Technical Report
IDP Project
Markerless Motion Capture in the Operating Room

October 2010 - June 2011

Benoit Diotte diotte@in.tum.de
Cédric Cagniart cagniart@in.tum.de
Slobodan Ilic ilic@in.tum.de
TU München, Garching

Abstract

The Goal of this IDP was to extend the patch-based surface tracking algorithm proposed by Cagniart et al.*, to a skeleton based solution. Such a solution could be used to track surgeons in an operating room and enhance human-machine interaction paradigms in this context. We implemented a model-based motion capture algorithm based on 3D point cloud data. An Expectation-Maximisation algorithm computes the point cloud/body parts assignment and the pose estimation is based on an Inverse Kinematics framework. This final report includes a brief introduction on the topic of motion capture, a discussion of the related works, a description of the method and its implementation, and concludes with some appreciation of the results.*

1. Introduction

For decades now, computer vision scientists have been looking at objects, and have been challenged by the search for algorithms allowing to find and recognize objects. In this work, we focus on systems capturing human motion. When human beings are the objects to track, many challenges arises. Humans are highly articulated objects, which means that pose changes (motion) are the result of complex transformations. The common models comprise around 14 body parts and at most 6 DOF per parts, if only rigid transformations are assumed. They can move fast and the parts with sharpest acceleration, i.e. hands, are relatively small compared to the rest of the body. They can adopt poses which result in body parts occlusions in the camera projections. They are nonrigid objects, their shape deforms. Humans share generally a common bone structure, but the size of those bones varies a lot from one instance to another and so does the shape of the flesh on the skeleton. In general scenes, people will wear a variety of clothing, occluding the body in various ways. So tracking their motion based on 2D optical images is an hard problem. No currently known systems solve all those challenges at once and it is common to simplify the problem by tuning the system on the expected input and desired output data.

In the next paragraphs, we give a brief overview of the different application areas using the taxonomy of Moeslund and Granum [17] and see how these drive the assumption making process.

Control Control applications typically offer gesture-driven interfaces to further subsystems. So the motion capture component must deliver fast and accurate output. Robustness may or may not be an issue if a controlled environment is an option.

Surveillance For surveillance applications on the other hand, typical scenes are cluttered and crowded, therefore robustness is one of the most relevant issue. The output of the motion capture component is not expected to be very precise and detailed in terms of motion description but informative enough in order to allow behavioural conclusions.

Analysis When it comes to designing analysis applications, accuracy is of absolute importance. Generally, analysis takes place in controlled environments and offline.

Markerless vs. Marker-based One way to simplify the task is the use of markers. Solutions using no markers on the body are prefered, since they allow more flexibility. So we focus on markerless motion capture (MMC).

Modelfree vs. Model-based Solutions may (Model-based) or may not (Modelfree) be using an explicit human

body model. The use of an a priori model surely add some robustness to the system, but is not a trivial pursuit. The design of an appropriate model is highly complex. Nonetheless, many works have showed that careful approximations give reasonable results. This work does use an explicit articulated model.

Depending on the nature of the desired input data representation, many camera configurations are in use. They range from a single camera, monocular, to two (stereo) or more point of views (multi-camera setups). We use a multi-camera setup. This setup helps to resolve ambiguities due to occlusions.

Our algorithm does the following motion assumptions: the subject remains inside the workspace, no camera motion, only one person in the workspace at the time, no occlusions from alien objects, slow and continuous movements and high sampling rate.

We see for this work potential uses in the field of control and analysis applications.

2. Related Works

It can be helpful for newcomers to look into the various surveys available on MMC. While giving a strong focus on taxonomy, like Gavrilla [10], Wang *et al*. [27], Poppe [22] and Moeslund *et al*. [17, 18], they report on the various challenges involved in motion analysis and the different approaches to tackle these. Espacially, Moeslund surveys report on about 500 publications published up to 2006. A comprehensive review of the field topics can be found in Forsyth *et al*. [8]. More recent surveys concentrate on a more restrained set of approaches to markerless motion capture (MMC), Tran and Trivedi [25] concentrates on model-based works using volumetric representation up to 2008, Ji and Liu [12] on view-invariant methods.

Following the lines of Forsyth *et al*. [8] and Moeslund [17, 18] the general problem of model-based MMC can be subdivided into the following common tasks: modeling, data preprocessing and pose estimation.

Modeling The first task for a model-based approach is to acquire a model of the objects in the scene, here those objects are humans. There is the question as to how the model is defined, and how it is initialized to represent a particular object. Typically the detail level of the model is driven by the desired output and chosen algorithm. Most methods use strong prior knowledge on the deformation of the observed object in the form of articulated models.

Pose Model

Generally the pose model encodes joint reference positions, body parts dimensions and orientations. Most commonly the desired output of MMC systems is a track of rigid transformations describing the full body configurations over time.

The influential works by Bregler *et al*. [2] introduced the use of the twist representation [19] and exponential maps. Pons-Moll and Rosenhahn[21] demonstrate that using a ball model to represent joints with 3 rotational degrees of freedom, like shoulders and hips, resolves gimbal lock problems resulting from using 3 adjacent revolute joints, which is equivalent to Euler angles. Some solutions do without a kinematic tree representation like the works of Sigal *et al*. [24] and Kakadiaris and Metaxas [13].

Shape Model

In order to find the object in the data, we also need a model of the body's shape. The shape model describes how the body should look like in the data.

Some methods use regular volumetric shapes, like ellipsoids or tapered cylinders [16], or more complex layered versions of these like in [20]. Most common is to use a 3D reconstruction of actual body shape like in Carranza *et al*. [4], Vlasic *et al*. [26] or more recently in Gall *et al*. [9].

The shape model of our tracking algorithm is a 3D reconstructed mesh to which we fit manually a twist based pose model with 25 DOF. See section 3.1.

There are solutions to initialize these models online, see [1], but most works focusing on the tracking task, as we do, initialize the model manually. Since the shape of the body in the data may change over time the shape model may have to be updated online in order to enhance tracking quality. In this work, the shape model do not change over time.

Data Preprocessing Once a body model is defined, the next step is to preprocess the raw video data such that it can be used in a sensible way for the pose estimation. This task is completely assumed a priori in this work, but we mention it for the sake of completeness.

Segmentation

At the lowest level, one must identify 2D image regions belonging to the projection of the subject (foreground) from regions that do not (background). Since in most studios, the background is assumed static and its appearance chosen to contrast well with the objects, usually some simply background substraction strategies are used to perform segmentation.

Representation

Most tracking algorithms don't work though directly on the basis of the lowest data level, but need some higher level representation. The representation tasks transforms the input data, pixels in 2D images, to a data representation fitting the model-data matching method.

Our system works directly on 3D reconstructions, like in Mikic *et al*. [16], Cheung *et al*. [5] and Cheng and Trivedi [23].

Pose Estimation After the data from the video is delivered in an adequate representation, the missing step towards a pose is to match the model to the data in a meaningful way and search for the model configuration that optimize this match according to some fitness measure, or cost function as measure of nonfitness.

Model-Data Matching

The role of the model-data matching procedure is to provide a matching and a measure of how good this matching is. The matching is the input to a cost function and its output is the measure.

Typical for solutions based on volumetric data is to establish correspondences between points in 3D space and to measure the fitness of the matching in terms of euclidean distance between those points, that is to cast the problem into a point cloud registration problem.

Since we have to deal with an articulated object, the correspondence assignment have to take into account that points are moving according to different rigid transformations.

A method for this registration problem is ICP, but this algorithm as well as its extensions lack robustness when confronted with outliers because of the determinism in the choice of point assignments.

Among the recent approaches addressing the problem in a probabilistic framework, are works by Horaud *et al*. [11] and Cheng and Trivedi [23]. These approaches use the Expectation-Maximization algorithm to iteratively reevaluate smooth assignments between the model and the data. We do as in Cagniart *et al*. [3] interleaving model-data matching and parameter search in an iterative algorithm. See section 3.3.

Parameter Search

Once a solution has a procedure to match model and data and an appreciation of this match, what is left is to determine a strategy to optimize the model parameters such that those represent the best possible match. This is a general cost optimisation problem. Since the function from parameter space to 3D body position space is nonlinear, two approaches can be taken. Either one assumes an high sampling rate, small changes in parameters, and a local linear approximation or few assumptions on motion but an expensive global search. Poppe [22] and Moeslund [17] refer to these as single or multiple hypothesis tracking.

Among the earlier works, a popular strategy was to use an extended Kalman filter as in Mikic *et al*. [16]. Due to its intrinsic strong linear motion prior, this approach would rapidly lose track of the object. Other local search strategies, based on curvature analysis of the energy function, are gradient-based as in Cheng *et al*. [23]. and stochastic meta descent (SMD) by Kehl *et al*. [14], where data is iteratively resampled.

More recent global approaches, on the other hand, sample the parameter space into multiple hypothesis. Among those are grid search as in Carranza *et al*. [4]. and particle filtering and its extensions, for instance the annealed particle filtering by Deutscher *et al*. [7]. These strategies however make the solution more expensive in proportion to the number of samples, or particles, tracked. Much effort is invested in heuristics to lower the number of samples needed.

As we aimed at a realtime solution, we adopted a local search strategy based on gradient descent and implemented the Gauss-Newton method. See section 3.2.

3. Method

3.1. Modeling

Our human body model is an articulated shape model. It is composed of a pose and a shape model.

Pose Model

The pose model can be compared to the human skeleton. It is composed of 5 open kinematic chains. A kinematic chain is a serial composition of segments attached together by joints. The chain is open if one end is loose (not constrainted by a joint). We assume that those joints, in the human body, can perform only rotations with different degrees of freedom. The root joint, here the torso, is a special joint, performing translations. Since all chains connect to this joint, they all perform the same pose translation. We parameterize those joints with the twist representation. See Appendix A.1 for an introduction into twist representation.

We found that we could build an approximative model of the human skeleton with the following three types of joints:

Translation The translational joint is parameterized with a vector $t = (\Delta t_1, \Delta t_2, \Delta t_3)$ representing the translation from the reference pose. It has 3 DoF.

