Simulation Support for CAN devices

In FRC-WPI Robot designs

(2016)

Motivation

One of the main factors that limits the more widespread use of simulation in FRC robot development and testing is the lack of support for CAN devices (specifically motor controllers) which are now the de-facto standard in new FRC designs.

Investigation Objectives

1. To develop a set of software tools that will allow robot source code to contain CAN class objects and functions that can be compiled for both simulation and deployment targets
2. To require as little additional coding effort as possible from student programmers to add simulation support to their robot project code
3. To require as little modification as possible to the Eclipse project environment or properties to support both deploy and simulation build types
4. To make sure that adding simulation support doesn't degrade efficiency (or in other ways impact) the code used for hardware deployment
5. To simplify simulating CAN motors in the Solidworks exporter and to also fix other problems that currently exist
6. To offer the solution (or solutions) obtained to as wide an audience as possible in the First community

1 Limitations

1. Only CANTalon devices will be supported (no CANJaguar etc.)
2. Eclipse C++ projects only initially (no Java support)
3. Not all CANTalon functionality will be supported (e.g. see below)
4. Performance and behavior in simulation will not necessarily match that observed on hardware

2 Supported features

1. Multiple PID controller channels (CANTalons currently offer 2)

2. Internal “Hard” limit switches
3. Built in QuadEncoder
4. Soft limits

3 Unsupported features

1. Other feedback devices (e.g. potentiometers)
2. Hardware based Limit switch behavior (e.g. normally open, closed etc.)
3. Hardware based encoder properties (in simulation all encoders are 360 ticks per revolution)
4. Motion profiles

Investigation Outline

There are several possible approaches that can be envisioned that would meet the stated objectives with varying degrees of success. It is planned that all of these will be investigated to some degree

1 Build Separate CAN dynamic library

Develop a stand-alone (Linux) dynamic link library (.so) and header file that adds support for CAN devices

Note: In the linux_simulate wpi source tree the header files for the CANTalon class are not included so there is no naming conflict if a new set of files that define a (simulation only) CANTalon class are used to build the library

Details

1. A new project “SimCanLib” was added to the 159 git MentorRepository
    - ~/MentorRepository/Software/workspace/SimCanLib
2. SimCanLib is a dynamic library project that includes “CANTalon.h” and CANTalon.cpp and produces SimCan.so as it's build artifact
3. As part of the SimCan project build process CANTalon.h and SimCan.so are copied to the user's ~/wpilib/simulation directory (which is where Eclipse looks for simulation libraries and header files by default)

Pros

1. Minimal impact on WPI simulation library build environment and process
2. Can be developed entirely as a special Eclipse project

3. Source control requires 159 github repository only

Cons

1. Requires some (minor) modifications to user source code
   - need to add: #include “CANtalon.h” to most source files (since it isn't included in “WPILib.h” in simulation)
2. Requires minor modifications to project build properties
   - need to modify c++ build properties to add the library SimCan
3. Difficult to migrate this solution upwards or merge in changes from wpi since it is not part of the “allwpilib” project

Status

1. This investigation has been completed and tested on a modified version of the 2016 “SimulatedRobot” project (SimCanLibTest)
   - see: ~/MentorRepository/Software/workspace/SimCanLibTest
2. All GPMotor objects and function calls in SimulatedRobot replaced by CANTalon equivalents
3. Added and tested “soft limit” code to Holder that uses built-in (emulated) CANTalon encoder

2 Add Simulated CAN support to standard WPI library

Modify the wpi code and build process to include support for CAN devices to the currently used simulation library

Details

1. checkout allwpilib
2. Create a local branch “test” in the github project “allwpilib”
3. Modify scripts in test branch to add CANTalon support in simulation build
4. Use CANTalon source and header files developed as described above

Pros

1. Requires no changes to the user source code or project build properties
2. Easier to merge in changes from wpi and offers a possible upward migration path

Cons

1. Requires building a customized version of the wpi simulation library
    - However, this needs to be done anyway since there are critical bugs in the current wpi simulation code that need to be patched
2. Requires installing modified files to user's ~/wpilib/simulation directory
    - Also currently needed because of reason given above
3. Unclear how to manage source code control between team 159 MentorRepository and allwpilib git repositories
4. Stand-alone library test project should(?) generate link errors once CANTalon.so has been included in wpi library (libwpilibcSim.so) unless

