Versor

versatile cursors

J. C. G. Sturdy

Published by John C. G. Sturdy.

Versor, the "versatile cursor" package, provides, and makes easily accessible, cursor movements in a variety of groups of "dimensions", such as:

		pages	lines	chars
	functions	depth of nesting	expressions	chars
paragraphs	sentences	phrases	words	chars
functions	statements	statement-parts	expressions	chars

Versor is one of a pair of companion packages. The other one, Languide, provides some high-level editing operations. Part of Versor's functionality is provided by the lower-level parts of Languide. See section "Languide and Versor" in *Languide: language guided editing*, for more information.

1 Introduction

Versor provides, and makes easily accessible, cursor movements in a variety of groups of "dimensions", such as:

- Characters, lines, pages
- Characters, expressions, depth of expressions, functions
- Characters, words, phrases, sentences, paragraphs
- Characters, expressions, statement-parts, statements, functions

Some of these operations are already present in GNUemacs, and some are added by Versor. Versor puts them all in easy reach on the arrow keys.

Clearly, more dimensions are mentioned above than will fit at the same time on the few arrow keys available (just two dimensions).

However, note that the available dimensions are conveniently listed in a two-dimensional table, as above.

Versor uses the arrow keys themselves, with a modifier key, to select a pair of adjacent dimensions from one of the rows above, and assign them to the arrow keys.

You will notice (unless your display cannot render this) that a unit of text, of the same size as the dimension you are using, will be highlighted, after each Versor command. This is what Versor regards as the cursor. The normal cursor appears at the start of it (or occasionally at the end), and all normal GNUemacs operations are available and will use the normal cursor. In emacs versions from 21 onwards, the colour of highlighting changes to indicate the current dimension; there is also an indication in the mode line.

The Versor highlighted cursor disappears at the start of every command (and is restored at the end of each Versor command), so when you use GNUemacs commands other than Versor commands, Versor quietly becomes invisible, until you use one of its commands again.

1.1 The aims of Versor

Although we have the computer power to edit programs in more abstract terms than characters and lines, and many "natural" programmers have the thought power to do so, editing is still often done character-by-character, which seems to be inefficient, unreliable, and a waste of human effort. Versor aims to move beyond that.

Versor is also part of a research project (see Chapter 10 [Versor and research], page 16), which looks into whether programmers really have moved on from thinking of edits in terms of the lines of characters that they see on paper listing and on editor screen.

Ideally (especially for those whose hands are becoming tired of keyboard use) it should be possible to enter and edit programs with very little handling of individual characters. Versor tries to get as close to this as possible. (It includes an analysis facility (see Chapter 9 [Analysis], page 15) to see well it is doing in this, with each particular programmer.) All that need be entered character by character are new identifiers and comments. It may even be possible in future versions to offer a selection of suitable names for variables in certain circumstances, such as `i` for a first index variable, as some of these are quite stereotyped.

2 Familiarization

The easiest way to start finding your way around Versor is to start trying things. You will first need to install Versor Chapter 8 [Setup], page 13, and call its initialisation function, indicating which set of arrow keys you wish it to use.

Versor can use either the keypad arrows, or the normal arrow keys. Because it is often useful to mix traditional movements with Versor movements, it is recommended that you give Versor only one set of arrow keys. The explanations here will be phrased in terms of the main arrow keys and the key cluster that, on typical keyboards, is just above them. See Chapter 8 [Setup], page 13, for how to set Versor up to use a particular set of keys.

The clearest demonstration of the basic facilities of Versor is probably to be had with a buffer containing a large number of Lisp functions, some of them with very deeply nested expressions. Since Versor provides commands for modification as well as navigation, we suggest you take a copy of your sample material!

Unless customised to do otherwise (see Chapter 8 [Setup], page 13), Versor will start in its 'cartesian" coordinate system, which is similar to the normal cursor keys: you can move left or right with the ⟨LEFT⟩ and ⟨RIGHT⟩ arrow keys, and up and down with the ⟨UP⟩ and ⟨DOWN⟩ arrow keys.

2.1 Zooming within a coordinate system

You can try zooming out along a series of coordinates by typing *M-LEFT*. Now the minor cursor keys (*LEFT* and *RIGHT* arrows) will select previous and next lines of the buffer, and the major cursor keys (*UP* and *DOWN* arrows) will move by whole pages. You will also see that when you move by lines, a whole line is highlighted in a particular colour. This is the current *versor selection*, and its colour provides feedback on the currently selected dimension.