Revolute The revolute joint is parameterized with a fixed axis ω and a scalar $\Delta\theta$ representing the rotation performed from the reference pose. It has 1 rotational DoF.

Ball The ball joint is parameterized with a vector $\omega = (\omega_1, \omega_2, \omega_3)$ giving the axis and amount of rotation induced by the joinr. It has 3 rotational DoF.

We represent the pose configuration with a vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_K)$, where the first 3 parameters represent the

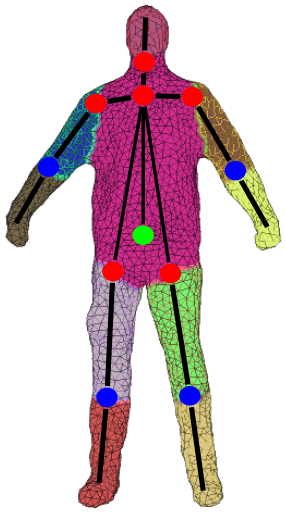


Figure 1: Red shows ball joints, green translational and blue revolute joints.

overall translation and the rest, the rotational parameters, represent the kinematic chains flatten such that all parameters have an higher index as their kinematic ancestors.

Given an open kinematic chain with n axes of rotation, all parameterized with twists $\hat{\xi}$, it can be shown that:

$$\bar{\boldsymbol{x}}(\boldsymbol{\theta}) = e^{\hat{\xi}_1\theta_1} e^{\hat{\xi}_2\theta_2} \dots e^{\hat{\xi}_n\theta_n} \bar{\boldsymbol{x}}(\mathbf{0}) \tag{1}$$

where $\bar{\boldsymbol{x}}(\boldsymbol{\theta})$ is the position of a point on the last link of the chain, in the configuration described by the n angles $\boldsymbol{\theta} = [\,\theta_1\ \theta_2\ \dots\ \theta_n\,]^\top$ (in homogenous coordinates); $\bar{\boldsymbol{x}}(\mathbf{0})$ represents the same point at a reference configuration of the chain. Equation (1) is called the product of exponentials formula for an open kinematic chain and it can be shown that it is independent of the order in which the rotations are performed. The angles are numbered going from the chain base toward the last link.

Shape Model

The shape model is given by the polygonal surface reconstructed from voxel data. The vertices of the model shape are assigned to their corresponding body parts. This process is sometimes called skinning and should be performed automatically. For this work, we used the free available 3D contruction software Blender[1] which offers this computation based on manually placed body parts envelopes. With those links from body parts to vertices on the body surface one can solve the forward kinematic subproblem of computing the position of vertices given some pose configuration, with eq. (1).

Figure 1 shows the pose and shape model overlayed.

[1] http://www.blender.org/

3.2. Parameter Search

In the further description of our method, let us reverse the process order. Since our method solves the pose estimation task in an iterative manner between model-data matching and parameter search, let us begin with the one that assumes the least. For this section, where we explain how parameter search is solved, we will assume a model-data matching given.

3.2.1 Inverse Kinematics through Energy Optimisation

The inverse kinematic problem is to find the parameter values of the joints of the kinematic chain such that points attached to the chain reach given positions. We solve this problem by iteratively improving the parameter approximations w.r.t. a cost function. The cost function is:

$$E(\boldsymbol{\theta}) = \sum_{y \in Y} \|\boldsymbol{x}(\boldsymbol{\theta}) - \boldsymbol{y}\|^2 = \sum_{y \in Y} \|r(\boldsymbol{\theta})\|^2 = \|\boldsymbol{r}(\boldsymbol{\theta})\|^2 \tag{2}$$

where $\boldsymbol{x}$ is a 3D point of the shape model and $\boldsymbol{y}$ is a corresponding 3D point in the observed data cloud. The cost function is basically modeling the sum of squared distances between point correspondences at model configuration $\boldsymbol{\theta}$. In order to find the model configuration minimizing this energy, we first approximate the value of E after an small update $\Delta\boldsymbol{\theta}$ by the first order linear approximation of $E(\boldsymbol{\theta} + \Delta\boldsymbol{\theta})$.

$$E(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = \|\boldsymbol{r}(\boldsymbol{\theta}) + J_{\boldsymbol{r}}(\boldsymbol{\theta})\Delta\boldsymbol{\theta}\|^2 \tag{3}$$

Such that the problem of minimizing $E(\boldsymbol{\theta})$ reduces to

$$\Delta\boldsymbol{\theta}^\star = \arg\min_{\Delta\boldsymbol{\theta}} \|\boldsymbol{r}(\boldsymbol{\theta}) + J_{\boldsymbol{r}}(\boldsymbol{\theta})\Delta\boldsymbol{\theta}\|^2 \tag{4}$$

In the least square sense, it amounts to solving the normal equations

$$\Delta\boldsymbol{\theta}^\star = -\left(J_{\boldsymbol{r}}(\boldsymbol{\theta})^\top J_{\boldsymbol{r}}(\boldsymbol{\theta})\right)^{-1} J_{\boldsymbol{r}}(\boldsymbol{\theta})^\top \boldsymbol{r}(\boldsymbol{\theta}) \tag{5}$$

The optimisation procedure is described in algorithm 1.

In order to avoid a badly conditioned $J_r^\top J_r$ matrix, we add on its diagonal a small constant, the damping parameter λ, i.e. $J_r^\top J_r \approx J_r^\top J_r + \lambda \boldsymbol{I}$.

Here, the attentive reader may point out, that we are adding as parameter updates rotations, and that there are no rotation representations being a vector space, i.e. close under addition. We handle this point in 3.2.2.

Algorithm 1 Solve($\boldsymbol{\theta}$, E, J_r)

$E \leftarrow E(\boldsymbol{\theta})$
repeat
  $scale \leftarrow 1$
  $\Delta\boldsymbol{\theta}^\star \leftarrow -(J_r^\top J_r)^{-1} J_r^\top E(\boldsymbol{\theta})$
  while $E_{swap} \geq E$ **do**
    $\Delta\boldsymbol{\theta} \leftarrow scale * \Delta\boldsymbol{\theta}^\star$
    $\boldsymbol{\theta}_{swap} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
    $E_{swap} \leftarrow E(\boldsymbol{\theta})$
    $scale \leftarrow scale/2$
  end while
  if $E_{swap} < E$ **then**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{swap}$
    $E \leftarrow E_{swap}$
  end if
  $numIter \leftarrow numIter + 1$
until $E < threshold$ **and** $numIter < maxIter$
return $\boldsymbol{\theta}$

3.2.2 Mesh Jacobian

We want to find the rate of change of points $\boldsymbol{x}$ attached to a rigid body parts expressed in spatial coordinates w.r.t. the parameters $\boldsymbol{\theta}$. $\boldsymbol{x}(\boldsymbol{\theta})$ is a vector function in 3 dimensions and depends on K parameters (given by the joints). That is, the derivative of $\boldsymbol{x}(\boldsymbol{\theta})$ is given by the jacobian matrix:

$$J_{\boldsymbol{r}}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial \boldsymbol{x}_1}{\partial \theta_1} & \cdots & \frac{\partial \boldsymbol{x}_1}{\partial \theta_K} \\ \frac{\partial \boldsymbol{x}_2}{\partial \theta_1} & \cdots & \frac{\partial \boldsymbol{x}_2}{\partial \theta_K} \\ \cdots & \cdots & \cdots \\ \frac{\partial \boldsymbol{x}_N}{\partial \theta_1} & \cdots & \frac{\partial \boldsymbol{x}_N}{\partial \theta_K} \end{bmatrix} = \begin{bmatrix} J_{\boldsymbol{x}_1}(\boldsymbol{\theta}) \\ J_{\boldsymbol{x}_2}(\boldsymbol{\theta}) \\ \vdots \\ J_{\boldsymbol{x}_N}(\boldsymbol{\theta}) \end{bmatrix} \tag{6}$$

$J_{\boldsymbol{r}}(\boldsymbol{\theta})$ is a $3N \times K$ matrix. We want to derive the entries of $J_{\boldsymbol{x}_i}(\boldsymbol{\theta})$. Notice that entries of $J_{\boldsymbol{x}_i}(\boldsymbol{\theta})$ are all zeros for parameters not affecting the position of $\boldsymbol{x}_i$, these are parameters not belonging to the kinematic chain of $\boldsymbol{x}_i$.

The new coordinates of a point $\boldsymbol{x_i}$ attached to a rigid body in spatial coordinates are given by:

$$\boldsymbol{x_i}(\boldsymbol{\theta}) = g(\boldsymbol{\theta})\boldsymbol{x_i}(\mathbf{0}) \tag{7}$$

where $g : \mathbb{R}^K \rightarrow SE(3)$ is a function giving the rigid transformation corresponding to parameters $\boldsymbol{\theta}$ according to the reference pose model. Differentiating yields:

$$\begin{aligned} J_{\boldsymbol{x}_i}(\boldsymbol{\theta}) &= J_g(\boldsymbol{\theta})\boldsymbol{x}(\mathbf{0}) = J_g(\boldsymbol{\theta})g^{-1}\boldsymbol{x}(\boldsymbol{\theta}) \\ &= \left[\frac{\partial g}{\partial \theta_1}g^{-1}, \ldots, \frac{\partial g}{\partial \theta_K}g^{-1}\right]\boldsymbol{x}(\theta) \\ &= \left[\frac{\partial g}{\partial \theta_1}g^{-1}\boldsymbol{x}(\boldsymbol{\theta}), \ldots, \frac{\partial g}{\partial \theta_K}g^{-1}\boldsymbol{x}(\boldsymbol{\theta})\right] \end{aligned}$$

Now that we have found an expression for each column of $J_{\boldsymbol{x}_i}(\boldsymbol{\theta})$, let us derive those concretely according to their joint type.

Translate If we parameterize the translate joint with a 3-dimensional vector, it can be shown that

$$\nabla_{i,j}^{\text{translate}(k)} = \begin{cases} 1 & \text{if } i = k, j = 4 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

is the 4×4 linear differential operator for the point affected by a translate joint w.r.t. parameter k.

Revolute A revolute joint is represented by a rotation of magnitude θ around a known axis ω. The rigid transformation given by a rotation of θ radians about the axis ω is given by the exponential map $e^{\hat{\xi}\theta}$.

