MouseoVeR:

A Virtual Reality Software Suite for the Laboratory

Jeremy D. Cohen, Mark A. Bolstad, and Albert K. Lee
Janelia Research Campus, Howard Hughes Medical Institute, Ashburn, VA, USA.

(JDC, MAB, and AKL designed the software; MAB programmed the software)

2017-07-11

Released with Cohen JD, Bolstad MA, Lee AK, eLife 2017:
"Experience-dependent shaping of hippocampal CA1 intracellular activity in novel and familiar environments"

Contents

Appendices

1. MouseoVeR

1.0 Introduction

MouseoVeR is a virtual reality software suite for the laboratory developed at the Howard Hughes Medical Institute (HHMI) Janelia Research Campus as part of Janelia's open-source virtual reality software platform (JOVIAN). MouseoVeR provides a user-defined and highly customizable virtual reality system in which external inputs drive navigation in the scene and relevant events are reported via continuous monitoring of activity. Virtual scene models are rendered in customizable display configurations that allow for fields of view up to 360°. In association with motion sensing inputs, (for example, position tracking of a treadmill), virtual scenes can be navigated. Each session's raw input data can be automatically stored to file, while a continuous data output stream provides information regarding current position, speed, orientation, and other user-defined information in the scene model.

1.1 Overview

MouseoVeR is a C++ application built from a number of open-source software components (Boost, Bullet, osgBullet, osgWorks, OpenSceneGraph, COLLADA, OpenGL, and Qt), making it capable of rendering 2-dimensional images and 3-dimensional (3-D) scene models. MouseoVeR can communicate with external software programs to receive inputs that affect the scene model (for example, to drive motion in the scene or to change scene properties), and send outputs that describe the current state of the system (Figure 1).

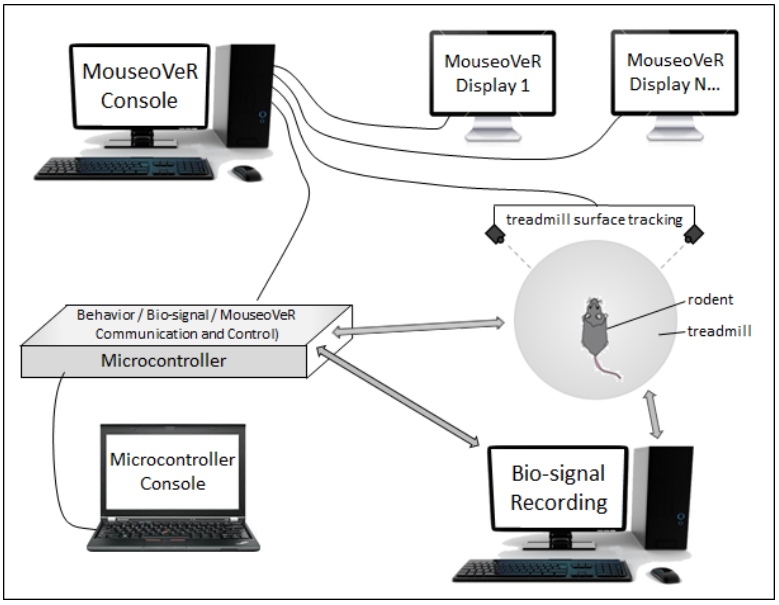


Figure 1. Overview of the MouseoVeR software suite integrated with an example virtual reality behavioral setup.

The hardware components overviewed in Figure 1 are more fully described in Appendix C – User Guide to MouseoVeR and the Virtual Reality Behavioral System: Electronic Devices and Basic Setup. In addition, a more complete description of the MouseoVeR software components and workflow can be found in Appendix D – JOVIAN and MouseoVeR Software.

1.2 Getting Started with MouseoVeR

1.2.1 Installation

To install, navigate to the MouseoVeR main directory, and follow the installation instructions located in README.txt:

…\eLIfe_release\Jovian\README.txt

After installation, to run MouseoVeR, locate MouseoVeR.exe in the folder:

C:\Program Files\Jovian\Release\bin\

To run the Remote Data Server (see section 1.3.1), locate RemoteDataServer.exe in the folder:

C:\Program Files\Jovian\Release\bin\

1.2.2 The Console GUI, Configuration File, and Initialization

The Console graphical user interface (GUI) provides access to all of the functionality of MouseoVeR. The configuration file (.cfg) uses JSON formatting, and contains a field for each parameter in the Console GUI. The .cfg file can also be manually edited by a text editor.

To load a pre-existing .cfg file, for example, the configuration file provided with MouseoVeR:

1) Select **File**, then **open config**.

2) Browse to **…\MouseoVeR\MouseoVeR_Configuration_Files** and select **JCVR_rodent_config.cfg**.

3) Click **Initialize** to apply all parameters and to initialize the graphics windows.

NOTE: In the **Setup** tab, clicking **Initialize** initializes the graphics windows and applies the configuration settings in the .cfg file (see sections 1.2.3 and 1.4.1-[6]). You must initialize the graphics windows before loading a graphics object, such as an image or a scene model.

4) To save the current configuration to file, at any time click **File**, then **Save** or **Save As**.

1.2.3 Displaying Images and 3-D Scene Models

MouseoVeR utilizes the open-source graphics engine OpenSceneGraph to render images to the displays. Blender (see Section 2) is a free open-source animation software application that can be used to create 3-D scene models. Scene models for MouseoVeR should be exported in COLLADA (.dae) format. (For more information on the COLLADA image format, see *https://www.khronos.org/collada/wiki/Main_page*). Exported .dae files can be edited in an advanced text editor such as NotePad++. MouseoVeR will load COLLADA files for display according to the properties of the objects in the scene (as described in Section 2) and according to the configuration file parameters. MouseoVeR can load many image formats for display; for example, .osg, .obj, .ply, .bmp, .tiff, .jpg, .tga, .svg, and .rbg. Several patterned .bmp images and 3-D .osg models are included with MouseoVeR, which can be useful for display calibration.

Example patterned images are provided in the MouseoVeR package, located at:

…\MouseoVeR\Visual_Test_Patterns\

Example scene models (.blend and associated exported .dae file) are located at:

…\MouseoVeR\Blender_scene_models\

1) **Load an image file**
   If desired, first load a configuration file (see section 1.2.2). In the **Console** dialog box, click the **Setup** tab. Under **Start Display With**, click **Image/Movie** (see section 1.4.1-[10]), and browse to or select the desired image file. Click **Initialize**, and the image will be displayed according to the parameters set in the configuration file. Users can click on the image in the display windows to rotate and/or zoom.

2) **Load a 3-D scene model (.dae)**
   If desired, first load a configuration file (see section 1.2.2). You must click **Initialize** to initialize the graphics windows (and also apply all parameters in the configuration file) prior to loading a scene model. After initialization, select **File**, then **Open**, and browse to the desired scene file. The scene model will be displayed according to the parameters set in the configuration file. Users can click on the image in the display windows to rotate and/or zoom.

3) **Change scene models from the console**
   In the **Setup** tab, under **Scene Files** select the desired scene model. The display will automatically update to the new scene. If multiple start objects (see section 2.1.2) are located in the scene, either the most recently used '_start_' location in the scene or the first location in the list of available start locations will be used.

   To change start locations (see section 2.1.2) within the same scene model from the console, click the **Setup** tab, then under **Start Locations**, select the desired '_start_' object from the available list, then click **Go to Location**.

1.3 Communicating with MouseoVeR

1.3.1 Input

There is one primary main input stream to MouseoVeR:

1) Remote Data Server (RDS)

The RDS (see Appendix D – D.1.7) is a stand-alone program that comes packaged with the MouseoVeR software suite. The RDS serves to continuously retrieve motion data from the treadmill tracking camera hardware (see Appendix C – C.1.1 and C.3), which can be used to drive motion in the virtual scene. The port selected for the RDS is set in the Console **Setup** tab, in the **Port** box. Port "22222" (a typically available port) is the default.

1.3.2 Output

There is one data output stream from MouseoVeR:

1) Data output stream via serial communication

When MouseoVeR is in the connected state, as set in Console **Setup** tab (see section 1.4.1-[12]), data describing the current state of the virtual camera (see section 2.1.1) in the scene model (current time relative to the time of clicking **Connect**, X-position, Y-position, Z-position, Speed, Heading Direction, and other user-defined events) is continuously printed out via serial port (set in the **Configuration** tab, under **I/O** (Input-Output Communication) by clicking the desired port from the **Output Port** list, or if undefined, is by default output to the Console terminal window).

The data output is an ASCII string comprised of comma-separated arguments. Each line of data corresponds to the activity of the virtual camera (see section 2.1.1) during (or at the end) of the previous frame. An example of a typical data line is:

3301, 24105, 42017, 200, 1615, 8556, 2, p1, p2/n/r/

This particular data line corresponds to:

Timestamp, X-position, Y-position, Z-position, Speed, Heading Direction, …Event Count, Event 1, Event 2

NOTE: Each data line is ended with "/n/r"

NOTE: Arguments 2-6 are multiplied by 100 for the data output stream only (NOT for internal use), and thus provide two decimals of precision represented as whole integer values. This ensures that the values in Arguments 2-6 are NOT required to be treated as floating-point numbers (typically with 8-decimel precision) across the software applications. To retrieve the actual values represented by Arguments 2-6 in the user-defined 'scene_unit' (that is, unit scale that is used for the Blender scene models, as well as that used for the treadmill radius value set in the Console **Configuration** tab, under **Ball Radius** (see section 1.4.5-[4])), divide by 100. For the example data line shown above, if the scene_unit is centimeters (cm), Arguments 1-6 should be read as:

3301 ms, 241.05 cm, 420.17 cm, 2.00 cm, 16.15 cm/s, 85.56 °, …

For a complete description of the data output stream format, see section 1.5.

1.4 Console Graphical User Interface (GUI) – Parameters and Functions

The Console graphical user interface (GUI) controls all of the functions available for MouseoVeR. Functions and parameters are grouped into six tabs: **Setup**, **Display**, **Lighting**, **Calibration**, **Configuration**, and **Heading Direction**.

1.4.1 Console – Setup

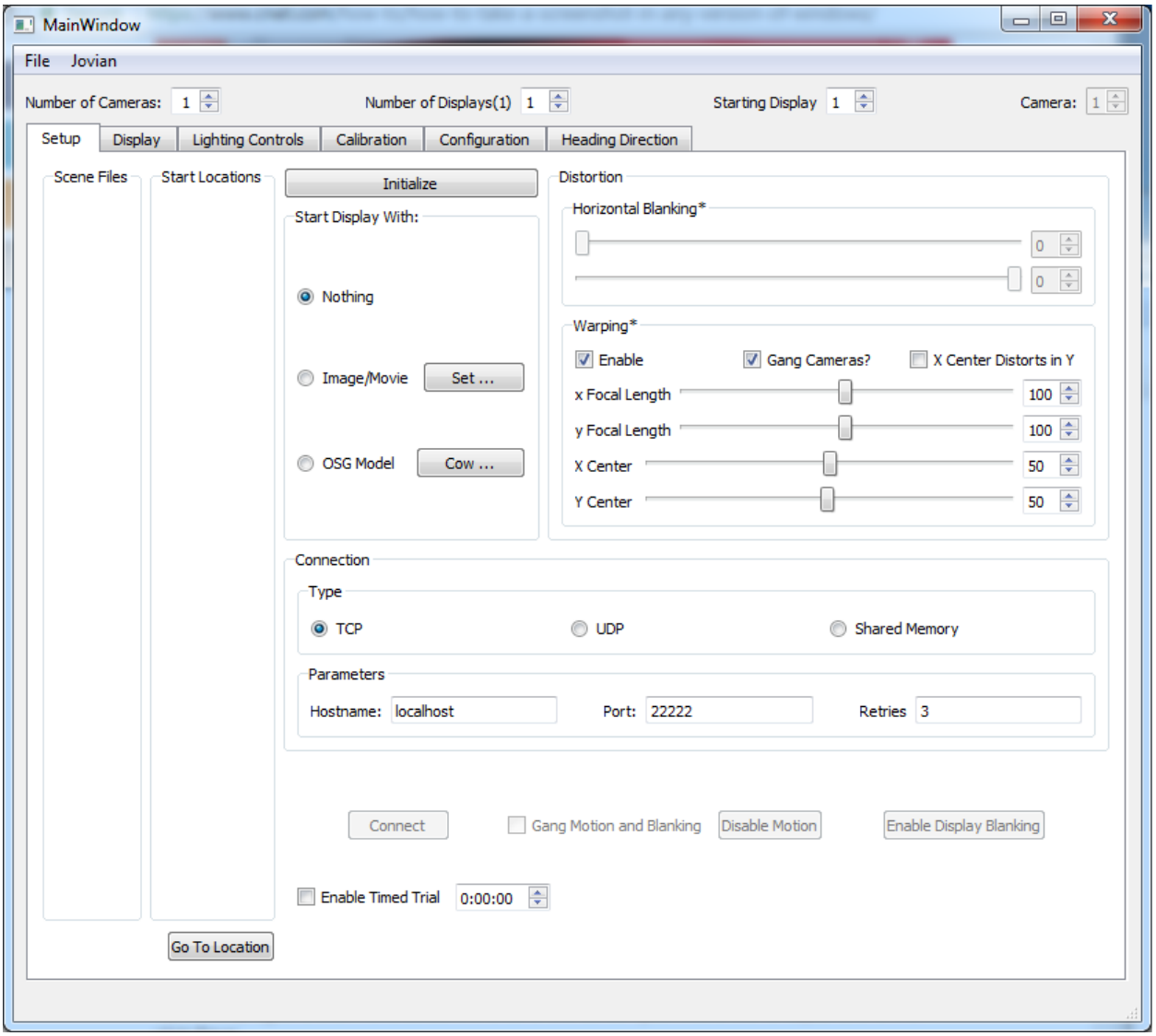


Figure 2. The Console – **Setup** tab. Parameters and features.

The **Setup** tab provides access to several essential parameters and features:

[1] Number of Cameras

In the **Number of Cameras** box, type or select the number of OpenSceneGraph (OSG) cameras in the scene to be used for capturing the final view of the scene model or image.

OSG camera objects can only capture information with a field of view (FOV) of <180°. Moreover, as the FOV for a single OSG camera approaches 180° (set in the **Configuration** tab by moving the **Field of View** slider, see section 1.4.5), the amount of image distortion at the edges of the FOV becomes undesirable. For cases where the final FOV of the scene is desired to be above approximately 90°, we recommend using multiple OSG cameras with a smaller FOV in order to capture the desired total FOV of the scene that is displayed across the desired **Number of Displays** (see [2]).

For example, if the desired FOV in the scene is 180°, we recommend using "3" OSG cameras, set to capture a FOV of 60° (with a **Camera Offset** amount that produces a contiguous image across the multiple displays, see section 1.4.5).

A simple configuration, as shown in Figure 2, uses a single OSG camera to capture a FOV of 135° (see section 1.4.5-[2]).

[2] Number of Displays

In the **Number of Displays** box, type or select the number of displays that are used for running MouseoVeR, which includes the optional additional display that contains the terminal window and Console GUI.

For example, in the simple configuration shown in Figure 2, there are **2** displays – one for the Console GUI and one for the scene. Typically, the **Number of Displays** value is set as "the number of physical displays that display the scene or image plus 1."

[3] Starting Display

In the **Starting Display** box, type or select the number of the first display that contains the scene or image.

For example, if using two displays for running MouseoVeR, with one being used for displaying the scene, set the **Starting Display** number to **2**. This will place the Console GUI on display 1, and the scene or image on display 2.

[4] Camera

In the **Camera** box, type or select the number of the specific OSG camera in the scene for which the user defines a camera-specific parameter.

For all OSG camera-modifying parameters accessible from the **Setup**, **Display**, and **Lighting Controls** tabs, the parameter values are only applied to the specified **Camera** number.

[5] Image/Movie
Under **Start Display With,** select **Image/Movie** to load images and scene models for display (see section 1.2.3).

[6] Initialize
Initialize the graphics windows and apply the current configuration settings (see section 1.2.3) *prior* to loading an image or a scene model file.

If desired, first load a configuration (.cfg) file, then click **Initialize**, then load the scene model.

[7] Port
RDS Communication port (see section 1.3.1 and Appendix D – D.1.7).

[8] Scene Files
The **Scene Files** area shows all scene models loaded (see section 1.2.3) and available for display.

[9] Start Locations
The **Start Locations** area shows, for each scene model, the list of available start locations (see section 2.1.2) to teleport to (see [11]).

[10] Start Display With
Select one of the options in the **Start Display With** area *prior* to initializing (see [6]) the graphics window objects (see section 1.2.3).

Click **Nothing** to initialize the displays (see [2]) with no image. This is the default.

Click **Image/Movie** to initialize the displays with an image file, such as patterned image for display calibration. Set the desired file by clicking **Set** and browsing the image file. Example patterned images are provided with MouseoVeR and can be located at:

…\MouseoVeR\Visual_Test_Patterns\

Click **OSG Model** to initialize the displays with an OSG 3-D model file. Set the desired file by clicking **Set** and browsing the image file. Example OSG models are provided with MouseoVeR and can be located at:

C:\Program Files\Jovian\OpenSceneGraph\

[11] Go To Location

To instantly change locations (teleport) within a given scene (see section 1.2.3), click the desired start location (see section 2.1.2) under **Start Locations** and click **Go To Location**.

To switch scene models, click the desired scene file name under **Scene Files.**

[12] Connect/Disconnect

Click **Connect/Disconnect** to connect or disconnect with the RemoteDataServer.

When connected, MouseoVeR's internal clock will be reset to 0, and the data output stream will be enabled (see sections 1.3.2 and 1.5). When disconnected, the data output stream will be disabled.

When switching scene models, but NOT start locations within a scene model, MouseoVeR's internal clock will be reset to 0.

[13] Gang Motion and Blanking

Select the **Gang Motion and Blanking** checkbox to group the actions of **Enable/Disable Motion** (see [14]) and **Enable/Disable Display Blanking** (see [15]) toggle buttons.

When unchecked, the two toggle buttons function independently.

[14] Enable/Disable Motion

Click on the **Enable/Disable Motion** toggle button to effectively enable or disable the treadmill motion from moving the camera in the scene.

Unlike **Connect/Disconnect** (see [12]), in this case the RDS will remain connected, thus continuing to report treadmill motion in the data output stream (see sections 1.3.2 and 1.5), and MouseoVeR's internal clock is NOT reset.

[15] Enable/Disable Display Blanking

Click on the **Enable/Disable Display Blanking** toggle button to effectively turn OFF/ON all of the displays used for displaying the scene or image. A black image is rendered across all scene displays when display blanking is enabled.

[16] Enable Timed Trial

The **Enable Timed Trial** checkbox enables simple versions of timed trials to be user-controlled from the Console GUI.

If the **Enable Timed Trial** checkbox is checked, then when **Connect** clicked, MouseoVeR will automatically disconnect (see [12]) and then teleport the camera back to initial start location (see section 2.1.2) after a fixed amount of time set in the edit box to the right.

1.4.2 Console – Display

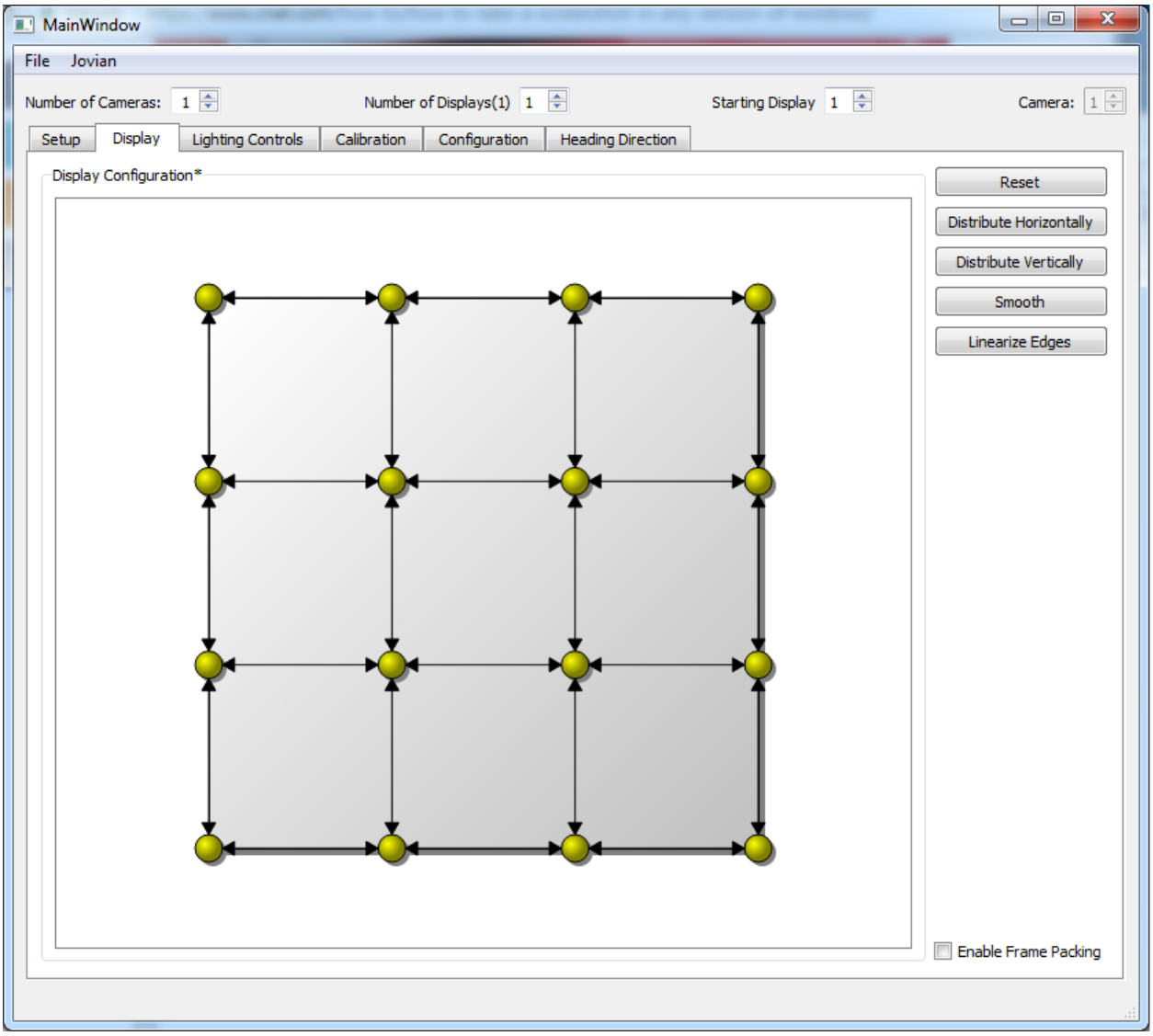


Figure 3. The Console – **Display** tab. Parameters and features.

The **Display** tab provides the user with a means to manually distort the displayed images:

[1] Display Configuration
Under **Display Configuration,** distort the images on the display(s) using the built-in Qt widget.

The display distortion widget contains control points for each display window, with the number of control points being defined in the edit box next to the widget. The control points are stored as a matrix of values in the configuration (.cfg) file.

For example, the distortion map shown for camera 1 in Figure 3 is configured for a "single projected image onto a toroidal display with 135° FOV" setup.

[2] Reset
Click **Reset** to reset (and thus unwarps) all control points to the default values.

[3] Distribute Horizontally
Click **Distribute Horizontally** to evenly space the control points in the horizontal plane (azimuth).

[4] Distribute Vertically
Click **Distribute Vertically** to evenly space the control points in the vertical plane (altitude).

[5] Smooth
The **Smooth** button performs an operation across all non-edge control points that serves to "equalize the length" (or "minimize the tension") between all of the lines surrounding the inner control points.

[6] Linearize Edges
Click **Linearize Edges** to take each pair of outer corner control points and make a new outer edge from a straight line between them.

Then, the two other control points on the outer edge between them are each moved to their nearest point on the new "linearized edge."

[7] Frame Packing Mode

Select the **Enable Frame Packing** checkbox to enable a special mode of MouseoVeR, commonly used in the "FlyoVeR" branch of JOVIAN (see Appendix D – D.1 and D.1.4). When enabled, instead of rendering three color frames (red, green, and blue) per physics update callback, we now perform the physics update callback on each frame render cycle, effectively increasing the displayed frame rate by three times. For example, if the color wheel is removed from the projector(s) used for the display(s), then MouseoVeR will render the scene model in a single grayscale channel, and fully operate at three times the projector frame rate.

NOTE: Depending on the number and type of physical interactions in the scene, the physics engine may not be able to keep up with desired frame rates. Therefore, it is not recommended to use this mode when running MouseoVeR in the more typical configuration described in this document.

