**Title:** Robot Operating System (ROS) Based Reconfigurable Robot Sensing Platform

**Name:** Daniel Steele

**Date:** March 31, 2017

Table of Contents

[Purpose 3](#_Toc480555781)

[Implementation 3](#_Toc480555782)

[Framework 3](#_Toc480555783)

[Source Code 3](#_Toc480555784)

[Sensor Management 3](#_Toc480555785)

[Command and Control 4](#_Toc480555786)

[Instillation 4](#_Toc480555787)

[Additional Notes 4](#_Toc480555788)

[Architecture 5](#_Toc480555789)

[Example Hardware 5](#_Toc480555790)

[Software 5](#_Toc480555791)

[Lessons Learned 6](#_Toc480555792)

[Potential Expansions 7](#_Toc480555793)

[Appendix A: Utilized Open Source Projects 7](#_Toc480555794)

[Appendix B: Software Package Descriptions 7](#_Toc480555795)

[Arduino Control 7](#_Toc480555796)

[Sensors 7](#_Toc480555797)

[Bridge 8](#_Toc480555798)

[Appendix C: Test Platform Information 8](#_Toc480555799)

[Test Platform Description 8](#_Toc480555800)

[Installing ROS on the Raspberry Pi 8](#_Toc480555801)

[Bill of Materials 9](#_Toc480555802)

[Images 9](#_Toc480555803)

[References 11](#_Toc480555804)

# Purpose

This project is to implement an expandable system in which multiple robotic platforms will receive command and control from a base station as well as relay sensing information back to the base station. These robotic platforms shall consist of an Arduino with the appropriate shields acting as a movement controller, a wireless router or repeater, and a sensing element. The implementation shall be designed such that multiple robots can connect to the network in the future as well as to be able to handle multiple sensor variants. The sensing board and element can be thought of as a payload on the robot.

# Implementation

## Framework

ROS was chosen as the framework for this project. ROS provides the message serialization, deserialization, and transport for most the project. The ease of use, fit for the project, documentation, and large user base all contributed to the decision to use ROS for the project (Open Source Robotics Foundation, n.d.).

## Source Code

The source code for this project was implemented in a combination of C++, C, Python, and Bash. When creating each individual portion of the project the programing language that best fit the scenario was utilized. The source code for the project has been posted publicly on GitHub under the MIT Open Source License and is located at <http://www.gihub.com/dksteele/Senior-Project>. The code was also coded such that it is either easily read and understood or well commented such that it can be understood easily. This allows the source code to self-document in many cases.

## Sensor Management

The sensor management and sensor nodes consist of a server request-response pattern for individual sensors to register with the management node to receive a unique topic that it should publish on. The registration also updates the GUI to add the sensor to its tree. The user can specify a node that it wants to have displayed in the GUI through selecting it in the tree. Currently Digital, Analog, and Camera (JPEG image stream) sensors have been implemented in the management node to allow some flexibility in what types of sensors can be utilized.

Python was chosen for this portion of the system because of its ability to work with Qt as well as pythons not using strong typing allowing for some abstraction.

## Command and Control

The Arduino UNO which is being used for command and control, does not have the ability to run a full ROS node due to the large amount of overhead that would be required. When researching different options rosserial seemed to be the prominent method for communication between an Arduino and ROS, but rosserial is built to be able to handle one device and could not easily be ported to multiple devices. As such, a custom method needed to be devised to communicate with the Arduino. rosserial message generation scripts were used to generate the Arduino libraries as they are protocol independent and provide a simple method for serialization and deserialization of native ROS serialized messages (Ferguson, n.d.).

Utilizing the current WiFi/Ethernet framework, it was decided that TCP was not the ideal protocol for the command and control as it is connection based and would have to create connections with each robot and individually send them information. Further UDP Broadcasting, although suitable to the needs of the project would send the information to every device on the network instead of just the devices needing the information. UDP Multicasting was chosen as it would allow for devices to listen to the multicast group for the messages and is also connectionless. UDP Multicasting uses groups of which there can be multiple on a single network, this could allow for multiple multicasted topics on a single network (Stevens, Fenner, & Rudoff, 2004). Using either a WiFi or Ethernet shield the Arduino UNO can connect to the network and receive the messages that it needs to control the motors, as long as the network contains a router that is capable of multicast routing.

C and C++ were chosen for this part of the code due to its flexibility when working with sockets.