So if θ_i parameterizes a revolute joint, it can be shown (see Appendix A.2) that:

$$\frac{\partial g}{\partial \theta_i}g^{-1}\boldsymbol{x}(\boldsymbol{\theta}) = \left[T_1 \ldots T_{i-1}\widehat{\xi}_i T_{i-1}^{-1} \ldots T_1^{-1}\right]\boldsymbol{x}(\boldsymbol{\theta}) \tag{9}$$

That is, the linear differential operator for the point affected by a revolute joint is:

$$\nabla^{\text{revolute}} = \left[T_1 \ldots T_{i-1}\widehat{\xi}_i T_{i-1}^{-1} \ldots T_1^{-1}\right] \tag{10}$$

Ball A ball joint represents a rotation about a unknown scaled axis $\theta\omega$.

Let the transformation induced by the ball joint i be R_i. Let its update be $R_i e^{[d\omega]_\times}$.
Then the rigid transformation after update $d\omega$, $T_i(R_i e^{[d\omega]_\times})$, is

$$\begin{aligned} T_i(d\omega) &= \begin{bmatrix} R_i e^{[d\omega]_\times} & (I - R_i e^{[d\omega]_\times})q \\ \mathbf{0}^\top & 1 \end{bmatrix} \\ &= \begin{bmatrix} R_i & (I - R_i)q \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} e^{[d\omega]_\times} & (I - e^{[d\omega]_\times})q \\ \mathbf{0}^\top & 1 \end{bmatrix} \end{aligned}$$

If we assume $e^{[d\omega]_\times} \simeq I + [d\omega]_\times$

$$T_i(d\omega) \simeq \begin{bmatrix} R_i & (I - R_i)q \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} I + [d\omega]_\times & -[d\omega]_\times q \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{11}$$

With this ball joint approximation, it can be shown (see Appendix A.2) that:

$$\frac{\partial x(\boldsymbol{\theta})}{\partial d\omega_i} = -\left[x(\boldsymbol{\theta}) - (R_{1\ldots i}q + t_{1\ldots i})\right]_\times R_{1\ldots i} \tag{12}$$

where $R_{1\ldots i}$ is the total rotation and $t_{1\ldots i}$ the total translation up to joint i. The ball joint is however a joint with 3 DoF and thus covers 3 columns of $J_{\boldsymbol{x}_i}(\boldsymbol{\theta})$. We would like to break the linear differential operator with respect to

the parameters of the rotation axis into 3 linear differential operators, that is $\frac{\partial \boldsymbol{x}(\boldsymbol{\theta})}{\partial \theta_1}$, $\frac{\partial \boldsymbol{x}(\boldsymbol{\theta})}{\partial \theta_2}$ and $\frac{\partial \boldsymbol{x}(\boldsymbol{\theta})}{\partial \theta_3}$, if we assume $d\omega = (\theta_1, \theta_2, \theta_3)^\top$.

After some computations we get a 4×4 linear differential operator w.r.t. each axis parameter j.

$$\nabla^{\text{ball}(j)} = \begin{bmatrix} \left[R^j_{1\ldots i}\right]_\times & -\left[R^j_{1\ldots i}\right]_\times (R_{1\ldots i} q + t_{1\ldots i}) \\ \mathbf{0}^T & 0 \end{bmatrix} \tag{13}$$

where $R^j_{1\ldots i}$ is the jth column of the total rotation up to joint i.

Hence the jacobian matrix can be built in a general way out of 3×1 vectors for each parameter (non-homogenous representation) and the computation of the jacobian column for parameter θ_i is simply the matrix-vector multiplication $\nabla^i x(\boldsymbol{\theta})$ for all joint types.

3.2.3 Joint Limits

To constrain the parameter space further, we add to the cost functions the energy term $E_r(\boldsymbol{\theta})$ penalizing joint configurations going over some limits.

Revolute We limit the revolute joint angle with the function

$$r(\theta) = \begin{cases} \theta - \min & \text{if } \theta < \min \\ \theta - \max & \text{if } \theta > \max \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Ball The reference pose prescribes the limiting vector x_B pointing, for instance, downwards for a hip. R is the current rotation of the joint. The solver detects that $x_B^\top R x_B < t$, for t a given threshold.

The regularization term is then the following expression

$$r(\omega) = \begin{cases} x_B^\top R x_B - t & \text{if } x_B^\top R x_B < t \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

For the $J^\top J$ and $J^\top r$, we have to do the derivative of $x_B^\top R x_B - t$ w.r.t. the update vector $d\omega$.

It can be shown (see Appendix A.3) that

$$J_r(d\omega) = (x_B \wedge R^\top x_B)^\top \tag{16}$$

3.3. Model-Data Matching

We continue the description of the method by motivating the bayesian modeling of our model-data matching approach and by explaining its resolution inside the E-step of an expectation-maximisation algorithm. The M-step of this algorithm consists in finding the parameters maximizing a likelihood function as done in section 3.2.

3.3.1 Bayesian Model

As discussed in Sections 1 and 2 we deal with data-driven surface fitting and cast the problem as the geometric registration of 3D point sets. In a Bayesian context, this means that given a set of observed 3D points and an estimate of the current pose of the mesh, we are faced with a maximum-a-posteriori (MAP) estimation problem where the joint probability distribution of data and model must be maximized:

$$\max_{\boldsymbol{\theta}} \ln P(\mathcal{Y}, \boldsymbol{\theta}), \tag{17}$$

where $\mathcal{Y} = \{y_i\}_{i=1:m}$ is the set of observed 3D points $\{\mathbf{y_i}\}_{i=1:m}$ and their normals.

According to Bayes law $P(\mathcal{Y}, \boldsymbol{\theta}) = P(\mathcal{Y}|\boldsymbol{\theta})P(\boldsymbol{\theta})$. For $P(\boldsymbol{\theta})$, we make the approximation

$$P(\boldsymbol{\theta}) \propto e^{-E_r(\boldsymbol{\theta})}, \tag{18}$$

where $E_r(\boldsymbol{\theta})$ is the energy term defined in Eqs.(14,42).

The likelihood $P(\mathcal{Y}|\boldsymbol{\theta})$ remains to be approximated to complete the generative model. This is done with a mixture of distributions parameterized by a common covariance σ^2, where each component corresponds to a bone B_k. This requires to introduce latent variables z_i for each observation $y_i \in \mathcal{Y}$, where $z_i = k$ means that y_i was generated by the mixture component associated with B_k. We also increase the robustness of our model to outliers by introducing a uniform component in the mixture to handle points in the input data that could not be explained by the body parts. This uniform component is supported on the scene's bounding box and we index it with $N_b + 1$.

$$P(y_i|\boldsymbol{\theta}, \sigma) = \sum_{k=1}^{N_b+1} \Pi_k P(y_i|z_i = k, \boldsymbol{\theta}, \sigma), \tag{19}$$

where the $\Pi_k = p(z_i = k|\boldsymbol{\theta}, \sigma)$ represent probabilities on the latent variables marginalized over all possible values of y_i. In other words they are prior probabilities on model-data assignments. We define them as constants $p(z_i = k) = \frac{1}{N_b}$.

The body part mixture component with index k must encode a distance between the position $\mathbf{y_i}$ and the body part B_k while accounting for the alignment of normals. For computational cost reasons, we model this distance by looking for each body part B_k in its current pose (this means the positions $\{\mathbf{x_i}(\boldsymbol{\theta})\}_{x_i \in B_k}$ and corresponding normals as shown in Fig. 2) for the closest vertex with a compatible normal v_i^k. We consider two points and normals to be compatible when their normals form an angle smaller than a threshold. In practice this threshold was set to 45° in all of our experiments. This leads to the following model for

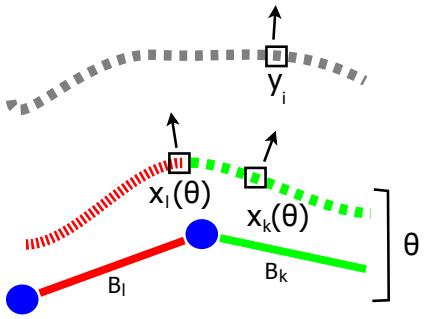


Figure 2: A point/normal y_i with position $\mathbf{y_i}$ from the observed data is associated to v_i^k, the closest vertex with a compatible normal for the body part B_k.

each component of the mixture:

$$\forall k \in [1, N_b],$$
$$P(y_i|z_i = k, \boldsymbol{\theta}, \sigma) \propto \begin{cases} \mathcal{N}(\mathbf{y_i}|\mathbf{x_i}(\boldsymbol{\theta}), \sigma) & \text{if } v_i^k \text{ exists,} \\ \epsilon & \text{otherwise} \end{cases} \tag{20}$$

where ϵ encodes a negligible uniform distribution defined on the scene's bounding box.

3.3.2 Expectation-Maximization

The variables z_i are unobserved but we can use the posterior distributions of Eq. (21) in the Expectation Maximization algorithm [6].

$$P(z_i = k|y_i, \boldsymbol{\theta}, \sigma) = \frac{\Pi_k P(y_i|z_i = k, \boldsymbol{\theta}, \sigma)}{\sum_{l=1}^{N_b+1} \Pi_l P(y_i|z_i = l, \boldsymbol{\theta}, \sigma)}. \tag{21}$$

The idea is to replace $P(\mathcal{Y}|\boldsymbol{\theta}, \sigma)$ with the marginalization over the hidden variables of the joint probability.

$$\ln P(\mathcal{Y}|\boldsymbol{\theta}, \sigma) = \ln \sum_Z q(Z) \frac{P(\mathcal{Y}, Z|\boldsymbol{\theta}, \sigma)}{q(Z)}, \tag{22}$$

where $q(Z)$ is a positive real valued function that sums up to 1. The concavity of the log function allows to write a bound on the function of interest:

$$-\ln P(\mathcal{Y}|\boldsymbol{\theta}, \sigma) \leq -\sum_Z q(Z) \ln \frac{P(\mathcal{Y}, Z|\boldsymbol{\theta}, \sigma)}{q(Z)}. \tag{23}$$

It can be shown that given a current estimate $(\boldsymbol{\theta}^t, \sigma^t)$, it is optimal to choose $q(Z) = P(Z|Y, \boldsymbol{\theta}^t, \sigma^t)$ in that the bounding function then touches the bounded function at $(\boldsymbol{\theta}^t, \sigma^t)$.

