Visualisation and Simulation of the Nao Robot using ROS

Abstract

Table of Contents

Introduction 1.

The purpose of this project is to investigative, to explore the viability and the usefulness of using ROS as a platform for the RoboEireann team. This would be of interest to the team and in terms of robotics in general because as a rule, debugging in robotics can be notoriously difficult.

Technical Background 2.

RoboEireann is Maynooth University’s team competing in Robocup’s Standard Platform League. Currently, the Nao robot, produced by Aldebaran Robotics, is used as standard. In this league, two teams of robots play soccer fully autonomously. Aldebaran’s Nao robot is a humanoid robot 58 centimetres tall. It is articulated with joints similarly to a human, and is fitted with a front and downward facing cameras, two sonars, microphones and speakers, and a variety of other sensors. By default it has a Linux-based operating system called Naoqi installed. For the purpose of Robocup, the RoboEireann robots run a stripped-down version of this Naoqi system alongside RoboEireann’s own codebase.

Robotics Concepts

Links, Joints and Kinematic Chains

A link is a rigid body in a robotic system. This can be a robot’s limb, or any other rigid object needing to be represented in a robotic system. Two links joined together produce a pair, e.g. a hinge or ball & socket joint. A link can be said to represent the system’s attachment to the world, and is referred to as the fixed link. A series of links and pairs together is called a kinematic chain.

Frames & Transformations

A frame is a set of axes used to describe a location and orientation in space, typically attached to a link. When working in three-dimensional space, a frame has 3 axes at right angles to each other, and 6 degrees of freedom. This means 6 numbers are needed to define its position and orientation, i.e., x, y and z position, as well as rotation around each axis. However a frame’s location information has no meaning without another frame to compare it to. This difference between two frames’ locations and orientations is called a transformation. When a number of frames are linked together by transformations any frame can be compared spatially to any other frame by following each transformation between them along the kinematic chain.

2.1 ROS

The Robot Operating System, or ROS as it will be called in this document, is an open source collection of tools and libraries built for the purpose of robotics. It has been developed since 2007 and has moved through alphabetically named versions since 2010. The version I was using for this project was Indigo Igloo, more commonly known simply as Indigo. At present, it is the most stable and widely supported version. As ROS Indigo was aimed at Ubuntu 14.04, that is the version of Ubuntu that was used in this project.

2.1.1 Packages and filesystem

ROS’s filesystem is divided into packages. Some of these come as part of the main ROS install, but many, many others created and maintained by developers in the community. There are currently over 2000 packages available, these supporting approximately 80 commercially available robots. Packages are atomic in that it is the smallest individual thing you can build in ROS, and are designed on the principle that each one has a certain amount of functionality, but not too much that it becomes difficult to use by other packages and software. In this way, the system is very well distributed and abstracted.

A package will contain, at minimum, a package.xml file, and a CMakeLists.txt file. The package.xml file lists metadata for the package, such as the package’s name, version, maintainers, as well as the packages dependencies. When building the package, the CMakeLists.txt file describes how to build the code and where to install it.

A package’s dependencies can be split into first-order and indirect dependencies. First-order dependencies are the ones listed in the package.xml file, directly from one package to another, but indirect dependencies are transitive. If package A depends on package B, and package B depends on package C, then A indirectly depends on package C. Because packages are made to be small, most packages have a lot of dependencies.

2.1.2 Nodes, Topics and Messages

At runtime, ROS operates by using nodes and topics. A node is an executable file within a ROS package. Nodes can publish or subscribe to topics, which are named buses for exchanging messages. A topic can have multiple publishers and subscribers.

2.2 The Aldebaran Nao

The robot used in this project, and by the RoboEireann RoboCup team, is the Nao V4, produced by Aldebaran Robotics. It is a humanoid robot standing at 58 centimetres tall. It is articulated with joints similarly to a human to give it a similar range of movement. It is also outfitted with two cameras, one facing straight out from its face and the other pointing from its face towards its feet. Two sonar sensors face out from its chest at slight angles to the left and right. Two microphones and two speakers sit on the sides of the robot’s head.

2.2.1 NAOqi

NAOqi refers to the name of the operating system running on the Nao robots, whereas the NAOqi framework is the programming framework used to program NAO. It addresses robotics-related needs such as parallelism, resources, synchronization, and events. The framework allows homogenous programming, information sharing and communication between modules.

The NAOqi framework:

* Is cross platform
* Is cross language, with an identical API for C++ and Python
* Provides introspection, i.e., it knows which functions are available in different modules

2.3 RoboCup Standard Platform League & RoboEireann

In RoboCup’s Standard Platform League, two teams of robots play soccer fully autonomously. Each team consists of no more than 5 playing robots and one coaching robot. The coaching robot observes the game from an elevated position and gives tactical advice to its teammates. The pitched played on measures 9 metres by 6 metres. The Aldebaran Nao is the standard platform used, to allow teams to focus on software development rather than the mechanics of the robots themselves.

