LINUX JOURNAL™
Since 1994: The Original Magazine of the Linux Community

Categorize Documents with Machine Learning

The Importance of Good Ticketing Systems for Sysadmins

Tips for Cutting the Cable Cord

THE SPACE ROBOTICS CHALLENGE and Linux

Solve Physics Problems with Linux

\+ Software Updates, Embedded Linux and the IoT

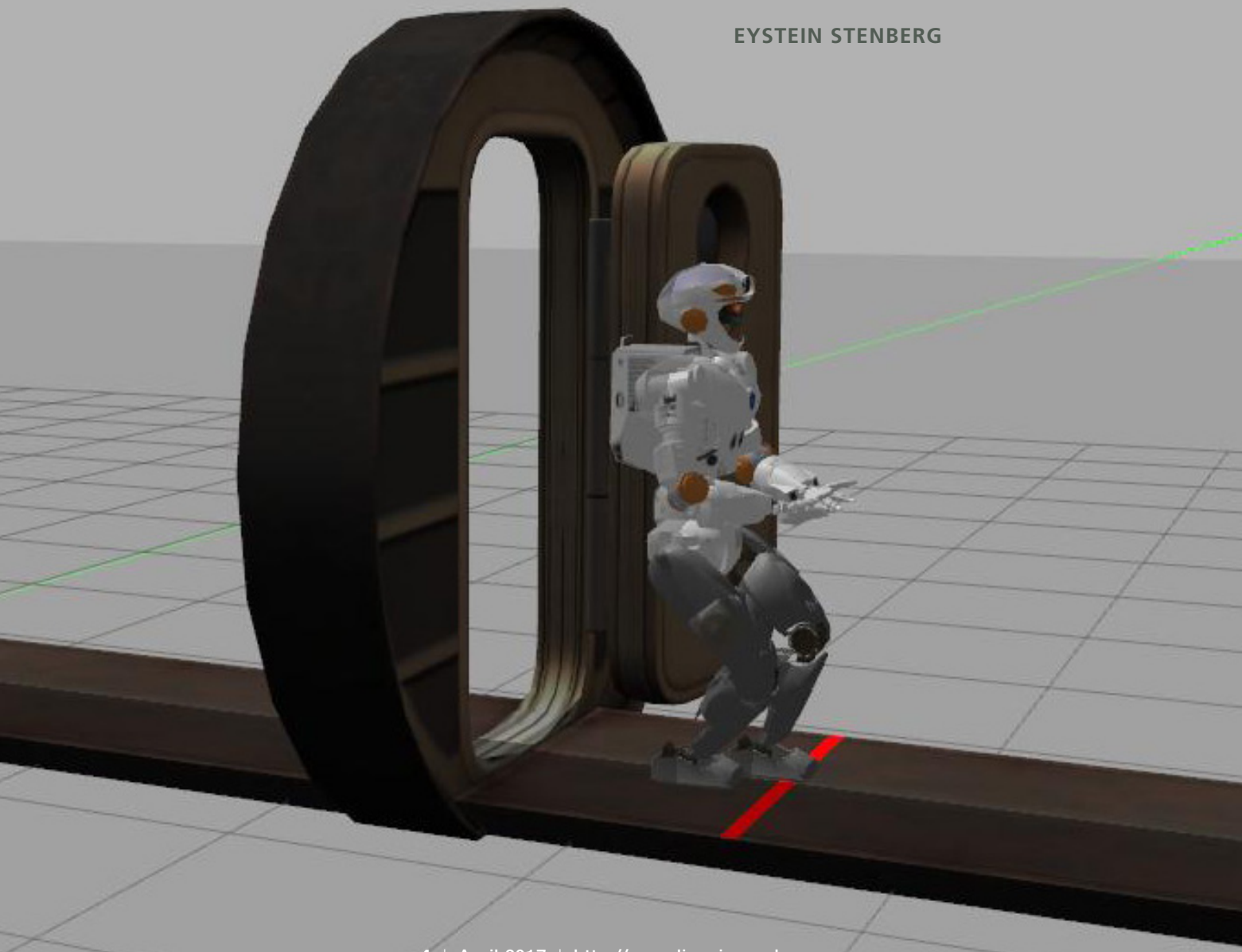
APRIL 2017 | ISSUE 276
http://www.linuxjournal.com

WATCH: ISSUE OVERVIEW

CONTENTS

APRIL 2017
ISSUE 276

FEATURES



ROBOTS AND LINUX, A MATCH MADE IN SPACE

As robots become more prevalent, they soon will be working alongside humans, relieving some of the burdens as well as taking on more of the dangers. The systems to make that happen are being built today, and they are using Linux in a major way.