This means that the bounding function should be the expected complete-data log-likelihood conditioned by the observed data:

$$-\ln P(\mathcal{Y}|\boldsymbol{\theta}, \sigma) \leq const - E_Z[\ln P(\mathcal{Y}, Z|\boldsymbol{\theta}, \sigma)|Y, \boldsymbol{\theta}^t, \sigma^t]. \tag{24}$$

We rewrite $P(\mathcal{Y}, Z|\boldsymbol{\theta}, \sigma)$ by making the approximation that the observation process that gave $\mathcal{Y}$ draws the y_i's from this distribution in an independent identically distributed way:

$$P(\mathcal{Y}, Z|\boldsymbol{\theta}, \sigma) = \prod_{i=1}^{m} P(y_i, z_i|\boldsymbol{\theta}, \sigma) \tag{25}$$
$$= \prod_{k=1}^{N_b+1} \prod_{i=1}^{m} \left[P(y_i, z_i = k|\boldsymbol{\theta}, \sigma)\right]^{\delta_k(z_i)} \tag{26}$$

The choice made for $q(z)$ then allows to write:

$$E_Z[\ln P(\mathcal{Y}, Z|\boldsymbol{\theta}, \sigma)|Y, \boldsymbol{\theta}^t, \sigma^t] = \sum_{k=1}^{N_b+1} \sum_{i=1}^{m} E_Z[\delta_k(zi)|Y, \boldsymbol{\theta}^t, \sigma^t] \ln[\Pi_k p(y_i|z_i = k, \boldsymbol{\theta}, \sigma)] \tag{27}$$

which finally leads to the expression of the bounding function we need to minimize:

$$-\ln P(\mathcal{Y}|\boldsymbol{\theta}, \sigma) \leq const - \sum_{k=1}^{N_b+1} \sum_{i=1}^{m} P(z_i = k|y_i, \boldsymbol{\theta}^t, \sigma^t) \ln P(y_i|z_i = k, \boldsymbol{\theta}, \sigma). \tag{28}$$

We use the Expectation-Maximization algorithm to iteratively reevaluate the $(\boldsymbol{\theta}, \sigma)$ and the posterior probability distributions on the latent variables $\{z_i\}$.

In the E - Step the posterior $P(z_i|y_i, \boldsymbol{\theta}^t, \sigma^t)$ functions are evaluated using the current estimation $\boldsymbol{\theta}^t, \sigma^t$ and the corresponding local deformations of the mesh. As defined in Equation (21), these functions require to find for each target vertex y_i and body part k the vertex index v_i^k of its nearest neighbor at the configuration of the body part. The complete E-Step amounts to the computation of a $m \times (N_b + 1)$ matrix whose lines add up to 1, as shown in Figure 3. This is an very parallel operation as all the elements of this matrix can be evaluated independently, except for the normalization of each line that takes place afterwards. In theory it would be tempting to use space partitioning techniques to speed up the nearest neighbor search. However the dependency on the orientation of vertex normals makes this cumbersome. In practice we run a brute-force search.

The M - Step requires to minimize the bounding function defined by the the soft data - model assignment weights that

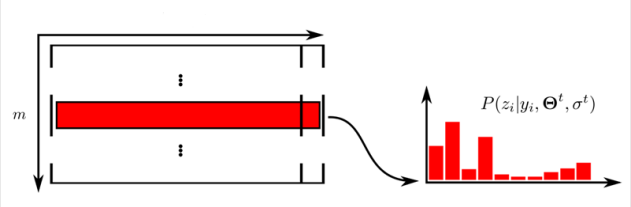


Figure 3: The soft assignment matrix holds the posterior body parts-assignment distributions for every vertex of the target point cloud. As such, the lines are normalized to add up to 1. The last column of the matrix corresponds to the outlier class.

were computed in the E-Step:

$$\boldsymbol{\theta}^{t+1}, \sigma^{t+1} = \arg\min \left[const + E_r(\boldsymbol{\theta}) - \sum_{k=1}^{N_b+1} \sum_{i=1}^{m} P(z_i = k|y_i, \boldsymbol{\theta}^t, \sigma^t) \ln P(y_i|z_i = k, \boldsymbol{\theta}, \sigma) \right] \tag{29}$$

In this bounding function, both data terms and joint limiting terms are weighted squared functions. This fits exactly in the framework defined in Section 3.2 and Equation (2). To prevent the appearance of degenerate mesh configurations, we however do not completely minimize the bounding function. Instead we just run one iteration of the Gauss-Newton algorithm, which amounts to minimizing the quadratic approximation of the objective function around ($\boldsymbol{\theta}^t$, σ^t).

It should also be noted that we do not solve Equation (29) in one maximization step but instead follow the Expectation Conditional Maximization (ECM) approach ([15]) that shares the convergence properties of EM while being easier to implement. The idea is to replace the M-Step by a number of CM-steps in which variables are optimized alone while the others remain fixed. Thus in the M-step, we use the mesh deformation framework to first optimize for $\boldsymbol{\theta}^{t+1}$, then update σ^{t+1}.

4. Implementation

The design of Kineben consists of 3 framework and 2 applications components. The component model is pictured in 4, with dependencies to 3rd party libraries.

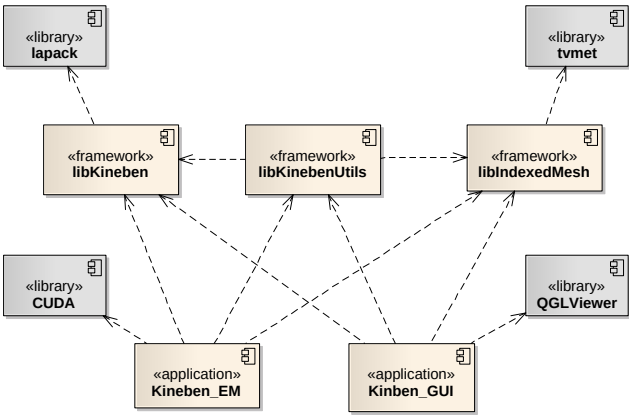


Figure 4: Component model of the software.

Component	Description
Framework	
libIndexedMesh	3D Mesh utility functions.
libKineben	Joint type definitions, kinematic computations, normal equations solver.
libKinebenUtils	Skeleton definition and skinning utilities, inverse kinematic solver. EnergyTerm interface and skeleton constraints implementation.
Application	
KinebenGUI	Graphical solver evaluation application based on 3D hard constraints energy term.
KinebenEM	Motion tracker based on 3D EM-based soft constraints energy term.

4.1. Kineben Framework

4.1.1 libKineben

To begin with, let us describe some important entities of the library.

`KBJoint` is a struct representing a joint in a kinematic chain. It has 2 members.

`m_type` stores the joint type: `KBJOINT_REVOLUTE`, `KBJOINT_BALL`, or `KBJOINT_TRANSLATE`.

`m_parentId` stores the index of the parent joint in its kinematic chain, or -1 if it is the root joint.

`KBJoint* tree` is an array of type `KBJoint` storing the skeleton structure. Each element has an index bigger than its member `m_parentId`.

Param objects store or compute reference pose informations.

`int* KBParamDim` stores the dimension of the reference pose description of the joint types.

`double* paramVec` stores the complete reference pose configuration, ordered as in `KBJoint* tree`.

`KBComputeParamDim` computes the size of `paramVec`.

State objects store or compute track informations.

`int* KBStateDim` stores the dimension of the track of the joint types.

`double* stateVec` stores the complete track, ordered as in `KBJoint* tree`.

`KBComputeStateDim` computes the size of `stateVec`.

Update objetcs store or compute track update informations.

`int* KBUpdateDim` stores the number of update parameters of the joint types.

`double* updateVec` stores all update values, ordered as in `KBJoint* tree`.

`KBComputeUpdateDim` computes the size of `updateVec`.

`double* Ti` is a 3×4 matrix representing the rigid transformation of KBJoint i.

`KBcomputeTis` computes all transformations in `KBJoint* tree`.

`double* TTi` is a 3×4 matrix representing the composite rigid transformation up to KBJoint i.

`KBcomputeTTis` computes all composite transformations in `KBJoint* tree`.

`double* DDi` is a 3×4 matrix representing the linear differential operator of KBJoint i.

`KBcomputeDDis` computes all linear differential operators in `KBJoint* tree`.

`KBsolve` The library solves the so-called normal equations of the Gauss-Newton method, Eq. (5) of section 3.2, in the following C routine:

```
int
KBsolve(int           numParams,
        const double* JTJ,
        const double* JTb,
        double        epsilon,
        double*       update);
```

`numParams` is the number of update parameters involved in the normal equations.

`JTJ` stores the $J^\top J$ matrix. Its size should be `numParams`$\times$`numParams`.

`JTb` stores the $J^\top b$ vector. Its size shoud be `numParams`.

`epsilon` stores the Marquardt parameter value.

`update` stores after function call completion the estimated update values. Its size should be `numParams`.

The user applies the estimated updates to the state vector with:

```
void
KBapplyUpdate( int             numJoints,
               const KBJoint*  tree,
               const double*   stateVec_old,
               const double*   updateVec,
               const double    updateScale,
               double*         stateVec_new );
```

The user includes this functionality with Kineben.h.

4.1.2 libKinebenUtils

This library build on the functionalities of libKineben and offers utility functions and classes to skeleton tracking applications. We describes here the main objects of this library.

IEnergyTerm is an abstract class specifying the interface of an energy term in the cost function, Eq. (2). It has the 2 following pure virtual functions:

```
virtual
void
addToJTJ_JTb( const double* sv,
              const double* Tis,
              const double* TTis,
              const double* DDis,
              const int     updateDim,
              const int*    updateDep,
              const int*    updateDep_bounds,
              double*       JTJ,
              double*       JTb)= 0;

virtual
double
computeEnergy( const double* sv,
               const double* Tis,
               const double* TTis ) = 0;
```

`addToJTJ_JTb` adds in the matrices $J^\top J$ and $J^\top b$ the contribution of the energy term.

