Inference of rotational kinematics from accelerometer signals

Data-processing model

Contents

General structure

data structures

input

$\vec{X}_i$ *i* = 1 to N   where *i* is the input position where $\vec{X} \stackrel{\text{def}}{=} \{A_x, A_y, A_z, \Delta t\}$ is the raw accelerometer sensor data

output

$\vec{O}_i$ *i* = 1 to M M <= N

the contents of $\vec{O}_i$ vary from one model to another, but typically represent the time evolution of constraint-parameter values and of the rotational kinematic variables.

window

The window selects a subset of the input data to process, generating a single output vector from it. It is the main processing element in the program. *K* consecutive input vectors treated in a single processing step. Other terms used analogously: "sliding window size", "filter size", "kernel size".

hyperparameters

- input length
- window size K, stride
- optimization parameters

execution model

1. Initialize
    a. load data
    b. set hyperparameter values
    c. initialize window
2. window-based processing
    a. update constants of the function evaluator from the current window data.
    b. call the optimization routine.
    c. write output
3. terminate
    a. visualize input, output, performance stats
    b. write output to file

function evaluation and differentiation

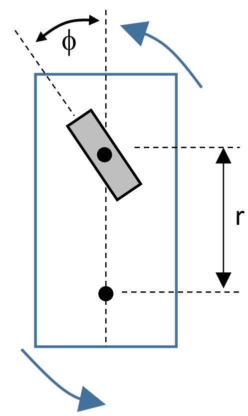
For these two capabilities we rely on the services provided by existing machine-learning libraries such as Theano or TensorFlow.

relation to machine learning

To describe the data processing, some vocabulary and notation from current machine-learning architectures is borrowed (CNN's and RNN's), but our learning model is very different in a number of important ways.

- the number of parameters in our models is very small (anywhere from 2 to about 10)
- both the parameters and the hidden variables correspond to human-interpretable, physically-based properties of a mechanical system.
- The system has no memory (outside of the current window). Based on a total error function for the current window position, we fully optimise the parameter values. The output, after each stride, is the (newly optimized) values the parameters.
- we do *not* rely on the usual operations such as convolutions or sigmoids. (Modeling of the low-level signal conditioning, a later extension, would introduce convolution layer(s).)
- in our equivalent of a hidden layer, a hidden (vector) variable is computed *independently* for each input vector. (There are no cross-connecting nodes in the computational graph). The values parametrising the function used to do this, however, are the same for each input vector within the current window position. This provides a form of local regularization.

Case 1: “Horizontal plane, planar non-alignment”



coordinate-transformation layer

For each input element $\vec{X}_k$ k = 1 to K within the current window position, we compute the *local kinematics* $\vec{l}_k$ associated with the point on the rigid body corresponding to the sensor location.

$$\vec{l}_k = \vec{T}_{\{r,\emptyset\}}(\vec{X}_k)$$

where $\vec{l}_k$ is the 4-vector

$$\vec{l}_k \stackrel{\text{def}}{=} \{a_r, a_t, v, \Delta v\}_k$$

composed of estimates for the radial acceleration a_r , tangential acceleration a_t, tangential velocity v and anticipated change in velocity over the current time interval Δt . Function *T* is composed of

$$\begin{pmatrix} a_r \\ a_t \end{pmatrix} = Rot_\emptyset \begin{pmatrix} A_x \\ A_y \end{pmatrix}$$

and

$$v = \sqrt{r a_r}$$

where

$$Rot_\emptyset = \begin{pmatrix} cos\varphi & -sin\varphi \\ sin\varphi & cos\varphi \end{pmatrix}$$

The vector function $\vec{T}$ is parametrised by the constraints parameters $\vec{c} \stackrel{\text{def}}{=} \{r, \emptyset\}$

whose values are determined though iterative minimization of an error function defined below.

(To avoid repeated calculations of the square-root operation over the iterations of the optimization loop, we will replace parameter *r* by $\rho \stackrel{\text{def}}{=} \sqrt{r}$ and, after optimization is completed, use

$$v = \rho\sqrt{a_r}$$

for the calculation of *v*.)