Notes

1. Checked out allwpilib from First repo
    - git clone https://usfirst.collab.net/gerrit/p/allwpilib.git
2. Created a test branch in allwpilib and switched to it
    - cd allpilib
    - git branch test
    - git checkout test
3. Checked bug fixes for simulated PIDController etc. into test branch

    modified: wpilibc/simulation/src/Notifier.cpp

    modified: wpilibc/simulation/src/PIDController.cpp

    modified: wpilibc/simulation/src/simulation/SimContinuousOutput.cpp
4. Was able to build a wpi simulation library (wpilibcSim.so) that included CANTalon.cpp and CANTalon.h by copying those files from the MentorRepository SimCanLib project into the allwpilib/simulate src and include directories and then running "./gradlew build -PmakeSim" from ~/allwpilib
    - Added #include "CANTalon.h" to the bottom of "WPILib.h"
        - Order is important since headers for member objects (Encoder etc) must in included first
    - Removed #include "CANTalon.h" from CANTalon.cpp
        - Not needed anymore since CANTalon.h is now included in WPILib.h
5. To package headers, plugins and libraries into a single directory (simulation) that can then be

copied (or tarred) into ~/wpilib execute the following from a command shell in the top level allwpilib directory:

- ./gradlew :wpilibc:wpilibcSimCopy -PmakeSim
- directories copied into ~/allwpilib/build/install/simulation

6. To create a zip file for installation
   - ./gradlew -PmakeSim simulation:zip
   - zip file created: ~/allwpilib/simulation/build/distributions/simulation-trusty.zip
   - to install into ~/wpilib execute from ~/allwpilib/simulation/build/distributions
     - unzip simulation-trusty.zip -d ~/wpilib/simulation
7. Tested SimCanTest using allwpilib headers and libraries
   - Removed SimCan from linker library list and did a clean build
   - gazebo simulation worked (as before when using libSimCan.so as additional library)

TODO

1. Come up with a method to combine allwpilib and 159 Mentor git repository trees and a better library installation plan

3 Add CAN device support to Solidworks exporter

Integrate CAN devices directly in both the Solidworks exporter and c++ Software components

Details

1. Set up a windows build environment to generate Solidworks "GazeboExporter" add-on
   - Obtain and install free FRC Solidworks student version
   - Obtain and install free Microsoft 2015 VisualStudio (community version)
2. Clone the gazebo exporter github project and create a local branch for testing
   - git clone https://usfirst.collab.net/gerrit/p/gazebo_exporter.git
3. Modify the Solidworks Gazebo Robot "Manage" UI to add a CANMotor option with additional entry fields to support built-in encoder and limit switch interfaces
4. Modify the add-on source code to produce special sections in the exported sdf file that support CAN devices

5. Create a new CAN Talon plugin that can be used to send and receive messages to gazebo
6. Modify the CANTalon source code to interact with the new plugin (instead of the dc_motor plugin that the base class Talon uses)

Pros

1. Offers integrated support for CAN devices at both CAD and software levels
2. Can merge in changes from upstream gazebo exporter project

Cons

1. For Solidworks support need to install custom built dynamic linked libraries (dlls) into end user's C:/Program Files/GazeboExporter/GazeboExporter directory
2. Need to develop a special CAN Linux plug-in and modify c++ code to use it
3. As in other strategies, requires installing modified files to end user's ~/wpilib/simulation directory (plugins etc)
4. Need to figure out how to do source control without pushing to remote wpi-first repo

Notes

1. Was able to import the github GazeboExporter c# project into VisualStudio, build it and test it in Solidworks
    - VS has a nice Debug feature that starts up Solidworks automatically for testing
2. Fixed an existing bug in the current code pertaining to internal limit switches
    - Original code prevents setting lower bounds for limit switches to something >0
3. Fixed a plug-in naming compatibility problem in the original code
    - current exporter uses older “libgz_dc_motor” (e.g.) naming convention for plug-ins but 2016 wpi plug-ins are now named “libdc_motor” etc.
4. Modified code to add a new CANMotor device to the object palette bar
5. Modified code to produce different sdf output for CANMotor than original Simple Motor object
    - replaced “./pwm/1” etc. in plugin section with “./can/1” etc.
6. Modified CAN Motor dialog box in Visual Studio to include extra fields for limit switches

TODO

1. Develop new CANTalon plugin and modify c++ code to use it

2. Add encoder and limit switch properties to CANMotor code in Solidworks exporter

Conclusions

1 Development

Of the three investigations that were carried out the second (add CANtalon support to allwpilib simulation build) seems to provide the best compromise between providing user functionality and the amount of additional support and development effort required to implement. Modifying the Solidworks exporter to include a separate CANmotor device and dialog might be something that could be considered in the future. However, it isn't clear what real benefit that would provide over the existing interface which uses PWM and DIO channels to map the functionality required to emulate the built in CANTalon motor, limit switches and encoder features. The current paradigm also has advantage that it better emulates the need to attach physical hardware (e.g. real limit switches and encoders) to a joint in the hardware phase of the design.