`computeEnergy` returns the value of this energy term given a state vector and transformations `Tis` and `TTis`.

Solver is a Gauss-Newton solver. To solve a given inverse kinematic problem, the user calls the following member function:

```
int
solve( int                                maxIter,
       int                                maxSubDiv,
       const std::list<IEnergyTermPtr>&   eTerms,
       const double*                      currSV,
       double*                            SV);
```

`maxIter` is the maximum number of converging iterations.

`maxSubDiv` is the maximum number of subdivisions of the shift vector, if divergence occurs.

`eTerms` is a list of `boost::shared_ptr` to IEnergyTerm objects, thus representing the cost function to minimize.

`currSV` is the starting state vector, i.e. the starting point in the search space.

`SV` stores after function completion the minimizing state vector. The expected size of this vector and `currSV` is computed in the constructor of the solver.

The interfaces to `IEnergyTerm` and `Solver` are included with `Kineben_solver.h`.

EnergyTerm_SkelConstraints is a derived class of IEnergyTerm. It implements the energy term constraining the joint orientations and angles, Eq. (42) and (14). This interface is included with `EnergyTerm_SkelConstraints.h`.

RiggedMesh is a data structure storing the assignments of reference mesh vertices to joints. Its important members are:

`coords` stores the vertices of the reference mesh. These are given in the constructor, as a .off file.

`vj` stores the assignment of a vertex to a joint, i.e. `vj[i]` $\leftarrow joint($`coords[i]`$)$. These are given in the constructor, as a .vj file.

`jv` stores the same information as `vj`, but group vertices by increasing joint order and store the `coords` index of the vertex. The order is specified by the `KBJoint` vector given as argument to the constructor.

`jv_bounds` stores the joint group bounds of `jv`. It has $N_b + 1$ slots. Thus `jv[jv_bounds[0]]` stores the first vertex belonging to the first joint and `jv[jv_bounds[1]]` the last (not inclusive)

Skel.h defines functions to read a .skl file and generate a skeleton tree.

`KBParse` parses the .skl file into `KBNodes`.

`KBGenJoints` generates from `KBNodes` a `KBJoint* tree`, as well as a `paramVec` and a `stateVec`.

KBBVHWriter writes a BVH file. The file contains two main parts, `HIERARCHY`, storing the skeleton structure, and `MOTION` storing the track. See Appendix A.6 for the format specification of BVH motion files.

`KBBVHWriter::writeFrame` allows to store one frame of the track.

4.2. Kineben Applications

4.2.1 KinebenGUI

For the evaluation of the body model, we developed the application KinebenGUI. The GUI framework is inherited from the 3rd party library QGLviewer[2]. Figure 5 shows

[2] http://www.libqglviewer.com/

the typical use case. The application loads the body model, that is the skeleton pose and the reference mesh. The user moves the mouse where he wishes a vertex of the mesh to be defined as constrained and press 'c'. A red box appears to show the constraint. To select a constraint, the user holds shift and clicks on the right button over the red box. To move it, he holds ctrl and right mouse button while he moves the mouse. A solution is computed by pressing ctrl+s. Holding everything while moving the constraint, exhibits the realtime response of the solver (under these few constraints).

`EnergyTerm_3DConstraint` is a derived class from the base class `Kineben::IEnergyTerm`.

`addToJTJ_JTb` loops over all M constraints $y_m \leftrightarrow x_m$ and compute the following sum:

$$J^\top J_{(i,j)} = \sum_{m=1}^{M} \delta_{i,j}(\boldsymbol{\theta}, x_m) \left[\frac{\partial x_m(\boldsymbol{\theta})}{\partial \theta_i}\right]^\top \left[\frac{\partial x_m(\boldsymbol{\theta})}{\partial \theta_j}\right]$$

$$J^\top b_{(i)} = \sum_{m=1}^{M} \delta_i(\boldsymbol{\theta}, x_m) \left[\frac{\partial x_m(\boldsymbol{\theta})}{\partial \theta_i}\right]^\top (x_m - y_m)$$

$$\delta_{i,j}(\boldsymbol{\theta}, x_m) = \begin{cases} 1, \text{if } \theta_i \text{ and } \theta_j \text{ in the kinematic chain of } x_m \\ 0, \text{otherwise} \end{cases}$$

`computeEnergy` loops over all M constraints $y_m \leftrightarrow x_m$ and compute the cost function, Eq. (2).

The arguments to the executable are listed in the following table:

-off	A **reference mesh file** (.off) containing vertices and triangles of the shape model.
-skl	A **skeleton definition file** (.skl) describing the pose model. It specifies the joint hierarchy, reference positions and orientations. See Appendix A.4 for the format specification of skeleton definition files.
-cst	A **joint constraints file** (.cst) describing the orientation and angle constraints of the joints. See Appendix A.5 for the format specification of constraints definition files.
-vj	A **vertex/joint assignment file** (.vj) is listing the vertex/body part assignments. The line number is the index of the vertex in the mesh file (.off) and the joint name appearing on this line is the body part to which this vertex belongs.

4.2.2 KinebenEM

The application KinebenEM is a skeleton pose tracker working on 3D Mesh inputs. The output is a BVH file, wherein the skeleton initial pose and track are written.

`IEnergyTerm_EM` is derived abstract class from the base class `Kineben::IEnergyTerm`. It specifies one additional pure virtual function.

`Estep` is a pure virtual function, whos intended purpose is to compute the current model-data matching, see section 3.3.2. Its implementing classes store the posterior matrix as in figure 3. We call the entries of this $M \times K$ matrix *weights*, w_m^k, storing the weight attributed to the match of observed vertex m with the generating vertex of joint k. `Estep` stores the index of the generating vertex in $index(m,k)$, let this vertex be x_m^k.

```
virtual
void
EStep( const float      sigma,
       const float      normThresh,
       const float      EOutlier,
       const double*    TTis ) = 0;
```

`addToJTJ_JTb` loops over all M observed vertices y_m and compute the following sum:

$$J^\top J_{(i,j)} = \sum_{m=1}^{M} \sum_{k=1}^{N_b+1} \delta_{i,j}(\boldsymbol{\theta}, x_m^k) w_m^k \left[\frac{\partial x_m^k(\boldsymbol{\theta})}{\partial \theta_i}\right]^\top \left[\frac{\partial x_m^k(\boldsymbol{\theta})}{\partial \theta_j}\right]$$

$$J^\top b_{(i)} = \sum_{m=1}^{M} \sum_{k=1}^{N_b+1} \delta_i(\boldsymbol{\theta}, x_m^k) w_m^k \left[\frac{\partial x_m^k(\boldsymbol{\theta})}{\partial \theta_i}\right]^\top (x_m^k - y_m)$$

$$\delta_{i,j}(\boldsymbol{\theta}, x_m^k) = \begin{cases} 1, \text{if } \theta_i \text{ and } \theta_j \text{ in the kinematic chain of } x_m^k \\ 0, \text{otherwise} \end{cases}$$

`computeEnergy` loops over all M observed vertices y_m and compute the value of the cost function:.

$$E = \sum_{m=1}^{M} \sum_{k=1}^{N_b+1} w_m^k (x_m^k - y_m)^\top (x_m^k - y_m)$$

`EnergyTerm_EM` implements the pure virtual function `Estep` of the abstract class `IEnergyTerm_EM`.

`EStep` runs the Estep of our Expectation-Maximisation algorithm, section 3.3.2, on graphic cards supporting the CUDA progamming language and is implemented in `CUDA_KERNEL_ENN_EStep.cu`.

(a) Load model. (b) Define constraints. (c) Move constraint. (d) Solve.

Figure 5: Model and Inverse Kinematic Solver Testing with KinebenGUI.

The arguments to the executable are listed in the following table:

The same 4 arguments as in KinebenGUI	
-cloudBasename	A path to numbered .off files (Note: put a printf numbering format for your file numbering, eg. %03d).
-outBVH	A file path for the output BVH file.
-F	First frame number.
-L	Last frame number.
-NThresh	Normal correspondence threshold.
-sigma0	Initial sigma value for the gaussian components of the mixture of gaussians in the Estep.
-Eoutlier	propability measure of outliers.

Output A **BVH Motion file**.

5. Results

We ran our tracking KinebenEM application on video sequences available at CSAIL[3]. This dataset offered us the possibility to test our tracker directly on 3D point clouds without having to implement the 3D reconstruction preprocessing. For each 5 seconds long video, we get then around 175 data clouds. The tracker performs at a frame rate of 1 Hz on a CUDA-enabled laptop PC; Intel®Core™i7-620M processor and a NVIDIA NVS 3100 graphic card.

We achieved the best results with Marquardt parameter set to 0.1 (still hardcoded) and command line arguments sigma0 set to 4.0, NThresh to 0.6 and Eoutlier to 0.1.

The *crane* sequence exhibits a character performing a crane walk. This is a simple case, where all body parts are moved, but very few occlusion phenomena occur (if the number of cameras in the multi-camera setup is high enough). By visual inspection we could confirm that the tracker delivered a successful track in this case. Figure 6 shows three frames from the skeleton track overlayed on images from one of the cameras.

In the *handstand* sequence the character flips 180° along the Z-axis to stand on his hands. The challenge of this sequence consists in tracking this flip, where all normals of the observed mesh stand in the opposite direction relative to the reference mesh. In this case also, we got a successful track.