error function

For each j = 1 to K-1 we predict the velocity associated with the next time step:

$$\tilde{v}_{j+1} = (a_t)_j \Delta t$$

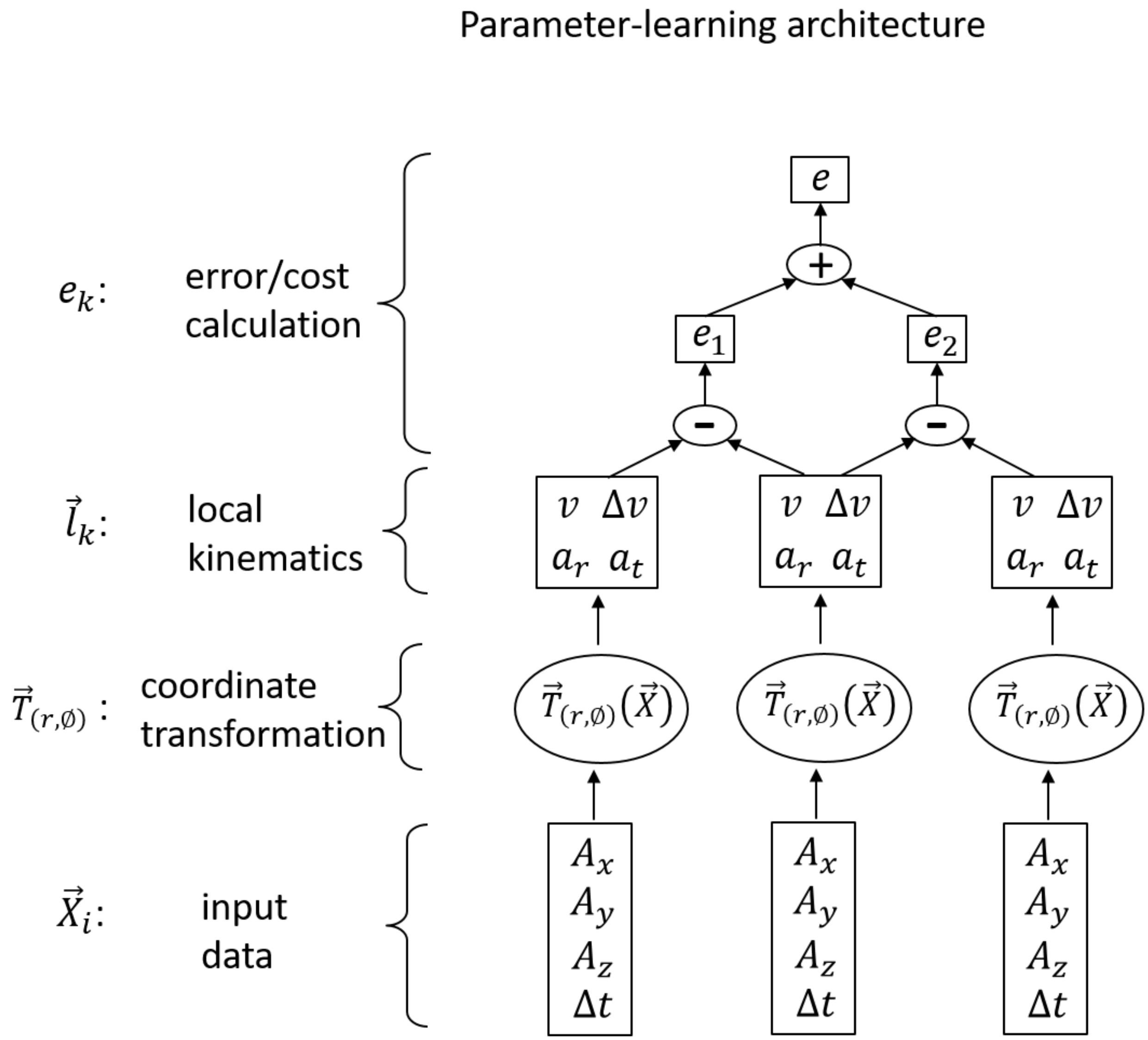
and the error e_j resulting from a comparison with the velocity is computed from the input data of time step j+1:

$$e_j = \tilde{v}_{j+1} - v_{j+1}$$

The total error is the direct sum of the e_j

$$e = \frac{1}{K-1} \sum_{j=1}^{K-1} e_i$$

The cost is minimized against the constraint parameters $\vec{c}$.



output:

As the kernel strides along the input data, it produces an output at each window position

$$i = 1 \; to \; N/stride$$

Outputs are new time-series sequences. There are two categories of outputs available:

1. constraint parameters $\vec{c}_i = \{r, \emptyset\}_i$ discussed previously; and