RoboEireann is Maynooth University’s RoboCup team, and currently Ireland’s only team. Originally the team entered in 2008 with the University of Newcastle, Austrailia, under the team name “NUMmanoids”, where they won first place. Since 2009 they have competed under the name RoboEireann.

Previously Completed Work

The University of Edinburgh’s RoboCup team “Edinferno” currently use ROS, however they use it in a capacity beyond the scope of this project. While this project is aimed at using ROS as a tool for visualisation and debugging, the Edinferno team install ROS on the Nao robots themselves and use this as the controlling software for the robot.

There are currently <INSERT NUMBER> of maintained ROS packages for the Nao, these include drivers, controllers and GUIs. The full list of ROS packages used in this project is listed <INSERT HERE>

The Problem

The hattrick codebase has been worked on for the past 7 years now, but despite this, it has its own limitations and drawbacks which have been caused by a number of factors. Over the years, hattrick has been worked on by a large group, as different students have come and gone. Any time a student writes code and leaves the team, the amount of legacy code grows. Many of these modules, and their dependencies, are badly documented. This means that editing or removing modules can have unforeseeable knock-on ramifications. In this sense the code is partially monolithic.

Another issue is with using external tools for things like calibration and debugging. With the tools and software currently used, many connect to hattrick differently, which means that a sizeable amount of code on the hattrick system is solely written for connecting with different tools.

In a standard RoboCup, there are three main stages the robot will be in.

1. Calibration & Configuration (not in play)
2. Debugging & Visualisation (not in play)
3. Gameplay

Before the game begins, the first two steps will be repeated; ideally until the team is satisfied, but typically until the game starts. For the first two stages,

The Solution

/\* ROS is a potential solution to this problem for a variety of reasons. As a suite purpose-built for robotics applications, its tools are suited to this problem. Rosbags could record robot sensor information during a game, to be played back and analysed at a later stage. Rviz can be used for simulating the robot while the rosbag is played, or connecting to the robot in real time. Both of these would be invaluable tools for debugging. /

* Idea for bringing nao information into ros

Once the nao\_bringup package was running, the robot could be visualised in Rviz and a variety of its sensor data could be accessed through ROS. Using rostopic all topics could be viewed that the robot’s sensor data was being published to, and using “echo” these messages could be viewed in terminal. In Rviz, all the robot’s frames and transformations between these frames were visible, giving a skeleton view of the Nao. Meshes for the Nao exist online as a package, and once these were downloaded, a colour model of the Nao was visible in Rviz.

Insert image

At this point, the NAOqi driver was interfacing with the robot, which was running the basic NAOqi configuration which is the factory configuration on the Nao. As the RoboEireann team strip out quite a lot of modules that aren’t necessary for RoboCup, these modules had to be stripped out to prove that the robot could still communicate with ROS without these. The only base modules that are left running in RoboEireann’s robots are called “albase”, “alsystem”, and “dcm\_hal”. Using SSH to access the Nao’s software, these three modules were disabled on the autoload file via comment. The autoload file lists the modules to be run when the Nao is booting up. Unfortunately, when the nao\_bringup was launched after these modules were disabled, the process failed. Error messages indicated an error with the naoqi\_driver package. The logical next step would be to try to get this naoqi\_driver to work with the RoboEireann codebase.

To do this, there were two options. Either a new ROS package would have to be developed to interface with the RoboEireann code, or the RoboEireann code would have to be changed so to the NAOqi driver it would resemble the NAOqi code. The latter was determined to be hugely unviable, due to the huge amount of legacy code in the RoboEireann codebase to familiarise myself with in a short timeframe, also that interfering with the RoboEireann code would hinder the RoboEireann team. The second option was chosen by default, however, after an attempt to reverse engineer the naoqi\_driver package from its source code on GitHub, it was discovered to contain over 10,000 lines of code. Because of this, this option was also deemed unrealistic with the given timeframe.

At this point it seemed like a dead end, as the project was initially intended for the benefit of the RoboCup team, and it seemed unviable to get ROS to work with the RoboEireann codebase. The decision was then made to continue on with the project as if the two were compatible, as a demonstration of what could be done with ROS and the Nao robots if a RoboEireann-compatible driver package was written for ROS. After researching potential avenues of investigation, a ROS package was found that contained AR Toolkit packaged for ROS use. After working through demos and researching AR Toolkit’s capabilities, the decision was made to use the ROS ar\_tools package to localise the Nao on a simulated RoboCup field using AR tags.