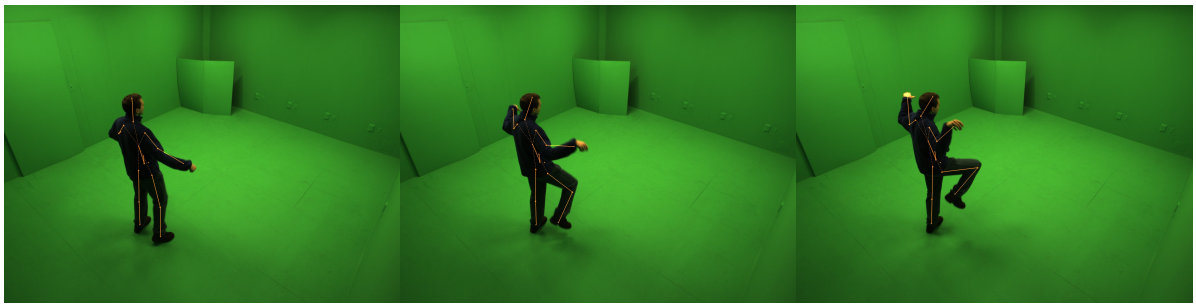
The *jumping* sequence challenges the tracker, in that it exhibits fast movements in all directions. Since the character brings his arms often near the torso, this sequence contains occlusions that are hard for the tracker to disambiguate. Near the frame 138, for instance, the reconstruction is noisy, due to the visual hull reconstruction approach, and vertices are assigned to wrong body parts. The result is that arms stick to the torso, until the character brings them back in a relative unambiguiate neighborhood.

6. Conclusion

We could achieve many of the goals we set. We

- ...gained an impressive knowledge of the literature on motion capture;
- ...derived a taxonomy for the multiple involved tasks;
- ...extended the patch-based method to body parts;
- ...gained experience in the exact mathematical derivation of self-designed solutions;
- ...got deep insights and experience in the programming languages C, C++, CUDA and Python;
- ...designed a CMake-based multi-platform reusable framework;
- ...could convince us that the method was working through applications build on the framework;
- ...presented the work to the chair;

[3] http://people.csail.mit.edu/drdaniel/mesh_animation/index.html



(a) (b) (c)

Figure 6: Crane Sequence

- ... wrapped up everything in this report;

As nature is complex and ever evolving, nothing is perfect and complete. We mention some problems unsolved and areas for enhancement:

- Comparative evaluation of the method with related solutions of other laboratories working on the subject.
- A global refinement of the locally estimated pose; maybe as in Gall *et al*. [9].
- A vertex/body part assignment prior that is not constant, since the proportion of vertices per body part is neither constant over time nor over all parts; see section 3.3.1.
- Investigate which CPU computations are portable to GPU and optimize time.
- Write code documentation.
- Refactor interfaces such that other solver algorithms are pluggable.
- Refactor interfaces such that solver algorithms working on 2D data are pluggable.
- Work out a solution to track people with loose clothes, multiple and/or unknown objects.
- Implement an automatic model initialisation solution.
- Couple to a multi-camera studio, to perform online tracking.

References

[1] I. Baran and J. Popović. Automatic rigging and animation of 3d characters. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[2] C. Bregler, J. Malik, and K. Pullen. Twist based acquisition and tracking of animal and human kinematics. *Int. J. Comput. Vision*, 56:179–194, February 2004.

[3] C. Cagniart, E. Boyer, and S. Ilic. Probabilistic deformable surface tracking from multiple videos. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 326–339, Berlin, Heidelberg, 2010. Springer-Verlag.

[4] J. Carranza, C. Theobalt, M. A. Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. *ACM Trans. Graph.*, 22:569–577, July 2003.

[5] K.-M. G. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette across time part ii: Applications to human modeling and markerless motion tracking. *Int. J. Comput. Vision*, 63:225–245, July 2005.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society, series B*, 39(1):1–38, 1977.

[7] J. Deutscher, A. Blake, and I. Reid. Articulated body motion capture by annealed particle filtering. *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, 2:126–133 vol.2, Aug. 2002.

[8] D. A. Forsyth, O. Arikan, L. Ikemoto, J. O'Brien, and D. Ramanan. Computational studies of human motion: part 1, tracking and motion synthesis. *Foundations and Trends in Computer Graphics and Vision*, 1:77–254, July 2005.

[9] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition : CVPR 2009*, pages 1746–1753, Miami, USA, 2009. IEEE.

[10] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding*, 73:82–98, 1999.

[11] R. Horaud, M. Niskanen, G. Dewaele, and E. Boyer. Human motion tracking by registering an articulated surface to 3d points and normals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:158–163, 2009.

[12] X. Ji and H. Liu. Advances in view-invariant human motion analysis: a review. *Trans. Sys. Man Cyber Part C*, 40:13–24, January 2010.

[13] I. Kakadiaris and D. Metaxas. Model-based estimation of 3d human motion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22:1453–1459, December 2000.

[14] R. Kehl, M. Bray, and L. Van Gool. Full body tracking from multiple views using stochastic sampling. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 129–136, Washington, DC, USA, 2005. IEEE Computer Society.

[15] x. Meng and D. B. Rubin. Maximum likelihood estimation via the ECM algorithm: A general framework. *Biometrika*, 80(2):267–278, June 1993.

[16] I. Mikić, M. Trivedi, E. Hunter, and P. Cosman. Human body model acquisition and tracking using voxel data. *Int. J. Comput. Vision*, 53:199–223, July 2003.

[17] T. B. Moeslund and E. Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding*, 81:231–268, March 2001.

[18] T. B. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104:90–126, November 2006.

[19] R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994.

[20] R. Plänkers and P. Fua. Tracking and modeling people in video sequences. *Comput. Vis. Image Underst.*, 81:285–302, March 2001.

[21] G. Pons-Moll and B. Rosenhahn. Ball joints for marker-less human motion capture. In *IEEE Workshop on Applications of Computer Vision (WACV)*, volume 0, dez 2009.

[22] R. Poppe. Vision-based human motion analysis: An overview. *Computer Vision and Image Understanding*, 108:4–18, October 2007.

[23] M. M. T. Shinko Y. Cheng. Articulated human body pose inference from voxel data using a kinematically constrained gaussian mixture model. *In CVPR EHuM2: 2-nd Workshop on Evaluation of Articulated Human Motion and Pose Estimation*, 2007.

[24] L. Sigal, M. Isard, B. H. Sigelman, and M. J. Black. Attractive People: Assembling Loose-Limbed Models using Non-parametric Belief Propagation. In *Advances in Neural Information Processing Systems*, 2003.

[25] C. Tran and M. Trivedi. Human body modelling and tracking using volumetric representation: Selected recent studies and possibilities for extensions. In *ICDSC08*, pages 1–9, 2008.

[26] D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 97:1–97:9, New York, NY, USA, 2008. ACM.

[27] L. Wang, W. Hu, and T. Tan. Recent developments in human motion analysis. *Pattern Recognition*, 36:585–601, 2003.

A. Appendix

A.1. Twist Representation

This section is for the most part a transcript from Mikic *et al*. [16], completed and corrected with a comparison to Murray [19].

Let us consider a rotation of a rigid object about a fixed axis. Let the unit vector along the axis of rotation be $\omega \in \mathbb{R}^3$ and $q \in \mathbb{R}^3$ be a point on the axis. Assuming that the object rotates with unit velocity, the velocity vector of a point $x(t)$ on the object is:

$$\dot{\boldsymbol{x}}(t) = \omega \times (\boldsymbol{x}(t) - \boldsymbol{q}) \tag{30}$$

This can be rewritten in homogeneous coordinates as:

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{\omega} & -\omega \times \boldsymbol{q} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = \hat{\xi} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \tag{31}$$

or, in a compact form,

$$\dot{\bar{\boldsymbol{x}}} = \hat{\xi}\bar{\boldsymbol{x}} \tag{32}$$

where $\bar{\boldsymbol{x}} = [\, \boldsymbol{x}\ 1\,]^\top$ is a homogeneous coordinate of the point $\boldsymbol{x}$, and $\omega \times \mathbf{z} = \hat{\omega}\mathbf{z}, \forall \mathbf{z} \in \mathbb{R}^{\mathbf{3}}$, i.e.,

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{33}$$

and

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & -\omega \times \mathbf{q} \\ \mathbf{0}^\top & 0 \end{bmatrix} = \begin{bmatrix} \hat{\omega} & \mathbf{v} \\ \mathbf{0}^\top & 0 \end{bmatrix} \tag{34}$$

is defined as a twist associated with the rotation about the axis defined by ω and $\mathbf{q}$. $\xi = (\omega_1, \omega_2, \omega_3, v_1, v_2, v_3)$ are called the twist coordinates of this rotation.

The solution to the differential Equation (30) is:

$$\bar{\boldsymbol{x}}(t) = e^{\hat{\xi}t}\bar{\boldsymbol{x}}(0) \tag{35}$$

$e^{\hat{\xi}t}$ is the mapping (the exponential map associated with the twist $\hat{\xi}$) from the initial location of a point $\boldsymbol{x}$ to its new location after rotating t radians about the axis defined by ω and $\mathbf{q}$. It can be shown that

$$\boldsymbol{T} = e^{\hat{\xi}\theta} = \begin{bmatrix} e^{\hat{\omega}\theta} & (\boldsymbol{I}_3 - e^{\hat{\omega}\theta})(\omega \times \boldsymbol{v}) + \omega\omega^\top \boldsymbol{v}\theta \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{36}$$

where

$$e^{\hat{\omega}\theta} = \boldsymbol{I}_3 + \frac{\hat{\omega}}{\|\omega\|}\sin(\|\omega\|\theta) + \frac{\hat{\omega}^2}{\|\omega\|^2}(1 - \cos(\|\omega\|\theta)) \tag{37}$$

is a rotation matrix associated with the rotation of θ radians about an axis ω. Eq. (37) is known as Rodriguez Formula.

$\mathbf{T}$ is a rigid transformation, i.e. $\mathbf{T} \in \mathbf{SE}(\mathbf{3})$. The term $\omega\omega^\top \mathbf{v}\boldsymbol{\theta}$ in the matrix block $\mathbf{T_{12}}$ is $\mathbf{0}$ for purely rotational twists.

A.2. Derivation of the joint type based linear differential operators

We saw in section 3.1 that the coordinates of a point $\boldsymbol{x}$ attached to a segment inside an open kinematic are given by

$$\bar{\boldsymbol{x}}(\boldsymbol{\theta}) = g(\boldsymbol{\theta})\bar{\boldsymbol{x}}(\mathbf{0}) \tag{38}$$

differenting yielded

$$J_{\boldsymbol{x}_i}(\boldsymbol{\theta}) = \left[\frac{\partial g}{\partial \theta_1}g^{-1}, \ldots, \frac{\partial g}{\partial \theta_K}g^{-1}\right]\bar{\boldsymbol{x}}(\boldsymbol{\theta}) \tag{39}$$

a differential $4 \times 4 \times K$ tensor multiplied to $\bar{\boldsymbol{x}}(\boldsymbol{\theta})$. The following gives the full derivation of the 4×4 components of this tensor.

