Matlab Toolbox for Motion Estimation

Emanuel Todorov

Version 1.0, 2/12/07

This Matlab Toolbox is an implementation of the motion estimation algorithm developed in:

> Todorov, E. (2007) Probabilistic inference of multi-joint movements, skeletal parameters and marker attachments from diverse motion capture data. *IEEE Transactions on Biomedical Engineering*, vol X, pp XX

General familiarity with the paper is needed to fully understand the documentation provided here. The full text of the paper is included as a PDF file in the software distribution. The abstract is:

> **We describe a comprehensive solution to the problem of reconstructing the multi-joint movement trajectories of the human body from diverse motion capture data. The problem is formulated in a probabilistic framework so as to handle multiple and unavoidable sources of uncertainty: sensor noise, soft tissue deformation and marker slip, inaccurate marker placement and limb measurement, and missing data due to occlusions. All unknown quantities are treated as state variables even though some of them are constant. In this way state estimation and system identification can be performed simultaneously, obtaining not only the most likely values but also the confidence intervals of the joint angles, skeletal parameters, and marker positions and orientations relative to the limb segments. The inference method is a Gauss-Newton generalization of the extended Kalman filter. It is adapted to the kinematic domain by expressing spatial rotations via quaternions and computing the sensor residuals and their Jacobians analytically. The ultimate goal of this project is to provide a reliable data analysis tool used in practice. The software implementation is available online.**

The software distribution `estimation_v1.zip` contains the following 11 files:

`manual.pdf` – this user's manual
`paper.pdf` – the above-referenced journal paper

`residual.m` – computation of sensor residuals and their Jacobians
`estimate.m` – Gauss-Newton estimation algorithm
`estimate_ekf.m` – a faster but possibly less accurate estimation algorithm (EKF)
`prepare.m` – automatic construction of the data structures needed to run the algorithm
`check.m` – numerical test for observability

`example1.m, example2.m, example3.m` – examples of running the algorithm
`hand_data.mat` – motion capture dataset of hand movements used in `example3.m`

Overview

The Toolbox performs movement estimation and self-calibration from motion capture data. The current version handles 3D position sensors, 3D orientation sensors, and virtual goniometers used to specify calibration postures. The estimated time-varying parameters are the global position and orientation of the body as well as all the joint angles. In addition, the Toolbox can estimate constant parameters such as limb sizes, axes of rotation, marker positions and orientations relative to the underlying segments. The latter computation corresponds to self-calibration and is performed simultaneously with the estimation of the time-varying parameters. Both the time-varying and the constant parameters are treated as elements of a state vector (x). All the sensor measurements are elements of a measurement vector (y).

The algorithm uses probabilistic inference. It computes not only the most likely values of the state variables but also their confidence intervals (or standard errors). This is done by fitting the posterior distribution of the state vector given all past measurements with a Gaussian, and propagating the mean and covariance of that Gaussian through time, via linearization of the measurement model. The algorithm also uses a dynamics model to specify a movement prior; currently this is a simple Brownian motion which captures movement continuity.

Using the Toolbox involves three steps:

1. Constructing a "generative model" which describes how the measurements depend on the state and how the state evolves over time. This includes a kinematic model specifying how the ideal sensor measurements depend on the state vector, a covariance matrix V characterizing the (Gaussian) distribution of sensor residuals, and a covariance matrix R characterizing the distribution of state changes in one time step (the latter distribution is assumed zero-mean).

2. Collecting motion-capture data compatible with the model created in step 1. Alternatively one can generate synthetic data using the Toolbox. The latter approach is illustrated in examples 1, 2 and is useful in quantifying the performance of the algorithm.

3. Applying the estimation algorithm to the (real or synthetic) data. Here one needs to specify the mean (x0) and covariance (S0) of the initial state, the kinematic model along with the matrices R and V, and certain data structures (map, info) which specify the types and addresses of the various model parameters in the state and measurement vectors.

