How to Use Showmotion

Contents

Showmotion is a Matlab function for displaying animations of rigid-body systems using Matlab's handle graphics. It is typically used to display the results of a dynamics simulation, but the motion data can come from any source, and does not have to be the result of a dynamics calculation. **Showmotion** can display an animation from any angle, at any magnification, and at any speed (including backwards); but the one thing it cannot do is display the data while it is still being calculated. In other words, **showmotion** is a post-processing tool: you use it after a simulation run has finished, or after it has been stopped or paused.

Showmotion is intended to be easy to use. So, why not type **showmotion(planar(2))** into the Matlab command window now, and see how much you can figure out without reading this document.

Calling Showmotion

showmotion(model [, tdata, qdata])	start a new animation
showmotion(filename)	load animation from file
showmotion('save', filename)	save current animation
showmotion('about' [, meta])	display/edit metadata

The first two calls start a new animation, terminating the current one if any. They can be issued at any time. The second two calls can be issued only if there is a current animation.

In the first call, **model** is a system model data structure containing an **appearance** field and, optionally, also a **camera** field. The former supplies the drawing instructions that **showmotion** needs in order to display the model; and the latter controls how the model is viewed (details here). **tdata** is a row or column vector of monotonically-increasing time values. It is not necessary for the time values to be strictly monotonic. However, the last element in **tdata** must be strictly greater than the first. **qdata** is a matrix containing joint position data. It can be either an **m×n** array or an **m×1×n** array, where **m** is the number of joint variables (i.e., **m=model.NB**) and **n** is the number of elements in **tdata**. The second option is the array output format of Simulink, so that Simulink outputs can be fed directly into **showmotion**. The contents of **qdata(:,i)** or **qdata(:,1,i)** (as appropriate) are interpreted as the vector of joint position variables at time **tdata(i)**. To achieve a smooth animation, **showmotion** uses linear interpolation to calculate the values of the position variables at intermediate times.

Example:

To view an animation in which the robot **planar(2)** moves from position **[0;0]** (=straight) at time **0** to **[1;-2]** at time **1**, and then returns to **[0;0]** at time **3**, type **showmotion(planar(2), [0 1 3], [[0;0] [1;-2] [0;0]])**

If the arguments **tdata** and **qdata** are omitted, then **showmotion** supplies a default animation in which each joint variable in turn is ramped from 0 to 1 and back to 0 in the space of one second (so the complete animation lasts **model.NB** seconds). This is useful when you just want to view an existing model, or when you are building your own models and want to see if you have got the drawing instructions right and put the joints in the right places.

In the second and third calls listed above, **filename** is the name of a Matlab '.mat' file. In the second call, **filename** is the name of an existing file containing a previously-saved animation that **showmotion** is to load. In the third call, **filename** is the name of the file to which the current animation is to be saved. (**Note: showmotion** does not check to see if the save-file already exists, and does not warn you if you are about to overwrite an existing file.) In both cases, the '.mat' extension is optional. Thus, to load an animation from the file 'results.mat', **filename** can be either 'results' or 'results.mat'.

The fourth call allows you to view and edit the metadata associated with an animation. The metadata consists of the following items: *title*, *author(s)*, *date stamp* and *description*. Calling **showmotion('about')** causes **showmotion** to print the metadata associated with the current animation in the Matlab command window. Calling **showmotion('about',meta)** lets you replace one or more items of metadata with new values. **meta** is a structure containing one or more of the following fields: **author**, **title** and **description**. For each field present, **showmotion** replaces the relevant item with the specified new value. All three fields are strings. However, whereas **author** and **title** specify the new value directly, **description** is the full file name of a plain text file containing the new description. The date stamp is set automatically when an animation is saved, and cannot be edited. Probably, the easiest way to set the metadata is as follows:

```
showmotion( 'about', struct( 'author', 'John Smith' ));
showmotion( 'about', struct( 'title', 'A Perfect Animation' ));
showmotion( 'about', struct( 'description', 'perfect.txt' ));
```

Viewing Commands 窗口命令

Showmotion's viewing commands make sense if you regard them as acting on the scene rather than the camera. For example, the left-arrow key (in the arrow-key cluster) performs the command *pan left*. If you press this key then the scene moves a little bit to the left. Now, to make this happen, it is necessary to turn the camera a little bit to the right, so one could argue that this is really a *pan right* command, since the camera has panned to the right. Nevertheless, the scene is what you see, and the scene has moved to the left, so **showmotion** calls this operation *pan left*.

The same idea applies when using the mouse. For example, to pan with the mouse, you press the middle mouse button (or left and right simultaneously if you only have a two-button mouse). If you now drag the cursor, you will see that the scene is dragged along with it. In other words, it is the scene, not the camera, that moves in the direction of the drag.

The same idea applies also to zooming. One way you can zoom is with the scroll wheel on the mouse (if you have one). Pushing the wheel away from you causes the scene to move away from you, so that things in the scene look smaller; and if you pull the scroll wheel towards you then you are pulling the scene towards you, so that things look bigger. Another way to zoom is to use the up- and down-arrows with the control key pressed. As the up-arrow is pointing away from you, it pushes the scene away, and the down-arrow pulls the scene towards you.