While experimenting with the ar\_tools package using the Nao’s camera, a problem was encountered. The topic that the Nao driver node publishes its camera data to was different to the topic that the AR Tools node subscribed to. Roslaunch has a remap command that can be specified in the launch file, however this did not seem to be working. To understand how this remap command worked a simpler publisher/subscriber example was written, where the remap command worked as expected. A remap command in the launch file will change the topics that the subsequent nodes publish to. Unfortunately, the remap command didn’t seem to work anywhere in the nao\_bringup launch file or the naoqi\_driver launch file, which was launched within the former. To remedy this, the header file within the ar\_tools package was edited so that it would subscribe to the same topic that the Nao driver node published to.

It was determined that this could be used to solve a RoboCup based problem, localisation. To test this, an AR tag was hung 80cm (the height of the goal crossbar in RoboCup) from the ground. A launch file was written to start the nao driver node, an AR Toolkit node that would recognise and plot a single AR tag, and Rviz configured to show both the robot and the AR tag’s frame. In Rviz, the robot was visible as well as a transform from the robot’s camera to the AR tag. The AR Toolkit node also drew a visualisation marker at the frame. Using the nao\_walker node teleop keyboard commands were sent to the robot and the movement between the robot’s odometry, the robot, and the tag could be seen.

Next, a transform was written between the AR tag and the pitch. The frame was placed where the centre of the pitch would be, and using Rviz’s libraries, a rectangle 9 metres by 6 metres (The dimensions of a RoboCup pitch) was drawn. Upon testing this, the pitch was not drawn accurately. This was because the AR tag used to represent the goal measured roughly 15cm by 15cm, and was statically transformed to a far larger object. This meant that any error in the pose and location of the AR tag would have a proportionally greater error in the pose and location of the pitch. The robot was seen to be floating above the simulated pitch, or clipping through it.

To solve this problem, a transform was needed that would make sure that the pitch’s position and rotation relative to the Nao wouldn’t change beyond the way a football pitch in real life normally would. These changes were determined to be along the x and y axes, and rotation around the z axis. A TF Listener object was created, which had methods to return the transformation between two specified frames at a given time. From this listener, the relevant transform information was pulled from the transform between the AR tag and the camera, and with this information, a transform from the pitch to the robot’s footprint was created.

This created another problem, this time relating to ROS’s TF system. The robot’s TF tree created by the naoqi driver listed “odom” the robot’s odometry frame as the parent of “base\_footprint”, the robot’s footprint. A frame can only have one parent, so when we created the transform from the pitch frame to the footprint frame, a conflict was created. This was easily remedied, by swapping the frames so that the transformation went from the footprint to the pitch. The TF library had a method to calculate the inverse of a transform, so making this change was easy.

* Impracticality of doing this with the robocup system
* Demonstrating Rviz and Rosbags, using the ar\_tools package
* Rosout for debugging

Evaluation

One of the issues that was encountered in this project

Conclusion

* Implications of my work

The Implications of this project open up new avenues for RoboEireann, and show others to be unrealistic.

* Very specific to robocup

The implications of this project are very specific to RoboCup and RoboEireann.

* Limitations and practicality of the ROS system
* Approach

The approach I took in this project was non-linear, as having a project with such an open-ended title,

* Future work

There are a number of paths the RoboEireann team could take in the future regarding ROS. The most obvious project that could be undertaken would be the writing of a hattrick-compatible driver. In this project, the naoqi-driver has been shown to open up new possibilities for working with the Nao robots. Having vision tools such as AR Toolkit, visualisation in Rviz, and using the keyboard to send teleop messages all from a single bridge, and having no contention between the different packages, would be an invaluable asset to the RoboEireann team. In addition to the tools shown in this project, ROS also has support for OpenCV, Gazebo, Moveit! and more.

Another, more complex route that could be taken, would be to run ROS directly on the Nao. If this project were to be completed, development for RoboEireann could be simplified greatly. This is because ROS had packages and tools to address a multitude of common robotics problems, solutions which would otherwise have to be written into the hattrick system.

References

Aldebaran NAOqi Documentation - <http://doc.aldebaran.com/1-14/dev/naoqi/index.html>

Official ROS website - [www.ros.org](http://www.ros.org)

Official ROS wiki - wiki.ros.org

Morgan Quigley, Brian Gerkey, & William D. Smart – Programming Robots with ROS, O’Reilly Media

Appendices

-tf frames

-hattrick

-semi-monolithic

-dark code

-legacy code

-need for abstraction

-constantly changing team

-extreme learning curve

-need for modularity

-ros allows access to multitude of other tools