1.4.3 Console – Lighting

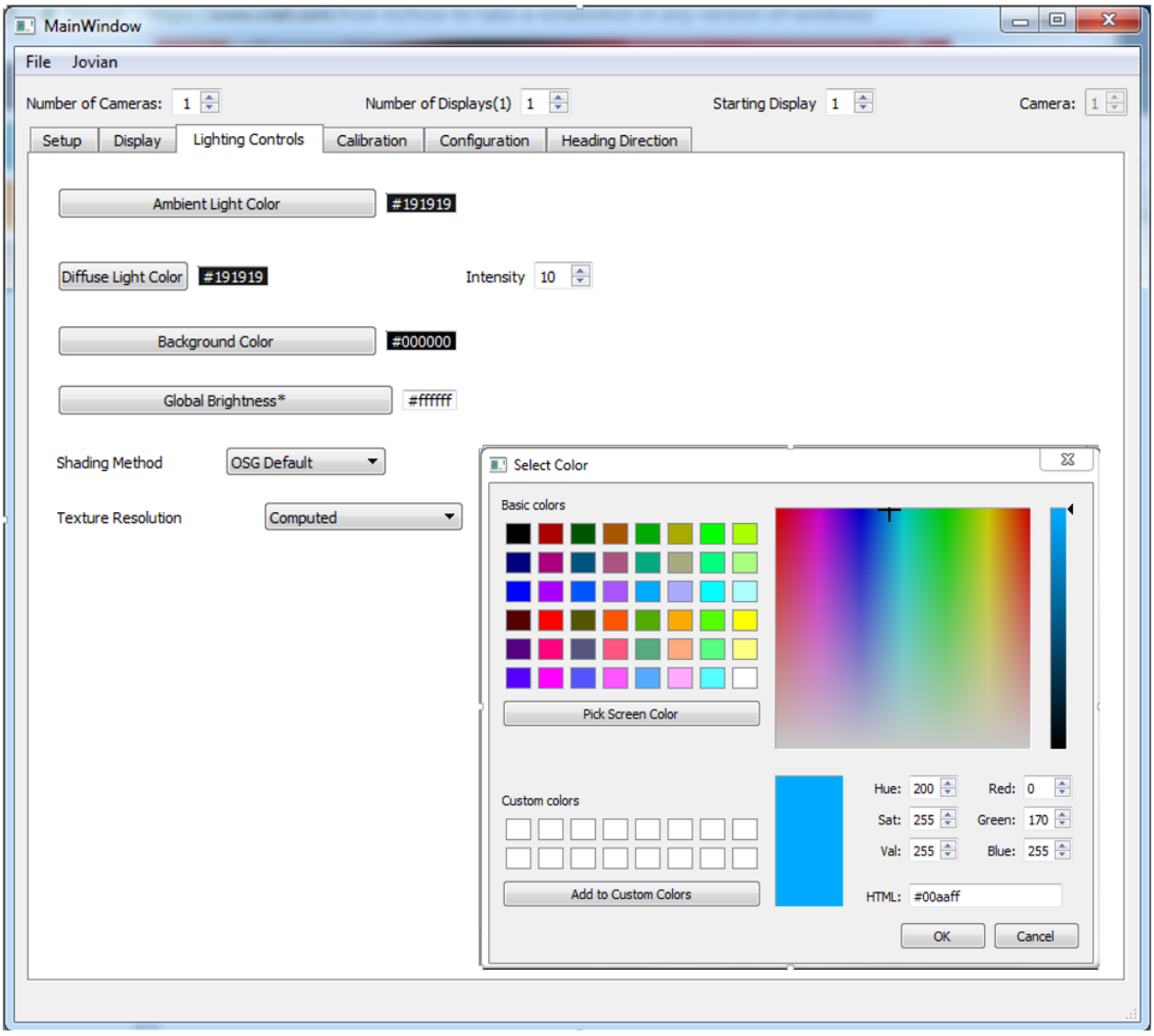


Figure 4. The Console – **Lighting** tab. Parameters and features.

Light in the scene is set by default to emit from five OpenSceneGraph (OSG) light objects that are placed at the top of the bounding box of the scene model; one light at each corner and one in the center. This configuration typically provides sufficient light in an approximately "10 square meter" or smaller scene model. However, if the scene model is much larger, or too many objects are located between the lights and the virtual camera object (see section 2.1.1) there may be a need for additional lighting modifications (see below).

Control the general lighting parameters for the rendered image or scene model using the following parameters:

[1] Ambient Light Color

Click **Ambient Light Color** to set the color of the light emitted by the lighting objects in the scene. Ambient lighting is heavily directional (that is, the rendered color depends on the interaction of the light with the object and the relative location of the object and the camera.

NOTE: Ambient lighting is only available for the following **Shading Method** options: **Per Vertex** and **Per Pixel** (see [6]). Ambient lighting is NOT functional when **OSG Default** is selected.

A typical configuration would use white (RGB = [255, 255, 255]) color.

[2] Diffuse Light Color

Diffuse light color acts as a multiplier to the color of every object in the scene model.

Click **Diffuse Light Color** to set the desired diffuse light color.

For example, if set at RGB = [51, 51, 51], the values are first converted from 0-255 to 0-1 (in this case [0.2, 0.2, 0.2]), and then each object's color is multiplied by the corresponding gain value. Diffuse lighting is heavily directional (that is, the rendered color depends on the interaction of the light with the object and the relative location of the object and the camera.

NOTE: Although this feature can be used to dim or brighten all objects, the result may not always be as intended, due to the directionality of the light. Diffuse lighting is available with all **Shading Method** options.

[3] Intensity

The intensity value is an integer multiplier to the color of every object in the scene model.

In the **Intensity** box, type or select the desired intensity value. The RGB color of all objects in the scene model will be multiplied by the intensity value.

NOTE: The RGB color values are capped at 255. For example, with an intensity of 3, an object with color RGB = [25, 50, 120], will be rendered as [75, 150, 255].

NOTE: Greater intensity values may be needed if the **Global Brightness** value (see [5]) is set to a value below RGB = [255, 255, 255]. If so, the brightness of the scene will be reduced. This is the primary software-based method to reduce the display brightness, which is useful for projector displays that do not have built-in brightness controls. If the brightness of the scene is reduced by, for example, RGB = [51, 51, 51], then the color of all objects in the scene will be reduced by the respective gain value (RGB = [0.2, 0.2, 0.2]). If an object has color RGB = [125 125 10], it will first be reduced to [25, 25, 2], then multiplied by the intensity value. In this example, we could use an intensity value of up to 10 before the object color begins to saturate in the red and green channels.

[4] Background Color

Click **Background Color** to set the color of the background in the scene model. This RGB color is applied to all regions of the scene that are not occupied by visible objects.

[5] Global Brightness

Every object's RGB color in the scene model is multiplied by the corresponding global brightness RGB values. Click **Global Brightness** to set the desired global brightness.

NOTE: Unlike ambient light color, global brightness is NOT directional, and thus will be applied evenly to all objects in the scene.

[6] Shading Methods

Three S**hading Method** options are available:

1) **OSG Default**
   OpenSceneGraph default light method. Ambient color features are not available.

2) **Per Vertex**
   All color features are available.

3) **Per Pixel**
   All color features are available. This is the most computationally expensive lighting method, and may affect overall performance.

Each method will produce a different style of lighting in the scene, and some lighting features may not be active for all **Shading Methods** options.

[7] Override Texture Resolution

The final displayed image ("texture") has a resolution, which is by default, automatically computed by MouseoVeR to be optimal for the given display when the **Computed** option is selected from the **Override Texture Resolution** list. However, in cases where a single projector is used to create a large size image (for example, via reflection off a spherical mirror), the size of each displayed pixel can be noticeably large. One method to compensate for this effect is to manually override the computed texture resolution by clicking a higher value number from the **Override Texture Resolution** list.

1.4.4 Console – Calibration

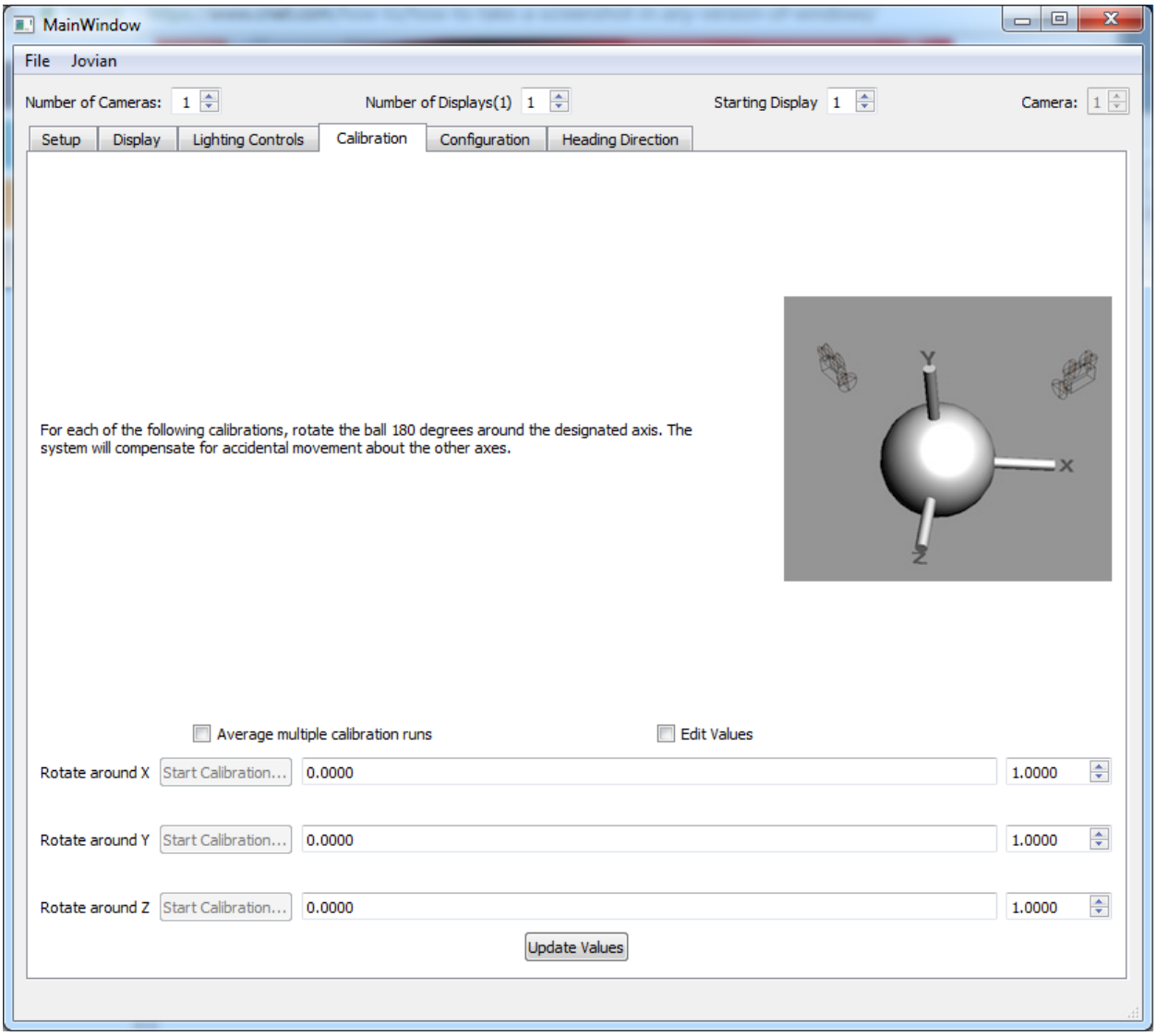


Figure 5. The Console – **Calibration** tab. Parameters and features.

In association with the treadmill tracking camera system (see Appendix C – C.3), the user can define how the raw motion of the spherical treadmill (see Section 3 and Appendix A) is converted into basic motion (see Appendix D – D.1.6 and D.1.6.1) in the scene model.

The conventions described in this document assume the configuration shown in Figure 5.1. The two treadmill tracking cameras should be located at the equator of the spherical treadmill, separated by 90°, with each camera being 45° off the Z-Axis centerline.

NOTE: If a fixed-position behavior system is used in association with the Trajectory-based method (see section 1.4.6) of manual-heading direction control (see section Appendix D – D.1.6 and D.1.6.3), the subject is assumed to be facing along the Z-Axis. The two cameras can be situated either in front or behind the subject's facing direction and will only affect the sign of the calibration value.

The three rotational axes are defined and operate as follows:

1) **X-Axis - Pitch**
   Forward-backward motion ("dollying," in filming terminology)

2) **Z-Axis - Roll**
   Side-to-side motion ("trucking," in filming terminology)

3) **Y-Axis – Yaw**
   Heading Direction/Turning motion ("panning," in filming terminology)

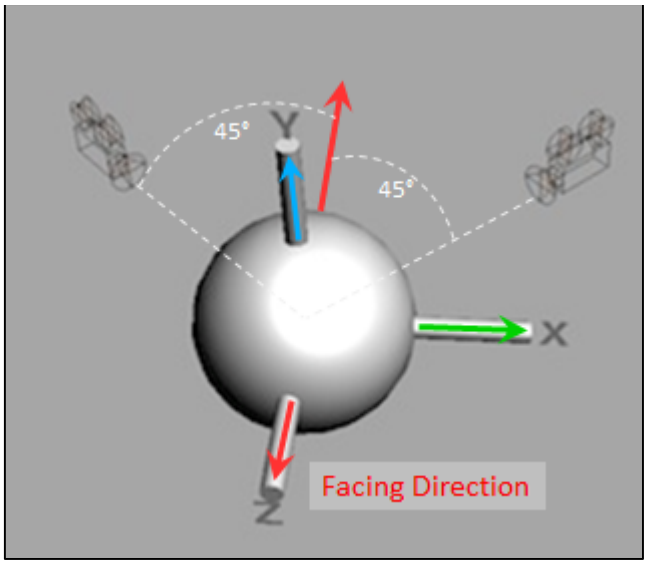


Figure 5.1. Configuration of the treadmill tracking camera system and the naming conventions used for the rotational axes (X-Axis: Pitch, Z-Axis: Roll, Y-Axis: Yaw) of the spherical treadmill.

Calibrating the spherical treadmill tracking camera system requires that the user create a mapping of the raw motion for each of the three rotational axes over 180-degrees of rotation. This procedure computes the number of camera image pixels moved (as determined in the treadmill tracking camera system (see Appendix C – C.3, and Appendix D – D.1.6) per 180-degree rotation.

For each axis of rotation, perform the following steps to run a single calibration trial:

1) Prepare to calibrate an axis of rotation by noting the initial position of the treadmill.

2) Click the **Start Calibration** toggle button to begin accumulating pixel displacement values associated with the rotation of the treadmill about the axis being calibrated.

3) Rotate the treadmill 180-degrees about the axis being calibrated.

   NOTE: The system will compensate for motion about the other axes.

4) Click the **Stop Calibration** toggle button to stop accumulating values.

The number of camera pixels of motion associated with the 180-degree rotation will be displayed in the corresponding edit box.

NOTE: If the **Average Multiple Calibration Trials** checkbox is checked, the user can use multiple 180-degree rotations to create an average calibration trial, derived from the individual calibration trials. We recommend using several calibration trials to create an average calibration value. Between each trial, click **Start Calibration**. MouseoVeR will automatically create the average as additional trials are collected.

NOTE: The calibration values can also be edited manually, when the E**dit Values** checkbox is checked.

Calibration of Gain Values

For each axis of rotation, in the corresponding spin box, the user can select or type a multiplier gain value that is applied to the calibrated raw motion inputs. The example configuration shown in Figure 5 applies the default gain value of 1 to the X-, Y- and Z- axes. To prevent any basic yaw-based heading direction control (see Appendix D – D.1.6 and D.1.6.1) in the scene, apply a gain of 0.0001 (effectively 0) to the Y-axis (yaw) motion.

This example configuration (X=1, Z=1, Y=0) is useful for enabling the following configurations:

1) The virtual camera (see section 2.1.1) heading direction remains fixed, while still able to move forwards, backwards and side to side.

2) In association with the second mode of manual heading direction control, trajectory-based heading (see section 1.4.6 and Appendix D – D.1.6.3), or with the auto-heading method path-based heading (see Appendix D – D.1.6.4), the subject can control heading direction using non yaw-based means.

1.4.5 Console – Configuration

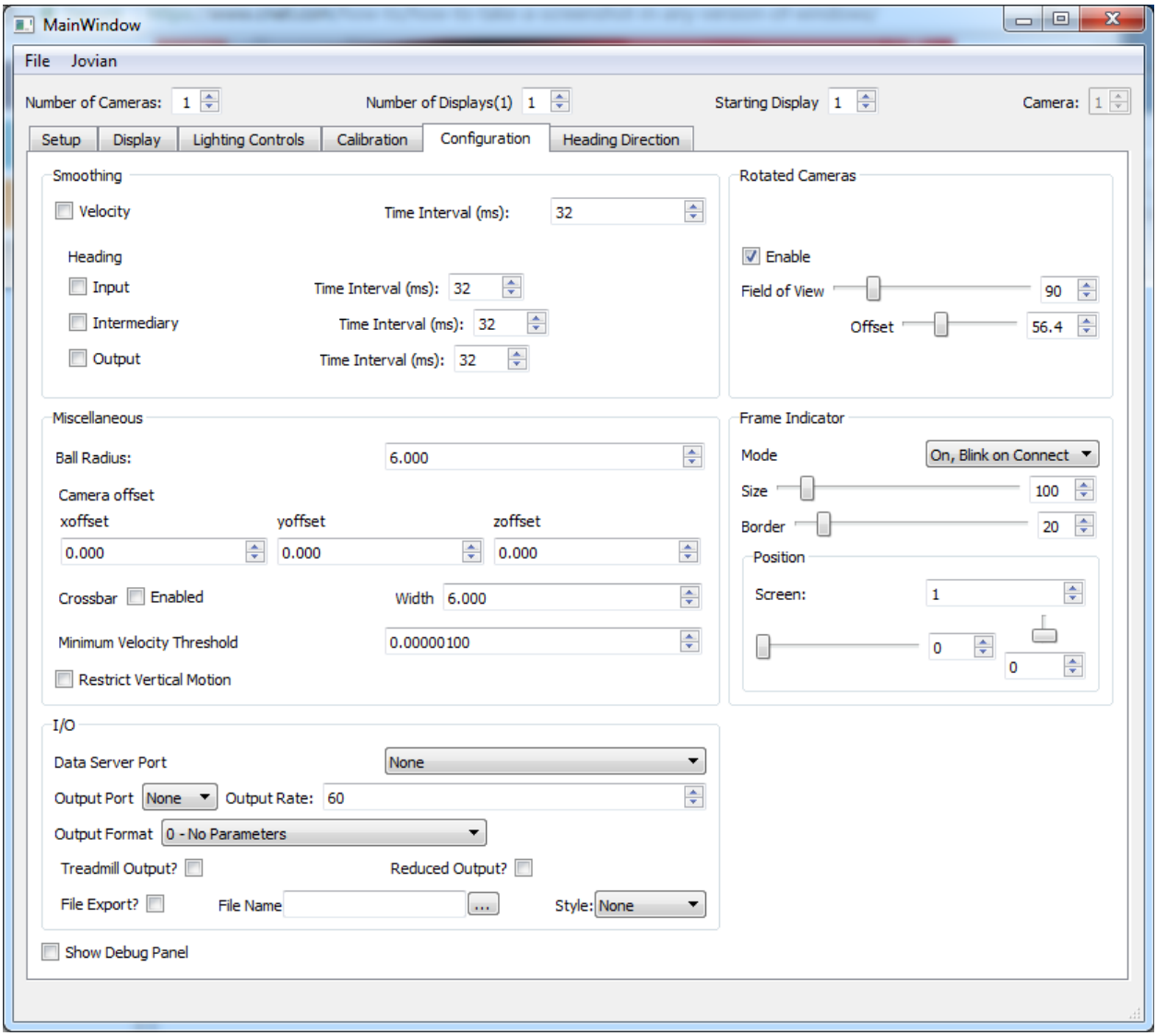


Figure 6. The Console – **Configuration** tab. Parameters and features.

The **Configuration** tab provides access to many parameters for the general configuration of motion, display, and communication. From the **Configuration** tab, the user can enable/disable parameters, and decide how much of a particular operation is performed. The parameters and associated functions are described below:

[1] Motion Smoothing
NOTE: For all **Smoothing** operations, **Time Interval (ms)** values of less than two frame windows in duration will result in NO smoothing.

1) Input Euler Angles
Independently smooth the three raw input Euler angles over the time window set in the corresponding edit box.

2) Velocity
After step (1), we independently perform velocity smoothing on the three Euler angles over the time window set in the corresponding edit box.

3) Trajectory-based Heading Direction
After step 2), we smooth the computed Running Trajectory (see section 1.4.6 and Appendix D- D.1.6.3) over the time window set in the corresponding edit box.

4) Auto + Manual Heading Direction
After step (3), we smooth the resultant of the Auto + Manual-heading components (further described in Appendix D – D.1.6.3), over the time window set in the corresponding edit box.

[2] Camera Field of View

1) Field of View
For all OSG cameras (see section 1.4.1-[2]), set the camera field of view (FOV) in degrees by moving the **Field of View** slider or by typing or selecting the desired FOV in the spin box.

2) Offset
If the **Use Rotated Cameras** checkbox is checked, all cameras other than camera_1 will be rotated (and thus have a heading direction offset from the center camera_1) accordingly to the value in the associated edit box, in degrees. This allows for configurations in which, for example, three cameras each with a FOV of 75°, and an appropriate **Offset** value , can create a total azimuthal FOV of 225°.

[3] Frame Indicator
The frame indicator is a square (of user-defined size and position) that alternates per frame as white and black pixels. If enabled, the box will effectively flash on/off as a marker of the actual displayed image. For example, the frame indicator can be detected with a photodetector (such as Thorlabs Inc. part 'DET36A' or 'DET10C').

1) Frame Indicator
In the **Frame Indicator** area, click the **On/Off** button to enable the frame indicator.

2) Size
To set the size of the square in pixels (N x N), move the **Size** slider or type or select the desired value in the spin box.

3) Border
To set the size of a black border around the frame indicator square in pixels, move the **Border** slider or type or select the desired value in the spin box. Incorporating a black box around the flashing frame indicator may aid the photodetection process.

4) Position
To set the location of the frame indicator in pixels, type or select the desired screen number in the **Screen** box. The **Screen** number refers to the **Camera** number relative to the **Starting Display** number (see section 1.4.1-[2], [4]).

[4] Miscellaneous

1) Ball Radius
In the **Ball Radius** box, type or select the radius of the spherical treadmill in 'scene_units'. For the example shown in Figure 6, the radius is set to 19.76 cm for a 15.5625 inch diameter treadmill.

2) Camera Offset
X-offset: In the **xoffset** box, type or select the forward/backward displacement of the camera relative to the default position centered on top of the '_camera_block_pm_' object (see section 2.1.1).

Y-offset: In the **yoffset** box, type or select the vertical displacement of the camera relative to the default position centered on top of the '_camera_block_pm_' object (see section 2.1.1).

Z-offset: In the **zoffset** box, type or select the sideways displacement of the camera relative to the default position centered on top of the '_camera_block_pm_' object (see section 2.1.1).

3) Crossbar
In the **Crossbar** area, if the **Enabled** checkbox is selected, the width of the crossbar object (see section 2.2.2) can be edited by typing or selecting a value in the **Width** box.
NOTE: If a crossbar object is present in the scene, the value in the **Width** box will correspond with the width of the object. If there is no crossbar object in the scene, the user can type or select a width value in the **Width** box to create a crossbar value, serving to restrain motion perpendicular to the path (see sections 2.2.1 and 2.2.2).

4) **Minimum Velocity Threshold** (for Trajectory-based Heading Direction)
To set the minimum velocity threshold, the minimum velocity required for motion, type or select a value in the **Minimum Velocity Threshold** box.

For example, to exclude the effects from spurious motion in place, such as grooming, or slight shifts while stopped, only motion above the threshold value will be sent to the physics engine to compute motion in the scene.

The trajectory-based heading direction input/output function (see section 1.4.6 and Appendix D – D.1.6.3) will only be enabled if the treadmill velocity is greater than the value in the corresponding edit box (in 'scene_units').

5) **Restrict Vertical Motion**
If the R**estrict Vertical Motion** checkbox is checked, the virtual camera object (see section 2.1.1) will experience a large gravitational force in the downward (negative in the Y-axis) direction, thereby serving to prevent upwards motion of camera as it collides with objects in the scene.

[5] **I/O** (Input/Output Communication)

1) **Data Server Port**
Click to view **Data Server Port** options and select the serial port for communication with the RemoteDataServer (RDS) (see section 1.3.1 and Appendix D – D.1.7).

2) **Output Port**
Click to view **Output Port** options and select the serial port for the data output stream (see sections 1.3.2 and 1.5).

3) **Output Rate**
In the **Output Rate** box, type or select the rate of the output data stream. The default rate is set to the frame rate of the scene/image displays.

4) **Output Format**
Click to view **Output Format** options (0, 1, or 2) and select the desired output format (see section 1.5 – Argument 7).

5) **Treadmill Output**
Select the **Treadmill Output** checkbox to store data lines to file at the data rate set in the **Output Rate** box in the following format:

Timestamp, cam_1 X-motion (dx), cam_1 Y-motion (dx), cam_2 dx, cam_2 dy, output format, etc…

6) **Reduced Output**
Select the **Reduced Output** checkbox to reduce the length of the data stored to file. If checked, the format of the data lines in the output file will be the following:

Timestamp, cam_1 X-motion (dx), cam_1 Y-motion (dx), cam_2 dx, cam_2 dy

7) **File Export**
Select the **File Export** checkbox to automatically store the data output stream to file (see sections 1.3.2 and 1.5) for each instance the RDS is connected (see section 1.4.1-[12]).

Select the **Treadmill Output** checkbox to include the raw treadmill input data to the exported file (see [5]).