The kinematic model is described in a compact format as an Nx5 matrix of integers where N is the number of segments (see next section). In the examples below this matrix is called **`segment`**. The numerical test for observability (see paper) is performed by calling **`check(segment)`**. The necessary data structures are created by calling **`[map,info] = prepare(segment,mode)`** where **`mode`** specifies which variables need to be estimated. This function can also construct the matrices S,R,V from a compressed description. Given data matrix Y with one column per time step, the estimation algorithm is called by **`[X,Err] = estimate(Y,x0,S0,map,info,R,V)`** where X is the matrix of state estimates (one column per time step) and Err is the matrix of standard errors (same size as X). One can also use **`estimate_ekf`** which is faster but less accurate. The core function **`residual(y,x,segment,map)`** is called internally by the estimation procedure. It computes the mismatch between the actual and predicted sensor measurements.

Kinematic modeling

The kinematic model is a tree of segments connected with joints. Segments in the tree can correspond to: (1) physical limb segments; (2) dummy segments needed to construct complex joints from the basic joint types allowed in the Toolbox; (3) virtual segments corresponding to sensors. Regardless of what it stands for, a segment is always associated with a 3D coordinate frame. A frame is described internally by an arbitrary 3D vector specifying the origin and three unit 3D vectors specifying the x,y,z axes. All these vectors are computed relative to the global frame (which is associated with the root segment). Such computation is called forward kinematics. Here it yields the position and orientations not only of the limb segments but also of the sensors (whose virtual segments and treated in the same way as "regular" segments). The expected measurements from 3D sensors are then equal to the position/orientation of the corresponding virtual segment's frame.

The tree is constructed by connecting segments with joints. A joint is a spatial transformation which determines the position and orientation of the child frame relative to the parent frame. The forward kinematics algorithm accumulates these local transformations along the kinematic tree. Any spatial transformation between a parent and a child frame is called "joint" even if it is constant. For example, forearm length is a (constant) parameter of a sliding joint which translates the frame from the elbow to the wrist. Every state variable is a parameter of some joint, and vice versa. In future releases, it will be possible to specify joint parameters that are known with certainty and do not need to be estimated. Presently, this effect can be accomplished by setting the prior variance of the corresponding state variables to a value near zero.

The types of joints supported by the Toolbox are the following:

	joint type	**description**	**dof**
1	S (liding)	translation in predefined direction	1
2	T (ranslation)	translation in any direction	3
3	H (inge)	rotation around predefined axis	1
4	R (otation)	rotation around any axis	3

1. Sliding joints correspond to sliding in a predefined direction. They have one parameter (included in the state vector) specifying the sliding distance. The sliding direction is one of the coordinate axes of the parent frame. This axis is specified as part of the model description and is not included in the state vector.
2. Translation joints correspond to arbitrary 3D translation of the child frame relative to the parent frame. They have three parameters which specify the translation vector.
3. Hinge joints correspond to rotation around a predefined axis. They have one parameter specifying the rotation angle. The rotation axis is given by one of the coordinate axes.
4. Rotation joints correspond to arbitrary 3D rotation. They have three parameters specifying an "angle-axis" vector. The length of this vector is the rotation angle (in radians) while the direction of the vector is the rotation axis.

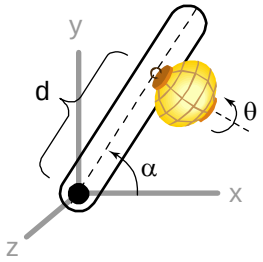
The construction of kinematic trees using these joint types is illustrated in the following examples, arranged in increasing order of complexity.

Example 1

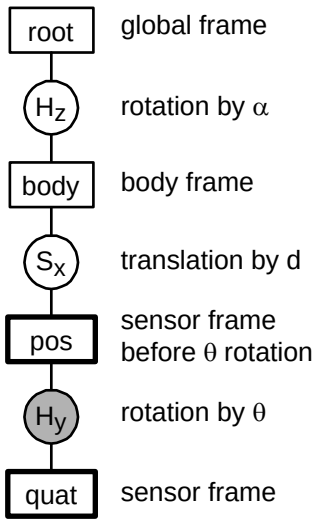
The simplest example is shown in the figure below. It involves a single rigid body rotating within the x-y plane around the z-axis. The rotation angle is α. A 3D position-and-orientation sensor is mounted on the body, at a distance d from the pivot point on the x-axis of the body (dashed). The distance d is also time-varying. The sensor is rotated by a constant angle θ around the y-axis of the body. Thus the state vector is 3-dimensional: x = [d;θ;α]. The measurement vector is 7-dimensional and contains the 3D position followed by a quaternion encoding the 3D orientation.

