FlyVR Documentation

Release 0.9

Andrew Straw

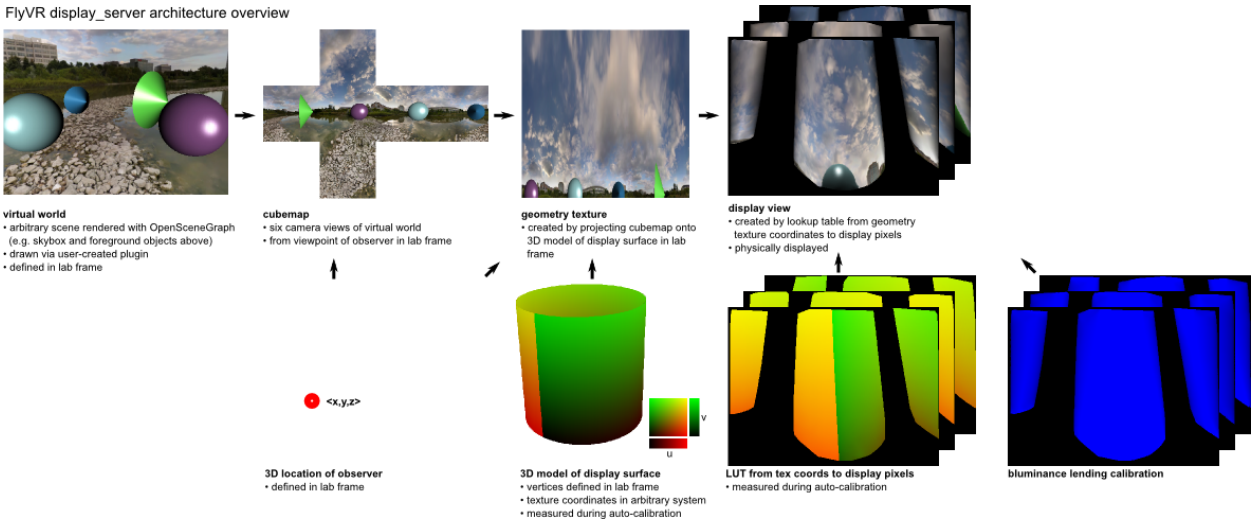
August 27, 2016

Contents

FlyVR is a virtual reality engine built on ROS and OpenSceneGraph. It manages multi-computer realtime tracking and display with the goal of being useful for scientific studies of vision.

CHAPTER 1

FlyVR pipeline overview



1.1 Installation and Getting Started

FlyVR is developed and tested on Ubuntu 12.04 on the amd64 architecture using NVIDIA graphics.

1.1.1 Installation

As the root user, in the bash command-line in Ubuntu 12.04, run `this script`. (Also, you can download the latest version of this script from here.) From the command line, this should work:

```
curl -sL https://raw.github.com/strawlab/flyvr/master/docs/install-flyvr.sh | sudo bash
```

1.1.2 Getting Started

Testing the basic installation

Once FlyVR is installed, you should be able to run a quick demo by typing:

```
# setup ROS environment variables
source /opt/ros/ros-flyvr.hydro/setup.bash
```

```
# launch the demo
roslaunch flyvr demo_display_server.launch
```

If that opens a graphics window showing a 3D scene, FlyVR is installed and running properly.

Configuring a new display

The most important part of FlyVR is the Display Server. This is the program that draws on a single display. If you need multiple physical displays, you will run multiple display servers. (A single display server can drive multiple virtual displays, as explained in the glossary.) We need to tell the Display Server about your display.

FlyVR uses ROS to handle configuration. To bootstrap a new system, begin by copying a default configuration file into a new location:

roscd flyvr/config cp rosparamconfig.yaml my_display.yaml

Edit this new `my_display.yaml` to reflect your display. Much of this YAML file should be self-explanatory. On initial setup, the most critical information is the contents of the `display:` key are the X windows parameters `displayNum` and `screenNum` and the window geometry parameters `x`, `y`, `width`, `height`, and `windowDecoration`. FlyVR does not switch your graphics mode, so set these values such that the display server will completely utilize your display.