## Instillation

Instillation is completed using a python script that generates a bash script that is run by a service at startup. The script ran by the service sets up the environment and starts a python program that starts the required nodes as sub processes to the service. This implementation allows for the user to specify the format of the instillation at install time so that it only starts the required items as well as have all node register their PIDs as part of the service for tracking and service manipulation. If GUI nodes are specified the instillation program also installs a script to begin them on log in.

## Additional Notes

* Individual robots can network with each other without the intervention of the base station. In this situation, the base station will assign, track, and communicate the topic transport information and then the rest will be up to the publisher and subscribers.
* Setting the TCP no delay flag when creating a publisher is important to provide uninterrupted communication.

# Architecture

## Example Hardware



Figure 1: Example Hardware Architecture

## Software



Figure 2: Software Architecture

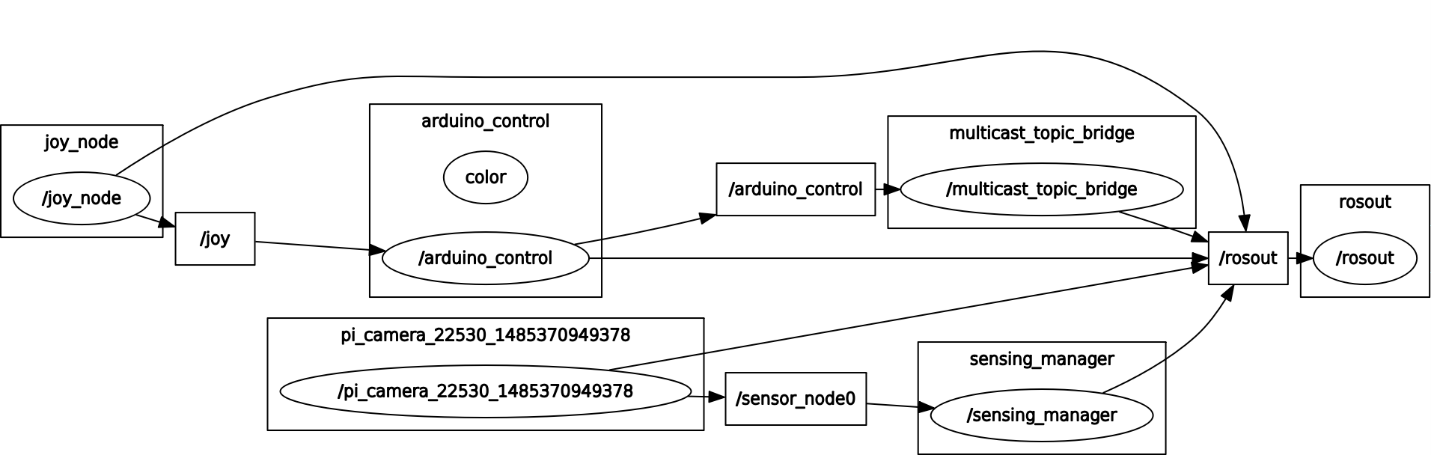


Figure 3: ROS Node Graph (1 Sensing Board with Raspberry Pi, Image Generated Using rqt\_graph)

# Lessons Learned

* Start out early with a source code previsioning and backup solution, nothing is worse than losing all your code in an accidental deletion
* Document as needed, over documentation does not help anyone
* When working with threads make sure that mutexes are closed properly no matter what happens in the code that utilizes the mutex
* Qt can be frustrating when working with threads, make sure to research if a program is thread safe before beginning to work with it.

# Potential Expansions

* Implementation of additional sensors
* Testing with multiple robots
* Data logging
* Variable command and control based upon robot id
* GPS location mapping (through command and control system)
* Sensors should be able to register a name to allow it to be easier to identify to the user.

# Conclusion

The goals of this project were achieved through the creation of a software product using ROS that provides command and control as well as sensing for an Arduino based modular robotic platform. A test platform was created to test the software product utilizing the Arduino for control and a Raspberry Pi and camera for sensing. Through testing the performance of the system was validated to fulfill the purpose of this project. Throughout development there were challenges such as setting up the Arduino to receive multicasted information, setting up the GUI for sensing to utilize multiple threads, and getting the Raspberry Pi set up to behave like an Access Point. This work could be applied in a situation where a user needs to have a robot platform with exchangeable sensors for testing of a hazardous environment, swarm based sensing, and others.

