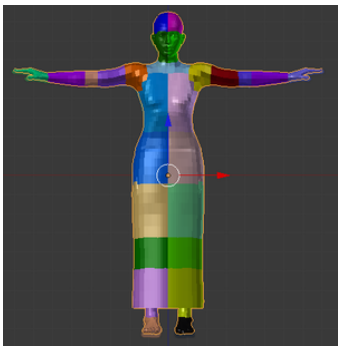

Figure 6: The MakeHuman base mesh showing vertex labels on clothing.

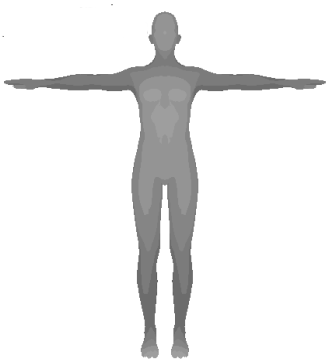

Figure 7: The depth image discretization.

3.4 Rigging

To apply a pose from a skeleton onto a mesh (in our case the human base mesh) we need to transfer the joint transformations (mostly rotations) onto the body vertices that are mapped to that joint. This can be done in a very simple way by defining a one-to-one mapping of each vertex to a bone and rigidly transforming those vertices along with their bones. This will, however, not result in a visually pleasing result. Therefore we use a technique called rigging [35, 36]. Rigging is a concept commonly used in 3D animation where vertices can be assigned to a number of bones, together with a weight value between 0 and 1. This weight is a scalar that determines how much of the bone rotation is applied to that vertex. Usually the sum of weights for a vertex amounts to 1 (eg. the weights are normalized). By storing the weight values in vertices of fixed length, we can increase the performance of rigging calculations using SIMD operations. The rigging needs to be optimized since it needs to be recalculated every frame of the animations. In real-time 3D applications, rigging is often calculated on the GPU using shaders (and referred to as hardware rigging), where a limited amount of bone weights is preferred (eg. 3).

In the component based implementation the rigging is done scripted in Blender and can be adjusted by the user with weight painting, in the MakeHuman implementation the rigging is fixed on the MHX mesh and is connected to the vertex indices.

3.5 Rendering

As a final step, the articulated avatar is placed in front of a virtual camera in an OpenGL environment to create an accurate virtual measurement. For this an accurate virtual representation of the camera was build taking into account the noise model and the quantization and triangulation effects that occur on the real camera. This step takes the intrinsic and extrinsic camera calibration as input together with the deformation matrix.

In the component based implementation this is done in a stand-alone OpenGL application that takes a calibration matrix and a deformed and posed mesh as input. This is done in a stand-alone fashion to avoid the overhead of Blender and makes it easier to distribute over a computer cluster as each unit in the cluster keeps it's own OpenGL environment and calculates a section of the poses.

The MakeHuman plugin version uses the OpenGL environment available in MakeHuman from which the Z-buffer is loaded on which the discretization effects are performed. This discretization can be seen in figure 7.

4 Results

The implemented pipeline was successfully tested to train randomized decision forests (RDF) and ferns to achieve human pose detection as described extensively by Shotton et al. [7] and Buys et al. [37]. As shown by Shotton et al. the depth image can be extensively discretized (up to the level of a simple binary image). We learned a RDF based on 180k images generated

Figure 10: Result image of the random decision forest labeling learned with virtual training data.

in the presented pipeline, a set of example images can be seen in figure 8 and figure 9.

An example of the end result labeling with the RDF based on real life input data can be seen in figure 10.

The framework presented in this paper will be available publicly at `https://github.com/KoenBuys`. A link to the output data will become available on `http://people.mech.kuleuven.be/~kbuys/` allowing for users to directly start their machine learning algorithms.

5 Conclusion

We've presented an approach for generating realistic sensor data of a human model with ground truth labeling. It is however important to not that we use this data only as positive input data to our machine learning algorithms. For negative training examples (e.g. examples where human is visible in the data) we still use real life data manually captured. However as the negative training examples don't need to be annotated, this process is not time intensive nor computationally. Future work includes adding more clothing types to the meshes and adding additional labeled environments (floor, ceiling, desks, ...) in the OpenGL environment in order to get more realistic depth images.

Acknowledgements

The authors would like to acknowledge Nvidia for their financial contributions and technical support for this project. The Flemish and the German government for financially supporting the authors.
Koen Buys is funded by KU Leuven's Concerted Research Action GOA/2010/011 Global real-time optimal control of autonomous robots and mechatronic systems, a PCL-Nvidia Code Sprint grant, an Amazon Web Services education and research grant, this work was partially performed during an intern stay at Willow Garage.
Anatoly Basheev is funded by Nvidia as support for the PointCloud Library.

References

[1] Amazon, "Amazon mechanical turk." `https://www.mturk.com`, 2013.

[2] A. Sorokin and D. Forsyth, "Utility data annotation with amazon mechanical turk," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pp. 1–8, 2008.