PAUL FERRETTI

PREVIOUS
New Products

NEXT
Feature: Key Considerations for Software Updates for Embedded Linux and IoT

I grew up in the sixties, during the height of the space race. I vividly remember watching the first moon landing, and I have loved space and all things space since those early days. Alas, I did not become an astronaut; instead, I ended up in the tech field writing software. And as much as I loved NASA, the closest I have come to working with NASA is vicariously through my nephew who works at the Johnson Space Center in Houston. I spent many summers with my nephew building and programming robots, and it's a hobby that I still enjoy very much.

So, when I became aware that one of NASA's Centennial Challenges was called the Space Robotics Challenge, I immediately reached out to several friends to see if they were interested in forming a team. After getting commitments from four other like-minded individuals, I registered our team to compete in the challenge.

NASA AND THE SPACE ROBOTICS CHALLENGE

The Centennial Challenges program is part of NASA's Space Technology Mission Directorate (STMD). The STMD creates various challenges in order to bring together members of industry, academia and the government with the goal being the advancement of innovation in key areas of technology important to the agency. These challenges are meant to engage individuals and teams from all walks of life. Whether you are a hobbyist, such as myself, or a member of a team from a tier-one technology lab, such as MIT, NASA has made these challenges accessible to everyone.

The technical coordinator for the Space Robotics Challenge (SRC) is NineSights, and according to its website, "The SRC focuses on developing