# Appendix A: Utilized Open Source Projects

* Robot Operating System (ROS): [http://www.ros.org/](http://www.ros.org/%20)
* rosserial: <http://wiki.ros.org/rosserial>

# Appendix B: Software Package Descriptions

## Arduino Control

* Description:
  + Translates joy messages such as those generated with joy\_node to arduino\_control messages that can be sent to control the arduino
  + The outputted message is composed of the direction and angle of magnitude with forward being 0°
* ROS Parameters:
  + arduino\_control/deadzone: Value of either axis at which to begin sending data. This allows the user to compensate for a non-zeroed axis (Default: 0.1)

## Sensors

* Description:
  + This package contains the following items:
    - Management utility for multiple sensor stream
    - Implemented Sensors:
      * Raspberry Pi Camera
      * Digital Testing Utility
* Environment Parameters:
  + All Sensors:
    - PLATFORM\_NAME: Designation of platform containing sensors
* ROS Parameters:
  + Pi Camera:
    - [plaform\_name]-pi\_camera/vflip: Raspery Pi Camera Vertical Flip (Default: False)
    - [plaform\_name]-pi\_camera/hflip: Raspery Pi Camera Horizontal Flip (Default: False)

## Bridge

* Description:
  + Utility for sending serialized ROS messages over a multicast channel
* ROS Parameters:
  + multicast\_topic\_bridge/topic: Topic to send over multicast (Default: multicast\_topic)
  + multicast\_topic\_bridge/mulicast\_group\_addr: Multicast group address to send to (Default: 224.0.1.1)
  + multicast\_topic\_bridge/sending\_port: Port to send on (Default: 12345)
  + multicast\_topic\_bridge/frequency: How often to send an updated message (Hz) (Default: 1)

# Appendix C: Test Platform Information

## Test Platform Description

The test platform for this project was a simple implementation with 2 motors and wheels controlled by the Arduino, and a Raspberry Pi with a Raspberry Pi Camera as the sensing board. It relayed images back to the base station in real time.

Note: The test platform that was utilized due to space and cost restrictions did not utilize a traditional router. Instead the Raspberry Pi was utilized as a WiFi router, WiFi access point, and Sensor Controller. In a field implementation, the robot would have a separate sensor board as well as a WiFi access point on it that connects to a router that is connected to the base station.

## Installing ROS on the Raspberry Pi

To install ROS on the Raspberry Pi, the Raspberry Pi should be version 2 or newer and have an SD card installed containing Ubuntu MATE (<https://ubuntu-mate.org/raspberry-pi/>) , then normal ROS instillation instructions can be followed to install the software (<http://wiki.ros.org/kinetic/Installation/Ubuntu>) .

Additional software consisting of hostapd and isc-dhcp-server are also required to set up the Raspberry Pi as an access point as was done for this project.

## Bill of Materials

|  |  |
| --- | --- |
| Quantity | Part |
| 1x | Sparkfun Magician Chassis |
| 1x | Arduino Uno and Cable |
| 1x | W5100 Arduino Uno Ethernet Shield |
| 1x | Arduino Dual MC33926 Motor Shield |
| 1x | Raspberry Pi 2 and Power Cable |
| 1x | Raspberry Pi Camera |
| 1x | Ethernet Cable |
| 1x | 8 GB MicoSD card |
| 1x | Panda Wireless PAU06 |
| 1x | Battery and Connectors |
| 1x | Laptop Base Station |
| 1x | Xbox Controller |

