CSE 145/237D Embedded Systems Design Project

Final Report: OpenROV Team
Spring 2015

Connor Brew (A09472240)
David Lane (A53064662)

Abstract

The majority of the surface of the earth is covered by water, and yet we know very little about what lies beneath the surface of the ocean because underwater exploration is so expensive and time consuming. OpenROV is a community of people looking to change that by developing a low cost remotely controlled vehicle for underwater research and exploration. Their platform is controlled by software written in Node.js and maintained by a relatively small group of people. Our project is to integrate the native software of the OpenROV with ROS, a robotics operating system that has a much larger community of people supporting and adding to it. Integration with ROS would give the OpenROV community access to many high level libraries that have already been developed in ROS. These libraries include algorithms for important robotics problems such as object detection in computer vision, point cloud construction of surrounding environment, and autonomous navigation. By integrating translation libraries on the OpenROV (roslibjs) and on the host laptop using ROS (rosbridge), we have enabled two way communication and given the OpenROV access to advanced robotics tools which it can quickly use to solve problems faster than developing solutions independently.

Introduction

The whole goal of OpenROV is to make underwater exploration accessible to as many people as possible. They do this by making a vehicle that is low cost and easy to assemble. This is done by an intelligent design that seeks to maximize low-cost materials and off-the-shelf components. This is markedly less than any other underwater platform available on the market and opens the door for scientists, academics, and even enthusiastic hobbyist to have a vehicle that can explore deep waters. This leads to a rich community of intellectuals and hackers who can all collaborate in an open source community, and firmly shows how OpenROV is succeeding in its goal to democratize underwater exploration.

While the OpenROV is a remarkable platform with a great supporting community, it still lacks resources compared to other open source projects. Robot Operating System (ROS) is an open source community that focuses on the software side of robotics. It was started in Silicon Valley and is supported by a strong open source community and is used in a number of commercial robotics products. They provide a set of tools for communication and synchronization that seek to create a standard for robots. In addition to the goal of standardization, there are also a large amount of advanced algorithms that are already implemented. These algorithms solve many common robotics problems such as visualization of an environment (via sensor point clouds or computer vision) and navigation within that environment.

This project seeks to maximize the best parts of both open source communities. OpenROV makes a fantastic underwater robotics platform, and ROS has a huge body of work that has already been developed for robotics software. We seek to develop a software interface between the two. The ideal interface is minimally intrusive to the existing OpenROV software and offers communication in parallel to it. The interface will be installed on the OpenROV as a plugin that can establish two way communication: listening to native OpenROV messages and converting them into ROS messages; and listening to ROS messages and using them to create OpenROV messages that can control the vehicle.

Technical Material

Before discussing the details of the implementation, we need to give an overview of the basics of the OpenROV and ROS.

OpenROV

The electronics of the OpenROV consist of an Arduino connected to a Beaglebone Black. The Beaglebone Black runs the native OpenROV software that uses node.js and communicates with JSON messages. An intuitive plugin design allows for interacting with messages that come to and from the Arduino in a global event loop.

Under typical use, the vehicle is tethered to a control station that runs another software called Cockpit. Cockpit is loaded into a browser and shows live video, sensor feedback, and provides driving controls. In addition to the primitive simple driving mechanisms that directly control the motors, there are the beginnings of auto-pilot mechanisms such as hold heading, or hold depth.

ROS

Rosbridge_suite

Rosbridge provides an API that can translate JSON messages into ROS messages. It acts as a ROS node that can publish or subscribe to messages. We communicated with it via a websocket connection and roslibjs.