The corresponding kinematic tree is shown on the bottom. Rectangles denote segments/frames while circles denote joints/transformations. Each child segment is connected to its parent segment with a joint specifying the spatial transformation between the parent and child frames. Joints are labeled with their type. Sliding and hinge joints (S and H) are indexed with the coordinate axis serving as their predefined direction. Shaded circles correspond to constant transformations while empty circles corresponds to time-varying transformations. Rectangles with heavy outlines denote frames whose position or orientation are being measured by 3D sensors. Note that a single sensor is represented with two virtual segments – one for the position measurement and another for the orientation measurement. In this case the position measurement depends on d but not on θ, thus it can be taken before the rotation by θ.

The kinematic tree can be represented as a table listing all the segments (and their number in the tree), the parameters of the corresponding joint, the parent segment, the type of joint connecting each segment to its parent, the type of sensor associated with this segment (if any), the joint axis (for slide and hinge joints only), and the joint category (varying vs. constant).



segment name (#)	joint parm	parent segment	joint type	sensor type	joint axis	joint category
root (1)	none	none (0)	none (0)	none (0)	none (0)	none (0)
body (2)	α	root (1)	hinge (3)	none (0)	z (3)	varying (1)
pos (3)	d	body (2)	slide (1)	pos (2)	x (1)	varying (1)
quat (4)	θ	pos (3)	hinge (3)	quat (4)	y (2)	constant (2)



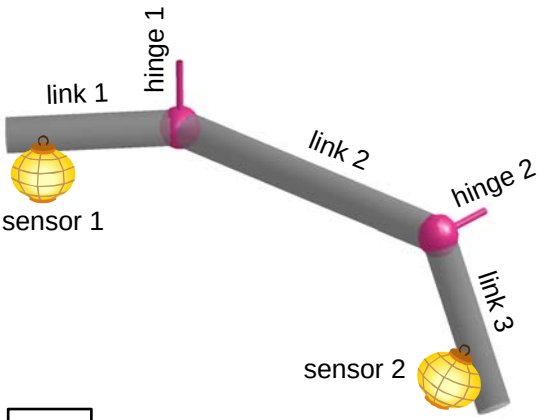
The first two columns and all the symbolic names in the above table are useful to make it human-readable. The user should draw such a table on a sheet of paper before encoding the model (ideally encoding will be done automatically in future releases). To encode this kinematic model in the Toolbox, all we need are the numbers in the shaded region. The corresponding 4x5 matrix of integers is created with the Matlab code shown below. This code can also be found in the file **`example1.m`**

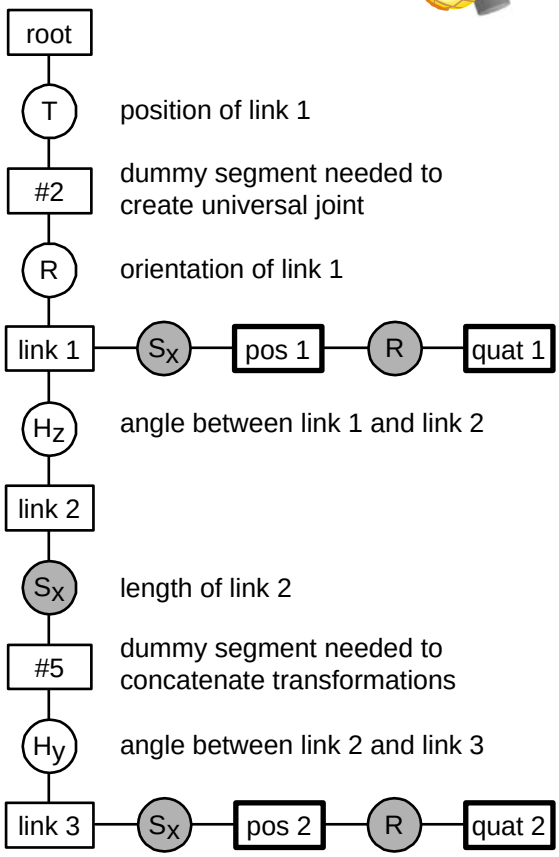
```
segment = [...
   0 0 0 0 0;  %1  root
   1 3 0 3 1;  %2  body
   2 1 2 1 1;  %3  pos
   3 3 4 2 2;  %4  quat
];
```

