Camera Usage Notes

March 17, 2011

Here are the directions for using the camera mounted on Jinx. We've setup the Logitech Webcam Pro 9000 currently mounted on the front. It is relatively secure, but try not to bump the camera, as the tilt angle is difficult to keep constant with it's current mount. Any change in tilt will require a recalibration of everybody's extrinsics for the plan view (bird's eye, top-down view, etc) transform. If you do bump the camera, please send a note to Eric and Chad (there is a mailing list for us... thechad@case.edu ...) so that we can notify the rest of the class that their extrinsics need recalibrated. This is also a motivation for you to develop a straightforward extrinisic calibration process that is as automated as possible - if all you have to do is wave the pole+puck around for a bit and then out pops a transform, that's better than a bunch of manual munging in Matlab.

We have already calibrated the camera's intrinsics, so you will not have to worry about dewarping the images if you subscribe to the proper topic.

1 Running on the robot

In an empty screen after carrying out the normal directions for starting up the robot, run the following:

```
roslaunch cwru_vision start_cameras.launch
```

This launch file with automatically start the appropriate cameras for whichever robot you are running on (the Logitech for Jinx or a Firewire camera if you were running on Alen or HARLIE) with the appropriate intrinsic calibration. Let's take a look at the launch file that actually starts up the camera so that you have a clue what's going on (called launch_cameras.launch). You shouldn't launch this file directly, but it is useful to know what it is actually doing.

```
<launch>
<node name="front_camera" pkg="uvc_cam" type="uvc_cam_node" respawn="true" >
    <param name="frame_rate" value="30.0" />
    <param name="camera_info_url" value="file://path/to/intrinsics.yaml" />
    <remap from="camera" to="front_camera" />
```

```
    <param name="camera_name" value="front_camera" />
    <param name="frame_id" value="front_camera" />
    <param name="exposure" value="3" />
    <param name="brightness" value="120" />
    <param name="contrast" value="30" />
    <param name="saturation" value="40" />
  </node>
  <node pkg="image_proc" type="image_proc" name="proc" ns="front_camera" />
  </launch>
```

You'll notice that we launch two nodes.

One of those nodes is the *uvc_cam_node* in the uvc_cam package. This is the actual camera driver node and is responsible for grabbing frames from the camera and publishing them on its *image_raw* topic along with the intrinsics that go along with that image. These two must be synced exactly in time - while most of our cameras have static intrinsics unless someone physically adjusts the lens, a camera that had a motorized focus or other on-the-fly lens adjustment would have variable intrinsics, so any camera dewarping code needs to know the intrinsics that go along with each publsihed image. Like all cameras, our UVC compatible Webcam Pro 9000 has many, many parameters, some of which we set. I'll go through what some of these mean in Table 1 on page 6. We also remap the *camera* namespace to *front_camera*, meaning that all topics for this particular camera node will be published like */front_camera/image_raw* or */front_camera/camera_info*.

The second node is an *image_proc* node from the image_proc package. As described on that page, this node subscribes to the raw image/intrinsics pairs from the camera driver and makes a number of different topics available based on this information. The one that you will care about right now is the */front_camera/image_rect_color* topic. *image_proc* provides a rectified, color version of the raw image on this topic. By subscribing to this image instead of the raw image from the camera driver, you won't have to worry about applying the camera intrinsics (rectifying) before doing further processing. Note that *image_proc* is smart and will not do any work until someone subscribes to one of it's advertised topics; once there is a subscriber for a topic, *image_proc* will start actually doing work on the input raw image and outputting images. This means that you don't have to worry about all of the extra topics that *image_proc* can output wasting CPU cycles rectifying images when nobody is listening.

Now that you know what the driver launch file is doing, how can you check if it's working? Use the following snippet to check if the driver is publishing images:

```
rostopic hz /front_camera/image_raw
```

You should see an output rate in Hz that is approximately what the requested frame_rate is. If you don't see anything, you'll have to check some of the troubleshooting steps available in Section 4. You can do a similar test for

/front_camera/image_rect_color and you should see approximately the same output rate as on */front_camera/image_raw*. If not, *image_proc* is having trouble keeping up with the camera driver and you should let Eric and Chad know, since it ought to be able to keep up on Jinx's computer.

2 Viewing Live Images

Now that you have the camera driver and processing pipeline up and running, you'd probably like to view the live images so you can see what the robot sees. We've told you many times that you subscribing to an image topic over WiFi is a terrible idea due to the sheer amount of data coming out of the camera (for example, 640x480@30fps of BGR8 images should be somewhere around 220 megabits per second, which the WiFi cannot hope to keep up with). You can get an idea for how bad this gets by running

```
rostopic hz /front_camera/image_rect_color
```

on your computer in a terminal that has been setup to remotely connect to Jinx (the whole source connect_harlie.bash dealio). When I did that when setting up Jinx's camera, I got somewhere between 2-5 frames per second when the camera was output 30 frames per second. However, there is a solution that allows you to view camera images at 30fps over the WiFi! By using image_transport, the entire ROS image pipeline is able to transparently output images either in their raw binary form (default and way too much data for WiFi) or compressed using JPEG encoding; as JPEGs, the images are much smaller and we can stream them over the WiFi. Here's how to open an *image_view* window on your laptop to view the rectified color images and telling *image_transport* to use JPEG compression since we need to stream the images over the WiFi:

```
rosrun image_view image_view image:=/front_camera/image_rect_color _image_transport:=compressed
```

After you run that, you should get a window with camera images in it and, if you wave your hand in front of the camera, you should see nice smooth updates. If you don't, you may not have properly specified JPEG compression and are dropping frames like crazy because the WiFi is too slow to keep up with raw images.