gscam

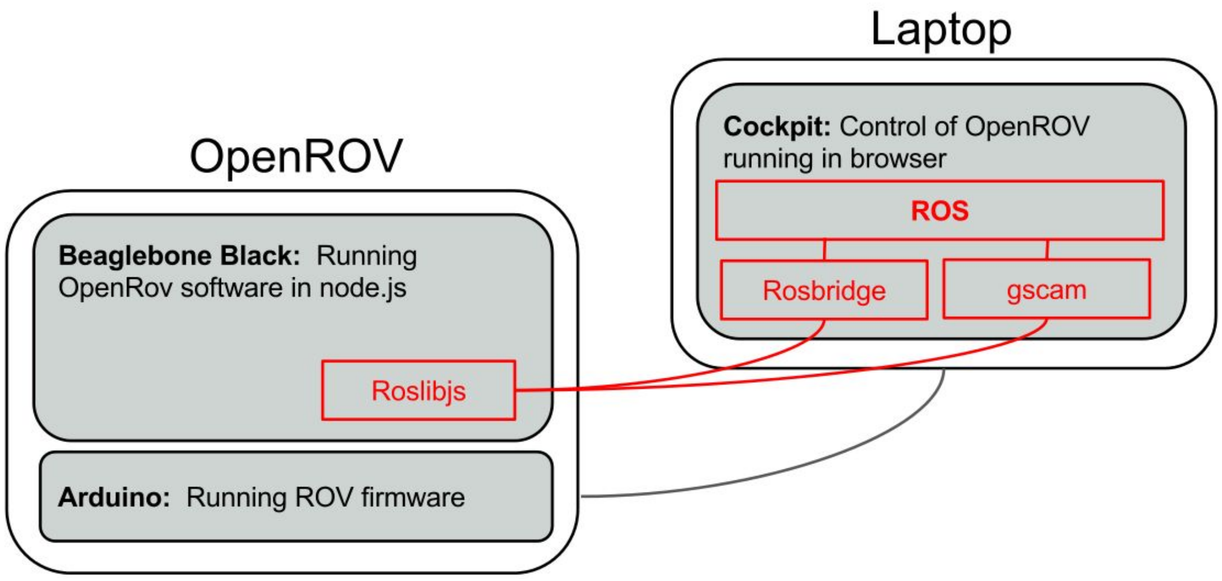
Gscam is a ROS package that imports video and converts it into a format that ROS can easily work with. It uses gstreamer which provides it with image processing tools.

Roslibjs

With Roslibjs ROS can interact with Javascript to publish or subscribe to ROS messages. It uses websockets to communicate with Rosbridge

Integration

Since the ROV is tethered to a laptop and the Beaglebone Black has limited computing resources, we decided to implement ROS on the laptop instead of directly on the ROV. In this setup, ROS Master will run on the laptop and interact with two ROS nodes that we have installed: Rosbridge and gscam. The ROV will contain an additional ROS node (Roslibjs) which will interact with Rosbridge for data messages.



Laptop
Cockpit: Control of OpenROV running in browser
ROS
Rosbridge
gscam
OpenROV
Beaglebone Black: Running OpenRov software in node.js
Roslibjs
Arduino: Running ROV firmware

Milestones

At the beginning of the quarter we set the following milestones and deadlines. We will discuss each of them in detail:

- Familiarize ourselves with the native OpenROV operating system and with ROS. Get newest version of OpenROV software running on our ROV. Our goal was to accomplish this by the end of week 4. This was done by the middle of week 5 when OpenROV released the latest version of their software for the ROV.
- Create an interface that can convert between the javascript events and ROS events to enable two way communication between ROS and OpenROV. Our goal was to have this done by the end of week 6. In practice we had trouble getting a library we needed working. We finished our interface early in week 10.

- Test, clean, document, and publish our work to the OpenROV community in order to get feedback and suggestions. We hoped to have a working interface early enough that we could do this by the end of week 8. In reality our integration took long enough that we started getting our plugin out to the OpenROV community at the end of week 10.
- Integrate with one or more ROS application that would demonstrate the value that is added by having ROS functionality. One such stretch goal would be to get the ROS 3D visualization tool, rviz, working. This could be extremely beneficial for future navigation projects. We wanted to do this by the end of the quarter so that when we shared our plugin with the general OpenROV community they would have an example use case they could immediately get working. As this was a stretch goal for us, we have focused more on documenting our work and coordinating with OpenROV to share our plugin with the community that this has been placed on hold for now.

<u>Familiarize ourselves with OpenROV and ROS</u>
Although it is hard to quantify because of the ambiguous target, we feel that we hit this milestone. That's not to say that we were not still learning things in the tenth week of the quarter but we were able to make some good progress early on thanks to the very good tutorials that ROS has available on their website. OpenROV also has very clear documentation - although it was not organized in a step-by-step manner, which meant a little more searching around in forums, github repositories, and Dozuki postings.