[3] Crowdflower, "The world's largest workforce." `http://crowdflower.com/`, 2013.

[4] HumanGrid GmbH, "Clickworker." `http://www,clickworker.com`, 2013.

[5] A. Torralba, B. Russell, and J. Yuen, "Labelme: Online image annotation and applications," *Proceedings of the IEEE*, vol. 98, no. 8, pp. 1467–1484, 2010.

[6] J. Shotton, M. Johnson, and R. Cipolla, "Semantic texton forests for image categorization and segmentation," in *CVPR*, 2008.

[7] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from a single depth image," in *CVPR*, 2011.

[8] Primesense. `primesense.com`, 2010.

[9] Microsoft XBOX Kinect. `xbox.com`, 2010.

Figure 8: Ground truth output images.

Figure 9: Depth output images.

[10] ASUS Xtion PRO. www.asus.com, 2011.

[11] M. Bastioni, M. Flerackers, and J. Capco, "MakeHuman." makehuman.org, 2012.

[12] K. Buys, D. V. Deun, T. D. Laet, and H. Bruyninckx, "On-line generation of customized human models based on camera measurements," in *International Symposium on Digital Human Modeling*, June 2011.

[13] D. V. Deun, V. Verhaert, K. Buys, B. Haex, and J. V. Sloten, "Automatic generation of personalized human models based on body measurements," in *International Symposium on Digital Human Modeling*, June 2011.

[14] Carnegie Mellon University, "CMU Graphics Lab Motion Capture Database." mocap.cs.cmu.edu/, 2011.

[15] L. Sumar and A. Bainbridge-Smith, *Feasability of Fast Image Processing Using Multiple Kinect Cameras on a Portable Platform*. PhD thesis, Department of Electrical and Computer Engineering University of Canterbury Christchurch, New Zealand, •.

[16] K. Khoshelham, "Accuracy and resolution of kinect depth data," 2011.

[17] *Hacking the Kinect*. Technology in action, 2012.

[18] M. Gschwandtner, R. Kwitt, A. Uhl, and W. Pree, "BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing," vol. 6939 of *Lecture Notes in Computer Science*, ch. 20, pp. 199–208, Berlin, Heidelberg: Springer Berlin / Heidelberg, 2011.

[19] M. Gschwandtner, *Support Framework for Obstacle Detection on Autonomous Trains*. PhD thesis, Department of Computer Science, University of Salzburg, Austria, 2013.

[20] M. Gschwandtner, "The blender sensor simulation." http://www.blensor.org/.

[21] Blender Foundation. blender.org/.

[22] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular openrobots simulation engine: Morse," in *Proceedings of the IEEE ICRA*, 2011.

[23] G. Echeverria, S. Lemaignan, A. Degroote, S. Lacroix, M. Karg, P. Koch, C. Lesire, and S. Stinckwich, "Simulating complex robotic scenarios with morse," in *SIMPAR*, pp. 197–208, 2012.

[24] S. Lemaignan, G. Echeverria, M. Karg, J. Mainprice, A. Kirsch, and R. Alami, "Human-robot interaction in the morse simulator," in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pp. 181–182, ACM, 2012.

[25] K. Buys, T. D. Laet, R. Smits, and H. Bruyninckx, "Blender for robotics: integration into the leuven paradigm for robot task specification and human motion," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2010.

[26] Open Source Robotics Foundation, "Gazebo." http://gazebosim.org/.

[27] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, pp. 2149–2154 vol.3, 2004.

[28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[29] Motion Capture Data. http://www.motioncapturedata.com/.

[30] The Motion Capture Society (MCS). http://www.motioncapturesociety.com/.

[31] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber, "Documentation mocap database hdm05," Tech. Rep. CG-2007-2, Universität Bonn, June 2007.

[32] J.-S. Monzani, P. Baerlocher, R. Boulic, and D. Thalmann, "Using an intermediate skeleton and inverse kinematics for motion retargeting," in *Computer Graphics Forum*, vol. 19, pp. 11–19, Wiley Online Library, 2000.

[33] A. Feng, Y. Huang, Y. Xu, and A. Shapiro, "Automating the transfer of a generic set of behaviors onto a virtual character," in *Motion in Games*, pp. 134–145, Springer, 2012.

[34] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen, "Real-time motion retargeting to highly varied user-created morphologies," in *ACM Transactions on Graphics (TOG)*, vol. 27, p. 27, ACM, 2008.

[35] I. Baran and J. Popović, "Automatic rigging and animation of 3d characters," in *ACM Transactions on Graphics (TOG)*, vol. 26, p. 72, ACM, 2007.

[36] M. Pratscher, P. Coleman, J. Laszlo, and K. Singh, "Outside-in anatomy based character rigging," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 329–338, ACM, 2005.

[37] K. Buys, C. Cagniart, A. Baksheev, T. D. Laet, J. D. Schutter, and C. Pantofaru, "An adaptable system for rgb-d based human body detection and pose estimation," *Journal of Visual Communication and Image Representation*, no. 0, pp. –, 2013.