Revolute If $\boldsymbol{\theta}_i$ is the angle parameter of a revolute joint:

$$\begin{aligned}
\frac{\partial g}{\partial \boldsymbol{\theta}_i}g^{-1}\bar{\boldsymbol{x}}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \boldsymbol{\theta}_i}(T_1 \ldots T_i \ldots T_K)(T_1 \ldots T_i \ldots T_K)^{-1}\bar{\boldsymbol{x}}(\boldsymbol{\theta}) \\
&= \frac{\partial}{\partial \boldsymbol{\theta}_i}(T_1 \ldots T_i \ldots T_K)T_K^{-1} \ldots T_i^{-1} \ldots T_1^{-1}\bar{\boldsymbol{x}}(\boldsymbol{\theta}) \\
&= \frac{\partial}{\partial \boldsymbol{\theta}_i}(T_1 \ldots e^{\hat{\xi}_i\boldsymbol{\theta}_i} \ldots T_K)T_K^{-1} \ldots e^{-\hat{\xi}_i\boldsymbol{\theta}_i} \ldots T_1^{-1}\bar{\boldsymbol{x}}(\boldsymbol{\theta}) \\
&= T_1 \ldots \hat{\xi}_i e^{\hat{\xi}_i\boldsymbol{\theta}_i} \ldots T_K T_K^{-1} \ldots e^{-\hat{\xi}_i\boldsymbol{\theta}_i} \ldots T_1^{-1}\bar{\boldsymbol{x}}(\boldsymbol{\theta})
\end{aligned}$$

$T_k T_k^{-1}$ factors vanish for all $k \geq i$, to give

$$= \left[T_1 \ldots T_{i-1}\widehat{\xi}_i T_{i-1}^{-1} \ldots T_1^{-1}\right]\bar{\boldsymbol{x}}(\boldsymbol{\theta})$$

Ball A ball joint represents a rotation about a scaled unknown axis $\theta\omega$. In section 3.2.2 we found that we can approximate the rigid transformation induced by a small axis update $d\omega$ by

$$T_i(d\omega) \simeq \begin{bmatrix} R_i & (I - R_i)q \\ \mathbf{0}^\top & 1 \end{bmatrix}\begin{bmatrix} I + [d\omega]_\times & -[d\omega]_\times q \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{40}$$

We look at the position of $\boldsymbol{x}$ after this update $d\omega$

$$\begin{aligned}
\bar{\boldsymbol{x}}(\boldsymbol{\theta} + d\omega) &= g(\boldsymbol{\theta} + d\omega)\bar{\boldsymbol{x}}(\mathbf{0}) \\
&= T_1 \ldots T_{i-1}T_i(d\omega)T_{i+1} \ldots T_n\bar{\boldsymbol{x}}(\mathbf{0}) \\
&= T_1 \ldots T_{i-1}\begin{bmatrix} R_i & (I + R_i)q \\ \mathbf{0}^\top & 1 \end{bmatrix}\begin{bmatrix} I + [d\omega]_\times & -[d\omega]_\times q \\ \mathbf{0}^\top & 1 \end{bmatrix}T_{i+1} \ldots T_n\bar{\boldsymbol{x}}(\mathbf{0}) \\
&= T_1 \ldots T_i\begin{bmatrix} I + [d\omega]_\times & -[d\omega]_\times q \\ \mathbf{0}^\top & 1 \end{bmatrix}T_{i+1} \ldots T_n\bar{\boldsymbol{x}}(\mathbf{0}) \\
&= T_1 \ldots T_i\left[\begin{bmatrix} I & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} + \begin{bmatrix} [d\omega]_\times & -[d\omega]_\times q \\ \mathbf{0}^\top & 0 \end{bmatrix}\right]T_{i+1} \ldots T_n\bar{\boldsymbol{x}}(\mathbf{0}) \\
&= T_1 \ldots T_i\begin{bmatrix} I & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}T_{i+1} \ldots T_n\bar{\boldsymbol{x}}(\mathbf{0}) + T_1 \ldots T_i\begin{bmatrix} [d\omega]_\times & -[d\omega]_\times q \\ \mathbf{0}^\top & 0 \end{bmatrix}T_{i+1} \ldots T_n\bar{\boldsymbol{x}}(\mathbf{0}) \\
&= \bar{\boldsymbol{x}}(\boldsymbol{\theta}) + T_1 \ldots T_i\begin{bmatrix} [d\omega]_\times & -[d\omega]_\times q \\ \mathbf{0}^\top & 0 \end{bmatrix}T_{i+1} \ldots T_n\bar{\boldsymbol{x}}(\mathbf{0})
\end{aligned}$$

bringing $\bar{\boldsymbol{x}}(\boldsymbol{\theta})$ on the left side

$$\begin{aligned}\bar{\boldsymbol{x}}(\boldsymbol{\theta}+d\omega)-\bar{\boldsymbol{x}}(\boldsymbol{\theta}) &= T_1\dots T_i\begin{bmatrix}[d\omega]_\times & -[d\omega]_\times q\\ \mathbf{0}^\top & 0\end{bmatrix}T_{i+1}\dots T_n\bar{\boldsymbol{x}}(\mathbf{0})\\ &= \begin{bmatrix}R_{1\dots i} & t_{1\dots i}\\ \mathbf{0}^\top & 1\end{bmatrix}\begin{bmatrix}[d\omega]_\times & -[d\omega]_\times q\\ \mathbf{0}^\top & 0\end{bmatrix}\begin{bmatrix}R_{i+1\dots n} & t_{i+1\dots n}\\ \mathbf{0}^\top & 1\end{bmatrix}\bar{\boldsymbol{x}}(\mathbf{0})\end{aligned}$$

where $R_{1\dots i}$ is the rotation block of the composite transformation from joint 1 up to i and $t_{1\dots i}$ its translation,

$$\begin{aligned}&= \begin{bmatrix}R_{1\dots i}[d\omega]_\times & -R_{1\dots i}[d\omega]_\times q\\ \mathbf{0}^\top & 0\end{bmatrix}\begin{bmatrix}R_{i+1\dots n} & t_{i+1\dots n}\\ \mathbf{0}^\top & 1\end{bmatrix}\bar{\boldsymbol{x}}(\mathbf{0})\\ &= \begin{bmatrix}R_{1\dots i}[d\omega]_\times R_{i+1\dots n} & R_{1\dots i}[d\omega]_\times t_{i+1\dots n}-R_{1\dots i}[d\omega]_\times q\\ \mathbf{0}^\top & 1\end{bmatrix}\bar{\boldsymbol{x}}(\mathbf{0})\end{aligned}$$

if we truncate the homogenous coordinate, we get

$$\boldsymbol{x}(\boldsymbol{\theta}+d\omega)-\boldsymbol{x}(\boldsymbol{\theta}) = R_{1\dots i}[d\omega]_\times R_{i+1\dots n}\boldsymbol{x}(\mathbf{0})+R_{1\dots i}[d\omega]_\times t_{i+1\dots n}-R_{1\dots i}[d\omega]_\times q$$

we apply $T\cdot a\wedge b = T\cdot a\wedge T\cdot b$,

$$= [R_{1\dots i}d\omega]_\times R_{1\dots i}R_{i+1\dots n}\boldsymbol{x}(\mathbf{0})+[R_{1\dots i}d\omega]_\times R_{1\dots i}t_{i+1\dots n}-[R_{1\dots i}d\omega]_\times R_{1\dots i}q$$

by distributivity of the factor $[R_{1\dots i}d\omega]_\times$, we get

$$\begin{aligned}&= [R_{1\dots i}d\omega]_\times\left(R_{1\dots i}R_{i+1\dots n}\boldsymbol{x}(\mathbf{0})+R_{1\dots i}t_{i+1\dots n}-R_{1\dots i}q\right)\\ &= [R_{1\dots i}d\omega]_\times\left(R_{1\dots i}(R_{i+1\dots n}\boldsymbol{x}(\mathbf{0})+t_{i+1\dots n})-R_{1\dots i}q\right)\end{aligned}$$

we replace the term $(R_{i+1\dots n}\boldsymbol{x}(\mathbf{0})+t_{i+1\dots n})$ with its equivalent $\mathrm{T}^{-1}_{1\dots i}\boldsymbol{x}(\boldsymbol{\theta})$,

$$= [R_{1\dots i}d\omega]_\times\left(R_{1\dots i}(R^\top_{1\dots i}\boldsymbol{x}(\boldsymbol{\theta})-R^\top_{1\dots i}t_{1\dots i})-R_{1\dots i}q\right)$$

since the inverse of a rotation is its transpose, we get

$$= [R_{1\dots i}d\omega]_\times\left(\boldsymbol{x}(\boldsymbol{\theta})-(R_{1\dots i}q+t_{1\dots i})\right)$$

we apply the rule $a\wedge b = -b\wedge a$,

$$= -[\boldsymbol{x}(\boldsymbol{\theta})-(R_{1\dots i}q+t_{1\dots i})]_\times R_{1\dots i}d\omega$$

and get finally,

$$\frac{\boldsymbol{x}(\boldsymbol{\theta}+d\omega)-\boldsymbol{x}(\boldsymbol{\theta})}{d\omega} = -[\boldsymbol{x}(\boldsymbol{\theta})-(R_{1\dots i}q+t_{1\dots i})]_\times R_{1\dots i}$$

With $d\omega$ small enough, we approximate the rate of change in position of $\bar{\boldsymbol{x}}(\boldsymbol{\theta})$w.r.t. the parameters of a ball joint with:

$$\frac{\partial\boldsymbol{x}(\boldsymbol{\theta})}{\partial d\omega}\simeq\frac{\boldsymbol{x}(\boldsymbol{\theta}+d\omega)-\boldsymbol{x}(\boldsymbol{\theta})}{d\omega} = -[\boldsymbol{x}(\boldsymbol{\theta})-(R_{1\dots i}q+t_{1\dots i})]_\times R_{1\dots i} \tag{41}$$

If we develop the expression on the rigth side further, we can even break the linear differential operator with respect to the parameters of the rotation axis into 3 linear differential operators, that is $\frac{\partial\boldsymbol{x}(\boldsymbol{\theta})}{\partial\theta_1}$, $\frac{\partial\boldsymbol{x}(\boldsymbol{\theta})}{\partial\theta_2}$ and $\frac{\partial\boldsymbol{x}(\boldsymbol{\theta})}{\partial\theta_3}$, if we assume $d\omega=(\theta_1,\theta_2,\theta_3)^\top$.