2.2 Switching between coordinate systems

Now try switching to a different coordinate system (series of dimensions), by typing *M-down*. This should select a "structural" coordinate system, and if your current dimension was the second one, "lines", you should now be in the second of the structural coordinates, which is "exprs", that is, "expressions". The minor cursor keys will now select previous and next s-expressions (you will see the current expression is highlighted), and the major cursor keys will select the depth (you will see the pair of brackets highlighted – Versor can use multipart selections).

You can control which dimensions are available in each mode Section 8.2 [Dimensions available], page 14.

2.3 Extending the selection

You can also extend the selection, using *C-right* and *C-left*. If you start by moving right, the extension grows from the end of the current selection, and moving left will shrink the extension. If you start by moving left, the extension grows from the beginning of the current selection, and moving right will shrink the extension.

3 Available dimensions

This section gives details of the groups of dimensions which Versor provides.

3.1 Cartesian navigation

Cartesian navigation is the Versor dimension most similar to the normal cursor keys. The main difference you will notice is that when you have selected "lines" as your current dimension, you will see a whole line highlighted.

The dimensions available in cartesian navigation are as follows:

characters

lines

pages

3.2 Structural navigation

Structural navigation uses bracketing characters to navigate by s-expressions. Moving in the "depth" dimension leaves the selection split into the opening and closing brackets.

The dimensions available in structural navigation are as follows:

chars

exprs

depth

defuns

3.3 Text navigation

Text navigation works in terms of natural language units such as words and phrases.

The dimensions available in the text navigation are as follows:

chars

words

phrases

sentences

paragraphs

Navigating by words is slightly different from the emacs `forward-word` command (see ⟨undefined⟩ [Words], page ⟨undefined⟩): it always leaves the cursor on the first character of a word.

Navigating by phrases uses the variable `phrase-end` to define the end of a phrase.

The other dimensions in this group are based closely on the underlying Emacs facilities. See ⟨undefined⟩ [Sentences], page ⟨undefined⟩, ⟨undefined⟩ [Paragraphs], page ⟨undefined⟩.

3.4 Structured Text navigation

Structured text navigation is similar to text navigation (see Section 3.3 [Text], page 4), with the addition of moving by nested blocks as used in many markup languages, in a manner similar to structural navigation (see Section 3.2 [Structural], page 4).

The definition of block syntax is mode-specific. For example, in HTML, paired tags are defined to begin and end blocks.

An associated command is `another-block` which analyzes the block before point (that is, from a closing-block construct back to the corresponding opening-block construct), collects up all the structuring constructs in it, and inserts a copy of those at point. For example, if point is just after an HTML table row (so that the tag most immediately before point is `</tr>`, the HTML table row ender), `another-block` will find all the HTML tags back to the corresponding `<tr>`, and insert them, but without the intervening non-tag text. (This specific example is probably the most useful use of this command, in the author's experience.)

This part of Versor is very much a work in progress, and you are encouraged to add to the definition of block syntax for your favourite markup language, and send it to the author of Versor. The block syntax mechanism is defined in the file '`nested-blocks.el`'.

3.5 Tables navigation

Table navigation provides movements based on cells and rows, in various modes for which these concepts are meaningful.

The dimensions defined in table navigation are as follows:

chars

cells

rows

3.6 Program navigation

Program navigation uses the syntax of the programming language to guide to the cursor movements. This uses the accompanying package, Languide, to provide the primitives for moving around source code. It is, of course, mode-specific.

Languide definitions are already reasonably comprehensive for Lisp, C, Perl and Java, partially written for sh, and planned for Haskell and Python. They would probably not be meaningful for PostScript. You are invited to contribute further or better Languide definitions, both for the languages already covered and for others.

The dimensions defined in program navigation are as follows:

chars

exprs

statement parts

statements

defuns

Apart from the statement-related dimensions, these are the same as those in structural navigation Section 3.2 [Structural], page 4.

Movement by statement parts moves the selection between parts of the current statement. Programming language statements can typically be divided into two or three parts, as follows:

1. The head of a statement is often a condition, such as in an `if` or a `while` statement.

2. The body of a statement is typically another statement, or a group of statements, such as the body of an `if` or a `while` statement.
3. The tail of a statement occurs in only a few statement types, such as `if-then-else` or `try-catch-finally`.