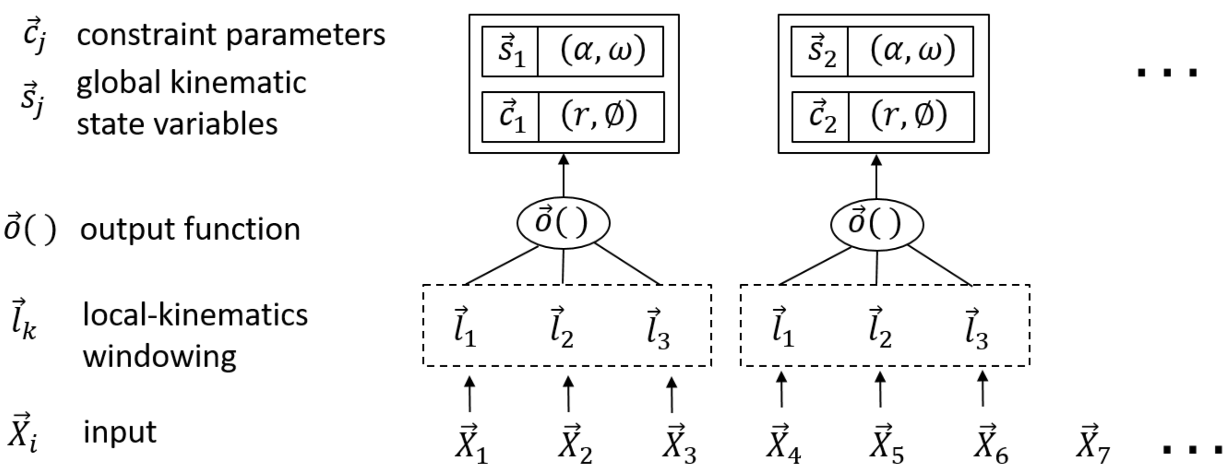
2. motion or system-state variables $\vec{s}_i = \{\alpha, \omega\}_i$ where

$$\alpha = r\overline{a}_t$$
$$\omega = \bar{v}/r$$

where the overhead bar indicates an average over all K-1 values in the current processing window, and *r* is calculated from the optimization parameter ρ via

$$r = \rho^2$$

Data processing and output



Other cases

Case 1b: “Horizontal plane, non-alignment about tangential axis”

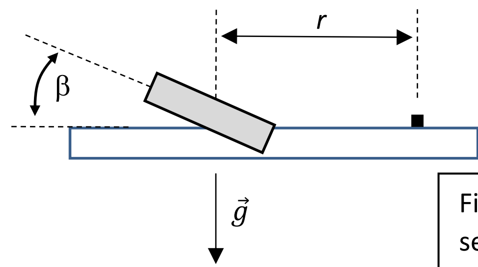


Fig. Side view of rigid body and sensor. Tangential axis points out of the page.

Student exercise.

Case 1c: “Horizontal plane, non-alignment about radial axis”

Student exercise

Case 2: “Vertical plane, aligned”

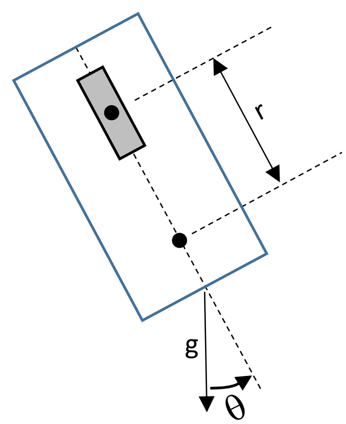


Figure 1 A system rotating in a vertical plane (viewed from the side, i.e. viewing axis parallel to the Earth's surface)

Here we begin with a sensor whose x-axis is aligned with the radial (outwards) direction.

Process is the same as before, only the function *T* is now with the following:

$$\begin{pmatrix} a_r \\ a_t \end{pmatrix} = \begin{pmatrix} A_x \\ A_y \end{pmatrix} - g\begin{pmatrix} cos\theta \\ sin\theta \end{pmatrix}$$

The calculations based on a_r and a_t proceed the same as before.

Optimization parameters are: (r, θ)

Output of state variable now includes θ as well: $\vec{s}_i = \{\alpha, \omega, \theta\}_i$

Case 2b: "Vertical plane, planar non-alignment"

Student exercise

Appendix: answers to student exercises

Case 1b: "Horizontal plane, non-alignment about tangential axis"

Here the sensor y-axis remains aligned with the tangential. We need only perform a rotation in the x-z plane to recover the radial component:

$$\begin{pmatrix} a_r \\ a_t \end{pmatrix} = \begin{pmatrix} cos\varphi & 0 & -sin\varphi \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix}$$

This should remove the gravitational component. One can verify this by evaluating the acceleration component perpendicular to the plane of rotation:

$$a_{\neg} = \begin{pmatrix} sin\varphi & 0 & cos\varphi \end{pmatrix} \begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix}$$

Its value should equal g.

Case 2b: "Vertical plane, planar non-alignment"

Here, function T is

$$\begin{pmatrix} a_r \\ a_t \end{pmatrix} = Rot_{\emptyset} \begin{pmatrix} A_x \\ A_y \end{pmatrix} - g \begin{pmatrix} cos\theta \\ sin\theta \end{pmatrix}$$

Optimization parameters are: $(r, \emptyset, \theta)$. For a fixed axis, *r* and $\emptyset$ are constant, but θ varies . For non-fixed axes, the first two parameters can be treated as slowly varying or piecewise constant.