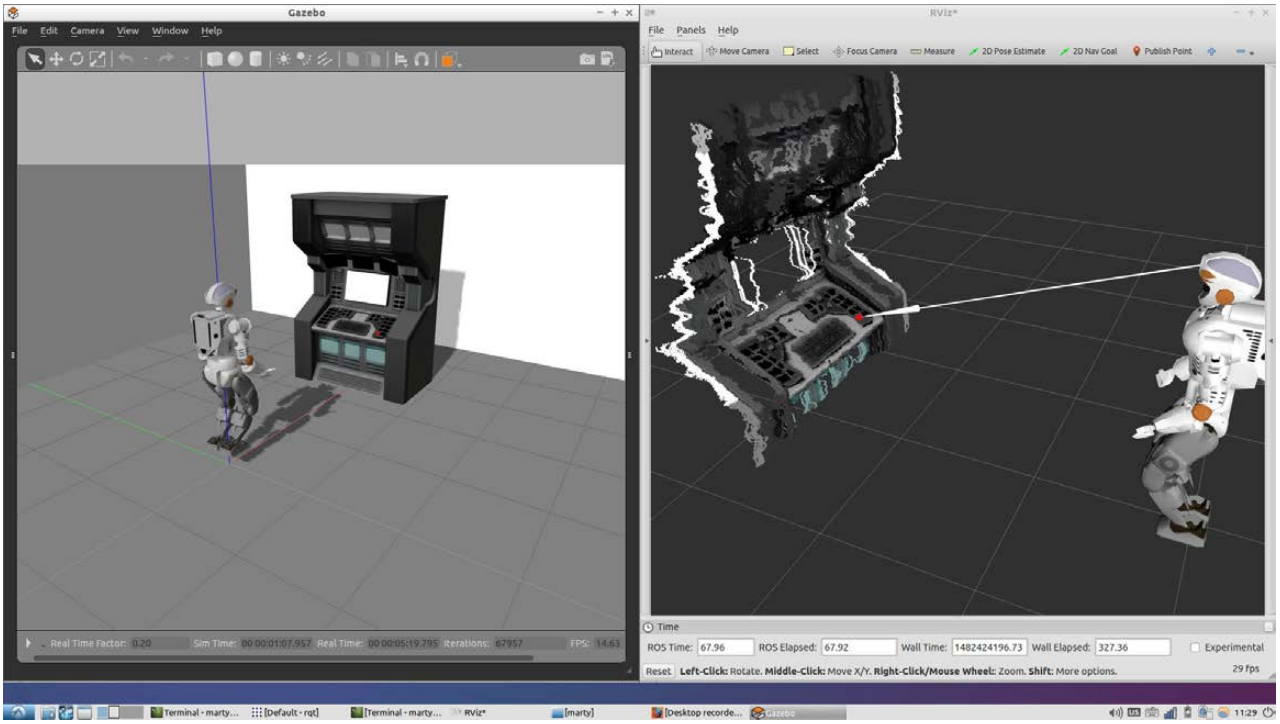


Figure 1. Using RVIZ to Check Line of Sight

software to increase the autonomy of dexterous mobile robots in humanoid format—specifically NASA's R5 robot—so they can complete specific tasks during space travel or after landing on other planets (such as Mars), as well as on Earth."

The SRC involved two rounds of competition: the first round, better known as the qualifying round, and the second round, referred to as the virtual competition. The qualifying round involved two tasks: a visual task and a mobility task. The visual task had the robot positioned in front of a panel that consisted of a number of LED lights. The lights would flash in a random sequence, and the task was to identify the location and color of the lights in the correct sequence (Figure 1).

The mobility task started with the robot standing behind a red line approximately four meters from a closed door. The robot needed to walk up to the door, press a button to open the door and then walk through the door for approximately one meter before ending up past a red line located on the pathway. The scoring for this task was based solely on the time it took the robot to walk from red line to red line (Figures 2 and 3).