While the JPEG compression option on *image_transport* is excellent for viewing images over WiFi, it cannot be used when doing image processing. JPEG encoding is a lossy compression and introduces artifacts into your images that, while you and I likely won't notice them unless we turn the JPEG quality wayyyyy down, OpenCV will most definitely notice them and your image processing algorithms may become confused by the JPEG artifacts.

3 Recording images to bags

Previously, we could get away with running something like

```
rosbag record -a -o blah
```

to record our data and not have any problems. With a camera and image_proc, this will pretty much cause rosbag record to explode and give you a bag with many, many missing messages where rosbag had to drop messages. For recording data with cameras, you'll want to pick a one or two image topics and explicitly record those. For example, the bagfile we made with HARLIE was made with something like the following:

```
rosbag record -o bagname /front_camera/image_raw /front_camera/camera_info /front_camera/image_rect_color /tf
```

Also keep in mind that bagfiles with images can become absolutely gigantic... the sample bag was only 640x480@15fps and somewhere between 50-70 seconds of data is over 1GB. Try to keep any bagfiles you take short and with a minimum of image topics (note that you can run *image_proc* on recorded data, so you don't technically need to record any of its outputs, just its inputs) and to make sure to delete your bagfile off Jinx after transferring it off the robot. Any bagfiles left lying around should be considered fair game to be deleted by any group if Jinx is running low on diskspace when they are running/recording and the group that recorded the bag is not around.

Also note that you'll have record bagfiles to use as sample images when developing your algorithms as Stage (at least the ROS wrapper) does not support simulation of camera sensors.

4 Troubleshooting

So, something didn't go according to plan, eh? Here's some things to keep in mind:

- *Problem*: You get some "unable to set control" errors when launching the driver, such as

  unable to set control : Input/output error [ERROR] [1300330664.357977358]: Problem setting exposure. Exception was unable to set control
  unable to get control : Input/output error [ERROR] [1300330664.658607089]: Problem setting absolute exposure. Exception was unable to get control
  unable to get control : Input/output error [ERROR] [1300330664.692137770]: Problem setting sharpness. Exception was unable to get control
  unable to set control : Input/output error [ERROR] [1300330664.987690476]: Problem setting white balance temperature. Exception was unable to set control
  unable to set control : Input/output error [ERROR] [1300330664.989188021]: Problem setting gain. Exception was unable to set control
  unable to set control : Input/output error [ERROR] [1300330665.005415843]: Problem setting saturation. Exception was unable to set control

unable to set control : Input/output error [ERROR] [1300330665.021200592]: Problem setting contrast. Exception was unable to set control
unable to set control : Input/output error [ERROR] [1300330665.023201428]: Problem setting brightness. Exception was unable to set control

  - *Solution*: This is okay as long as the driver is streaming images. The driver is a bit finicky and video4linux is a huge pain in the butt, so it often throws a bunch of errors that we can safely ignore.

- *Problem*: No images streaming, with or without the above errors.

  - *Solution*: Is there a red ring on the front of the camera? Whenever the driver has the camera open and is streaming images, this ring should be lit. If not, try starting the driver up again. Try unplugging the camera, waiting 15 seconds and plugging it back in, then relaunching the driver. Try rebooting the computer (last resort).

- *Problem*: Camera driver is off but the red ring is still lit.

  - *Solution*: Are you 100% positive the driver has quit? If so, it's possible the camera itself has crashed or otherwise locked up. To fix that, unplug the camera, wait 15 seconds and then plug it back in. The red ring should not be lit at this point. Now that the camera has been reset, you should be able to continue with whatever you were doing. Note that some things that can cause this are attempting to set the width/height of the image to an invalid pair.

- *Problem*: HALP! I've tried all the steps on here and I'm still not getting any images!

  - *Solution*: Reboot the computer. If that doesn't help, email Eric and Chad or hop on GChat and see if one of them is available.

5 Tables and such

Parameter Name	Description	Notes
frame_rate	Number of frames per second	30 or 15 are good values. The max frame_rate can depend on other things such as resolution or exposure time.
camera_info_url	Path to the yaml file containing the camera's intrinsics	URL listed above shortened to avoid wrapping. See the actual file for the real URL.
frame_id	TF reference frame to include set in the output image's header	You are going to find the transform between pixels in this frame and 3D points in the base_link frame when you find the camera's extrinsics
exposure	The exposure mode the camera is in	Possible values: 1 for Manual, 3 for automatic . All others are unsupported by the Webcam Pro 9000
brightness	Brightness value to set on the camera	
constrast	Constrast value to set on the camera	
saturation	Saturation value to set on the camera	
absolute_exposure	Exposure value to use when *exposure* is set to Manual mode	Not currently set in the launch file, but may be useful later if auto-exposure screws things up

Table 1: uvc_cam Important Camera Parameters