8) **File Name**
The string entered into the **File Name** box will be appended to the beginning of the filename of the exported file. The remainder of the name will be automatically generated by MouseoVeR to include the name of the S**cene File** (see section 1.4.1-[8]) and other information, such as the current timestamp, that the user can select with the **Style** button.

9) **Style**
Click **Style** to view and select the type of information (e.g., current time stamp, counter) to append to the exported filename.

[6] Show Debug Panel
When the **Show Debug Panel** checkbox is checked, the hidden **Debugging** area will appear (Figure 6.1). Under **Debugging**, options provide the ability to display invisible or hidden objects in the scene, such as all of the user-defined invisible objects (see section 2.0.1), and other information showing some basic physics engine parameters controlling motion.

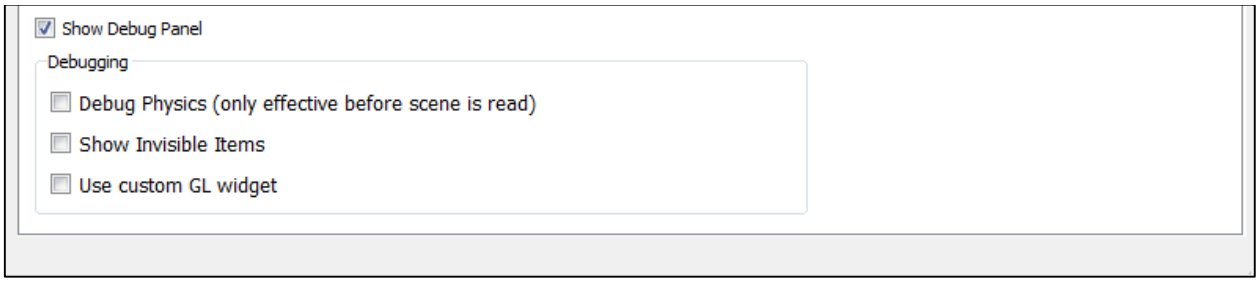


Figure 6.1. The Console – **Configuration** tab – **Show Debug** Panel. A hidden area which is made accessible by checking the **Show Debug Panel** checkbox.

1.4.6 Console – Heading Direction

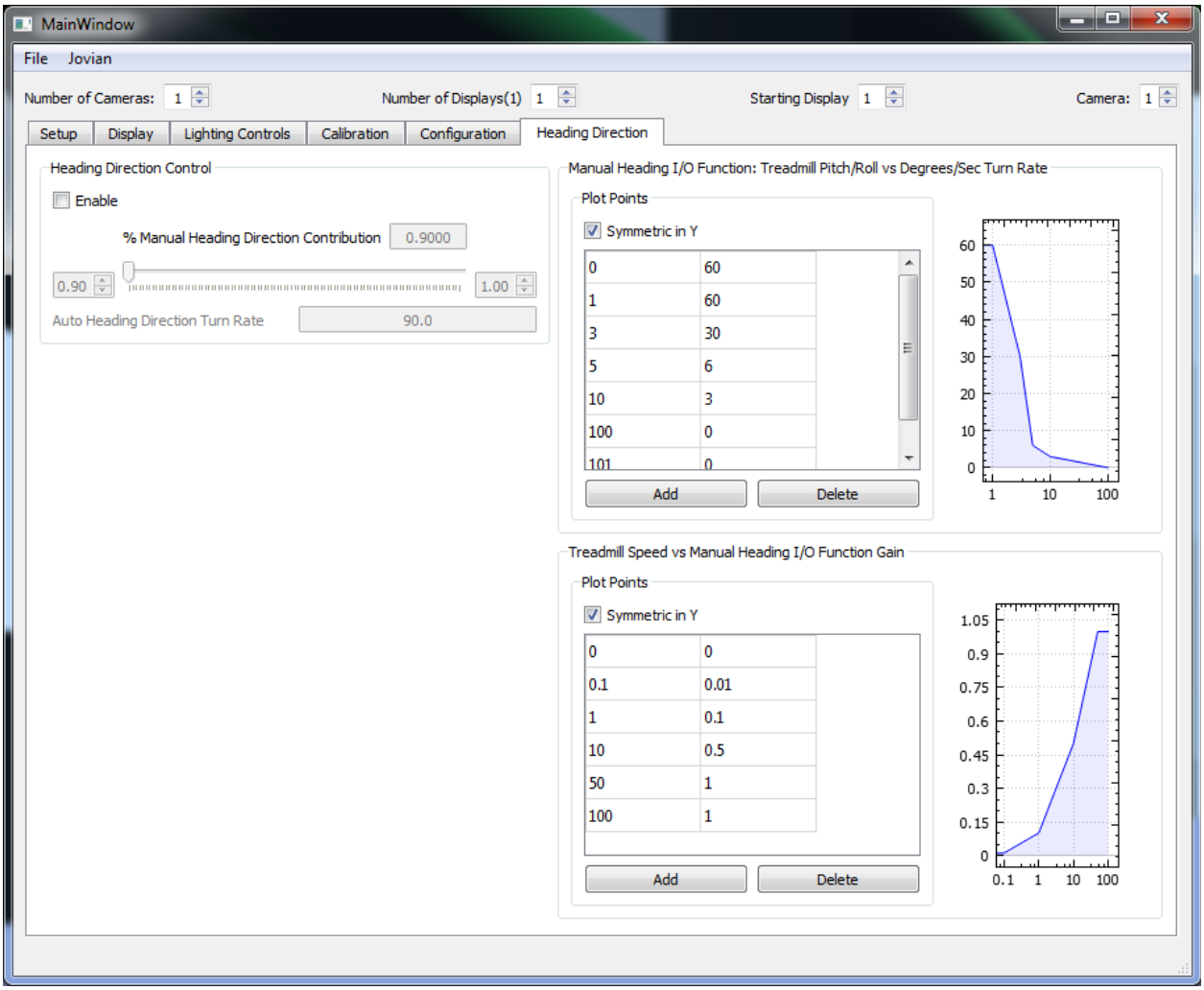


Figure 7. The Console – **Heading Direction** tab. Parameters and features.

The **Heading Direction** tab provides access to all of the parameters that define the Trajectory-based mode of manual heading direction control (Figure 7.1), as fully described in Appendix D – D.1.6 and D.1.6.3.

[1] Trajectory-based Heading

Check the **Trajectory Heading Direction** checkbox to enable or disable this feature. When enabled, the ratio of the spherical treadmill's raw pitch versus roll motion values (Running Trajectory) is first computed, then an angular rate of rotation about the Y-axis (yaw) is determined by a user-defined lookup table, set in the table under **Heading Direction Input-Output Function**. The lookup table can be edited at any time.

NOTE: Settings will be applied automatically. To store all current configuration settings, save the Configuration (.cfg) file.

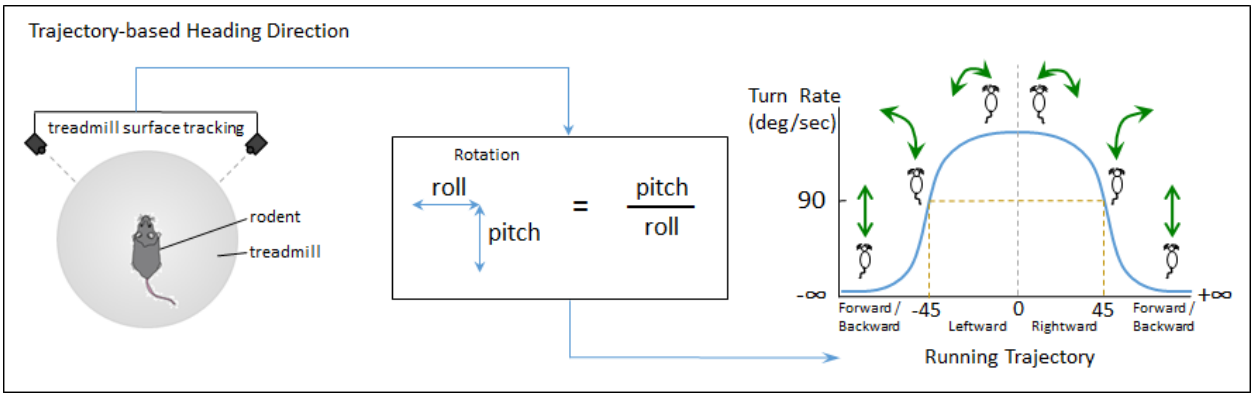


Figure 7.1. Trajectory-based heading direction control. The ratio of the spherical treadmill motion in the pitch versus roll axes (Running Trajectory) is converted into a rate of rotation (Turn Rate) about the Y-axis (yaw) in degrees per second. The user sets the Input-Output function that determines the conversion of running trajectories into heading direction turn rates.

[2] Heading Direction Input-Output Function

The **Heading Direction Input-Output Function** area contains a plotting widget that displays three types of information:

1) The Running Trajectory-to-Turn Rate input-output function, derived from the values in the associated lookup table on the left.

2) When in the connected state (see section 1.4.1- [12]), the current raw treadmill Running Trajectory (pitch-versus-roll ratio) before any smoothing has been applied, is shown as a black vertical line.

3) When in the connected state, the current smoothed Running Trajectory (as defined by the third smoothing value located in the **Configuration** tab, under **Smoothing**, see section 1.4.5-[1]) after the second heading direction smoothing operation has been applied, is shown as a green vertical line.

[3] Speed versus Heading Direction Gain

This panel provides the user with a means to dynamically control the magnitude of the Turn Rate function (that is, the amount of heading direction change in degrees per second) based on the speed of the treadmill. This function was designed to allow for different rates of turning when rotation of the treadmill at the same running trajectory occurs at different speeds. For example, using the default settings provided in the example configuration file JCVR_rodent_config.cfg, if the subject is running on the treadmill at relatively high speeds, this function permits the maximum magnitude of Turn Rate, as defined in the upper panel. In contrast, at slower speeds, such as those

which may be occurring during more stationary behaviors and when performing slow and sideways treadmill motion (that is, low pitch-to-roll ratios), this function will globally reduce the magnitude of the Running Trajectory-to-Turn Rate function, serving to prevent large and rapid changes in heading direction.

Together, the default settings in the configuration file JCVR_rodent_config.cfg permit high Turn Rates when moving above relatively moderate speeds (> 10 cm/s), yet prevent high Turn Rates at relatively slow running speeds (< 4 cm/s). These settings allow for rapid control of heading direction during navigation across large distances, while reducing extremely rapid Turn Rates during slower behaviors that are not accompanied by larger changes in distance.

1.5 Data Output – Format and Description of the Arguments

This section describes the format of the data output stream, which occurs via serial port (the **Output Port** selected in the **Configuration** tab, under **I/O** (Input/Output Communication)). See section 1.3.2 for an introduction to the data output stream.

An example of a typical data line is as follows:

3301, 24105, 42017, 200, 1615, 8556, 0, 2, p1, p2/n/r/

NOTE: Each data line is ended with "/n/r"

NOTE: Arguments 2-6 are multiplied by 100 for the data output stream only (NOT for internal use), and thus provide two decimals of precision represented as whole integer values. This ensures that the values in Arguments 2-6 are NOT required to be treated as floating-point numbers (typically with 8-decimel precision) across the software applications. To retrieve the actual values represented by Arguments 2-6 in the user-defined 'scene_unit' (that is, the unit scale used for the Blender scene models, as well as that used for the treadmill radius value set in the Console **Calibration** tab, under **Ball Radius** (see section 1.4.5-[4])), divide by 100. For the example data line shown above, if the scene_unit is centimeters (cm), Arguments 1-6 should be read as:

3301 ms, 241.05 cm, 420.17 cm, 2.00 cm, 16.15 cm/s, 85.56 °, …

This particular data line corresponds to:

- **Argument 1: Timestamp**
  Timestamp at the moment of rendering the previous frame. Upon being connected or switching scene models, MouseoVeR's internal clock is reset to 0.

- **Argument 2: X position**
  X position (in scene_unit*100) at the moment of rendering of the previous frame.

- **Argument 3: Y position**
  Y position (in scene_unit*100) at the moment of rendering of the previous frame.

- **Argument 4: Z position**
  Z position (in scene_unit*100) at the moment of rendering of the previous frame.

- **Argument 5: Speed**
  Speed (scene_unit/s*100) of the virtual camera (see section 2.1.1) in the scene model at the moment of rendering of the previous frame. This speed value is the final computed speed in the scene model, which is based on all parameters set in the configuration file, and the interaction of the virtual camera with objects in the scene (for example, with walls). The speed value is the absolute difference in virtual camera position over time. Divide by 100 to

retrieve actual value in scene_unit/s.

- **Argument 6: Heading Direction**
  Heading Direction (degrees*100) of the virtual camera (see sections 1.4.6 and 2.1.1) in the scene model at the moment of rendering of the previous frame. This heading direction value is the final computed heading direction in the scene model relative to due east (0°), which is based on all parameters set in the configuration file, and the interaction of the virtual camera with objects in the scene (for example, with walls). The heading direction value is the absolute heading direction in the scene. Divide by 100 to retrieve actual value in degrees.

  NOTE: East = 0°, North = 90°, West = +/- 180°, South = -90°

- **Argument 7: Format Tag for Additional Arguments**
  This argument contains a tag that defines the format of optional additional arguments. Currently, there are two Format Tags: "0" or "1". The option is set in the **Configuration** tab, under **Output Format** (see section 1.4.5-[5]).
  - **IF Argument 7 = 0**
    For example:

    3301, 24105, 42017, 200, 1615, 8556, **0**, 2, p1, p2/n/r/

    - **Argument 8: Number of Events**
      The number of physical contact events between the virtual camera object (see section 2.1.1) and other user-defined objects (see section 2.0.5) that occurred during the previous frame.
    - **Argument 9: IF Argument 8 > 0, THEN Argument 9 = Event 1**
      For example, if the camera object contacts an object with the name 'p1' (as defined by the rules outlined in section 2.0.5-2.0.10), 'Event 1' = 'p1'.
    - **Argument 10-n: IF Argument 8 > 1, THEN Argument 10-n = Event 2 – Event n…**
      For example, if the camera object contacts two objects with names 'p1' and 'p2', 'Event 1' = 'p1' and 'Event 2' = 'p2', etc…

  - **IF Argument 7=1**
    For example:

    3301, 24105, 42017, 200, 1615, 8556, **1**, 2343, 3445, 2, p1, p2/n/r/

    - **Argument 8: Raw treadmill speed**
      This speed value is the length of resultant motion vector based on the accumulated rotation of treadmill over the previous frame (in scene_unit/s*100). This value is what is fed into the physics engine (Bullet), which is used to drive motion in the scene. Divide by 100 to retrieve actual value in scene_unit/s.

- **Argument 9: Raw treadmill heading direction**
  This heading direction value is the angle of the resultant motion vector based on the accumulated rotation of treadmill over the previous frame (in degrees*100) based on the coordinate scheme described below. This is what is fed into the physics engine (Bullet), which is used to drive motion in the scene. Divide by 100 to retrieve actual value in degrees.

  NOTE: East = 0°, North = 90°West = +/- 180°, South = -90°

- **Argument 10: Number of Events**
  The number of physical contact events between the virtual camera object (see section 2.1.1) and other user-defined objects (see sections 2.0.5 - 2.0.10) that occurred during the previous frame.
- **Argument 11: IF Argument 10 > 0, THEN Argument 11 = Event 1**
  For example, if the camera object contacts an object with the name 'p1' (as defined by the rules outlined in sections 2.0.5-2.0.10), 'Event 1' = 'p1'.
- **Argument 12-n: IF Argument 10 > 1, THEN Argument 12-n = Event 2 – Event n…**
  For example, if the camera object contacts two objects with names 'p1' and 'p2', 'Event 1' = 'p1' and 'Event 2' = 'p2', etc...

- **IF Argument 7 =2**
  For example:

  3301, 24105, 42017, 200, 1615, 8556, **2**, 2343, 14260, 7801, 1120, … 2, p1, p2/n/r/

  - **Argument 8: Raw treadmill speed**
    This speed value is the length of resultant motion vector based on the accumulated rotation of treadmill over the previous frame (in scene_unit/s*100). This value is what is fed into the physics engine (Bullet), which is used to drive motion in the scene. Divide by 100 to retrieve actual value in scene_unit/s.
  - **Argument 9: Raw treadmill Z-Axis, ROLL**
    This heading direction value is the component of the raw treadmill motion vector about the Z-axis, ROLL
  - **Argument 10: Raw treadmill X-Axis, PITCH**
    This heading direction value is the component of the raw treadmill motion vector about the X-axis, PITCH
  - **Argument 11: Raw treadmill Y-Axis, YAW**
    This heading direction value is the component of the raw treadmill motion vector about the Y-axis, YAW
  - **Argument 12: Number of Events**
    The number of physical contact events between the virtual camera object (see section 2.1.1) and other user-defined objects (see sections 2.0.5 - 2.0.10) that occurred during the previous frame.
  - **Argument 13: IF Argument 12 > 0, THEN Argument 13 = Event 1**
    For example, if the camera object contacts an object with the name 'p1' (as defined by the rules outlined in sections 2.0.5-2.0.10), 'Event 1' = 'p1'.

- **Argument 14-n: IF Argument 12 > 1, THEN Argument 14-n = Event 2 – Event n…**
  For example, if the camera object contacts two objects with names ‘p1’ and ‘p2’, ‘Event 1’ = ‘p1’ and ‘Event 2’ = ‘p2’, etc..

2. Blender - Creating 3-D Scene Models for MouseoVeR

Blender is a multi-platform, free open-source 3-dimensional (3-D) animation software application that can be used to create the 3-D scene models displayed by MouseoVeR.

Official website: *www.blender.org*
Blender Manual: *https://www.blender.org/manual*
Blender hotkeys: *http://en.wikibooks.org/wiki/Blender_3D:_HotKeys/All*

2.0 Naming Codes and Associated Functions

When creating a scene model in Blender for MouseoVeR, special naming conventions must be followed in order to correctly parse the exported file and utilize the full functionality afforded by MouseoVeR.

Various characters appended to the names of objects allow them to carry specific functions or properties in MouseoVeR. Together, they provide the user with an array of optional features that define how the virtual camera (see section 2.1.1) behaves as it interacts with objects in the scene. Using these conventions, we can create scenes with visible and invisible objects that may or may not interact with the camera. Physical contact events between the camera and special objects (see sections 2.0.5 – 2.0.10) in the scene can also be reported in MouseoVeR's data output stream (see section 1.3.2), serving as markers of location or special events.

The naming conventions and associated functions are outlined in sections below:

*2.0.1 _name_*

If an object name in the scene is flanked by underscores, it will be invisible when displayed by MouseoVeR.

NOTE: All Invisible objects in the scene can be made visible by checking the **Show Debug Panel** checkbox in the Console **Configuration** tab, then selecting the **Show Invisible Items** checkbox in the hidden **Debugging** area (see section 1.4.5-[6]).

2.0.2 name_p

If an object name ends in '_p', the object will be physics enabled. That is, 'name_p' objects will interact physically during collisions with the virtual camera object (see section 2.1.1). By default, the 'name_p' object will be (i)mmovable. For example, these objects can be used for creating visible barriers, such as walls.

Objects without the '_p' tag will not be physics enabled and thus, will not interact with the camera object. The camera will traverse through objects that are not physics enabled.

*2.0.3 _name_p_*

'_name_p_' objects will be physics enabled and invisible. By default, the 'name_p_' object will be (i)mmovable. For example, these objects can be used for creating invisible barriers, such as guide walls for constraining motion within a larger area.

2.0.4 name_pm

'name_pm' objects will be physics enabled and movable. That is, the virtual camera object (see section 2.1.1) will interact with the movable object in complex ways that depend on the properties (e.g., location, size and shape) of all objects in the physical interaction (collision).

2.0.5 name_pr

'name_pr' objects will be physics enabled and reportable. That is, when the virtual camera object (see section 2.1.1) contacts the object 'name_pr', the 'name' portion of the object's name will be reported in the next data output line (see section 1.3.2). By default, 'name_pr' objects will NOT interact physically with the camera. The camera will traverse through objects with this tag, and will report the contact events.

For example, similar to the use of visible landmarks in real-world environments, these visible objects can be used for reporting special locations of the camera in the scene model.

*2.0.6 _name_pr_*

'_name_pr_' objects will be physics enabled, reportable, and invisible. That is, when the virtual camera object (see section 2.1.1) contacts the invisible object '_name_pr_', the 'name' portion of the object's name will be reported in the next data output line (see section 1.3.2). By default, '_name_pr_' objects will NOT interact physically with the camera. The camera will traverse through objects with this tag, and will report the contact events in the data output stream (see section 1.3.2).

For example, similar to the use of infrared photobeams in real-world environments, these invisible objects can be used for reporting special locations of the camera in the scene model.

2.0.7 name_pir

Similar to 'name_pr' objects (see section 2.0.5), the 'name_pir' object will be physics enabled, immovable, and will report collisions with the virtual camera (see section 2.1.1) in the data output stream (see section 1.3.2). Unlike '_pr' objects (sections 2.0.5 and 2.0.6), the camera WILL physically interact with the immovable '_pir' object.

2.0.8 name_pmr

'name_pmr' objects will be physics enabled, movable, and will report collisions with the virtual camera (see section 2.1.1) in the data output stream (see section 1.3.2). These objects WILL interact physically with the camera. Similar to 'name_pm' objects (see section 2.0.4), the camera will interact with the movable object in complex ways that depend on the properties (e.g., location, size and shape) of all objects in the collision.

*2.0.9 _name_pir_*

Like 'name_pir' objects, (see section 2.0.7), '_name_pir_' objects will be physics enabled, immovable, and will report collisions with the virtual camera (see section 2.1.1) in the data output stream (see section 1.3.2). '_name_pir_' objects will be invisible.

*2.0.10 _name_pmr_*

'_name_pmr_' objects will be physics enabled, movable, invisible, and will report collisions with the virtual camera (see section 2.1.1) in the data output stream (see section 1.3.2). Similar to 'name_pm' objects (see section 2.0.4), the camera will interact with the invisible and movable object in complex ways that depend on the properties (e.g., location, size and shape) of all objects in the collision.

2.0.11 name_pic

The '_pic' tag may be required for correct motion computation when creating complex shaped arenas from single objects. This is often the case for enclosures with floors AND walls.

For any object that 1) has a bounding box greater than the virtual camera object's (see section 2.1.1) 3-D size, and 2) contains a region that the camera might be located, the name of the object must include a '_pic' tag. The object will be physics enabled and immovable. The 'c' portion of the tag denotes that the object is (c)oncave and that the camera can be inside the 3-D bounding box region. This tag is therefore required for correct motion computation while traversing inside of boxes, tunnels, and similar type shapes that enclose areas.

SUMMARY: If the sides of an arena (floor, ceiling, and walls) are created as individual objects, they only need to be named as 'name_p' objects to prevent traversal through the object. However, if a single object is used to create all sides of the arena, and the camera is to traverse the inside of the object, then the object name must be tagged with '_pic' for correct motion computation.

*2.0.12 _name_pic_*

Similar to 'name_pic' objects (see section 2.0.11), '_name_pic_' objects will be physics enabled, immovable, and denoted as (c)oncave, in order for the virtual camera object (see section 2.1.1) to correctly compute motion while traversing the inside of the object. The '_name_pic_' object will also be invisible.

For example, a '_name_pic_' object could be used to create an invisible track, channel or tunnel, serving to constrain the camera's motion within a larger space.

2.0.13 name_pgX.Y

Whenever the virtual camera object (see section 2.1.1) is in contact with an object named 'name_pgX.Y', the raw treadmill motion gain values (originally set in the Console **Calibration** tab, see section 1.4.4) will be overwritten with the values X.Y, which represent the gain multiplier value in the format X.Y.

For example, to create an invisible object with an X-axis (pitch) gain factor of 3.2, effectively increasing the amount of forward/backward motion created by a given raw treadmill motion by a factor of 3.2, set the object name to 'name_pg3.2'). For complete instructions on setting the motion gain values using collisions with these objects, see section 2.2.3).

2.1 Mandatory Objects

2.1.1 Camera

Each scene model must contain one virtual camera object, named '_camera_block_pm_'. The camera represents the virtual subject in body size and location, and should be made from a UV sphere object (*https://www.blender.org/manual/modeling/meshes/primitives.html#uv-sphere*), to ensure the smoothest rolling motion on all surfaces.

The camera object requires flanking underscores in the name, '_camera_block_pm_', thereby making the camera physics enabled, movable, and invisible.

By default, the perspective of the camera is from the top of the '_camera_block_pm_' object and faces parallel to the Z-axis (0° of pitch).

NOTE: The size of the '_camera_block_pm_' object will affect how it interacts in the scene. For example, if the width of an open space is smaller than the width of '_camera_block_pm_', the camera will not be able to traverse through that region.

The user can change the height (Y-axis translation) of the camera (set in the Console **Configuration** tab, under **Camera Offset** (see section 1.4.5-[4])) serving to raise or lower the camera perspective in the scene.

2.1.2 Start

Objects with the name '_start_' create pre-defined locations in the scene to which the virtual camera object (see section 2.1.1) can be instantly moved to (teleported). Teleportation to '_start_' locations is user-controlled from the Console **Setup** tab (see section 1.4.1-[11]).

The name of '_start_' objects must contain the flanking underscores, and are thus invisible. Although any type of object can be used to make a start object, a square mesh plane is recommended for simplicity.