As you can see, this challenge had two very important features: robots

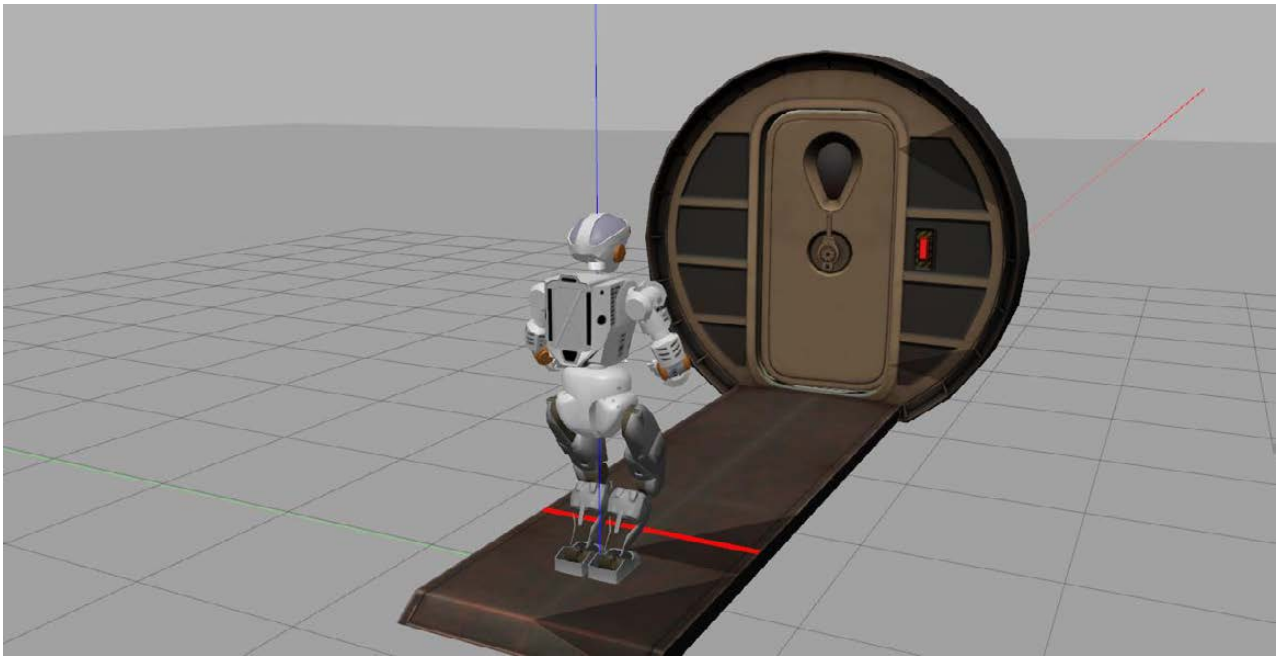


Figure 2. Robot in Starting Position

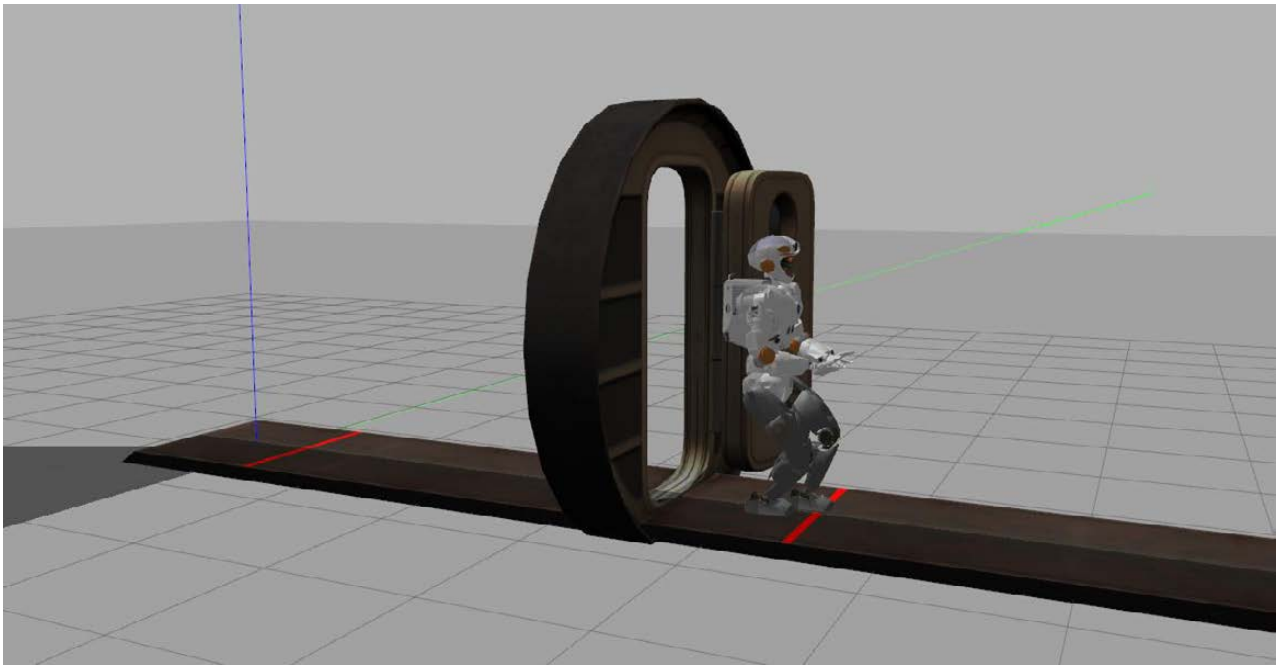


Figure 3. Robot in Finished Position

and space. That was all I needed, but I was soon to find out the challenge included a third element that was of great interest to me. The operating platform that the challenge would be working in was Linux-based. In addition, the challenge would be using the Robotics Operating System

(ROS), a simulation software called Gazebo and the Space Robotics Challenge Simulation (SRCSim), a predefined simulation that included the environment for the qualifying tasks, created for the Space Robotics Challenge by the Open Source Robotics Foundation (OSRF).

ROBOTICS AND LINUX

Robots have been around for a long time, and during most of that time, if you wanted to build your own system, you had to build the hardware, and you also had to create the software that would control the robot. This is no small task, and it required several skill sets that not everyone had. Unfortunately, the need for special skills was a deterrent that kept potential participants from jumping in to the robotics arena.

