To: Professor Merz
From: Benjamin Nitkin
Subject: IGVC Progress Report
Date: October 23, 2013

This past week, the programming team has been working on integrating the cameras into ROS and RoboRealm. (RoboRealm's been troublesome, as the computer refuses to run two identical webcams at once.) This memo will lean heavily towards what we've learned about ROS. So far, we have the cameras communicating with the computer under ROS. Based on camera setup, adding additional nodes to ROS, such as sensors and serial communication, should be fairly simple.

ROS is a framework of applications designed to interface a computer with assorted robotic hardware. It includes libraries for common tasks, such as positioning, divergence (depth) mapping, hardware drivers, and serial communication. ROS is built around a node / publisher / subscriber model. Every piece of ROS software, whether it's a camera driver, mapping service, or image viewer, is called a *node*. Nodes manipulate data by publishing or subscribing to *topics*. For instance, the camera driver we're using publishes images and camera calibration data to the */gscam/image_raw* and */gscam/camera_info* topics, respectively. Other nodes can subscribe to these topics; a viewing window for each camera subscribes to the */cameras/left/image_raw* and */cameras/right/image_raw* topics. (We remapped the default topic paths to more sensible ones; more on that later.) This publisher-subscriber model simplifies data transport – new nodes can query the central ROS server (*roscore*) for the data they want. If a node starts before data it's subscribed to, it waits patiently for data, rather than crashing.

The simplest way to start a node is *rosrun.* The user simply specifies which *node* to start, any relevant options, and hits return. For instance, after initializing settings, *rosrun gscam gscam* starts a camera. *rosrun* becomes unwieldy for more complicated tasks. Starting a camera is simple enough, but starting two cameras, two viewing windows, stereo processing, and a viewing window for depth output is unwieldy. ROS provides a solution in *.launch* files. A launchfile is an XML file that specifies which nodes to start, what parameters to run them with, and where to place their topics in the hierarchy. A launchfile replaces dozens of shell commands with a single script, to be invoked by *roslaunch.*

Over the weekend, the cameras were set up on ROS. After some tinkering and learning on the fly, we devised a launchfile that started the cameras and stereo processing, with output windows to debug. Setting up one camera was easy, but whenever the second camera started, the first would exit (they were publishing to the same topic). ROS allows remapping of topic names, which let us move each camera from */gscam/image_raw* to a more descriptive path. With the name conflict solved, both cameras happily ran together.

As of Tuesday, the cameras were mounted in a crude stereo jig. Moving forward, a calibration pattern will be used to correct for camera distortion. Once the cameras are characterized, ROS should be able to generate a depthmap by comparing the left and right undistorted images.

As an aside, I looked into giving the cameras unique names on disk. Linux handles everything as a file – when a camera's plugged in, it appears on the hard drive as */dev/video0* (a second camera is */dev/video1*, and so on). These paths aren't unique to the camera, though. The software needs to know which camera is left and which is right, or all depths will be inverted. Unfortunately, the cameras are identical (they carry neither unique ID, nor serial, nor different version numbers). The only way to tell them apart is by the USB port they're plugged into. If the cameras had a unique ID, I could write some code to give each its own path (say, */dev/videoleft* and */dev/videoright*). I can't, so we'll have to make sure to keep the cameras in the same USB ports and identify them that way.