The actual location moved to is the geometric center (centroid) of the '_start_' object. The initial heading direction of the camera after teleportation is the mean direction of all normals extending from the object (*https://en.wikipedia.org/wiki/Normal_(geometry)*).

To view the object normal in Blender, select the object and enter edit mode (toggle the tab button). Go to the **Mesh Display** section in the **Transform Properties** panel (the transform properties panels can be shown/hidden by pressing the N key and scrolling down to find the **Mesh Display** section). Check the appropriate boxes to display the normals extending from the object's vertices and/or faces. To view the normals clearly, it may be helpful to increase the **size** to a relatively large number (in scene units), such as 10.

NOTE: Multiple start locations can be included in a single scene model. The name of the available start locations for each scene model will be displayed in the Console **Setup** tab, under **Start Locations**. For example, if three start locations are desired in 'Scene_1', they could be named '_start_1_', '_start_2_', and '_start_3_'. In this case, the three available start locations will be listed in the **Start Locations** area under the heading 'Scene_1' as 'start_1', 'start_2', and 'start_3'.

To teleport to a start location within the list of available scenes under **Start Locations**, select the desired '_start_' object, then click **Go To Location**.

NOTE: When changing scenes, if the user selects the scene name, then the most recently used '_start_' location in that scene will be used, or the first in the available list.

2.2 Optional objects

2.2.1 Path

A path object can be used to guide the virtual camera (see section 2.1.1) heading direction in the scene – termed "auto-heading" – in a user-defined manner. For a more complete description of auto-heading and path-based heading, see Appendix D – D.1.6.4.

Briefly, if a '_path_' object is present in the scene, the camera heading direction is continuously computed as the tangent to the '_path_' at the nearest point along the '_path_' to the camera.

NOTE: The user determines the relative % contribution of the auto-heading component (see Appendix D – D.1.6.4) versus the manual-heading component (see Appendix D – D.1.6, D.1.6.1 – D.1.6.2) to the final heading direction of the motion vector (set in the Console **Heading Direction** tab, by moving the **Trajectory Heading Direction** slider (see section 1.4.6-[1])). If no path object is detected in the scene, MouseoVeR implements 100% manual-heading mode.

NOTE: Only one path object may be present in a scene. It is recommended to create a '_path_' from a NURBS circle object. The path object must be named '_path_' and must be a closed-loop line object (not a surface). Once the desired path shape is set, the '_path_' must then be converted to a mesh object. To convert the object from curve to a mesh, select the '_path_' in object mode, press ALT + C (to convert), and select **from curve to mesh**.

2.2.2 Crossbar

In association with the '_path_' object (see section 2.2.1), the user can create a '_crossbar_' object that serves as a constraint on the motion of the virtual camera object (see section 2.1.1).

The '_crossbar_' must be two mesh planes (we recommend two small mesh squares that face each other), where the length of the crossbar is defined as the distance between the centroids of the two planes. The length of the crossbar variable can be overwritten by clicking the Console **Configuration** tab and selecting the **Enabled** checkbox in **Crossbar** area (see section 1.4.5-[4]). The crossbar can be located anywhere in the scene.

If a path and a crossbar object are present in the scene, the camera location is continuously constrained to the '_path_' +/- the length of the '_crossbar_' (in scene_unit) perpendicular to the '_path_'. If no crossbar object is present, there will be no such perpendicular motion constraint on the camera.

2.2.3 Velocity Gain

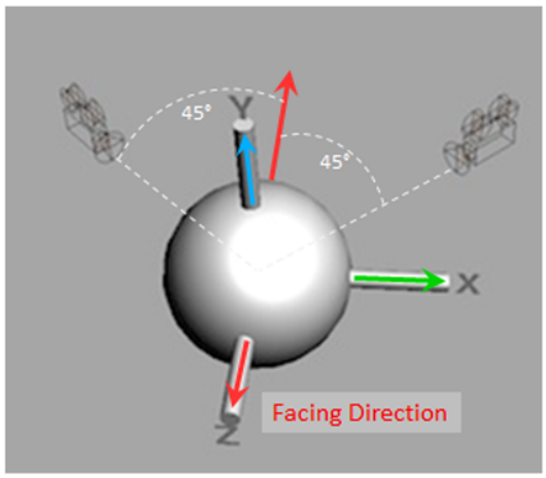
The user can create '_name_pgX.Y' objects in the scene that, when contacted by the virtual camera object (see section 2.1.1), will update the treadmill rotational motion gain calibration settings (originally set in the Console **Calibration** tab, see section 1.4.4). Gain values will return to their original value when the camera is not in contact with a gain object.

'_name_pgX.Y' objects must be created from a mesh triangle object.

NOTE: Multiple triangles can be placed in the scene to create more complex gain-modulating regions. Gain values are determined from the three color channels of the '_name_pgX.Y_' object. That is, the gain values are continuously determined from the values of the three color channels (Red, Green, and Blue) for the '_name_pgX.Y' object at the point of contact (Figure 8).

The color channel values range from 0-255, and are converted to gain values that range from 0-1. The gain values from 0-1 are then multiplied by the gain factor 'X.Y', defined in the objects' name. In this manner, the object can be used to decrease (values < 1.0) or increase (values > 1.0) the treadmill rotational gain for a given axis of motion.

NOTE: Multiple gain objects can be present in the same location. In this case, for each contacted gain object, the color channel values are converted from 0-255 to 0-1, and then multiplied by the objects' gain factor 'X.Y'. Then, the individual gain values for each axis are summed (additive) across the multiple gain objects to determine the final roll, pitch, and yaw gain values that are applied to the camera object.



Axis	Color Channel
ROLL, Z	RED
PITCH, X	GREEN
YAW, Y	BLUE

Figure 8. Gain calibration values for treadmill motion in the roll, pitch, and yaw axes can be updated during physical collisions between the virtual camera and special named objects in the scene. Roll, pitch, and yaw gain values are encoded by the color channels: red, green, and blue, respectively.

2.3 Exporting Blender Scene Models

Prior to exporting Blender files for MouseoVeR, the user must perform the following steps:

1) Enter object mode, **de-select all items**, and then **select all items** in the scene by pressing the A key twice. This ensures that all objects in the scene are selected.

2) Convert all objects to mesh objects by pressing ALT + C, and selecting the **convert curve to mesh** option. This ensures all objects are mesh objects.

   WARNING: Because you have converted all curves to meshes, if you save the .blend file in this state, all curves will be lost! Therefore, be careful not to overwrite the .blend file.

3) Go to **File**, select **Export**, and save the file in COLLADA (.dae) format.

It is now recommended that the user load the previously saved .blend version that contains the original object states (with curves, etc.).

2.4 Modifying the exported Blender file

Exported Blender files must be in COLLADA (.dae) format. This is a text-based file format and can be read and edited in an advanced text editor such as Notepad++.

Blender may export the file with the shading property of all objects set to Phong (also called "Phong interpolation"). However, sometimes we find Lambert shading (also called "Lambertian reflectance") to be more appropriate for rendering in MouseoVeR. Using Lambert shading, the apparent brightness of all object surfaces will appear constant to an observer regardless of the observer's viewing angle (also called "isotropic luminance" or "diffuse reflection"). If desired, the user can ensure all objects shading settings are set to Lambert by opening the exported .dae file in Notepad++, searching for the term "phong," and replacing the found items with the term "lambert." Finally, save the .dae file to store the changed parameter. Load the new file and compare the difference in appearance.

Acknowledgments: We thank Vivek Jayaraman, Hannah Haberkern, Marc Olano, Ari Blenkhorn, Christopher Bruns, and Sean Murphy for contribution to the development of JOVIAN and MouseoVeR. This work was supported by the Janelia Research Campus and Howard Hughes Medical Institute.

Appendix A

Large Spherical Treadmill System
Parts and Design files

Jeremy D. Cohen, Nicholas J. Sofroniew, and Albert K. Lee
Janelia Research Campus, Howard Hughes Medical Institute, Ashburn, VA, 20147

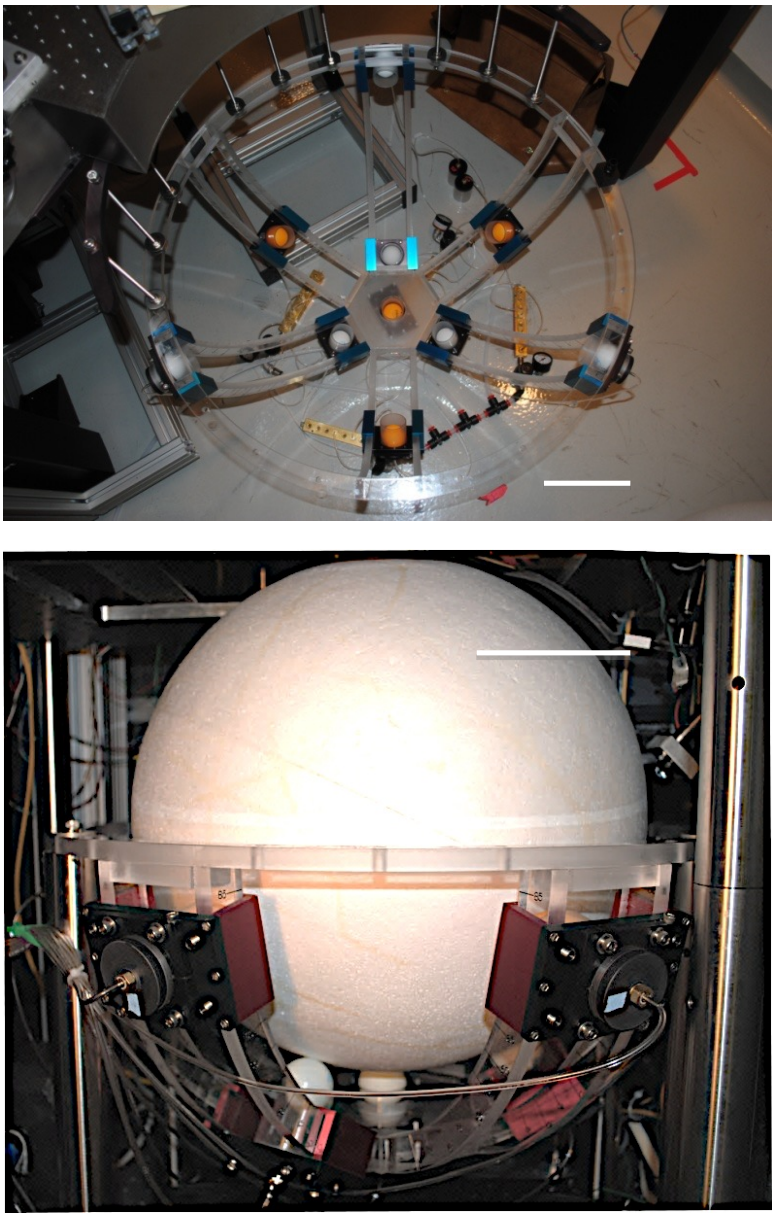


Figure A1. Overview of the principal components of the spherical treadmill system. A high-density polystyrene (HDP) sphere is supported by several independent air-cushioned ping-pong ball bearings (air cannons) clamped inside a lightweight acrylic treadmill frame. Loft is produced by continuously providing regulated air pressure (~5-15 psi) through small diameter tubing to each air cannon. (Top) 24" treadmill frame configured with ten air cannons. This treadmill system, hung from the underside of an air table, was designed for rats (60-150 g). (Bottom) A 16" treadmill system designed for mice (20-30 g). The 16" treadmill frame is configured with ten air-cannons and houses a 16" treadmill weighing ~65 g (scale bars, 6"). See Figures A2-A6 for parts and Figures A7-A15 for part designs.

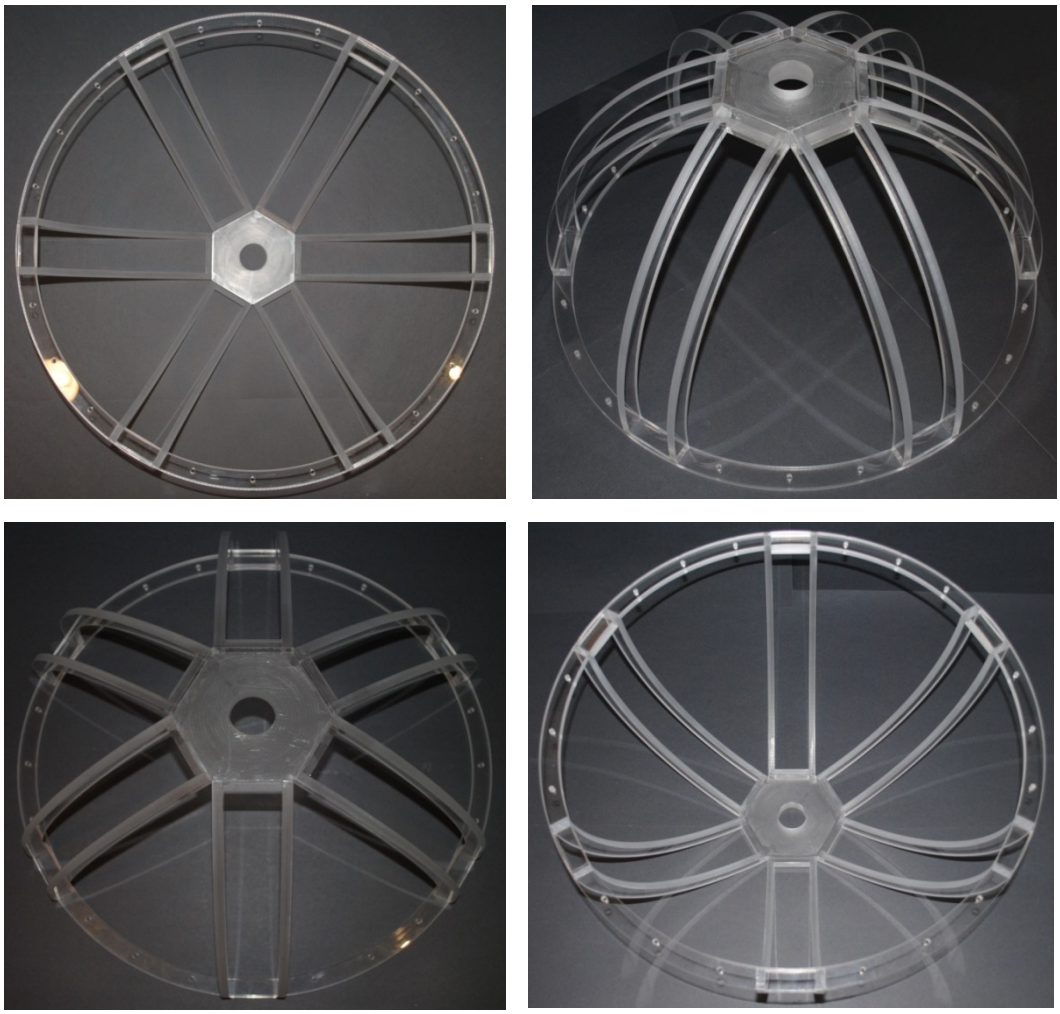


Figure A2. The treadmill frame. All panels, example of a fully assembled 24” treadmill frame.

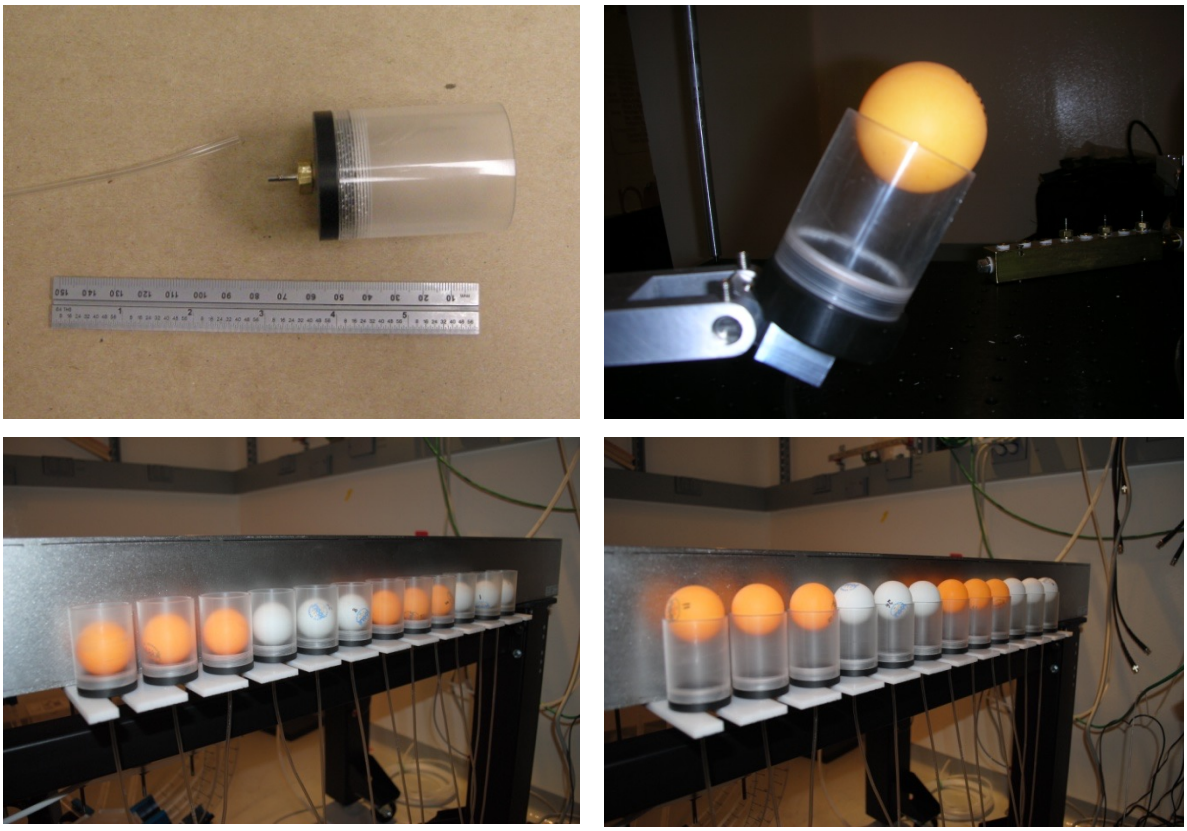


Figure A3. Air cannons. Each air cannon consists of:
1 x acrylic tube (Figure A10)
1 x delrin bottom tube caps (Figure A11)
1 x 1/8-27 NPT to 1/16” I.D. barbed pipe nipple
1 x ping pong ball (Gold 3-Star)
1 x 1/16” I.D. tubing

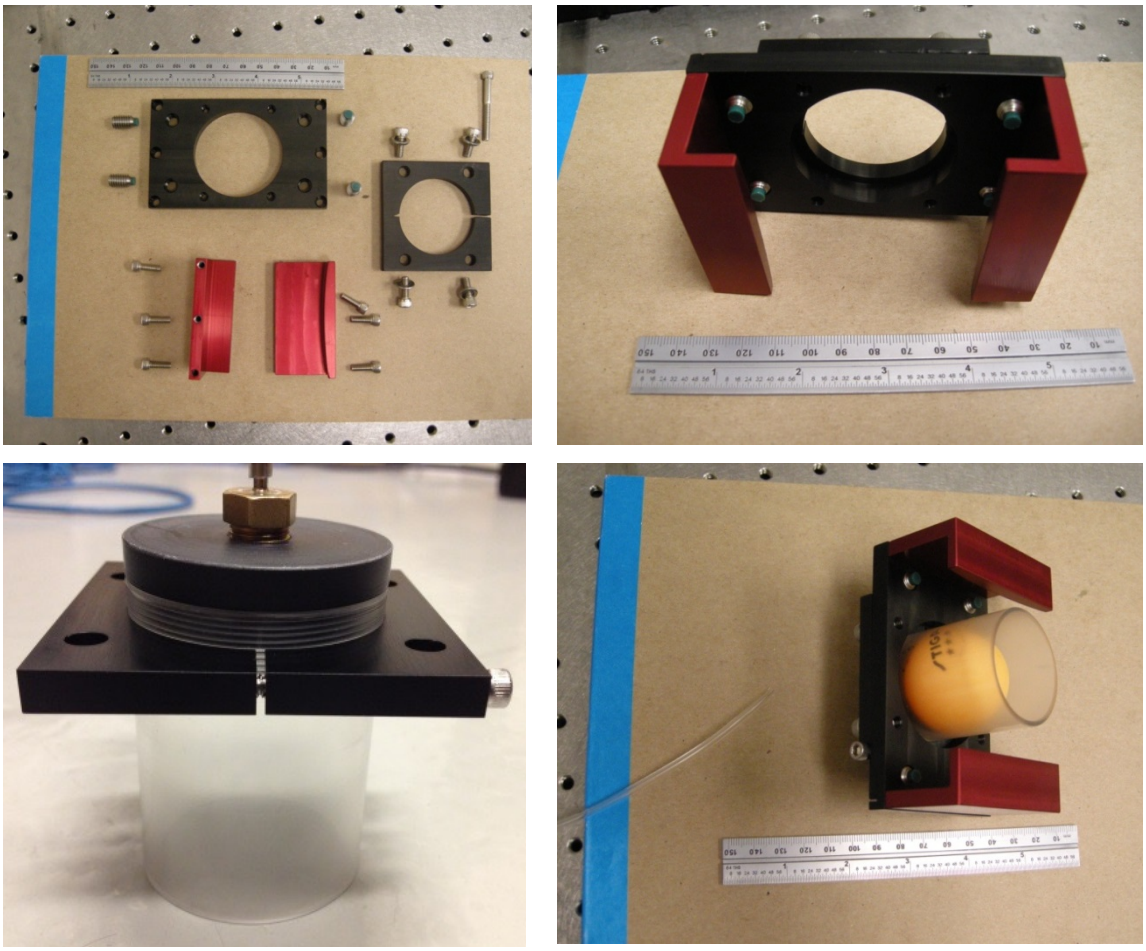


Figure A4. Air cannon clamp and bracket system. Each air cannon clamp and bracket system consists of:
1 x Cannon Clamp (Figure A12)
1 x Clamper Plate (Figure A13)
1 set of two Frame Clampers (Figure A14):
A) 2 x 9” radius Frame Clampers (Figure A14, top, shown here in red)
B) 2 x 13” radius Frame Clampers (Figure A14, bottom, shown in blue throughout the document)
6 x 8-32 ½” Socket Cap Screws
1 x 8-32 1-1/2” Socket Cap Screws
4 x 10-32 1-1/2” Socket Cap Screws (with washers)
4 x 5/16”-18 1-1/2” Nylon Tip Set Screws

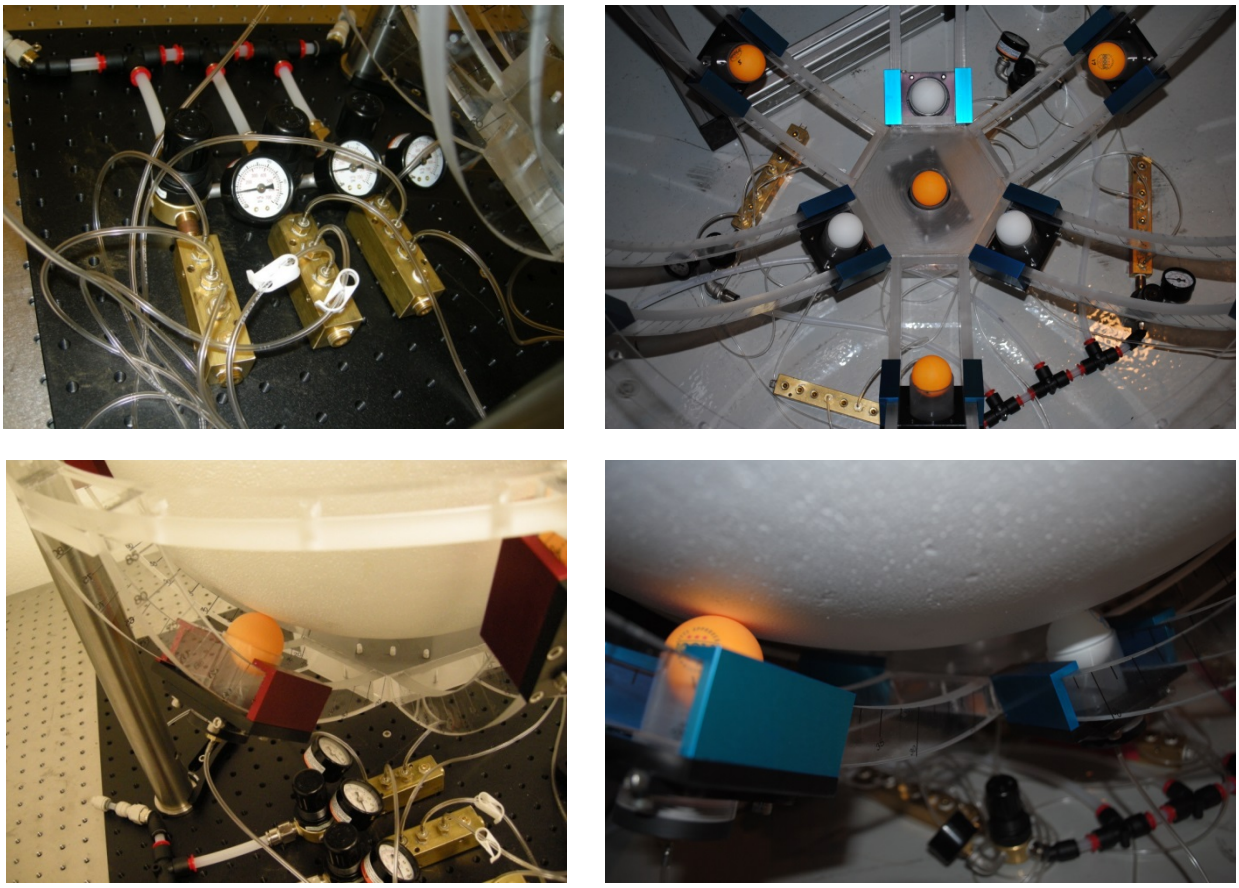
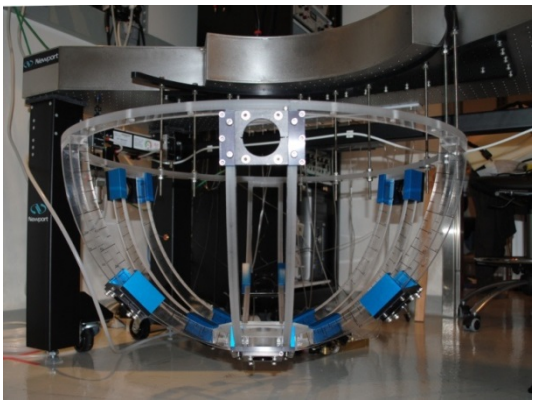
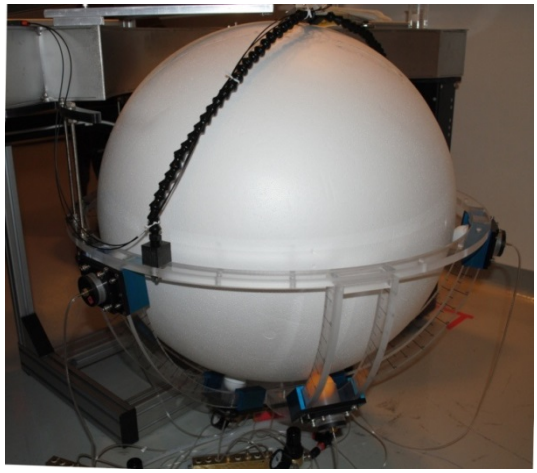