## Images

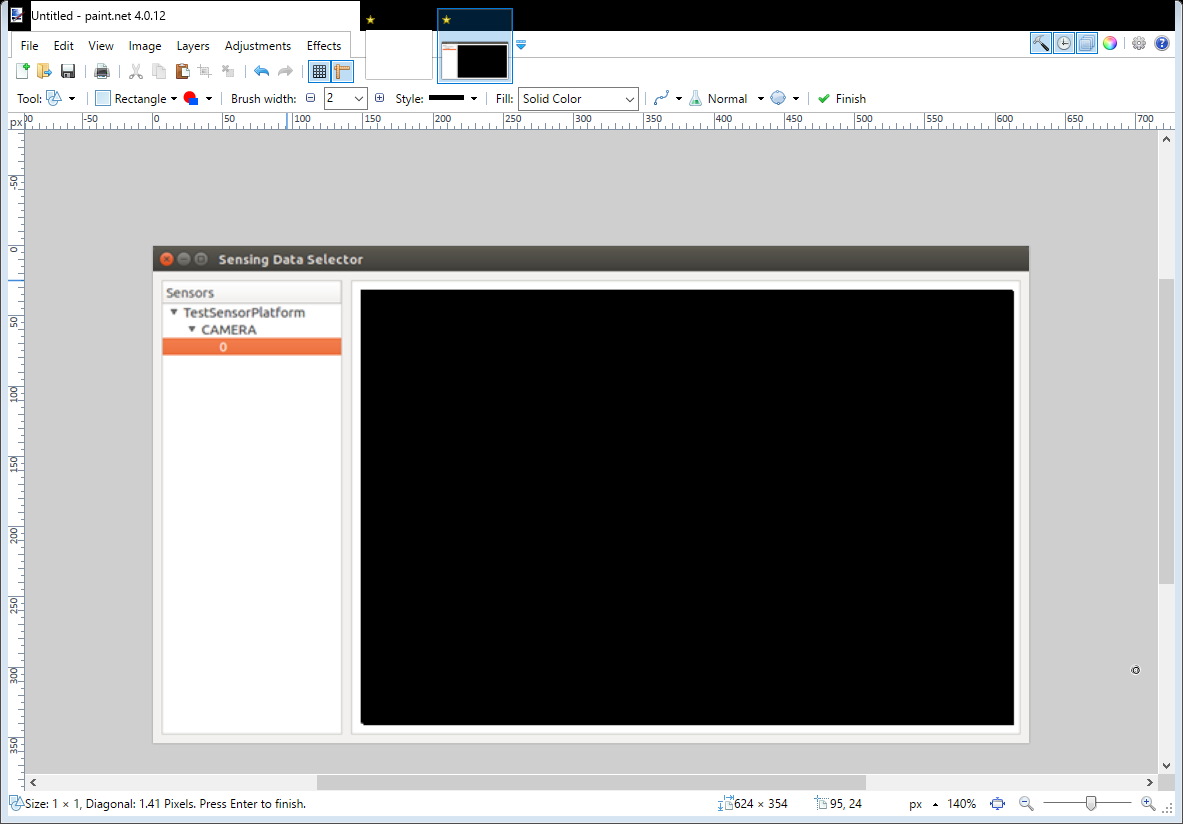


Image 1: Test Platform Sensing Manager

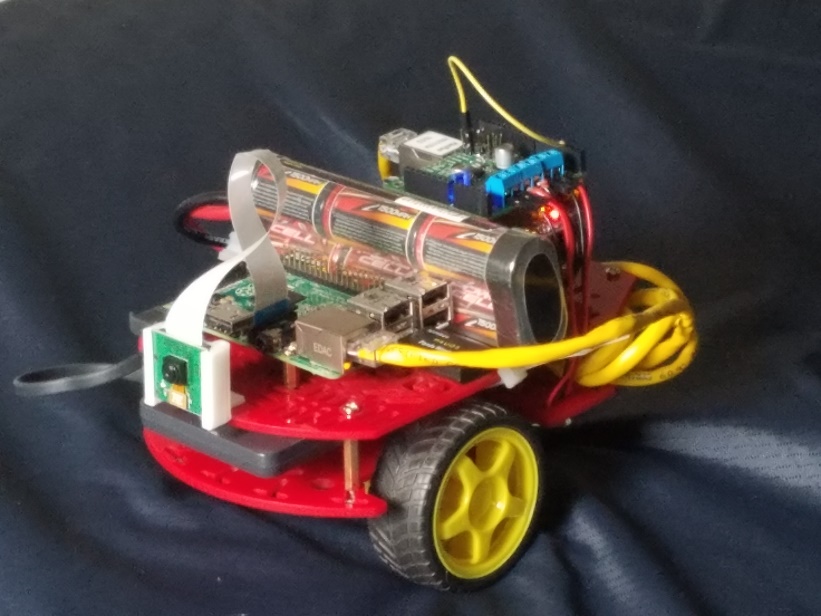


Image 2: Test Platform

# References

Ferguson, M. (n.d.). rosserial. Retrieved from http://wiki.ros.org/rosserial

Open Source Robotics Foundation. (n.d). Is ROS for me. Retrieved from

http://www.ros.org/is-ros-for-me/

Stevens, W., Fenner, B., & Rudoff, A. (2004). Unix network programing. (3rd ed.). Boston, MA:

Pearson Education.