The "statement parts" dimension will move between these in turn, and can also moved to the *container* of the current statement, that is, a grouping statement surrounding it.

If using voice input (see Section 6.3 [Voice input], page 10), it is convenient to define commands such as "head", "body",and "container", so that you can jump directly to the one you want rather than treating it as part of a sequence. The commands `navigate-this-head`, `navigate-this-body`, `navigate-this-tail`, `navigate-this-whole`, and `navigate-this-container` are available for binding directly to voice commands (or, for that matter, to keys of their own).

The "statement" dimension moves between successive statements.

4 Further commands

As well as redefining the arrow keys, Versor takes over a few other keys as well. The most noticeable of these is that *DEL* now deletes the current Versor selection.

4.1 Deletion

Versor defines the *DEL* key to delete the current selection. Note that if this is a multipart selection, such as a pair of opening and closing brackets, all parts of it are deleted.

Deleting a multipart selection puts the parts of it into separate entries in the kill-ring. This is compatible with Versor's insertion commands (see Section 4.2 [Insertion], page 7) while also keeping compatibility with normal Emacs kill-ring use.

4.2 Insertion

Pressing the *INS* key gives you a choice of several ways to insert, which you choose between by pressing one of the Versor arrow keys.

The *LEFT* and *RIGHT* arrow keys let you insert before or after the selection.

The *UP* arrow key inserts two entries from the kill ring, one before and one after the selection; that is to say, it inserts around the selection. This is compatible with Versor's way of deleting a multipart selection such as a pair of opening and closing brackets Section 4.1 [Deletion], page 7.

These are defined in `versor:insertion-placement-keymap`.

Having selected where to insert something, you then select what to insert.

You can type any deletion-type key to insert the top item on the kill-ring; or a digit key to insert the second, third... item.

You can type *C-s* to insert the latest search string.

You can type *f* to insert the name of the file in the next window (if there is only one window, the name of the file in that window is inserted).

You can type any kind of opening bracket to insert a bracket, or, if inserting around the selection, a pair of brackets.

If inserting around the selection, you can type *?* to wrap an `if` statement around the selection, `@` to wrap a `while` statement around it, `=` to wrap a scoping construct around it (like `let` in Lisp), *&*, `|` or *!* to wrap an `and`, `or`, or `not` expression around it. These are very basic string insertions; Languide (see ⟨undefined⟩ [Alterations], page ⟨undefined⟩) has more sophisticated ones, which use the template and skeleton insertion systems, and can maintain indentation and spacing better.

4.3 Alteration

The "alterations" system provides a Versor style interface to changing the text of the current Versor selection. While you are doing alterations, the ⟨LEFT⟩ and ⟨RIGHT⟩ cursor keys change the text within the selection, between several possible values of the same kind. The ⟨UP⟩ and ⟨DOWN⟩ cursor keys change which kind of thing you are choosing between.

The effect is similar to turning the selection into a little window behind which you are scrolling a two-dimensional grid of possible values.

For example, you could use the ⟨LEFT⟩ and ⟨RIGHT⟩ keys to put any local variable which is currently in scope into the selection, and ⟨UP⟩ and ⟨DOWN⟩ keys to switch between selecting local variables, global variables, and expressions wrapped around a variable.

When you have got the text you want into the selection, you can use a "select" key to accept that selection, or a "reject" key to go back to the original value.

The selection of types of value depends on the context. Possible types include:

local variables

global variables

tags in your tag table

functions defined in this file

reserved words in a programming language

The data used by Versor's "alteration" feature is provided by the companion package, Languide (Language Guided Editing). See ⟨undefined⟩ [Alterations], page ⟨undefined⟩ for details.

4.4 Do What I Mean

Sometimes Versor deviates from its most logical design, to make sure that the cursor ends up where you are likeliest to want it.

An example of this is that if you are moving by s-expressions, and move forward from the last one in the enclosing expression, the cursor moves to the end of the last expression, and then on past any whitespace and comments, ending up where you are likely to want to type the next s-expression.

However, it is still not always possible to get to exactly the right place without using character-level movements. For these circumstances, Versor provides a "Do What I Mean" (DWIM) command, which moves the cursor to a point likely to be of interest that is otherwise hard to get at using the current Versor dimension. This command is normally bound to the key *M-home*. Alternatively, it can be accessed with *M-x versor:dwim*.