Figure A5. Air regulation. Each set of three air cannons is regulated by one air regulation system. Each air regulation system consists of:
1 x air regulator (rated to 0-50 psi recommended)
3 x 1/8-27 NPT to 1/16” I.D. barbed pipe nipples
3 x pieces of 1/16” I.D. tubing





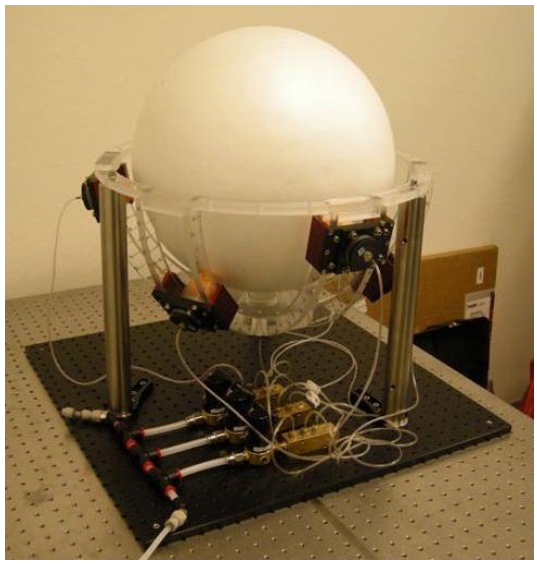


Figure A6. Examples of assembled treadmill suspension. (Top) 24” treadmill frame hung from the underside of an air floatation table - designed for rats. (Middle) 24” treadmill in the frame, hung from the underside of the air table. (Bottom) 16” treadmill system resting on three 1.5” diameter stainless steel posts - designed for mice.

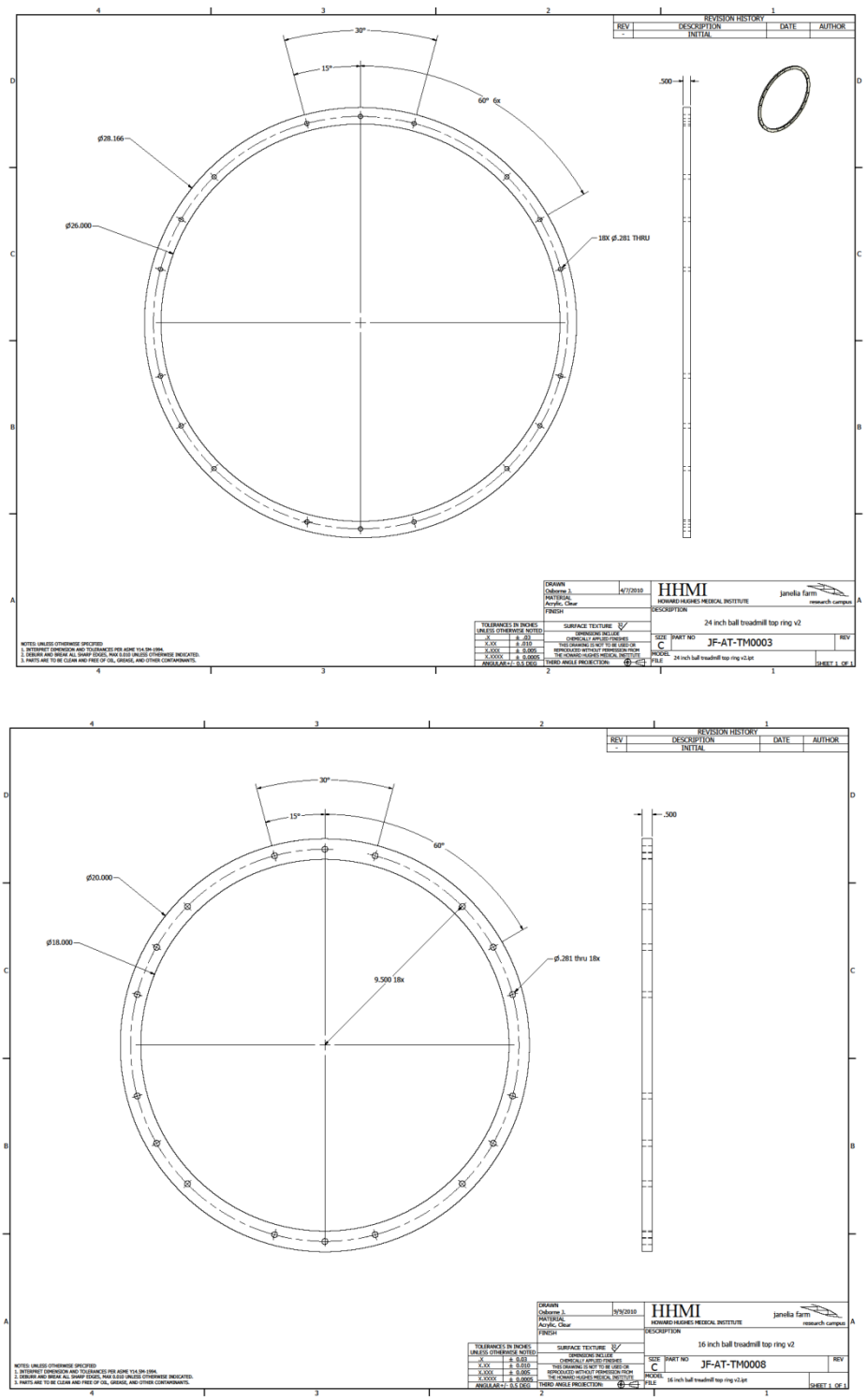


Figure A7. Treadmill frame top ring. (Top): 24” treadmill frame top ring. (Bottom): 16” treadmill frame top ring.

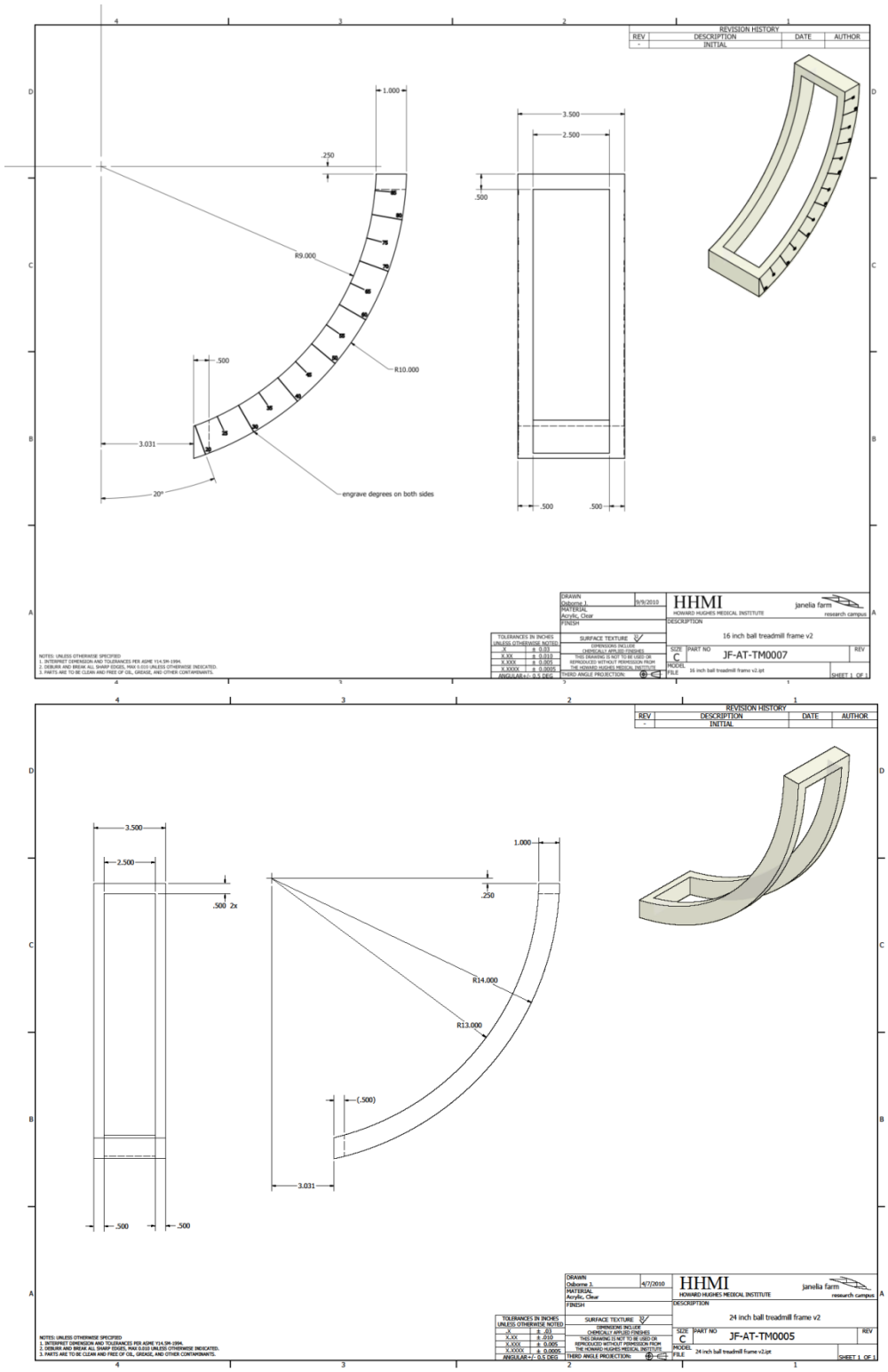


Figure A8. Treadmill frame arms. (Top) Note that the 16” treadmill requires 9” radius curvature arms for the 18” I.D. treadmill frame. (Bottom) Note that the 24” treadmill requires 13” radius curvature arms for the 26” I.D. treadmill frame. (6 x arms required).

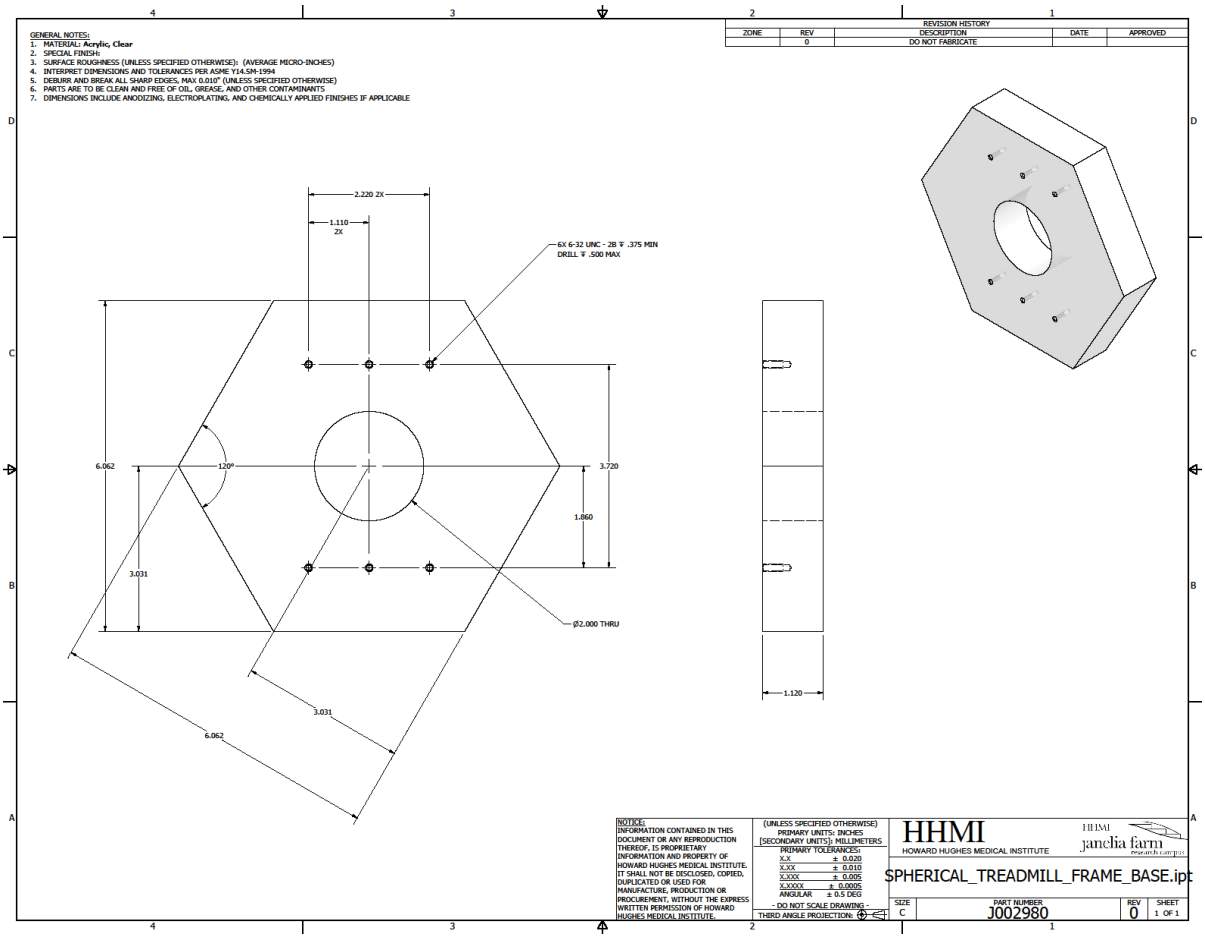


Figure A9. Treadmill frame base. (one size only)

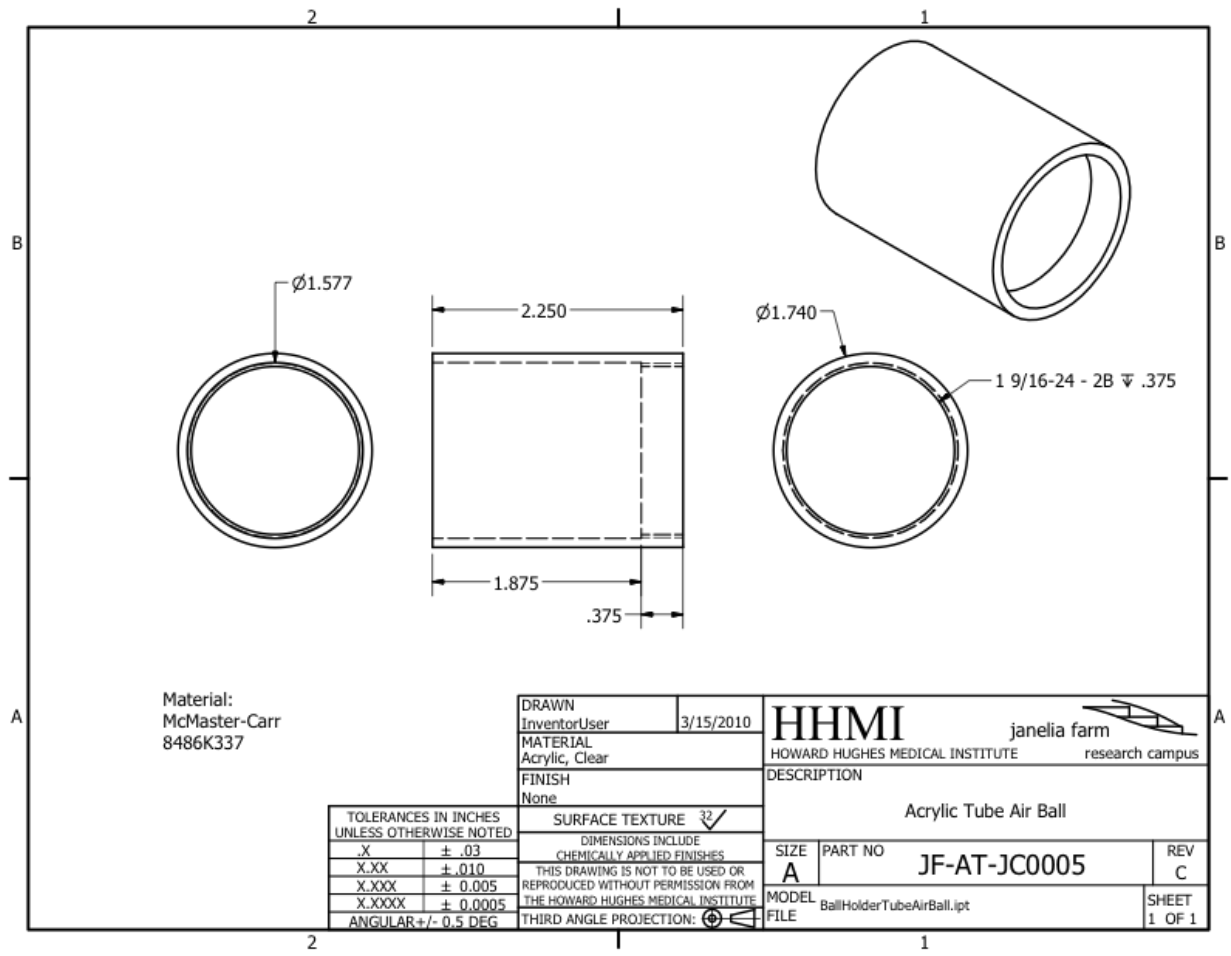


Figure A10. Air cannon tube.
(one size only)

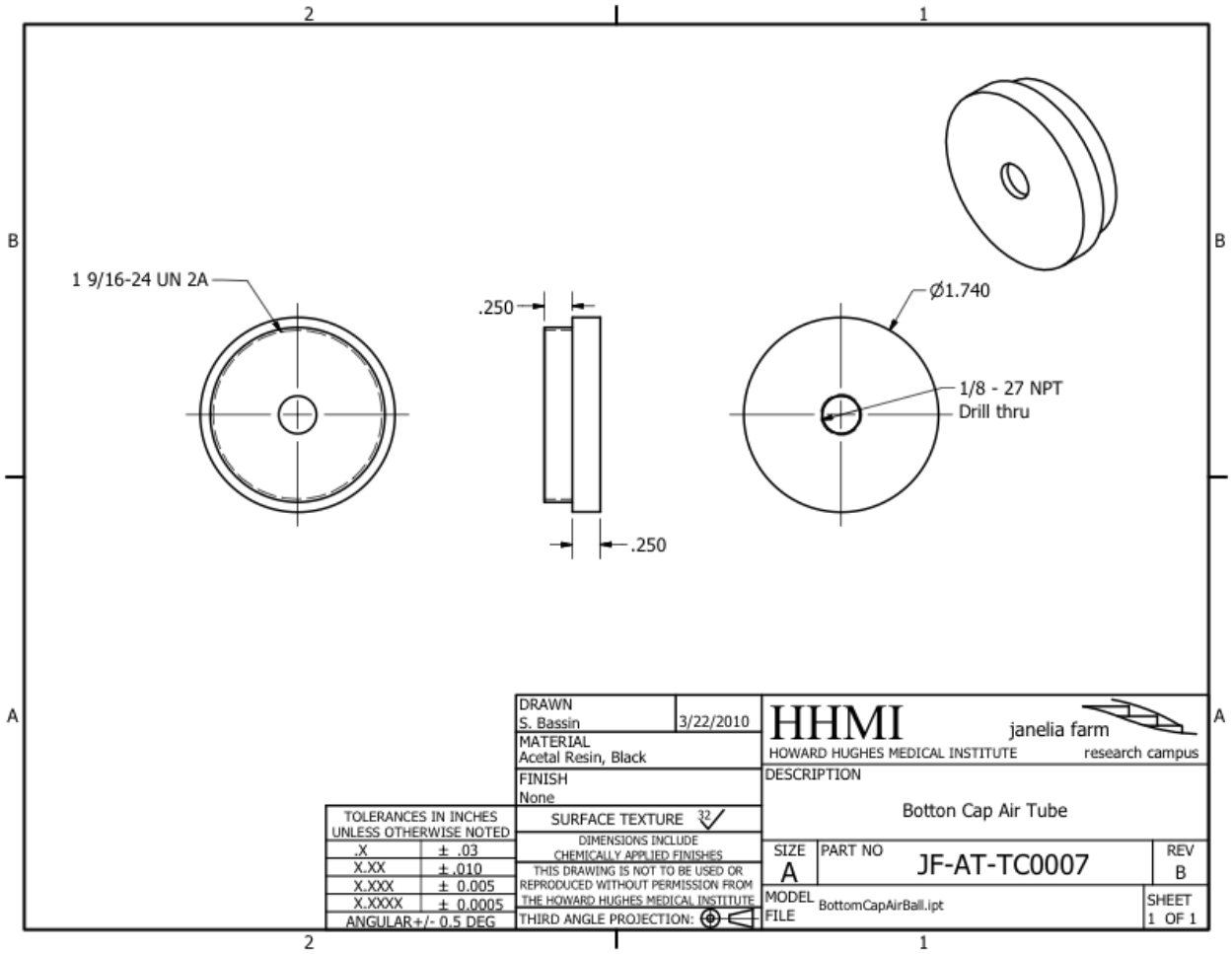


Figure A11. Air cannon tube cap
(one size only)

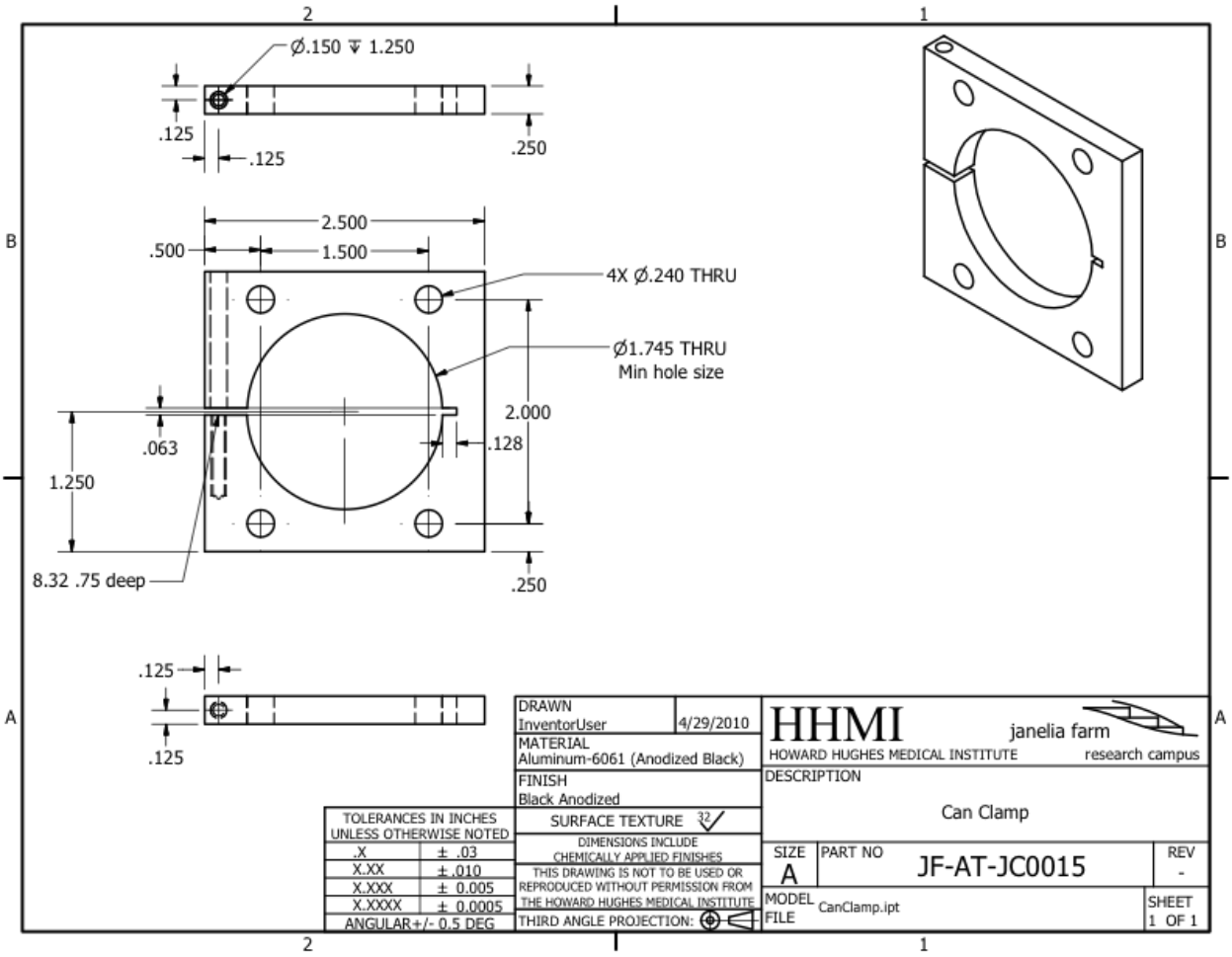


Figure A12. Air cannon clamp.
(one size only)