These days, there's a tremendous number of robot kits, like those made by Vex Robotics and Makeblock, which utilize an Arduino controller. Robotics kits like these greatly have reduced the need to fabricate your own robot; however, if you possess the necessary talent to fabricate a robot from scratch, you still can do so. For a long time, what the robotics community was lacking was a solid software platform with which to work.

In 2006, Microsoft was the first major player to enter the mainstream robotics software field with Robotics Studio version 1.0. Later, in 2008, Microsoft released Robotics Developer Studio (MRDS). The MRDS is a Windows-based environment for robot control and simulation. As with all Microsoft software, it is closed source. The primary programming language for MRDS is C#. Besides the closed-source nature of the MRDS, it has been described as overly complicated and requires a great deal of overhead. Along with that, the latest version, RDS4, has not had an update on its website since June 2012. A more sustainable solution was needed. Software sustainability increases when the maximum number of people are able to access it. Enter the Open Source community, led by Linux.

In 2007, Willow Garage, a robotics incubator in California, stepped in to extend work that was done by others, mainly at universities such as Stanford, to create a well tested implementation of an open-source operating environment for robotics. Linux, which has a long successful history on which to rely, was the platform utilized to provide an open-source foundation.

It makes complete sense that Linux would emerge as the dominant platform for robotics. The open-source nature of Linux, and by extension ROS, allows a

IT MAKES COMPLETE SENSE THAT LINUX WOULD EMERGE AS THE DOMINANT PLATFORM FOR ROBOTICS.

greater audience to access, develop and maintain a robust environment for the development and distribution of robotics-related software. Linux is widely used in academia as well as by home hobbyists. It's free, it's stable, and it runs on a variety of platforms, without the need for cutting-edge hardware.

REQUIREMENTS FOR THE SRC

The Technical Coordinators for the Space Robotics Challenge had specific software requirements for the participants of the challenge. The challenge organizers required teams to use the Linux-based Robot Operating System (ROS) and the 3D multi-robot simulator software package called Gazebo.

ROBOT OPERATING SYSTEM

ROS is a flexible framework for writing robot software. It is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. The ROS ecosystem now consists of tens of thousands of users worldwide, working in domains ranging from tabletop hobby projects to large industrial automation systems.

ROS is mainly targeted for the Ubuntu distribution of Linux. However, the source code is available for compilation in the many different flavors of Linux. The Ubuntu distribution provides the easiest implementation and the most support.

GAZEBO

Robot simulation is an essential component in every roboticist's toolbox. A well designed simulator makes it possible and quick to test algorithms, design robots, perform regression testing and train artificial intelligence (AI) systems using realistic scenarios.

Gazebo is a 3D multi-robot simulator that generates both realistic sensor feedback and physically plausible interactions between objects, and it

GAZEBO IS A 3D MULTI-ROBOT SIMULATOR THAT GENERATES BOTH REALISTIC SENSOR FEEDBACK AND PHYSICALLY PLAUSIBLE INTERACTIONS BETWEEN OBJECTS, AND IT INCLUDES A HIGHLY ACCURATE SIMULATION OF RIGID-BODY PHYSICS.

includes a highly accurate simulation of rigid-body physics. Gazebo also offers the ability to simulate populations of robots in complex indoor and outdoor environments accurately and efficiently. It includes multiple robust physics engines, high-quality graphics and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant user community that is constantly adding new simulations as well as extending existing ones.

Gazebo comes with many built-in robot simulations that are ready to use. Some of the more interesting ones include the Atlas Robot from Boston Dynamics; Baxter, an industrial robot, built by Rethink Robotics; and NASA's Robonaut2. Many more simulations are available, and all of them are modifiable and extendible by anyone.

INSTALLING ROS

For the SRC, the challenge rules mandated that a specific version of ROS on a specific version of Ubuntu be used. The moderators of the challenge wanted everyone to be working in the same environment and, therefore, face the same challenges and have the same benefits. Specifically, teams participating in the SRC were going to be using the ROS-Indigo version targeted for Ubuntu Desktop 14.04 (Trusty 64-bit amd64).