2 Implementation

Development Status

1. The c++ code required to implement the (limited) CANtalon functionality described previously has been written and is provided in two files (CANTalon.h and CANTalon.cpp) that have been added to the allwpilib/wpilibc/simulation include and src directories respectively. The version of WPILib.h in the simulation include directory has also been modified to add “CANTalon.h” to the list of included header files.
2. The allwpilib source tree has also been modified to fix the Bugs described above for the simulated PIDController class, etc.
3. A test application project (SimCanTest) has also been created and is based on the SimulatedRobot project that was developed in parallel to the team's 2016 competition robot project that uses (simulated) CANtalon functions exclusively to control and access the encoders and limit switches that were used in the design.

Source Code Control Details

1. The repository that contains the modified allwpilib source code and test project is available at the FRCTeam159 github website (https://github.com/FRCTeam159/MentorRepository)
2. The allwpilib source code is contained as an eclipse project at Software/workspace/allwpilib in the MentorRepository. The source tree in the project directory was based on the latest released allwpilib version first checked out from https://usfirst.collab.net/gerrit/p/allwpilib.git and then modified to include the bug fixes and simulated CANTalon support described above.

   When the WPI allwpilib git project was added to the team's MentorRepository only the original and modified source tree was committed (since git doesn't push nested ".git" directories)

3. A special branch "cantest"was created to contain the Solidworks Gazebo exporter investigation code. The visual studio gazebo_exporter project was first cloned from https://usfirst.collab.net/gerrit/p/gazebo_exporter.git and added to the MentorRepository at Solidworks/GazeboExporter/gazebo_exporter.

   As for the case of allwpilib only the source tree was committed to the team's repo (not the First/WPI cloned ".git" repository directory)

   The "master" branch contains the code changes that support the bug fixes and CANTalon simulation enhancement consistent with the previous Solidworks exporter paradigm

4. Merging and pushing from/to the external 3rd party repositories

   If the Team159 MentorRepository is checked out on a new system the (modified) source trees for the First/WPI allwpilib and gazebo_exporter projects will be available in the subdirectories described previously. It should then be possible to add the original repositories for these projects using a clone or other operation which *should* support command line merges etc. from the remote URLs while in the directory that contains the .git subdirectory (note: this hasn't actually been tried yet)

Building

1. allwpilib

   for simulation purposes the allwpilib project needs to be built from an Ubuntu 14.04 operating system environment (either native or using a virtual machine)

   the allwpilib build process uses "gradle". Instructions can be found in the README.md file in the top level directory.

   - ./gradlew build -PmakeSim   (creates build/install sub-directory)
   - ./gradlew wpilibc:allcsim -PmakeSim (builds libraries)
   - ./gradlew simulation:zip -PmakeSim (packages header files into a single directory)

   Following the above operations most of the files needed for simulation will be present in allwpilib/build/install/simulation but to make a complete tar file for export the gz_msgs lib and header files should also be built and copied into this directory

   - ./gradlew wpilibc:gz_msgs -PmakeSim
   - cp -R ~/allwpilib/build/simulation/gz_msgs/generated/simulation/gz_msgs ~/allwpilib/build/install/simulation/include
   - cp ~/allwpilib/build/simulation/gz_msgs/libgz_msgs.so ~/allwpilib/build/install/simulation/lib

2. gazebo_exporter

   This project must be build in a Windows environment using Visual Studio (e.g. the free “Community” version that can be downloaded from Microsoft)

   Check out the “cantest” branch in the MentorRepository to continue the work-in-progress for a built in Solidworks CANMotor device or the “master” branch to just get the bug fixes for the existing exporter (e.g. fix limit switch problem etc)

   To build the project, start VS and open the “GazeboExporter.sln” solution project in MentorRepository/Solidworks/GazeboExporter/gazebo_exporter

   Building either the Debug or release configuration will produce a GazeboExporter.dll file that can be copied to the C:\Program Files\GazeboExporter directory on the Windows computer for testing or use from outside Visual Studio

Installation

1. For convenience a tar file (wpilib.simulation.tar) that includes linux libraries and header files that support CANtalon simulation has been placed in ~/Robotics/MentorRepository/patches
2. To install in Ubuntu, untar wpilib.simulation.tar file into ~/wpilib (e.g. tar -xf wpilib.simulation.tar -C ~/wpilib)