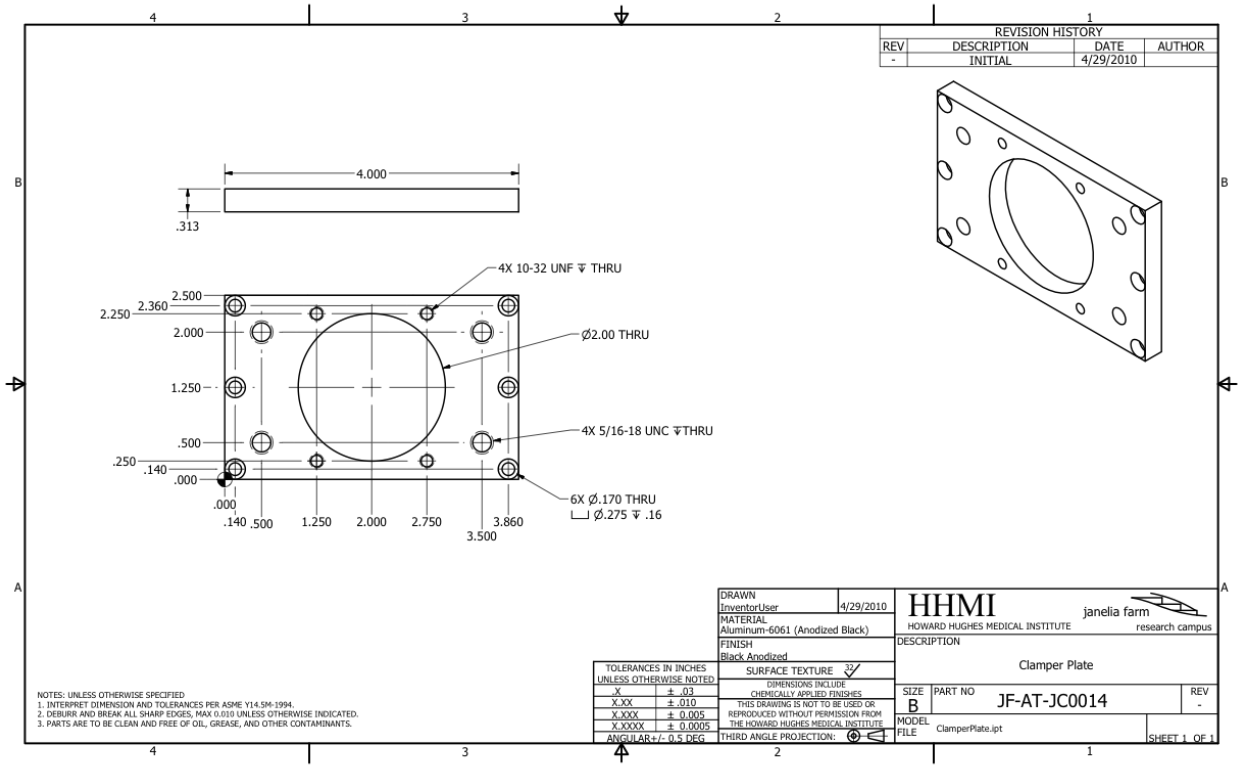


Figure A13. Treadmill clamper plate
(one size only)

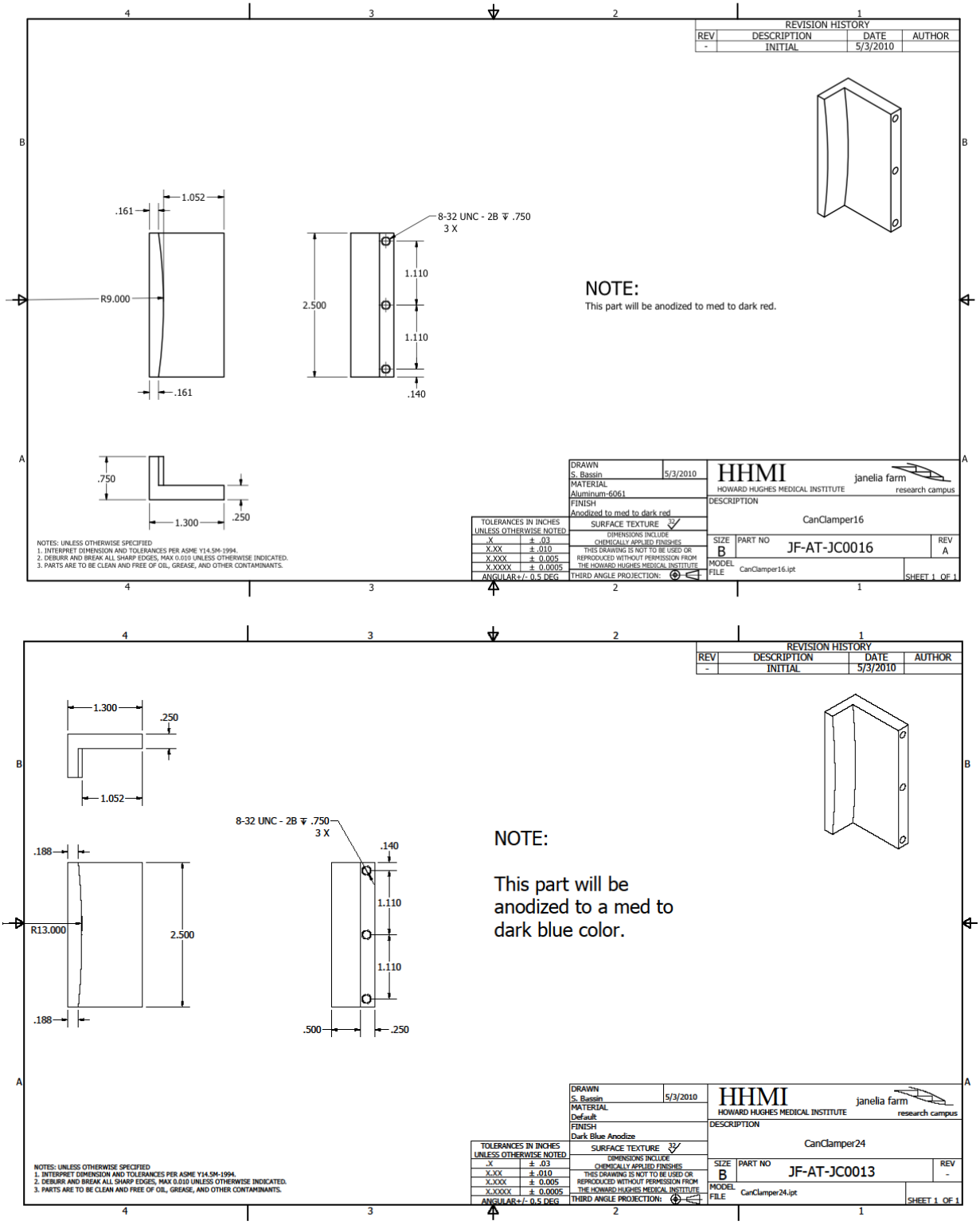


Figure A14. Treadmill frame clampers. (Top) Note that the 16” treadmill requires the 9” radius curvature clampers for the 18” I.D. bowl. (Bottom) Note that the 24” treadmill requires the 13” radius curvature clampers for the 26” I.D. bowl.

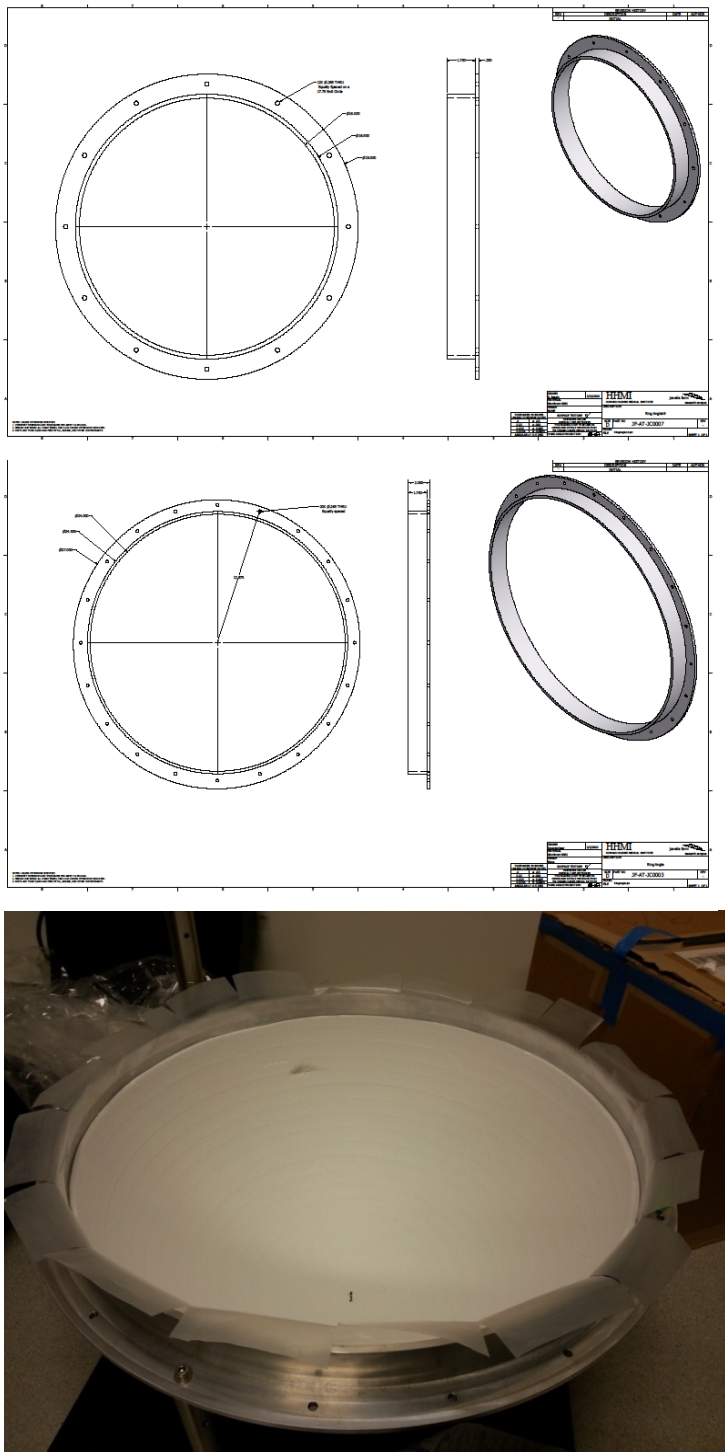


Figure A15. Treadmill gluing ring. (Top) 16” gluing ring. (Middle) 24” gluing ring. (Bottom) 24” half-sphere in the 24” gluing ring (surrounded by parafilm). The gluing ring is resting on three 1.5” thick, 11” high pillars. This configuration places the equator of the forged sphere in the center of the gluing ring. (**NOTE:** Oddly, the 16” spheres from Smoothfoam.com are actually 15.5625” diameter. We recommend a gluing ring of 15.6” I.D. for that specific product).

Appendix A

Acknowledgements: We thank Jason Osborne and Steve Bassin for mechanical engineering, design, and machining work; Art Lee for assistance with air regulation; Carmen Robinett for helpful discussions regarding the floatation mechanism; Nicholas J. Sofroniew for his work as an early user of the treadmill system and for assistance with documentation.

Appendix B

Large Spherical Treadmill System
Essential Parts and Quote Estimates

Jeremy D. Cohen and Albert K. Lee
Janelia Research Campus, Howard Hughes Medical Institute, Ashburn, VA, 20147

Appendix B

Essential Parts and quote estimates for MouseoVeR and the spherical treadmill system

See Appendix A for technical drawings and example pictures, and Appendix C for a user-description of the essential parts and basic setup.

Part # (quantity)	QUOTE
JF-AT-TM0007 (x6) - frame arms, 16”	~$150 each (Precision Plastics)
JF-AT-TM0008 (x1) - frame top ring, 16”	~$500 each (Precision Plastics)
JF-AT-TM0009 (x1) - frame base	~$200 each (Precision Plastics)
Treadmill frame assembly - frame assembly	~$600 (Precision Plastics)
JF-AT-JC0005 (x10) - air cannon, acrylic tube	~$60 each (Machining Services)
JF-AT-TC0007 (x10) - air cannon, bottom cap	~$30.00 each (ID&F Machining)
JF-AT-JC0014 (x10) - air cannon, clamper plate	~$64.50 each (Jones Mfg)
JF-AT-JC0015 (x10) - air cannon, can clamp	~$59.50 each (Jones Mfg)
JF-AT-JC0016 (x20) - air cannon, can clamper, 16”	~$29.10 each (Jones Mfg)
JF-AT-JC0007 (x1) - aluminum ring	~$1025 each (Jones Mfg)
HD Polystyrene sphere - 16” diameter - Part #: 10073	~$40 each (Smoothfoam)
Ping-pong Balls (x24)	~$20 (Joola Gold 3-star)
Air regulation, various parts - see Figures A2-A4	~$1000 (McMaster-Carr)
FlyFizz ball-tracker (x2) *http://wiki.int.janelia.org/wiki/display/~liujin/Ball+Tracker+%28or+Treadmill%29+project*	~$800 each (HHMI – Janelia, ID&F)
8mm Megapixal lenses (x2) Part#: NT56-786	~$235 each (Edmund optics)
8mm Megapixal lenses (x2) Part#: NT56-786	~$235 each (Edmund optics)

Appendix B

Part # (quantity)	QUOTE
64-bit Windows computer For VR system	~$1500 (Dell)
Displays for VR system e.g., multiple projectors or monitors	~$2000 (various options available)
Video Card for VR system GIGABYTE GV-N770OC-2GD GeForce GTX 770 2GB 256-bit GDDR5 PCI Express 3.0 HDCP Ready WindForce 3X 450W	~$379 each (Newegg)
MouseoVeR Virtual Reality Software Suite	Free, open-source (HHMI – Janelia Research Campus)
	TOTAL COST ESTIMATE ~$18,606

Appendix C

User Guide to MouseoVeR and the Virtual Reality Behavioral System: Electronic Devices Order List and Basic Setup

Xinyu Zhao, Jeremy D. Cohen, and Albert K. Lee
Janelia Research Campus, Howard Hughes Medical Institute, Ashburn, VA, 20147

C.1 Electronic Devices Order List

C.1.1 Treadmill Tracking System

FlyFizz ball-trackers (Originally developed at HHMI Janelia by Gus K Lott III, Nir Dutta, Jason Osborne, Eugenia Chiappe, Johannes Seelig, Michael Reiser, and Vivek Jayaraman).

- The treadmill tracking camera documentation is located at:

  …Documentation\Treadmill_Tracking\

  - One set contains one MCU board and two cameras
  - Only one set is needed for the configuration described in this document

- Order two IR LED lights for illumination:
  - CMVision IR30 WideAngle IR Illuminator
    http://www.amazon.com/CMVision-IR30-WideAngle-IR-Illuminator/dp/B001P2E4U4/ref=sr_1_5?ie=UTF8&qid=1411592819&sr=8-5&keywords=infrared+spotlight

- Order 2X 12V DC adapter for the power:
  - 12V power adaptor (DC, 2.1mm)
    http://www.amazon.com/DC-Power-Adaptor-12-2-1MM/dp/B002O9QM5C/ref=pd_sim_hi_1?ie=UTF8&refRID=05APQW5VVND36XY3C8ZX

C.1.2 Lick Detection

- Optical sensor from SunX (Panasonic) is used for lick detection
  - Fiber: FT-V23
    http://www.sunxsensors.com/products/product/2908-ft-v23.html
  - Amplifier: FX-301H
    http://www.sunxsensors.com/products/product/919-fx-301h.html

C.1.3 Lick Port

The lick port system contains one lick port holder, one lick port stand (to connect the holder to the servomotor) and two optic fiber clamps (to secure the optical sensor).

- Lickport design files are provided and located in the folder:

  …\MouseoVeR\Lickport_Holder_Documentation\

- The lick port holder and two fiber clamps (as mirror images to each other) are 3D-printed parts:
  - Lickport_Holder_v3.ipt
  - FiberClamp_v2.ipt
  - FiberClamp_v2_MIR.ipt

- Example lick port stand:
  - Lickport_Stand.ipt

- Order antibacterial tubing from Eldon James:
  http://tubingandfittings.eldonjames.com/item/categories-tubing-clamps-medical-biomedical-tubing/flexelene-tubing-fxag/fxag-1-2?&plpver=10&origin=keyword&by=prod&filter=0

- Optional: The lick port can be mounted on Zaber actuator motors
  - 2X T-LS28E
    http://www.zaber.com/products/product_detail.php?detail=T-LS28E

- Other accessories needed for the example lick port:
  - 2mm OD glass capillary
  - 1/16" ID regular tubing
  - WPI barb-to-tubing coupler kit
    http://www.wpiinc.com/products/laboratory-supplies/500890-barb-to-tubing-coupler-kit/?query=%28name.like.barb.or.misc0.like.barb.or.misc2.like.barb.or.misc3.like.barb%29&xsearch_id=products_search_all&xsearch[0]=barb&back=products

C.1.4 Solenoid Water Valve

- EV-2-12L from Clippard:
  http://www.clippard.com/part/EV-2-12-L

- Optional driver circuit for a solenoid valve:
  - CoolDrive ONE solenoid driver (161D1X250) from NResearch:
    http://www.nresearch.com/
  - You can find the product at Accessories->CoolDrive Valve driver family
  - The valve and driver may break easily. We recommend ordering additional parts for backup

C.1.5 Optional - Pulse Generator

- A.M.P.I. Master-9 Pulse Stimulator
  http://www.ampi.co.il/master9.html

C.1.6 Example Electrophysiology Recording System

- Amplifier: BVC-700 from Dagan
  http://www.dagan.com/bvc-700.htm
  with Universal Pipette Holder for use with 1.0 to 1.8 mm glass: HB-180
  http://www.dagan.com/holders.htm
  - The amplifier comes with an intracellular headstage
  - Order an extracellular headstage (8024-7001) if necessary
    http://www.dagan.com/8024.htm

- DAQ board: InstruTECH ITC-1600e Acquisition Interface from HEKA
  http://www.heka.com/products/products_main.html#acq_itc1600

- Manipulator from Luigs & Neumann:
  - UNIT Junior 4RE (210-100 000 0085)
  - Remote control system SM8 (200-100 900 9080)
  - Control system SM7 for 4 Axis (up to 9) (200-100 900 7411)
  - Special adapter to mount the UNIT Junior 4 axis to Stereotaxic (19 mm bars) (200 100 500 0060)
  - Mounting clamp for headstage for the Heka, NPI, Axon Instrument, Warner Instruments (LN-JUNIOR) (200-100 500 0250)

- Oscilloscope: Tektronix TDS 2012C (100MHz)
  http://www.tek.com/oscilloscope/tds2000-digital-storage-oscilloscope

- Extech 407910: Heavy Duty Differential Pressure Manometer (29psi)
  http://www.extech.com/instruments/product.asp?catid=15&prodid=222

C.1.7 Computers

- Computers for electrophysiology recording (e.g., Dell Precision Tower 5810)
  - Must run 64-bit Window 7, equivalent or better
  - The computer must have at least one PCI-e slot for the ITC DAQ board

- VR computer (e.g., Dell Precision Tower 7910)
    - Video cards for this computer: for ex, GV-N770OC-2GD (need 2 cards)
      http://www.gigabyte.com/products/product-page.aspx?pid=4629#ov
      Or, NVIDIA GeForce GTX 970 (1 card)
      http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-970
    - If you order a different model of computer or video card, several requirements need to be met:
        - Each video card requires 600W power supply, so the computer must have >1200 W power
        - The computer also needs to have at least two PCI-e 3.0 slots for the video cards
        - This computer should run 64-bit Windows 7. The current VR software (Jovian) may not work with Win8.
        - In addition, confirm with Dell to ensure that the case of the computer is large enough to install two video cards

C.1.8 Monitors (for MouseoVeR Display)

- For example, NEC X463UN
  http://www.necdisplay.com/p/large--screen-displays/x463un?type=support

- Various projector configurations can also be used

C.1.9 Accessories

- Several BNC cables, F/F adapters and T-connectors (from McMaster Carr)

- Several 12V DC power adapters (from Amazon)
    - DC power connectors could be tricky, since it has so many different sizes
    - The two most commonly used types are: OD5.5mm/ID2.1mm and OD5.5mm/ID2.5mm
        - M/F connectors for these two types are not interchangeable. Be careful when ordering power adapter or soldering connectors. Make sure they match the connectors on devices.

- Several USB A-B cables, USB extension cables and USB to mini-USB cables (from Amazon)

- At least 2 Keyspan USB to RS232 adapter (USA-19HS)
  http://www.tripplite.com/high-speed-usb-to-serial-adapter-keyspan~USA19HS/

- Mouse, keyboard (from Amazon)

- Console monitor: Dell UltraSharp 1708FP 17-Inch LCD Flat Panel Monitor (from Amazon) *http://www.amazon.com/Dell-UltraSharp-1708FP-17-Inch-Monitor/dp/B001P510CQ/ref=sr_1_1?ie=UTF8&qid=1426555608&sr=8-1&keywords=Dell+UltraSharp+1708FP+17-Inch+LCD+Flat+Panel+Monitor*

- StarTech 4 Port DVI USB KVM Switch (from Amazon) *http://www.amazon.com/StarTech-com-Port-Switch-Audio-SV431DVIUA/dp/B000VS6RZS/ref=sr_1_7?ie=UTF8&qid=1426555898&sr=8-7&keywords=DVI+KVM+startech*

  NOTE: Find a KVM that works with your OS.

- Several ethernet cables (from Amazon)

- Several DVI-D M/M cables, DVI-D M/F extension cables and HDMI to VGA adapters (from Amazon)
  - Be sure to order DVI-D (digital) but NOT DVI-A (analog) cables
  - DVI-I (integrated) is OK too since it can work with both digital and analog signals

C.1.10 Useful Thorlabs Parts

Order several Thorlabs parts to install IR spotlights and the lick port:

- 4X Ø1/2" Universal Post Holder, Spring-Loaded Locking Thumbscrew, L = 3" (UPH3)
- 2X Ø1/2" Optical Post, SS, 8-32 Setscrew, 1/4"-20 Tap, L = 10" (TR10)
- 2X Ø1/2" Optical Post, SS, 8-32 Setscrew, 1/4"-20 Tap, L = 6" (TR6)
- 1X Aluminum Breadboard, 8" x 24" x ½", ¼"-20 Double-Density Taps (MB824)

C.2 Spherical Treadmill System

C.2.1 Polystyrene Sphere

- Order 10X 16" foam balls (20 halves) from Smoothfoam (or similar): *http://www.smoothfoam.com/product/10073.html*
- Directly ship them to WeCutFoam (or similar): *http://www.wecutfoam.com/#!contact/c1d94*
- Have WeCutFoam to hollow the halves uniformly from the inside to a final weight of approximately 32-33 g

C.2.2 Air Regulation system

- Obtain four sets of air regulation systems (see Appendix A – Figure A4)
    - Each set contains:
        - 1 brass manifold
        - 3 brass nipples (barbed, pipe size: 3/8”, hose ID: 1/16”)
        - 1 air regulator
    - Sets are connected using 3/8” plastic hose and couplers

- The air regulation manifolds are connected to the air cannons through soft rubber tubing from McMaster-Carr:
    - High-Temperature Silicone Rubber Tubing, Soft, 1/16” ID, 1/8” OD (Catalog #: 5236K81)
    - Order 100 feet

- Order 12X Clamp-Style Pinch Valves for Tubing (Catalog #: 5330K23) from McMaster-Carr

C.2.3 Useful Thorlabs Parts

The following Thorlabs parts are used to mount the frame onto the air table:

- 4X Ø1” Imperial Pillar Posts (L=12”, ¼-20 tap) (Catalog #: RS12)
- 4X Ø1” Pedestal Pillar Posts (L=0.5”, ¼-20 tap) (Catalog #: RS05P)
- 4X Long Clamping Fork, 1.76” Counterbored Slot, Universal (Catalog #: CF175)
- 4X 1/4"-20 Stainless Steel Setscrew with Hex on Both Ends (Catalog #: SS25E63D)
- 4X ¼-20 1” screws

C.2.4 Treadmill Installation

C.2.4.1 Air Table and Air Regulation System

1) Use tape to mark the position where the air table should be located. Ask facilities department to install the air table.