USING DOCKER

Complying with the SRC technical requirements meant using ROS-Indigo and the 14.04 version of Ubuntu. At the time, I was running a dual-boot Windows 10 (I know, forgive me), Ubuntu 16.04 Xenial 64-bit system.

Initially, I thought this would not be a problem, as I had figured I simply could install a Docker container running Ubuntu 14.04 and the required versions of ROS and Gazebo.

Docker containers allow you to wrap software in a container that keeps everything separate from the base operating system. The nice thing about Docker containers is that there are tons of containers already available for running various applications. I found a Docker container that was already configured with Ubuntu 14.04. All I had to do was install Docker on my 16.04 distribution and then install the 14.04 container. The install was fast and easy, and even though I had not used Docker previous to this, I found plenty of instructions on the web on how to set up my container as well as how to save various versions of the container as I installed other pieces of software. This allowed me to roll back to a working version of the software whenever I messed things up.

Unfortunately, even though I was able to install ROS and Gazebo in my Docker container, I never was able to get the SRC Simulation to run properly. I spent the better part of a weekend trying to solve the graphics problems, but in the end, simply gave up in lieu of an easier, albeit less elegant solution. I installed an additional hard drive in my computer and created a tri-boot system. Now I could boot into a separate version of Ubuntu 14.04 and install the necessary software without worrying about conflicting drivers. As a side note, I could have run Linux from a USB stick, but I figured I would get better performance using a hard drive. Plus, I had a spare 1 terabyte drive lying around, so why not use it.

Installing ROS was a breeze. I simply navigated to the ROS website and followed the instructions to install the version of ROS targeted for the indigo distribution of Ubuntu. In less than five minutes, I had a working version of ROS, complete with tutorials to get me up and running with total ease.

GAZEBO INSTALLATION

The SRC also mandated that teams use a specific version of Gazebo. The version of Gazebo packaged with ROS-Indigo is version 2.x.x; however, for the SRC, we needed to remove that version and upgrade to the latest version, which at the time of the SRC was version 7.0.x.

After loading the correct versions of ROS and Gazebo, the next step was to set up the SRC Simulation. The Open Source Robotics Foundation

THE ROS ENVIRONMENT ALLOWS YOU TO CODE IN MANY LANGUAGES. IN FACT, YOU CAN CODE IN MULTIPLE LANGUAGES AND HAVE THEM WORK TOGETHER SEAMLESSLY THROUGH THE CREATION OF SEPARATE ROS NODES.

(OSRF) created and maintained the SRCSim and they provided a web page with all of the necessary steps to perform the SRCSim installation and setup. Links to all of the software needed to run the simulation are found at the Resources section at end of this article.

QUALIFICATION ROUND

Once the install of the ROS, Gazebo and SRCSim software was completed successfully, the teams would be ready to start the actual process of coding solutions to the two qualifying tasks. The ROS environment allows you to code in many languages. In fact, you can code in multiple languages and have them work together seamlessly through the creation of separate ROS nodes.

ROS nodes are basically pieces of code that help control the robot. Robot control usually involves multiple nodes. Nodes can control things like a laser range-finder or the motors associated with wheels on the robot. Nodes also are used to perform path planning or to provide a graphical view of the robot's world. For the SRCSim, our team had nodes that controlled the robot's walking and the robot's arm movement. We also had a node that controlled the vision system for identifying LED lights.

The most often used languages for programming nodes are C/C++ and Python, although C#, Java, Ada and Assembly, along with a variety of other languages also will work. Our team chose Python mainly because most of us already had some experience with the language, and because it is a fairly easy language to pick up if you have never used it before.

The qualifying round lasted approximately three months. Three months is not a long time when you are working a full-time job and are trying to

find any amount of spare time to work on maximizing robot performance. In order to utilize our time as efficiently as possible, we split the team into two sections with each section being responsible for a single qualifying task. We also had an individual team member float between the two tasks to pitch in wherever needed.

Working with ROS in the Linux environment involves a good deal of command-line interaction. The ability to open multiple terminals and have each one dedicated to various operations was a real plus. Many times, I would have one terminal to launch the simulation, another terminal to produce output from the robot sensors and still another window to send commands via messaging to the robot. It was not unheard of to have five or more terminals open at a time all communicating via the ROS messaging network.