You can test your new configuration by creating a new ROS launch file which will load these parameters.

roscd flyvr/launch cp demo_display_server.launch my_test.launch

Edit this new `my_test.launch` file and change the name of the configuration .yaml file to refer to the file you created above. Now, run this new launch file:

roslaunch flyvr my_test.launch

The displayed window should now have the properties you specified in `my_display.yaml`.

Running the pinhole calibration wizard

(To be continued...)

1.2 Using the joystick for input

Several FlyVR programs use a joystick for input (e.g. `joypose`, `joystick_cursor`, `spacenav_pose`, `pinhole_wizard.py`). Specifically, they listen to the ROS `/joy` topic. To ensure that your joystick is running, you can do this from the command line:

rostopic echo /joy

1.2.1 Using a web browser instead of a physical joystick

If you don't have a real joystick, you can run an emulated one:

rosrun browser_joystick web_control.py

This starts a webserver running on the local machine and prints the URL. Open this URL with a modern browser and the webserver should now emit messages on the ROS `/joy` topic.

Note: this requres the installation of python-tornado (> 2.4.x) and python-sockjs-tornado packages.

1.2.2 Using a physical joystick

If your joystick is device `/dev/input/js0`, use ROS to emit `/joy` messages like this:

```
rosrun joy joy_node /dev/input/js0
```

Using a PS3 joystick

A PS3 joystick can be run like a physical joystick, but there are a couple of tricks to get it connected. The ROS `ps3joy` package facilitates this. The required steps are:

For initial setup, perform bluetooth pairing with the joystick (Use the `sixpair` program.)

Then, for daily use, run the bluetooth listener which write the output into the linux device system. Run `python ps3joy.py`.

Contents

1.3 Theory of operation

A moving observer has a pose within a global coordinate frame. Objects within the global frame may also move or be updated (e.g. a moving grating). Six camera views with a fixed relationship to the observer are used to build a cube map, showing the scene surrounding the observer without regard to the projection surface.

This cube map is then projected onto a 3D shape model of the display surface. From there, this image is warped to the physical display output.

1.4 flyvr nodes

1.4.1 display_server - the FlyVR display server

The FlyVR display server node runs locally on the computer(s) connected to the physical display. During a typical experiment, it will be running an experiment plugin. Each experiment plugin updates the graphics engine on the basis of the fly's current position. Given the scenegraph and the calibrated screen layout, the node will compute the images shown on the projectors.

1.4.2 viewport_definer.py - FlyVR viewport definer

Runs a GUI program that allows the user to interactively define the viewports for all connected projectors.

1.5 Glossary

Display Coordinates - the native pixel indices on a physical display. These are 2D.

World Coordinates - the 3D coordinates in lab space of physical (or simulated) points. (May also be represented as a 4D homogeneous vector *x,y,z,w* with nonzero *w*.)

Physical Display - a physical device capable of emitting a large, rectangluar block of pixels. It has display coordinates - the 2D locations of each pixel. (A physical display does not have world coordinates used for the VR mathematics. On the other hand, A virtual display does have world coordinates.)

Virtual Display - a model of a physical display which relates world coordinates to display coordinates. The model consists of a linear pinhole projection model, a non-linear warping model for lens distortions, viewport used to clip valid display coordinates, 3D display surface shape in world coordinates, and luminance masking/blending. Note that a physical display can have multiple virtual displays, for example, if a projector shines onto mirrors that effectively create multiple projections.

Viewport - vertices of polygon defining projection region in display coordinates (x0,y0,x1,y1,...). It is used to limit the region of the physical display used to illuminate a surface. (The FlyVR Viewport corresponds to a 2D polygon onto which the image of the projection screen is shown.)

Display Surface - a physical, 2D manifold in 3D space which is illuminated by a physical display (either by projection or direct illumination like an LCD screen).