Example 2

This is a more complex example involving a kinematic chain with 3 links tracked by 2 position-and-orientation 3D sensors. Note that the sensor data cannot possibly tell us the physical lengths of links 1 and 3. Instead, all we can hope to estimate are the distances from the hinge joint axes to the attachment points of the sensors. The sensors are attached on the long (x) axes of links 1, 3 and have constant but arbitrary orientation which needs to be estimated. The distances from the hinge joints as well as the length of link 2 also need to be estimated. Thus there are 5 constant joints.

The variable joints include the two hinge rotations as well as to the global position and orientation of link 1. The latter is encoded with a "universal joint" composed of a 3D translation followed by a 3D rotation. Since two segments can only be connected with a single joint and the Toolbox does not allow a universal joint type, a dummy segment (#2) is introduced. Another dummy segment (#5) is needed to compose a sliding joint (length of link 2) with a hinge joint (angle between links 2 and 3).





As in the previous example, a 3D sensor measuring both position and orientation is treated as two separate sensors. Orientation measurements must be supplied as unit quaternions (see paper for a review of quaternions and their relation to 3D rotations). Most hardware sensors have an option for generating quaternion data. If that is not the case, the user should transform the orientation data to quaternion format before using the toolbox.

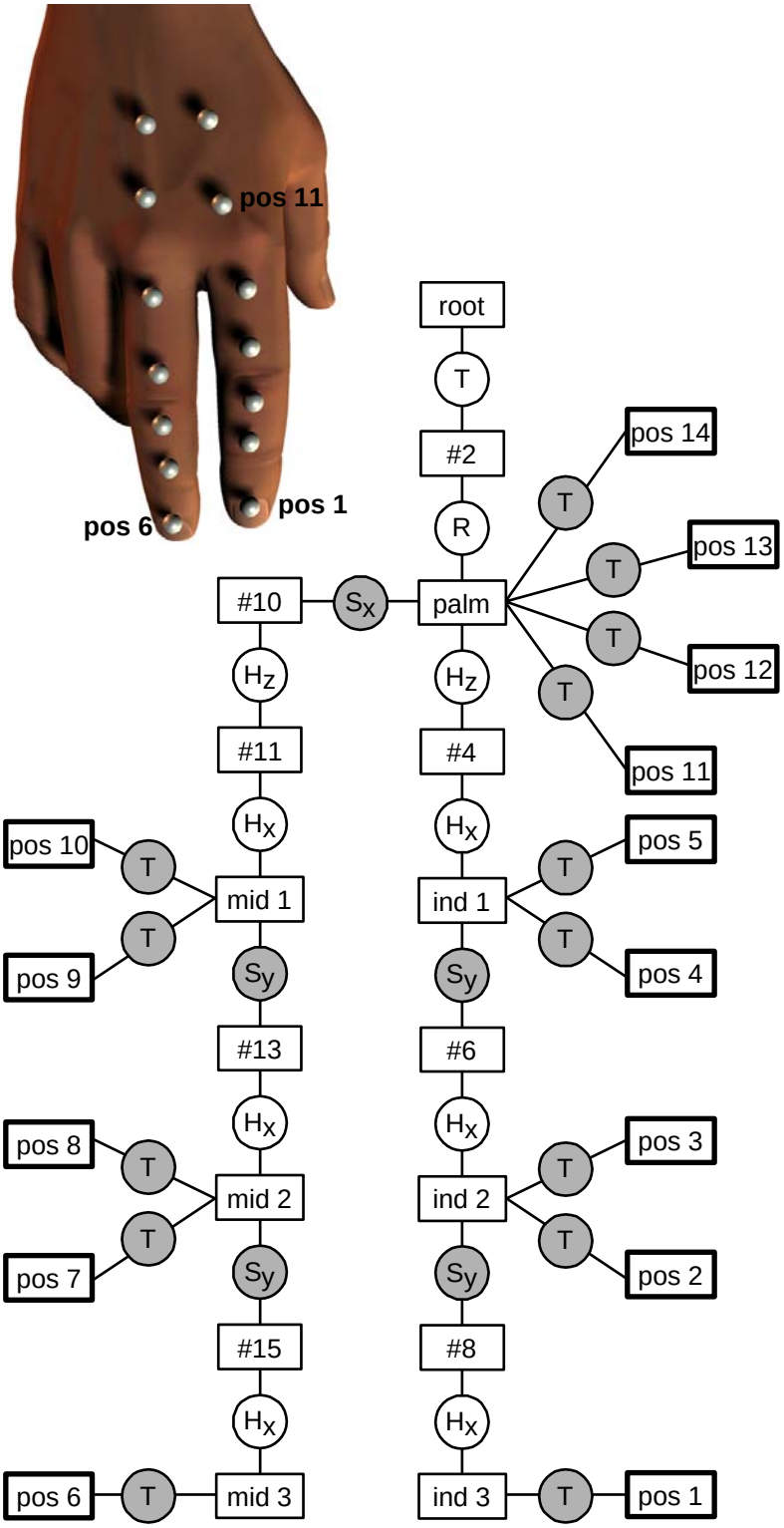
The Matlab code which creates the 10x5 matrix describing this model is given below. This code can also be found in the file **`example2.m`**

```
segment = [...
   0 0 0 0 0;  %1  root
   1 2 0 0 1;  %2
   2 4 0 0 1;  %3  link 1
   3 3 0 3 1;  %4  link 2
   4 1 0 1 2;  %5
   5 3 0 2 1;  %6  link 3
   3 1 2 1 2;  %7  pos 1
   7 4 4 0 2;  %8  quat 1
   6 1 2 1 2;  %9  pos 2
   9 4 4 0 2;  %10 quat 2
];
```