CHALLENGE RESULTS

More than 400 teams from 55 countries around the world pre-registered for the Space Robotics Challenge. Of those 400-plus teams, only 92 of them submitted valid solutions for both qualifying tasks. From the 92 qualifying submissions, 20 teams qualified to move on to participate in the Virtual Competition set for June 2017.

As a team, our goals for the challenge were simple. First and foremost, we wanted to have fun. Second, we wanted to learn. Each of us had a desire to become more proficient using ROS and Gazebo, while others wanted to get some experience with Linux and Python, having never used either of them. Finally, we wanted to be competitive. Our goal was to submit real solutions to the two qualifying tasks in the hope that we would be in the top 20 and move to the next round of the competition.

I am proud to say, we met all of our goals—well, almost all of them. We did submit solutions for both tasks. Unfortunately, we fell short of qualifying, but we definitely were competitive. All in all, it was a great experience, and I recommend that anyone who has an interest in robotics look seriously at utilizing the Linux stack of software mentioned in this article as a starting point.

TEAM MEMBERS

For the Space Robotics Challenge, my team, named Patriot Robotics, consisted of myself and four other talented individuals: Erica Kane, Dan

Sionov, Marty Kube and William Marrieta. They were an absolute pleasure to work with. All of us are amateurs in this arena, and although some brought a bit more experience to the team than others, we all worked together very cohesively. I want to thank each of them for making the experience both rewarding and enjoyable.

CONCLUSION

Working on the Space Robotics Challenge and collaborating with my team was great fun. The truly remarkable thing about participating in the SRC was that it cost nothing. In fact, for nothing more than the cost of your time, you can utilize the software mentioned in this article to run the same simulations that the Space Robotics Challenge used in its qualifying round. The basics of ROS and Gazebo are pretty easy to pick up. Without much effort and by using the supplied tutorials as well as other on-line

The TurtleSim Tutorial

One of the best things about ROS is that it has a very rich tutorial library. The tutorials are a great way to come up to speed with ROS and Gazebo. If you are interested in learning ROS, one of the first tutorials you should try is the turtlesim.

The turtlesim teaches you how to create ROS packages, work with multiple nodes, publish and subscribe to topics, and work with ROS services.

The tutorial provides step-by-step instructions as well as several online videos to walk you through the process of setting up and running the simulation. The simulation involves hands-on programming and command-line interaction through ROS. Learning how to make the simulated turtle move, rotate or spawn copies of itself are just some of the features of this tutorial.

The turtlesim is a fun and easy way to get ROS up and running. If for no other reason than to just control a simulated turtle, I encourage you to give it a try.

resources, anyone with little or no experience can set up and run the software. You even can try your hand at creating solutions and seeing if you can score well enough to have qualified for the Space Robotics Challenge. If you do, feel free to email me your results. Good luck!■

Paul Ferretti has more than 20 years of experience as a software engineer. He is a Senior Member of the IEEE and a former Vice Chair for the Washington DC/Northern Virginia Chapter of the IEEE Robotics and Automation Society. He welcomes your comments sent to pferretti@ieee.org.

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.

RETURN TO CONTENTS

Resources

ROS History: http://www.ros.org/history

About ROS: http://www.ros.org/about-ros

ROS.org, Installing from Source: http://wiki.ros.org/indigo/Installation/Source

Space Robotics Centennial Challenge: https://ninesights.ninesigma.com/web/space-robotics-challenge

STMD: Centennial Challenges: https://www.nasa.gov/directorates/spacetech/centennial_challenges/index.html

Space Center, Houston: https://spacecenter.org

Johnson Space Center: https://www.nasa.gov/centers/johnson/home/index.html

Open Source Robotics Foundation: http://www.osrfoundation.org

The Institute for Human & Machine Cognition (IHMC): https://www.ihmc.us

Patriot Robotics: http://www.patriotrobotics.net

Gazebo: http://gazebosim.org

Tutorials for Gazebo: https://bitbucket.org/osrf/srcsim/wiki/tutorials

tutlesim: http://wiki.ros.org/turtlesim