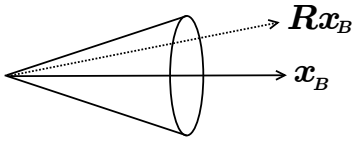


Figure 7: $R\boldsymbol{x}$ is constrained by the dot threshold cone.

The operator is:

$$\frac{\partial \boldsymbol{x}(\boldsymbol{\theta})}{\partial d\omega} = -\left[\boldsymbol{x}(\boldsymbol{\theta})) - (R_{1\ldots i}q + t_{1\ldots i})\right]_\times R_{1\ldots i}$$

if we focus on the columns and let R^i be the ith column of R, we get

$$\begin{aligned}
&= -\left[\begin{array}{c|c|c} (\boldsymbol{x}(\boldsymbol{\theta}) - (R_{1\ldots i}q + t_{1\ldots i})) \times R^1_{1\ldots i} & (\boldsymbol{x}(\boldsymbol{\theta}) - (R_{1\ldots i}q + t_{1\ldots i})) \times R^2_{1\ldots i} & (\boldsymbol{x}(\boldsymbol{\theta}) - (R_{1\ldots i}q + t_{1\ldots i})) \times R^3_{1\ldots i} \end{array}\right] \\
&= \left[\begin{array}{c|c|c} R^1_{1\ldots i} \times (\boldsymbol{x}(\boldsymbol{\theta}) - (R_{1\ldots i}q + t_{1\ldots i})) & R^2_{1\ldots i} \times (\boldsymbol{x}(\boldsymbol{\theta}) - (R_{1\ldots i}q + t_{1\ldots i})) & R^3_{1\ldots i} \times (\boldsymbol{x}(\boldsymbol{\theta}) - (R_{1\ldots i}q + t_{1\ldots i})) \end{array}\right] \\
\frac{\partial \bar{\boldsymbol{x}}(\boldsymbol{\theta})}{\partial d\omega} &= \left[\begin{array}{c|c|c} \frac{\partial \bar{\boldsymbol{x}}(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial \bar{\boldsymbol{x}}(\boldsymbol{\theta})}{\partial \theta_2} & \frac{\partial \bar{\boldsymbol{x}}(\boldsymbol{\theta})}{\partial \theta_3} \end{array}\right]
\end{aligned}$$

where

$$\frac{\partial \bar{\boldsymbol{x}}(\boldsymbol{\theta})}{\partial \theta_i} = \begin{bmatrix} [R^i_{1\ldots i}]_x & -[R^i_{1\ldots i}]_x(R_{1\ldots i}q + t_{1\ldots i}) \\ \mathbf{0}^\top & 0 \end{bmatrix} \bar{\boldsymbol{x}}(\boldsymbol{\theta})$$

A.3. Energy Term constraining ball joints

The reference pose prescribes the limiting vector x_B pointing, for instance, downwards for a hip. R is the current rotation of the joint. The solver detects that $x_B \cdot Rx_B < T$, for T a given threshold. Figure 7 conveys this simple idea.

The regularization term is then the following expression

$$r(\omega) = \begin{cases} x_B^\top Rx_B - t & \text{if } x_B^\top Rx_B < t \\ 0 & \text{otherwise} \end{cases} \tag{42}$$

For the $J^\top J$ and $J^\top r$, we have to do the derivative of $x_B^\top Rx_B - t$ w.r.t. the update vector $d\omega$.

We procede by finite difference:

$$r(\omega + d\omega) - r(\omega) = x_B^\top Re^{[dw]_x}x_B - t - x_B^\top Rx_B + t$$

if we take $e^{[dw]_x} \simeq (I + [dw]_x)$, and take canceling thresholds out, we get

$$= x_B^\top[R(I + [dw]_x)x_B] - x_B^\top Rx_B$$

reordering terms, the right hand side becomes

$$\begin{aligned} &= x_B^\top R(dw \wedge x_B) \\ &= (dw \wedge x_B)^\top R^\top x_B \end{aligned}$$

then the rule $(a \wedge b) \cdot c = -(c \wedge b) \cdot a$,

$$= -(R^\top x_B \wedge x_B)^\top dw \qquad\qquad = (x_B \wedge R^\top x_B)^\top \cdot dw$$

thus

$$\frac{\partial r(\omega)}{\partial d\omega} \simeq \frac{r(\omega + d\omega) - r(\omega)}{d\omega} = (x_B \wedge R^\top x_B)^\top.$$

A.4. Skeleton Definition File

⟨SKELDEFFILE⟩→ **KBSKEL** ⟨INT⟩ ⟨GRAPH⟩ ⟨COORDINATES⟩

⟨GRAPH⟩→ (⟨IDENTIFIER⟩ ⟨TYPE⟩ (⟨IDENTIFIER⟩ | **NULL**))+

⟨IDENTIFIER⟩→ [0-9A-Za-z][.]+

⟨COORDINATES⟩→ (⟨TRANSLATE_COORDINATES⟩ | ⟨REVOLUTE_COORDINATES⟩ | ⟨BALL_COORDINATES⟩)+

⟨TRANSLATE_COORDINATES⟩→ ⟨IDENTIFIER⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩

⟨REVOLUTE_COORDINATES⟩→ ⟨IDENTIFIER⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩

⟨BALL_COORDINATES⟩→ ⟨IDENTIFIER⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩

⟨FLOAT⟩→ [–]?[0-9][0-9]+[[̇0-9]+]?[[eE][0-9][0-9]+]

⟨INT⟩→ [–]?[0-9][0-9]

A.5. Constraints Definition File

⟨CSTFILE⟩→ **KBCST** ⟨INT⟩ ⟨CONSTRAINTS⟩

⟨CONSTRAINTS⟩→ (⟨TRANSLATE_CONSTRAINTS⟩ | ⟨REVOLUTE_CONSTRAINTS⟩ | ⟨BALL_CONSTRAINTS⟩)+

⟨TRANSLATE_CONSTRAINTS⟩→ ⟨IDENTIFIER⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩

⟨REVOLUTE_CONSTRAINTS⟩→ ⟨IDENTIFIER⟩ ⟨MIN_ANGLE⟩ ⟨MAX_ANGLE⟩

⟨BALL_CONSTRAINTS⟩→ ⟨IDENTIFIER⟩ ⟨HEADING_AXIS⟩ ⟨MAX_ANGLE⟩ ⟨BANK_AXIS⟩ ⟨MAX_ANGLE⟩

⟨MIN_ANGLE⟩→ ⟨FLOAT⟩

⟨MAX_ANGLE⟩→ ⟨FLOAT⟩

⟨HEADING_AXIS⟩→ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩

⟨BANK_AXIS⟩→ ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩

⟨FLOAT⟩→ [–]?[0-9][0-9]+[[̇0-9]+]?[[eE][0-9][0-9]+]

⟨INT⟩→ [–]?[0-9][0-9]

A.6. BVH Format

⟨BVHFILE⟩→ ⟨HIERARCHY⟩ ⟨MOTION⟩

⟨HIERARCHY⟩→ **HIERARCHY** ⟨ROOT_JOINT_DECL⟩

⟨ROOT_JOINT_DECL⟩→ **ROOT** ⟨IDENTIFIER⟩ ⟨JOINT_BODY⟩

⟨IDENTIFIER⟩→ [0-9A-Za-z][.]+

⟨JOINT_BODY⟩→ { **OFFSET** ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ **CHANNELS** ⟨INT⟩ ⟨CHANNEL⟩+ (⟨JOINT_DECL⟩ | ⟨END_SITE⟩) }

⟨JOINT_DECL⟩→ **JOINT** ⟨IDENTIFIER⟩ ⟨JOINT_BODY⟩

⟨END_SITE⟩→ **End Site** { **OFFSET** ⟨FLOAT⟩ ⟨FLOAT⟩ ⟨FLOAT⟩ }

⟨CHANNEL⟩→ **Xposition** | **Yposition** | **Zposition** | **Xrotation** | **Yrotation** | **Zrotation**

⟨MOTION⟩→ **MOTION Frames :** ⟨INT⟩ **Frame Time :** ⟨FLOAT⟩ ⟨FRAME⟩*

⟨FRAME⟩→ ⟨FLOAT⟩+

⟨FLOAT⟩→ [–]?[0-9][0-9]+[[̇0-9]+]?[[eE][0-9][0-9]+]

⟨INT⟩→ [–]?[0-9][0-9]

A.7. Skinning with Blender

Setup

1. Load reference mesh (.off). Doing this **deselect** all options.
2. Load or construct skeleton. Make sure that mesh and skeleton share the same origin.
3. Run the script "addJointTypeToBonePanel.py" to add the joint type selection to the bone properties panel.

Skinning Envelopes Setup Rigging Envelopes. Modus Edit; go to object Armarture, select Display Envelope For each bone:

1. select bone
2. make sure envelope is big enough to enclose all vertices belonging to this bone. **If one vertex is missing, it won't appear in the output vj.txt file and it becomes inconsistent with the ref mesh. The script should fail, because a vertex has no vertex group.**

3. if a bone should have no vertex, like the root bone, **deselect** the deform options (this bone does not deform the mesh).
4. select the bone's type in the bone properties panel.

Skinning Once everything is set up, you can compute the skinning

1. select the mesh.
2. hold shift and select the armature.
3. press Ctrl+P (for setparent)
4. in the context menu choose set parent with envelopes.
5. this creates vertex groups in the armature.

Output skinning files

1. Output .vj file by running export_groups.py.
2. Output the .skl file with export_skel_file.py.