<u>Create an interface between OpenROV and ROS</u>
This was the main goal of the project and we did not hit our rather optimistic goal of the end of week six. Installing ROS and the necessary packages on the laptop was straight-forward and went quickly. It was also simple to get a working copy of roslibjs running in a browser listening and publishing messages back and forth with Rosbridge. We stalled however when it came to moving Roslibjs over to the ROV.

The major problem was that roslibjs was written to work from within a browser - not node.js. We researched potential problems with it and tried a few unsuccessful manual fixes before we found that there existed a package available from Node Package Manager (npm).

Even after finding the npm package, we still had some difficulties installing it. Inconsistent use of sudo during the installation corrupted the package, and we also learned that by default npm does not interact with the proxy internet the OpenROV uses. In the end these were all easy fixes, but combination of errors and gaps in knowledge ultimately caused a large setback.

Once Roslibjs was finally communicating with Rosbridge, the focus of the project turned to getting deeper knowledge of the OpenROV software (beyond just installing a working plugin) to find out what types of messages the system was using and how we could hook into those messages. It also lead to additional research about ROS to speculate which ROS packages would be useful to OpenROV and determine if we could get the appropriate data from the OpenROV. We struggled for some time to finally get the details of interacting with the OpenROV software, and with the help of Jim Trezzo and Brian Adams were able to establish two way communication.

<u>Clean up documentation and publish to OpenROV</u>
We kept notes and took screenshots throughout most of the installation process. Since we were working on the interface well beyond the six week deadline, the collating and cleaning of the documentation got pushed to a lower priority. After we finally got two way data communication working in week 8, we realized that video might not be as difficult as we originally thought. In the next week we were able to successfully integrate video, but at the expense of putting off the documentation. This seemed like a reasonable tradeoff since the overall project was clearly still a work in progress.

Even though we realize that documentation and presenting the installation procedure was one of the primary goals of the project, we felt more urgency in getting presentable results for the final presentation, video, and report. The instructions were not cleaned up and presented to OpenROV until the last week of school. Also, there will likely need to be some cleaning up of the instructions in the future. Jim Trezzo visited during finals week and we ran into a few problems that we had not seen before which should be addressed in the revised instructions.

<u>Create a flashy application to demonstrate worth</u>
Regrettably we did not have the time to implement a good concrete example. We would have liked to provide a good, easy to setup example to go along with the installation guide because we feel that it would go a long way towards getting people excited about the possibilities with ROS. Since we will both be in the Bay Area this summer, Jim has suggested that we visit OpenROV to present our work to the whole team. This might give us the opportunity to get an application up and running.

Conclusions
The primary goal of our project when we set out was to integrate ROS with the existing OpenROV software in a simple to reproduce approach so that the OpenROV community could begin to use the capabilities that ROS provides. We accomplished this primary goal and have already received feedback from the OpenROV community that they are eager to begin using our plugin. By focusing on our primary goal of making a plugin that works with the existing OpenROV system we were able to accomplish the most important things we set out to accomplish even when we hit unexpected difficulties. While we still need to work on documentation for the openROV community, by having a working plugin that demonstrates major use cases, we can get members of the OpenROV community set up with our plugin and work with them on documentation for the entire community. Our plugin forms the vital first step in demonstrating the usefulness of ROS to the OpenROV community.

General References

Rosbridge:	http://wiki.ros.org/rosbridge_suite
gscam:	http://wiki.ros.org/gscam
Roslibjs:	http://wiki.ros.org/roslibjs
Adapting Roslibjs:	https://github.com/RobotWebTools/roslibjs/pull/104

Additional Project Information

Github OpenROV:	https://github.com/DCLane/OpenROV
Github ROS:	https://github.com/DCLane/OpenROV-ros
Blog:	http://145openrov.blogspot.com/

Youtube:
https://www.youtube.com/watch?v=5aW-hfu1LAk&feature=youtu.be&ab_channel=ConnorBrew