Rotation is a bit more complicated. Nevertheless, it too follows the same basic idea. To rotate using the mouse, you press the left mouse button. As soon as you do this, a large *crystal ball* appears and remains visible until you release the button. What happens next depends on where the cursor is relative to the crystal ball. If the cursor is somewhere near the middle, then dragging the cursor causes the crystal ball to rotate as if you were rotating it with your finger (i.e., it's as if the cursor is touching the ball, and the contact point on the ball follows the cursor as it is dragged). Meanwhile, the scene behaves as if it is fixed to the crystal ball, and therefore undergoes the same rotation. The overall effect is that things in the scene which are closer to you than the centre of the crystal ball move in the same direction that the cursor is being dragged, while things that are further away move in the opposite direction. The same effect can be obtained using the arrow keys with the shift key pressed. In each case, the scene rotates such that the foreground moves in the direction of the arrow.

If, instead, the cursor is not over the crystal ball, then dragging the cursor causes the scene to spin about its centre point in step with the orbital motion of the cursor. For example, if you drag the cursor clockwise then the scene rotates clockwise by the same amount. If the cursor is near the edge of the crystal ball then you get a rotational behavoiur that is intermediate (in 3D space) between the two behaviours.

Refocus
重调焦距

Showmotion provides one more viewing command, which is a bit unusual. It is called *refocus*, and what it does is alter the depth of the crystal ball. You perform a refocus using the scroll wheel while the left mouse button is pressed (i.e., while the crystal ball is visible). Pulling the scroll wheel towards you will pull the crystal ball towards you, and pushing the scroll wheel away will push the crystal ball away. However, because the crystal ball automatically resizes itself to fill a fixed fraction of the window, you won't see a change in the apparent size of the crystal ball. The only way you can tell it is moving is by watching the intersections between the crystal ball and objects in the scene, or by performing an exploratory small rotation to see if the depth is where you want it to be.

Refocus solves the following problem. Suppose you want to zoom in on some particular object in the scene and study it from more than one direction. Panning will bring the object to the centre of the window, and zooming will make it larger; but if you try to rotate it then the result may not be quite what you really wanted. Ideally, you would like the centre of the crystal ball to be in the middle of the object, so that rotating the scene makes the object rotate in place, as if you were holding it in your hand. However, it is possible (and quite likely) that the crystal ball is substantially in front of or behind the object of interest, in which case attempting to rotate it causes it to shoot off to one side. Refocus solves this problem by letting you adjust the depth of the crystal ball to be the same as that of the object you want to study.

Tips

1. If you are having trouble clicking on the sliders in the control panel then see bug 1
2. You can use mouse and keyboard commands simultaneously. For example, you can press the left mouse button to make the crystal ball visible, and then use the arrow keys (with the left mouse button still pressed) to pan an object of interest accurately to the centre of the crystal ball.
3. When the crystal ball is visible, zooming can be done using 'z' and 'Z', or ctrl-up/down-arrow.
4. You do not have to stop an animation simply to alter the speed, viewpoint, lighting, etc.: nearly all commands still work while an animation is playing.
5. Keyboard commands work in all three windows created by **showmotion** (main window, control panel and help window).
6. **Showmotion** does not use the numeric keypad. Use the arrow keys in the arrow-key cluster and the digits on the main keyboard.
7. If you have typed **showmotion(model,tout,qout)** to see the results of one simulation run, and then run Simulink again to get new data, then you have to re-issue the **showmotion** command to see the new data.

Command Summary

Keyboard Commands

space bar	play/pause toggle
0 (zero)	pause (if playing), rewind (if paused), skip to end (if rewound)
r	auto-repeat on/off
p	display/hide control panel (see bug 1)
h, ?	show the help window, which contains a shorter version of this command summary
comma, dot	increase/decrease animation speed
slash	backwards/forwards toggle
1 (one)	reset animation speed to +1
arrow keys	pan
shift-arrow keys	rotate (change viewing direction)
z, Z, ctrl-up/down	zoom
ctrl-left/right	rotate (spin about viewing direction)
i	restore initial view
l, L, o, O	decrease/increase intensity of camera headlight (l, L) and overhead light (o, O)
n, m, N, M, ctrl-n, ctrl-m	adjust time backwards/forwards in big (n, m), medium (N, M) and small (ctrl-n, ctrl-m) steps

Mouse Commands (main mindow)

left button	rotate using the crystal ball
shift-left button	pan
middle button	pan (UNIX)
left & right	pan (Windows)

buttons	
scroll wheel	refocus if crystal ball is visible, otherwise zoom
right button	pop-up menu with self-explanatory options

Control Panel Commands

triangle	play/pause toggle
square	pause (if playing), rewind (if paused), skip to end (if rewound)
loop icon	auto-repeat on/off
help	show the help window, which contains a shorter version of this command summary
two-arrow icon	backwards/forwards toggle
speed slider	adjust animation speed
time slider	coarse time adjustment
fine time slider	fine time adjustment

Bugs, Problems, Etc.

1. If you are using Matlab on a Linux machine, you may find that the place where you have to click on an icon or graphic in the control panel is about 5 pixels higher than where it appears on the screen. This is an intermittent bug, so your options are (1) put up with it, and aim high; (2) delete and recreate the control panel, using the keyboard command 'p', until you get one that works properly; or (3) just use the control panel as a status display.
2. Issuing a Matlab **clear all** or **clear global** command will clear the global variable used by **showmotion** to store its data. If this happens, then you will have to delete the **showmotion** windows manually before starting a new animation. (The main window will die as soon as you move the cursor over it.)
3. Certain viewing parameters cause the scene to vanish (i.e., the main window goes black). This bug is repeatable, but affects only a very small percentage of viewing parameters. If this happens to you, then the remedy is to move the camera a bit.
4. Drawing models with a lot of bodies is slower than it should be. For example, drawing a 100-link robot in which each body is drawn as a single box is significantly slower than drawing a 10-link robot in which each body is drawn as ten boxes.

Page last modified: June 2012
Author: Roy Featherstone