This function reads the user's mind, using the following algorithm:

1. Tell the user what has been done each time;
2. Eventually, the user will come to expect the behaviour of this function;
3. Reading what the user wants Versor to do should then usually be trivial.

Aspects of mental state not necessary for figuring out where to leave point are factored out of the calculations.

`versor:dwim` is mode-specific in its behaviour. For example, in programming language modes, it can move point in and out of string constants and comments.

4.5 Refactoring

Some more sophisticated commands, replacing long sequences of manual labour by the programmer, are provided by the companion package, Languide ⟨undefined⟩ [Top], page ⟨undefined⟩

5 Advanced features

5.1 Text in code

You can make Versor remember different current dimensions for actual code, and for the insides of comments and string literals. You can turn this on by requiring the feature `versor-text-in-code` and setting the variable `versor:text-in-code` to anything other than `nil`; or by including the symbol `text-in-code` in the arguments to `versor:setup`, which does both of the above.

5.2 Per-buffer dimensions

You can make Versor remember different current dimensions for each buffer. You can do this by requiring the feature `versor-local` and setting the variable `versor:per-buffer` to anything other than `nil`; or by including the symbol `local` in the arguments to `versor:setup`, which does both of the above.

5.3 Per-mode dimensions

You can make Versor remember different current dimensions for each mode. You can do this by requiring the feature `versor-modal` and setting the variable `versor:auto-change-for-modes` to non-nil; or by including the symbol `modal` in the arguments to `versor:setup`, which does both of the above.

6 Accessibility

Versor is designed to be usable not only through a conventional keyboard, but also through narrow interfaces such as pedals, and through voice recognition.

6.1 Using pedals

The setup used by the author requires six pedals, arranged in two sets of three. The pedals are daisy-chained into the keyboard connector, and duplicate the actions of selected keys. One set provides the modifiers ⟨Control⟩, ⟨Shift⟩ and ⟨Alt⟩, and the other provides three types of action, referred to here as ⟨Other⟩, ⟨Move⟩, and ⟨Select⟩.

If using the common commercially available pedals in which each unit has a large central pedal and a smaller pedal along each side, the recommended setup puts ⟨Shift⟩ and ⟨Move⟩ onto the large pedals.

⟨Move⟩ is the main action, moving forward in the current dimension, and corresponds to the ⟨Right⟩ cursor key. *S-Move* moves in the other direction in the same dimension.

⟨Other⟩ and *S-Other* provide the ⟨DOWN⟩ and ⟨UP⟩ actions of Versor.

⟨Select⟩ brings up a menu, using the Text Mode Menus (see ⟨undefined⟩ [Menu Bar], page ⟨undefined⟩), and you can then move along the menu using ⟨Move⟩ and *S-Move*, and select an entry using ⟨Select⟩, which will either bring up a further menu, or run the selected command.

Some of these commands read their arguments through a system built on top of completing-read, that is designed to allow fast selection of possibilities without needing typing. See Section 6.4 [Flexi-choose], page 10, for details of this.

6.2 Reversing the cursor motions

To make Versor usable with a very small number of keys (for example, a mouthswitch), it is possible to reverse the motion of Versor's "forward" and "back" commands (of all kinds, both dimensions and both meta-dimensions (ways of choosing dimensions). This is available through the command `versor:reverse`. If reversing is used, it appears in the mode line as an arrow at the appropriate end of the dimension indication.

This is an extreme solution, for extreme problems. It's unlikely to be useful if you can use Versor in other ways.

6.3 Using voice input

Some parts of Versor are particularly suited to use with voice input, such as vr-mode `http://emacs-vr-mode.sourceforge.net/`.

Some of the voice or menu commands read their arguments through a system built on top of completing-read, that is designed to allow fast selection of possibilities without needing typing. See Section 6.4 [Flexi-choose], page 10, for details of this.

6.4 The flexible chooser

Flexi-choose splits a list of choices into a tree, and allows navigation down the tree without having to either type parts of an entry, nor having to scroll past all the ones before the one you want. It uses similar techniques to Text Mode Menus (see ⟨undefined⟩ [Menu Bar], page ⟨undefined⟩), but constructs the menu tree on the fly.