2) Assemble the air regulation system at the same time or separately.

3) Adjust the appropriate pressure for the air table (approximately 25 psi). When more things are mounted on the table, it may be necessary to re-adjust the pressure if one or more of the legs touches the bottom. In this case, slowly increase the pressure to the extent that just lift all legs up, and then further increase the pressure by 5-10psi. There are valves on each leg that allows independent adjustment of pressure. Once configured, **DO NOT** adjust them individually. Always use the main air regulator to change pressure.

4) The 3/8” plastic hose goes into the coupler directly. No need to tape the hose. There is a ring of “teeth” inside the coupler that will hold the hose. It may be necessary to temporarily disconnect the air regulation system during the installation of monitors. To do this, first release any pressure in the system and then push the ring backward to release the hose.

C.2.4.2 Treadmill Assembly

Refer to Appendix A – Figures A3-A4 for the air cannon and bracket system parts.

1) Assemble Air Cannons

Screw the nipple onto the cap. Use a hex socket driver or a wrench to tight it. **DO NOT** need to push the entire threaded part of the nipple into the cap (Figure C1, left). Over-tight the nipple may crack the cap. Screw the cap onto the acrylic tube. Clamp the tube with the clamper. **DO NOT** over-tight the clamper; it can crack the tube. The position of the clamper: the bottom surface of the clamper is about 0.5” from the bottom of the cap. Make sure clampers on all cannons are at the same position. Use any method to achieve this. In my case, I first clamped one tube, and used it as a reference to set all other clampers to the same level (Figure C1 right). Once the tube is clamped, it will be hard to unscrew the cap or nipple; loose the clamper first if those things need to be adjusted later. The air cannon that goes to the bottom of the frame should have a different clamper position since no brackets would be used (Figure C4). The clamper position of this one can be manually adjusted later when all other parts are ready and the foam ball is on.

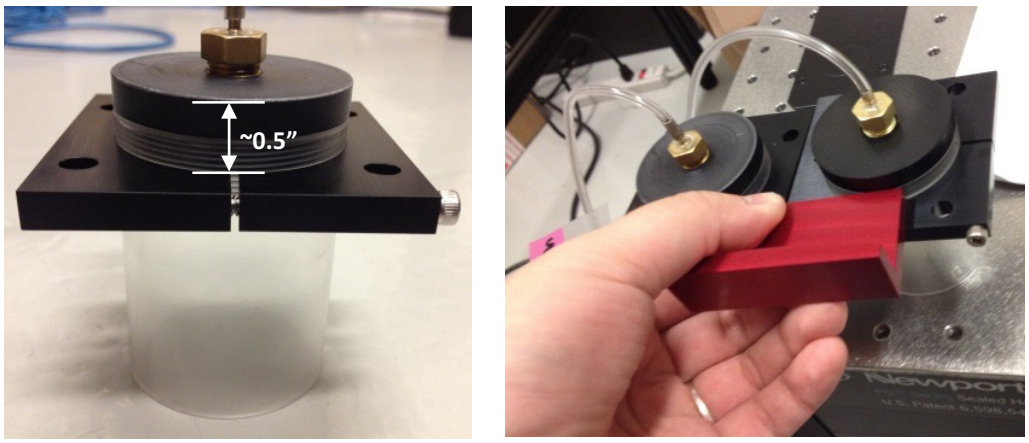


Figure C1. Assembly of air cannons.

Cut the soft rubber tubing into 10x 8ft pieces. Plug one end into the nipple of the manifold, put a pinch valve on each tubing and plug the other end to the nipple of the air cannon. The friction could make it very hard to plug the tubing to brass nipples. It is helpful to clean the nipple first and put some lubricant around it before plugging. Use 3 manifolds for 9 (3X3) cannons. The rest cannon, which will be mounted to the bottom of the frame, connect to a nipple on the fourth manifold. Cut two pieces of short tubing. Plug them to the other 2 nipples on that manifold and use pinch valves to seal them. Organize the tubing so they do

not entangle with each other. Use any method to label each tubing at both ends. So it will be easy to tell which manifold nipple is connected to which cannon (Figure C2).

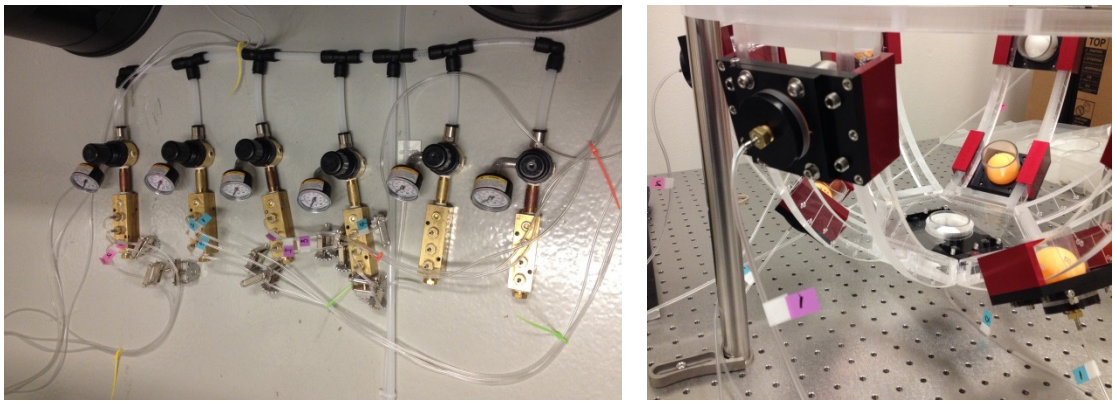


Figure C2. Air regulation system and labels.

2) Test Ping Pong Balls

First seat all air cannons (already connected to the air regulation system) on a flat surface. Use some rack to do this (see Appendix A – Figure A3) - or as what I did - use clamper plates as a temporary tool to mount all cannons on the air table's edge (Figure C3).

Ping pong balls are not manufactured in a very precise way. First use a spare acrylic tube to test if a ball can easily go into the tube without large friction. Put passed balls into air cannons. Turn on the air to float all of them. Use the same pressure for all manifolds except the one only connected to one air cannon (~1/3 of the pressure as for other manifolds should be used since only one outlet is open). Push every ball down and feel the force. If a ball is significantly smaller than others, one should feel less resistance when pushing it down. Replace balls until one feels a similar force pushing every one. Mark all chosen balls to avoid confusion in future.

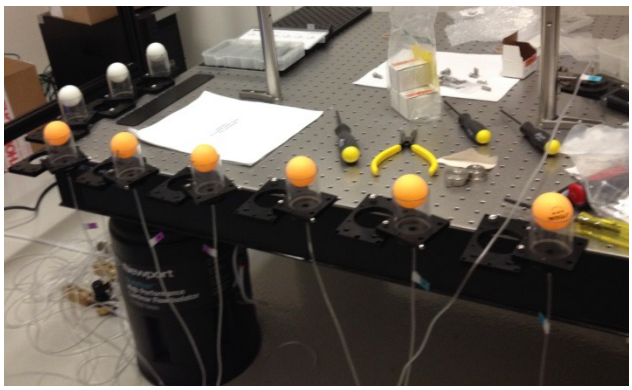


Figure C3. Test ping pong balls.

3) Mount Air Cannons onto the Frame

First screw one clamper plate to the bottom plate of the frame. No bracket ("bowl clamper") is needed for this one. Screw one bowl clamper on each clamper plate. Then put the whole thing on the frame's arm and screw the bowl clamper on the other side (when bowl clampers are installed on both sided, it cannot go into the arm anymore). Here are two example arrangements of air cannons on frame arms: 1) Three are placed close to the rim and three are placed close to the bottom. In this case, seven air cannons were used for one treadmill; 2) Six are placed close to the rim and three are placed close to the bottom. The second configuration makes the treadmill more stable but also a little harder to turn due to the extra friction. I personally used the second configuration (Figure C4). There are reference marks on each arm of the frame. The upper edge of the clamper plate is at 85°/45°. Tighten nylon-tip set screws to secure the positions.

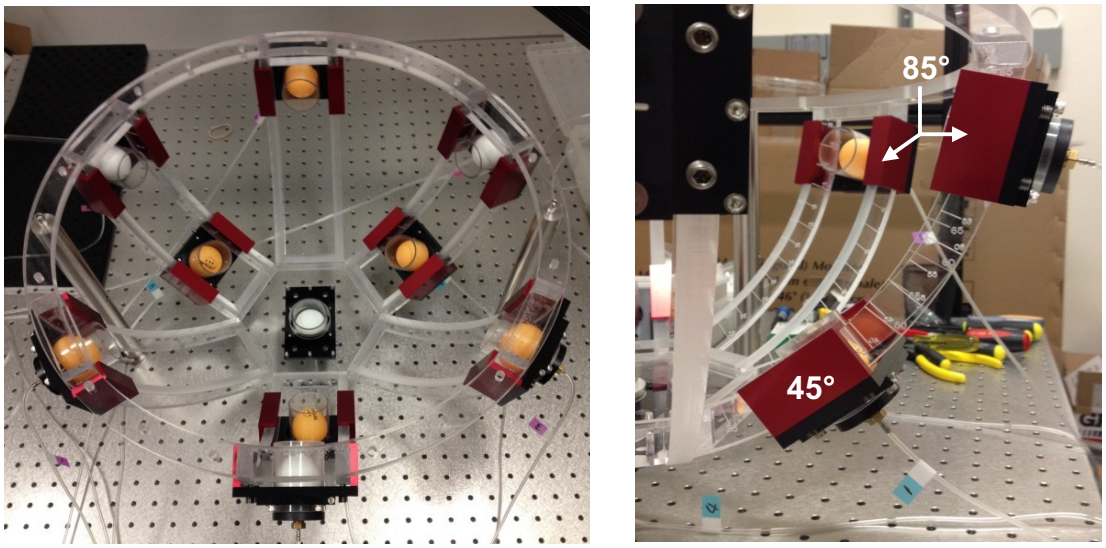


Figure C4. Air cannons and bracket systems on the frame.

C.2.4.3 Mounting the Treadmill

1) Connect the post pedestals with the 12" posts with set screws

Pedestals come with hollow set screws. Those hollow screws are designed to make it able for a screwdriver to go through to access other screws. However, I find these hollow screws easy to break when tightening to the post. Also, it can be difficult to take any broken pieces out of the hole. So I suggest to use conventional solid set screws in this case.

2) Put posts at their approximate position with clamp forks

The plastic treadmill frame is very easy to deform, so use the following sequence to install:

1. Tighten one fork
2. Use a ¼-20 screw to fasten the treadmill frame to that post
3. Fasten the frame to other posts
4. Slightly adjust the position of other posts to release any torque and then tighten their forks

C.2.4.4 Gluing the Treadmill Half Spheres

NOTE: Always handle foam spheres gently - they are very delicate!

1) Weigh the Half Spheres

Carefully take sphere halves out of package. Weight each one with a scale. They should be 32-33g as required. They may range from 30-35 g. Use a marker pen to write down the weight on the inner surface. Halves with the same weight should be glued together to make sure the sphere will have uniform weight distribution.

2) Prepare the Aluminum Ring

Cut several pieces of parafilm (each spans approximately five sections). Use the parafilm to cover the inner surface of the ring's rim (Figure C5). Make sure there are no gaps.

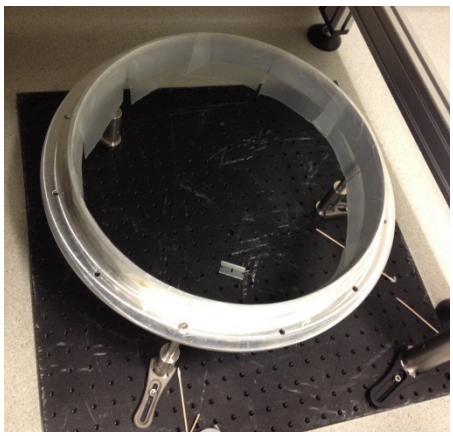


Figure C5. Cover the aluminum ring with parafilm.

3) Initial Gluing Process

Shake the foam glue (from FoamFusion) while holding the cap. Use a Q-tip or needle to open the tip if necessary. Squeeze some glue into a weighing dish. Take one half sphere and use a cotton-tipped probe to carefully cover the rim with a layer of glue (Figure C6). If the glue drips onto the outer or inner surface, quickly remove it.

Put the half sphere into the ring. Let the bottom touch the floor (Figure C6). Make sure it is level (that is, not tilted). Push the other half into to ring until rims of the two halves firmly touch each other. Carefully swing the plastic plate to the top of the sphere to apply light pressure. The height of this plate was pre-set so that it should just touch the top. If the plate is blocked by the sphere, it means the two halves may not touch each other at all positions. Turn the black threaded ring (Figure C6, red arrow) 1-2 rounds to move the plate upwards a little. This will release extra pressure on the sphere. Leave the sphere overnight to dry.

Figure C6. Initial gluing of foam sphere.

4) Second Gluing Process

After the sphere is dry from the initial gluing, carefully take the sphere out of the ring. Peel off the parafilm. A substantial portion of the rim may not be actually glued. This is OK as long as the two halves are seamlessly glued together at some positions. Tip: Use some small pieces of tape across the seam to hold the sphere during this second gluing step.

Put something (box, foam block etc.) on the floor plate at the ring's center and seat the sphere on top so the seam won't touch the ring (Figure C7). Use a cotton-tipped probe to cover the unglued seam with a thicker layer of glue. Wait for 2-3 min to let the glue soak into the gaps and settle. Gently brush the seam with the probe to flatten the glue. Be careful not to let the glue touch the pieces of tape. It may be helpful to place the sphere in a way that the seam plane is perpendicular to the floor. In this case, glue the top half of the seam, wait for it to dry, then rotate to do the other half. This also allows gravity to help get the glue into the seam.

Wait 4-6 hours until the glue is dried. Gently remove all tape. Repeat the step above to apply the glue on previously taped areas and any gaps. Make sure no gaps remain as in the case of rodents, the subject's paw can easily get stuck into a gap.

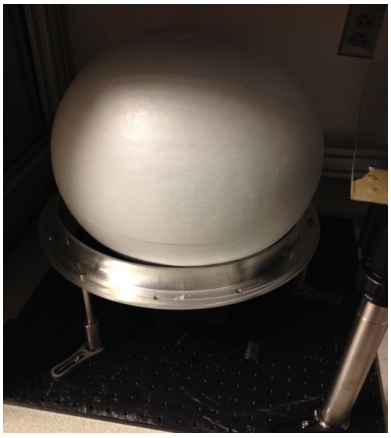


Figure C7. Second gluing process.

C.3 Basic Setup and Calibration

C.3.1 Treadmill Tracking System and Initial Calibration

Also see the original treadmill tracking system documentation LottTreadill_AssemblyGuide.pdf, located in the folder:

…\MouseoVeR\Treadmill_Tracking\

1) Use the metal parts to mount cameras onto the ball frame (toward the VR display side). Each camera is placed at a 45° angle from the front facing roll axis. Cameras should be well aligned so that their axes point towards the center of the spherical treadmill. The surface field of view for each camera should be set at approximately 4 x 4 mm^2.

2) Plug the two cameras into the MCU board, and plug the MCU board to the VR computer using a USB cable.

3) The IR spotlight used here has a sensor at the center. The light turns on only when the environment is dark. Unscrew the glass lid and cover the sensor with opaque tape or aluminum foil so that it will be constantly on when powered. Use Thorlabs parts to install spotlights next to each camera.

4) Install MATLAB on the VR computer. Use MATLAB code for calibrating the tracking system:

   …\MouseoVeR\Treadmill_Tracking\TreadmillDoc\TreadmillDocumentation\Software \Matlab Code\

5) Download treadmillVideo.m and motiondisplay.m.

6) Turn on IR spotlights and the MCU board.

7) Turn on the air to float the treadmill – or create the conditions used during experimentation.

8) Run treadmillVideo.m. You will see images acquired by the two cameras (polygon-shaped texture of the foam ball. Each camera has two adjustable rings. The back one is the aperture while the front one is the focus. Loose screws on both and optimize the focus. Adjust the positions of IR spotlights to achieve maximal illumination. And then adjust the aperture to optimize the contrast of images. Sometimes you may see one or two bad pixels on the image. It is typically a bug of the MCU board but not cameras, since it will be gone when you refresh the video. It is not a problem for our use.

9) Close treadmillVideo.m. Run motiondisplay.m. Select the rotation mode. You should see a ball model at the center and two bars for each camera (one for shutter speed and the other for signal quality). Adjust focus and aperture to further optimize the signal quality ("squal"). Keep the shutter speed below the line shown on the bar to have the system work properly. The shorter shutter speed the better. But do NOT open the aperture too much to achieve shorter shutter speed. Large aperture causes limited view depth, which may make the image quality very bad when the ball is jittering. In my system the shutter speed is ~60 µs (this is short enough for our purpose). Once everything is optimized, fasten screws on the camera.

10) Troubleshooting: If you cannot see any images with treadmillVideo.m, check all connections. Make sure IR spotlights and the MCU board are turned on. Adjust aperture and focus (it could be a saturation or out-of-focus issue). If you can see good images but cannot run motiondisplay properly (no shutter speed or signal quality are shown, the MATLAB code says "Packet dropped"), it is most likely a jumper problem. Turn off the MCU board and unplug the USB cable, open the case, and see if it the jumper is set correctly.

11) Rotate the spherical treadmill by hand. See how the model ball on the screen follows. If it rotates in the wrong direction, most likely it is due to wrong camera orientation or reversed connection to the MCU board. Check the camera orientation (cable should be on the right side). Switch camera plugs on the MCU board. Switch to the **independent motion** mode. You should see trajectories of image flow through each camera. Rotate the treadmill to ensure both trajectories are correct. Note that if you rotate the treadmill side-to-side (about the roll axis), the image flow should be up for one camera and down for the other.

Appendix D

JOVIAN and MouseoVeR Software

Mark A. Bolstad, Jeremy D. Cohen, Albert K. Lee
Janelia Research Campus, Howard Hughes Medical Institute, Ashburn, VA, 20147

D.1 Introduction

JOVIAN (Janelia Open VIrtual reality eNvironment) is a C++ library designed as a flexible set of software components. JOVIAN is built from a number of open-source software components that work together to generate a system that allow a user to create configurable experimental visual environments. MouseoVeR is a branch of JOVIAN, refined for use with small rodents. Additional branches may be available, such as FlyoVeR, which is refined for use with Drosophila. Figure D1 shows the various packages and their dependencies required to build and run JOVIAN.

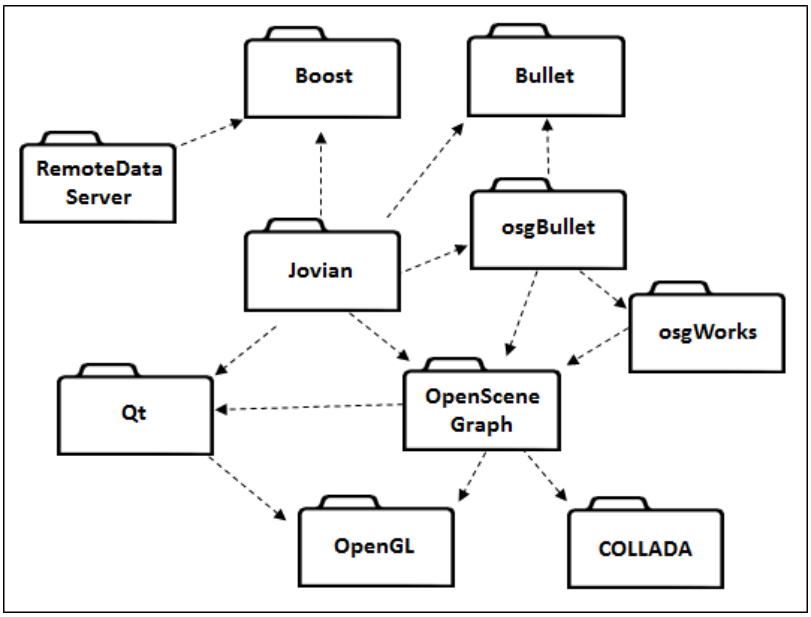


Figure D1. System level dependencies amongst the JOVIAN components. Arrows show an "include" (or "build") dependency. For example, OpenSceneGraph requires OpenGL and COLLADA to build, and osgBullet requires Bullet.

D.1.1 Overview

JOVIAN is built around a standard Model-View Controller (MVC) architecture. The controller (here called the "Console") is responsible for all user-interaction and configuration that is necessary to configure the displays and system for the current experiment. The Scene_Model is responsible for all parts of the system related to the internal representation of the graphics and physics related data. The model takes requests from the Console and configures them for use by the Viewer (viewingWindowQt) and the various external subsystems (OpenSceneGraph and Bullet). In an ideal world there would be strict separation between the MVC components, but some functionality has been moved between the components to either satisfy convenience or performance. For example, the Viewer handles frame input/output (I/O) reporting such as time, position, speed, and collisions. The following sections will discuss each component of the MVC architecture in more detail. Figure D2 shows an overview of how the various system components interact.

D.1.2 Console Graphical User Interface (GUI)

The Console GUI is the controller of the system and the primary way the user interacts with the system. Figure D2 shows the basic flow of information through JOVIAN. The two main inputs to the system are 1) the RemoteDataServer (RDS), carrying treadmill motion information, and 2) the user, via widgets in the Console GUI. Briefly, during each display frame rendering cycle, the Scene Model is updated by information from the RDS and the Console GUI. JOVIAN updates the Scene Model, captures an image from the perspective of the virtual camera according to the configuration settings, and then sends the image to the Viewer in order to update the displays.

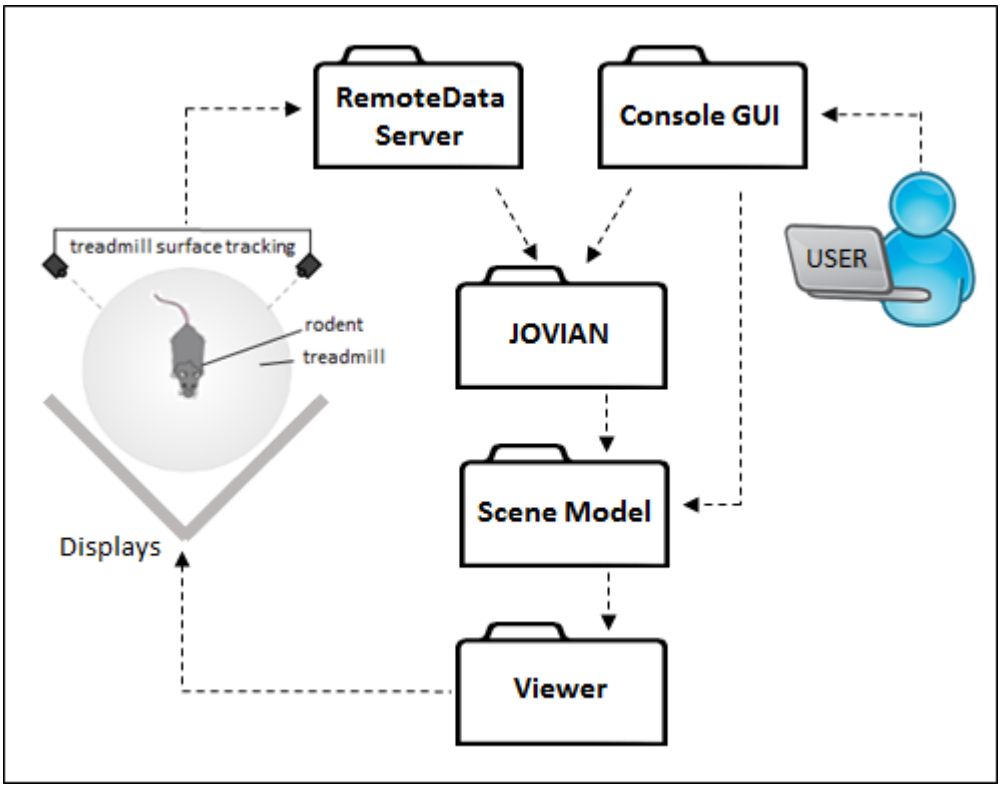


Figure D2. JOVIAN system overview showing the interaction of the high-level components of MouseoVeR and the virtual reality behavioral system. Arrows show the basic flow of information through the system.

The target platform is Windows X (currently X=7). However, the system was built for cross-platform compatibility. Accordingly the GUI toolkit used to implement the Console GUI is Qt [1]. Qt, like most GUI toolkits, uses a collection of callbacks to connect a widget's action or to a user-defined behavior. In Qt's terminology, the widget's action "emits" a "signal" which is then caught by a "slot." As a class, the "Console" has a minimal set of public methods. The public interface consists of a handful of queries (the hostname, the port used for communication, etc...) and a small set of commands (enable/disable motion, blank the display, etc...) that needed to be exposed so they can be accessed through keyboard shortcuts. Most of the class is in private (not visible outside of the class) slots that are attached to the signals emitted by the widgets. The majority of these methods are to take data from the user's action and from other parts of the Console GUI and transfer it to the Scene Model, serving to update the Viewer.