Example 3

This example illustrates hand tracking with 14 position sensors attached to a kinematic structure with 7 rigid segments (the palm and the three phalanges of the index and middle fingers). The constant parameters are the lengths of the phalanges, the distance between the proximal joints of the index and middle fingers, and the positions of all markers relative to the underlying segments. The time-varying parameters are the 8 hinge rotations (4 per finger) and the palm position and orientation.

The Matlab code which creates the 30x5 matrix describing this model is given below. This code can also be found in the file **`example3.m`**




```
segment = [...
   0 0 0 0 0;  %1  root
   1 2 0 0 1;  %2
   2 4 0 0 1;  %3  palm

   3 3 0 3 1;  %4
   4 3 0 1 1;  %5  ind 1
   5 1 0 2 2;  %6
   6 3 0 1 1;  %7  ind 2
   7 1 0 2 2;  %8
   8 3 0 1 1;  %9  ind 3

   3 1 0 1 2;  %10
  10 3 0 3 1;  %11
  11 3 0 1 1;  %12 mid 1
  12 1 0 2 2;  %13
  13 3 0 1 1;  %14 mid 2
  14 1 0 2 2;  %15
  15 3 0 1 1;  %16 mid 3

   9 2 2 0 2;  %17 pos 1
   7 2 2 0 2;  %18 pos 2
   7 2 2 0 2;  %19 pos 3
   5 2 2 0 2;  %20 pos 4
   5 2 2 0 2;  %21 pos 5

  16 2 2 0 2;  %22 pos 6
  14 2 2 0 2;  %23 pos 7
  14 2 2 0 2;  %24 pos 8
  12 2 2 0 2;  %25 pos 9
  12 2 2 0 2;  %26 pos 10

   3 2 2 0 2;  %27 pos 11
   3 2 2 0 2;  %28 pos 12
   3 2 2 0 2;  %29 pos 13
   3 2 2 0 2;  %30 pos 14
];
```