If used with a voice extension (contact the author for details), it is possible to say a word which occurs in the entry you want, and have the list of choices redisplayed with only the entries that match that word. Saying another word will narrow it down further, and so on. When the list is down to a single entry, that one is then selected automatically.

6.5 Other input devices

As well as pedals (see Section 6.1 [Pedals], page 10) it should be possible to extend versor to use other input devices, such as joysticks and mouthswitches.

For commands suitable for use with very "narrow-channel" input devices, see Section 6.2 [Reversing], page 10.

A complex joystick (at least with twist as well as X and Y movements) could prove to be an excellent way to use versor. The main stick movements could be the main versor movements, and a side twist could either change dimensions or do alterations. Alternatively, a Fire button could switch the main co-ordinates between movement and alterations. Some joysticks have a smaller joystick mounted on the main joystick; that would be another way to access alterations or dimension changes. A button on the side of the joystick could also change the main movements between navigating the buffer and navigating the space of navigation dimensions.

7 Versor and Languide

Language provides "language guided editing", guiding editing operations using the syntax of the file being edited.

The statement-based dimensions of Versor use the lower levels of Languide.

Languide is a cross-language package, with definitions for several popular programming languages.

See section "Languide and Versor" in *Languide: language guided editing*, for languide's end of the story..

8 Setup

To install Versor, unpack the tarball into a suitable directory, and put that directory on your load-path.

From your .emacs, for the default setup, call

```
(require 'versor)
(versor:setup)
```

8.1 General configuration

You can control which keys are used, and which other facilities are turned on, by giving some symbols as arguments to `versor:setup`.

The arguments can be any (combination) of

`arrows` for the main cursor keys

`arrows-misc`
: for insert, delete etc

`keypad` for the keypad cursor keys

`keypad-misc`
: for keypad insert, delete etc

to select which keys are set up to do Versor commands.

You can turn on further facilities by including the following symbols amongst the arguments:

`modal` remember a different dimension for each mode

`local` remember a different dimension for each buffer

`text-in-code`
: switch dimensions for string literals and comments, allowing code-oriented movement in actual code, and text-oriented movement in embedded natural language text

`menu` define a menu of Versor commands

The recommended default setup is:

```
(versor:setup 'arrows 'arrows-misc 'modal 'text-in-code 'menu)
```

You may then want to customize it further, like this:

```
;; preset the dimensions for some modes
(setq versor:mode-current-levels
  (mapcar 'versor:mode-levels-triplet
  '(
    (emacs-lisp-mode "structural" "exprs")
    (lisp-interaction-mode "structural" "exprs")
    (c-mode "program" "statement-parts")
    (text-mode "cartesian" "lines")
    (html-helper-mode "text" "words")
    )))
```

8.2 Dimensions available for each mode

The variable `versor:meta-dimensions-valid-for-modes` controls which meta-dimensions are valid for which major modes.

If t, all meta-dimensions are allowed in all major modes.

Otherwise, it is an alist mapping modes to sublists describing the meta-dimensions allowed in that mode.

Each sublist should begin with t, to indicate that only the meta-dimensions listed are to be allowed, or nil, to indicate that all meta-dimensions except those listed are allowed.

The rest of the sublist is the meta-dimensions allowed or blocked for that mode.

The head of the node may also be a list of major modes for which this rule applies.

A sublist for a major mode t gives the defaults.

9 Versor use analysis

Versor provides some facilities for measuring how much you are using it, and how much you are using non-Versor commands, and how the two are mixed. This is partly because Versor was developed as part of some research on the psychology of programming Chapter 10 [Versor and research], page 16, and partly so that you can tune your own computer use, and partly to get feedback on what needs to be added to Versor to make it a complete way to edit programs.

10 Versor and research

Versor was developed as part of some research on the psychology of programming Chapter 10 [Versor and research], page 16, to see how much people editing programs will use the facility to work at a higher level of abstraction than the character-by-character, line-by-line level, and how much, and when, they resort to traditional commands despite more powerful ones being available.

Versor will be used in controlled experimental conditions for this, but to gather wider information, the author encourages you to use Versor's use analysis features (see Chapter 9 [Analysis], page 15) and send in the results, once you have been using Versor for a while.

Command Index

(Index is nonexistent)

Concept Index

(Index is nonexistent)

Table of Contents