D.1.3 Scene Model

The **scene model** contains the internal representation of all data required for the system's operation. More than half of the member functions in this class are for processing Collada files – the file format used by JOVIAN to process data from modeling programs such as Blender. As schematized in Figure D3, we utilize the OpenSceneGraph Collada parser to ingest the data and generate a preliminary scene graph. The scene graph can best be thought of as an N-branch tree structure with the intermediate nodes being either Group or Transform nodes, and the leaves being some type of Geometry (a "Geode" node in OpenSceneGraph terminology). For the next phase of input, we traverse the tree in a depth-first manner, accumulating the transforms between the root node and each geometry leaf. For each piece of geometry, we compute the centroid, translate the centroid to the origin, and then proceed to transform the vertices and normals of the geometry by the accumulated matrix. Once all the geometry has been processed, to extract whether it is to be passed to the Bullet physics system, we parse the geometry based on the name (see section 2.0). If it is physics-enabled (for example, see section 2.0.5), it is prepared and passed on the physics processing routines (Bullet). Once all the processing is complete we create a new set of transform nodes and put the transformed geometry into the scene graph.

Parameters for the Console GUI are saved and loaded from the configuration file system. When switching between scene models, we use the auxiliary class 'Config_Memento', incorporating a software design pattern that provides the ability to restore an object to its previous state [2].

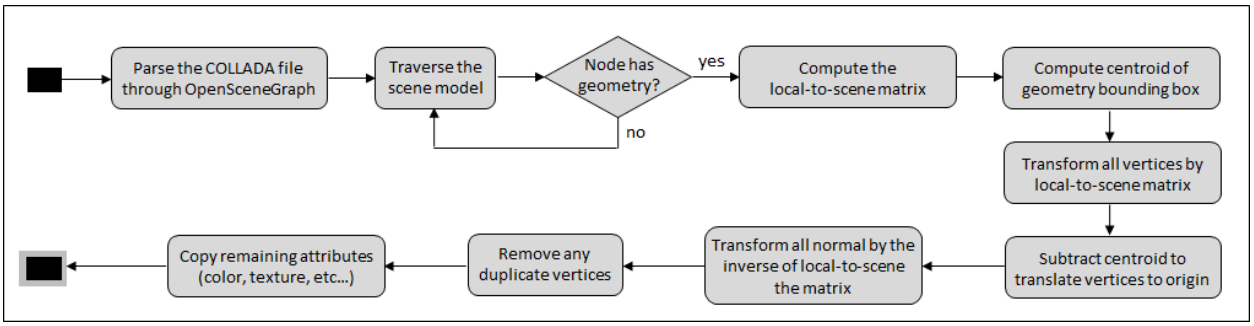


Figure D3. Processing of COLLADA input files (for example, a Blender scene model). The generated scene model is traversed, "flattening" the transforms and removing duplicate vertices, in order to prepare the geometry for processing by the physics engine, Bullet.

D.1.4 Viewer

The Viewer strictly takes the scene graph as provided by the model and renders it to the display(s). The Viewer uses render-to-texture to enable the distortion mapping the user requests from the Console GUI. Render-to-texture is a technique in which the scene graph is first rendered into an image map in memory (that is, a texture), instead of sending the rendered scene graph directly to the display device. We then map that texture onto an array of contiguous screen-aligned quadrilaterals ("quads" in OpenGL terminology) that have been distorted to match the Display Configuration widget in the Console GUI. The majority of the methods in this class are to set the mapping between the requested field of view (FOV), the number of displays, and the requested distortion to the user's physical display configuration.

Additionally, there is support in the viewer for a high-speed mode – a special rendering mode initially designed for insects with high fusion flicker rates.[1] For example, using specially modified projectors that have had the color wheel removed, the high-speed mode generates one frame of camera movement for each of the color channels, effectively tripling the frame rate (and removing the capability to render in full color).

D.1.5 Information Flow Logic

JOVIAN has essentially two main loops that run alternately. The first is the Qt event loop that processes all events from the Console GUI and transfers them to the appropriate callbacks. The second is the rendering loop that runs as a timeout event (that is, an event that schedules itself every N milliseconds) within the Qt loop. In theory, this means that a long running event from the Console interface will stall the rendering (for example, loading a new scene model). However, in practice, it has not been an issue to date.

The rendering loop, shown in Figure D4, is at the core of the display system. The JOVIAN render loop is nearly identical to the inherited methods from osg::Viewer, with some modifications for the physics engine, Bullet [3], and for the high-speed frame mode described above.

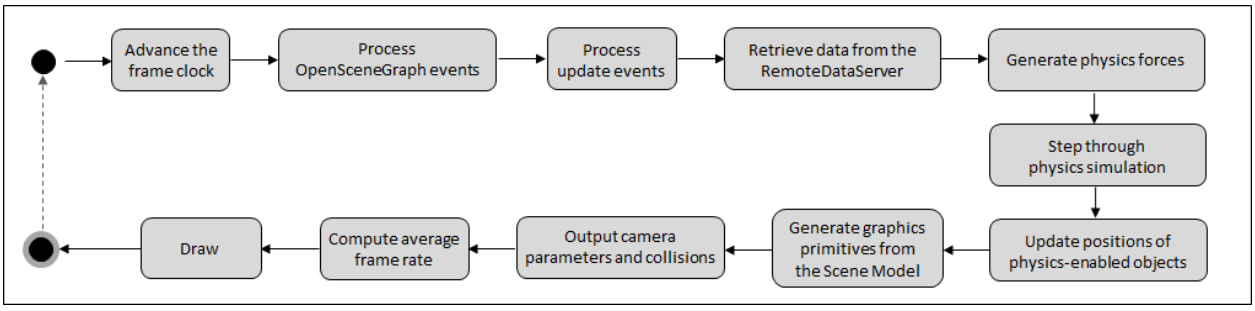


Figure D4. The rendering loop

[1] Fusion flicker rate is the minimum refresh rate of a display such that the observer can no longer discern individual frames.

D.1.6 Motion Computation and Heading Direction Control

For each iteration of the rendering loop (Figure D4), Jovian communicates to the RemoteDataServer (RDS) (see D.1.7) to retrieve the accumulated motion values since the last request (Figure D7). The values coming from the RDS are defined as Euler angles in RDS's coordinate space. We require the user to create mappings of 180-degree rotations of the treadmill along each of the three axes in real-world space into RDS's coordinate space. This is the treadmill calibration process (see section 1.4.4, and Appendix C – C.3). We then use the inverse of these values to scale the raw treadmill inputs into real-world space, where the Euler angles are converted into a quaternion [4]. From the quaternion we extract the axis and magnitude of rotation of the treadmill. As we know the radius of the treadmill (parameter is set in the Console GUI), we convert the rotational magnitude into an arc length to create a vector of the magnitude of motion. This motion vector is then passed to other routines for manipulation before being sent to the physics engine – Bullet [3]. These other routines are described in the sections below.

D.1.6.1 Basic Motion

If MouseoVeR and the treadmill tracking system are configured as described in this document, motion about the Z-axis (roll) moves the camera side-to-side (trucking), motion about the X-axis (pitch) moves the camera forwards and backwards (dollying), and motion about the Y-axis (yaw) rotates the camera heading direction (panning). Basic motion is a form of manual-heading control (Figure D5), as the degree of heading direction is determined by the behavior of the subject on the treadmill.

Each axis of rotation is also controlled by a user-defined gain value, set in the Console **Calibration** tab (see section 1.4.4), where the value acts as a multiplier to the magnitude of the raw treadmill rotational component. For example, a Y-axis (yaw) gain value of 0.0001 would effectively turn off the rotational yaw component (panning), permitting only forward, reverse, and side-to-side motion, with no turning. A Z-axis (roll) gain value of 2.5 would increase the side-to-side motion by a factor of 2.5.

NOTE: Use of basic motion yaw-based heading direction control (Figure D5, top) requires that the gain of the Y-axis (yaw) rotational component be set to a value greater than 0. In this case, the degree of rotation about the Y-axis is used to compute the change in the virtual camera (see section 2.1.1) heading direction in the scene model.

The gain values for each axis of rotation can also be updated via collisions between the virtual camera and a special user-defined object in the scene (see section 2.0.13 and 2.2.3).
In association with basic motion, there are three potential modes of navigation in the scene: 1) Free motion-based heading 2) Trajectory-based heading, and 3) Path-based heading. Each mode is described in D.1.6.2, D.1.6.3, and D.1.6.4, respectively.

D.1.6.2 Free Motion

In association with basic motion (see D.1.6.1), free motion computation is the simplest of the four available modes. First, the motion vector is transformed into the Scene Model coordinate system. Then, if the user has requested, for example, that the velocity is to be smoothed (set in the Console **Configuration** tab, see section 1.4.5-[1]), we compute the smoothed velocity for the time window, update the motion vector, and pass it into Bullet.

In general, we set the linear velocity of the camera to the velocity scaled by the frame rate, and pass that value to Bullet. However, one optimization is that we first estimate the distance to be moved, and then use Bullet to check for any collision objects within that distance. If there are no collisions, we physically place the camera at the computed distance and have Bullet apply gravity. This method is much faster than in the case of when we do detect any collisions, for which we simply set the linear velocity, and let Bullet compute the new camera position.

D.1.6.3 Trajectory-based Heading

Another form of manual-heading control is the trajectory-based heading mode (Figure D5, bottom). Trajectory-based heading is a user-defined extension that can be applied in addition to basic motion (see D.1.6.1). In this mode, the magnitude of the angle that the subject is running off the Z-axis (roll) centerline of the treadmill (running trajectory) is used to determine an angular rate for turning (rotating about yaw) in the virtual scenes.

NOTE: This mode was designed to allow for configurations where the user sets the gain of the Y-axis (yaw) component to 0 (set in the Console **Calibration** tab, see section section 1.4.4), eliminating the ability of the treadmill yaw rotation to control heading direction in the scene. The user may then use trajectory-based heading to solely control the rate of yaw rotation in the scene. However, the user can apply a mix of the raw treadmill yaw-based and trajectory-based heading direction controls as desired.

To enable trajectory-based heading, in the Console **Heading Direction** tab, check the **Trajectory Heading Direction** checkbox (see section 1.4.6-[1]). When enabled, the ratio of the treadmill's raw pitch to roll values (the running trajectory) is first computed, then an angular rate of rotation about the Y-axis (yaw) is determined by a user-defined lookup table, set in the Console Heading Direction tab.

We can additionally smooth the magnitude of the rotational component, as well as the rotational component at three different locations before sending the data to Bullet. Each of the smoothing components is under user control with the parameters set in the Console **Configuration** tab (see section 1.4.5-[1]), defining whether to perform a particular smoothing operation, and by how much. The first smoothing location is on the raw Euler angles from the RDS. The second location is after the computation of the trajectory-based angular rate. The third smoothing location is after the blending of the trajectory-based angular rate with the free motion-based Y-axis (yaw) rotational component. The motion vector is then updated and passed into Bullet, with the remaining steps being identical to those described for free motion heading direction control. In this mode, the computation of the magnitude of motion (via the

quaternion calculation) is briefly delayed to prevent the duplicate computation of some of the intermediate variables that are needed for this specialized mode.

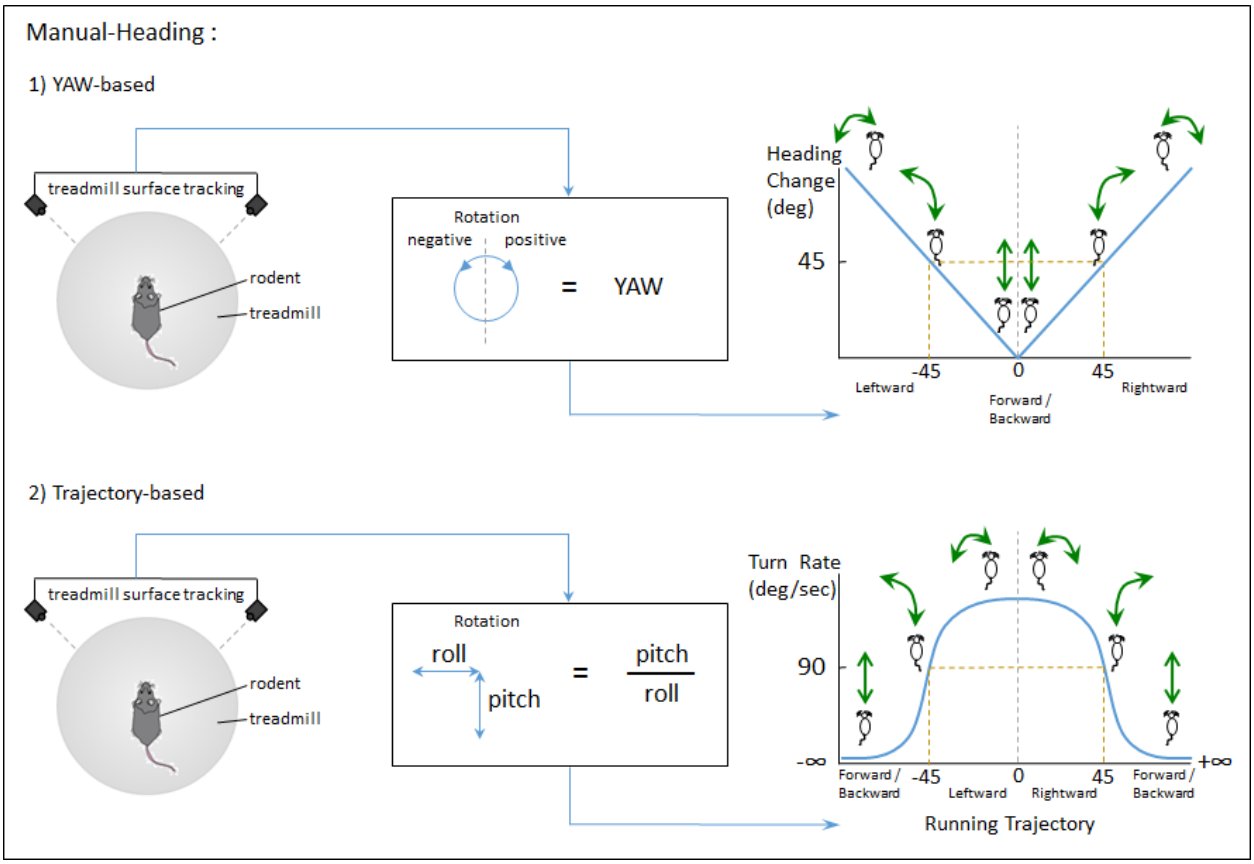


Figure D5. The two modes of manual-heading direction control. 1) Yaw-based heading uses the treadmill Y-axis (yaw) rotational component to determine the degree of heading direction change. For this example, the Y-axis gain (set in the Console **Configuration** tab) is set to a value of 1. 2) Trajectory-based heading uses the ratio of treadmill motion in the pitch versus roll axes (Running Trajectory), which is then converted into a rate of rotation (Turn Rate) about the Y-axis (yaw) in degrees per second. The user sets the I/O function that determines the conversion of running trajectories into heading direction turn rates in the Console **Heading Direction** tab.

D.1.6.4 Path-based Heading

When the Collada scene file contains a specially labeled path object (see section 2.2.1), a different computation for motion and heading direction is carried out. Path-based motion is a form of auto-heading direction control (Figure D6) as it serves to guide the virtual camera (see section 2.1.1) heading direction at each location in the scene model. Path-based heading (Figure D6) adds two additional restrictions on motion. First, the crossbar (see section 2.2.2) width (a legacy term from the first version of the software) is the distance that the camera is allowed to travel perpendicular to the path. Second, path-based heading is a mode where raw treadmill turning inputs can be ignored (or combined), and the camera's heading direction is continuously aligned parallel to the path in the direction of motion (that is, parallel to the tangent at the nearest point to the path).

The virtual camera can be configured to use auto- (that is, path-based) heading control as described in this section, manual- (that is, yaw-based and trajectory-based) heading control (see D.1.6.1. and D.1.6.3), or some blended combination of the modes. The amount of contribution from auto-heading and manual-heading control is set in the Console **Heading Direction** tab (see section 1.4.6) using a slider that moves from a position of 0 (meaning 100% auto-heading direction control) to 100 (meaning 100% manual-heading direction control). For example, a slider value of 75 sets the mixing function to 25% auto-heading and 75% manual-heading direction control, and the final motion vector's rotational component is computed as the combination of the all heading vectors by their relative weights:

Heading_Direction = [[Auto_Heading * 0.25] + [Manual_Heading * 0.75]]

As in the free motion scenario, we first transform the motion vector into the Scene Model coordinate system. Then we compute the distance that the camera is projected to move along the path and perform interpolation to find the control points on the path that bound the projected point. We then extract the rotations from the control points and apply them to a special invisible physics object called the "sliding body." The sliding body, as the name implies, can only slide back and forth along the path. The virtual camera is then tethered to the sliding body (in Bullet terminology, the tether is called a Generic 6 DOF Constraint). The length of that tether is the crossbar width, and it determines how far to each side of the path the camera can wander.[2] The crossbar can be an object in the scene with a special name (see section 2.2.2), or a variable that is set in the Console **Configuration** tab (see section 1.4.5-[4]). Setting or updating the variable in the configuration file will override the crossbar object in the scene model.

After we have determined the position and rotation of the "sliding body," we need to update the virtual camera's position and rotation. Just as in the free motion case, we can perform smoothing on input velocity and any additional smoothing on the rotational component of motion. We can smooth the rotational component at three different locations before we send the data off to Bullet. Each of the smoothing components is under user control, deciding whether to perform a particular smoothing operation, and by how much. The first smoothing location is on the raw Euler angles from the RDS. The second location is after the computation of the auto-heading (path-based) component. The third smoothing location is after the blending of the auto-heading angle with any manual- (that is, trajectory-based heading and/or basic yaw-based) heading direction control. The two component vectors (auto- and manual-based heading directions) are then blended together as described above, and processed for Bullet, as in the free motion mode.

[2] In actuality, the length of the tether describes the radius of a circle that bounds the extents of the camera's motion. However, since we are updating the sliding body on every frame, the camera is practically perpendicular to the path measured from the centroid of the sliding body.

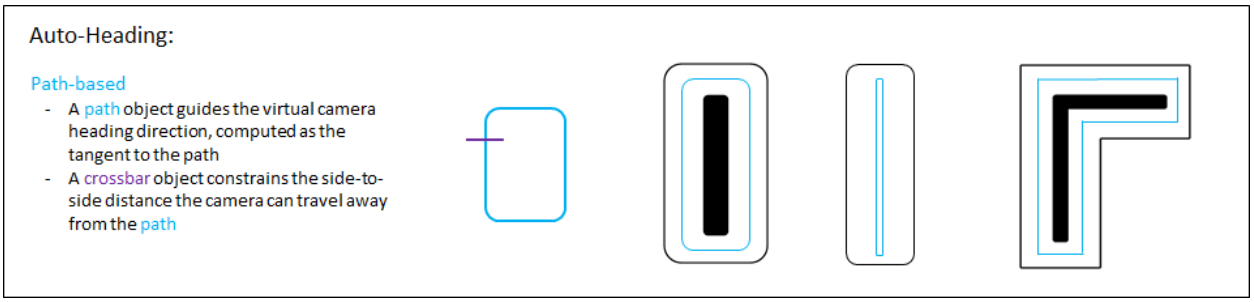


Figure D6. Auto-heading direction control: Path-based heading. For path-based heading, the crossbar width value is set by a crossbar object in the scene (see section 2.2.2), or in the Console **Configuration** tab. For example, the 3rd diagram represents a straight track that turns the view around 180 degrees when reaching each end.

D.1.7 Remote Data Server (RDS)

The RemoteDataServer (RDS) is a stand-alone C++ application that interfaces between the treadmill tracking system (see Appendix C – C.3) and JOVIAN. The RDS does not have any direct dependency on the JOVIAN library. This means it can be used independent of a JOVIAN-based system. The only requirement for JOVIAN is that the RDS provide a set of Euler angles for the movement since the last frame.

Seven arguments are retrieved from the RDS in the following format:

roll, pitch, yaw, cam1_dx, cam1_dy, cam2_dx, cam2_dy

Roll, pitch and yaw are the three Euler angles, followed by the camera_1 image raw change in X (dx) and Y (dy) motion, and the camera_2 image raw change in dx and dy.

The RDS relies heavily on mutex threading [5] and the Boost Asynchronous I/O (asio) library [6] to provide high throughput processing of the treadmill tracking camera data. Figure D7 schematizes how messages are passed between JOVIAN and the RDS components over time.

Briefly, the workflow is as follows. The RDS is initialized via the function 'main', which initializes the class 'data_interface'. The 'data_interface' communicates with the mutex to ensure a secure environment for data retrieval from the camera tracking system (see Appendix C – C.3).

Communicating occurs via the classes 'tcp_server' and 'session' with 'boost asio', which provides continuous asynchronous communication with the camera tracking system. On each frame draw callback, JOVIAN sends read commands to 'session', which reads the data from 'boost asio', and then writes it back to JOVIAN. In Figure D7, The short boxes under 'session' represent the handshaking between 'tcp_server' and 'session', ensuring complete data transmission, and the long boxes represent the read/write cycles between JOVIAN and the camera tracking system via 'boost asio' and 'session'.

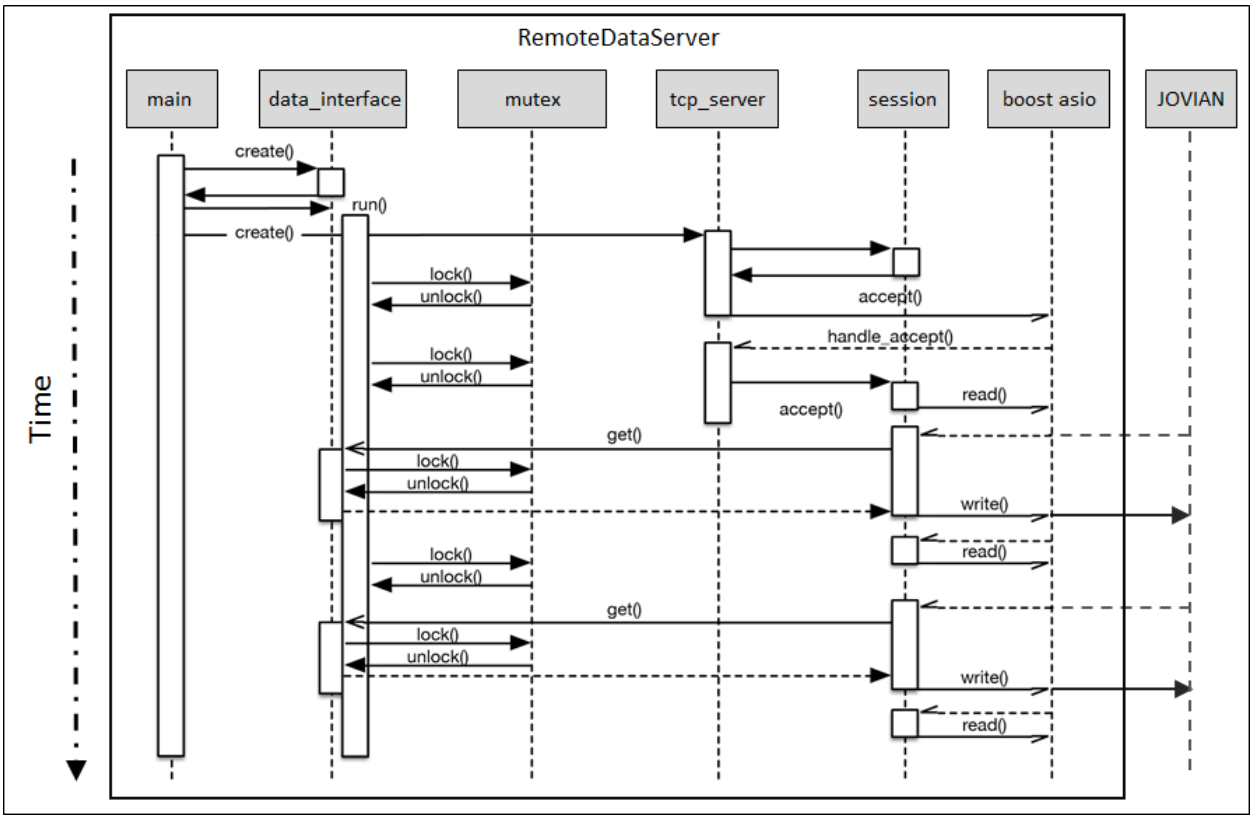


Figure D7. Sequence diagram for the RemoteDataServer (RDS) and its communication with JOVIAN.

References

1. http://www.qt-project.org.
2. Erich Gamma, Richard Helm, Ralph E Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
3. http://bulletphysics.org
4. For more information on Euler angles, quaternions, and the conversion process, see https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles)
5. https://en.wikipedia.org/wiki/Mutual_exclusion
6. http://www.boost.org/doc/libs/1_62_0/doc/html/boost_asio/overview/core/basics.html

Acknowledgements: We thank Yuko M. Kawamura for editing the MouseoVeR documentation.