Automated Code Generation of Simulink Models as ros_control Controllers

G.A. vd. Hoorn
Cognitive Robotics
Delft University of Technology
g.a.vanderhoorn@tudelft.nl

Murat Belge
Development Lead
MathWorks, Natick USA
mbelge@mathworks.com
Introduction

Code Generation

```c
#include "AutomatedParkingValetAlgorithm.h"
#include "AutomatedParkingValetAlgorithm_private.h"

int32_t dir_x32_floor(int32_t numerator, int32_t denominator)
{
  int32_t quotient;
  int32_t absDominator;
  int32_t tempAbsQuotient;

  absDominator = 0 < denominator ? (denominator < 0) ? (-static_cast<int32_t>(denominator)) : denominator : static_cast<int32_t>(denominator);
  if (absDominator == 0)
  { return 0; }
  tempAbsQuotient = absNumerator / absDominator;
  if (absDominator > 0)
  {
    tempAbsQuotient++;}
  if (quotientNeedsNegation) quotient = -quotient;
  return quotient;
}

void AutomatedParkingValetModelClass::sAVU_xenInit_real_T(sxarray_real_U* x)
{ *x = 0; 
  for (int i = 0; i < numDimensions; i++)
  { 
    *x = *x + dir_x32_floor(int32_T x[i].floor, int32_T x[i].numerator, int32_T x[i].denominator); 
  }
}
```
Motivation

**Idea:** Design a controller in Simulink and use automated code generation to create a ros_control controller

- Many benefits
  - Model-based design
  - Virtual testing and validation in simulation environment
  - Automated C++ code generation from a graphical model of the controller eliminates hand-coding

- MathWorks and Delft University of Technology partnered to bring Simulink and ros_control together
Agenda

- ros_control overview
- Simulink C/C++ code generation
- ros_control as a codegen target for Simulink models
- Summary
What is ros_control

- Generic set of tools for control of robots (mostly PID)
- Generalization of pr2_mechanism controllers (PR2)
- ROS 1 and ROS 2
- Used by enthusiasts and OEMs (Clearpath, PAL, UR, ABB, Franka Emika, ..)
- Two main parts:
  - Abstracted hardware access (including simulators)
  - Controllers with standard ROS interfaces (FollowJointTrajectory, etc)
- Decoupled execution policies
- Hard real-time compatible

**Main goal**: promote reuse of control code
ros_control: abstract hardware

- No direct access to hardware
- RobotHW class wraps:
  - Registers / memory
  - OEM API / SDK
  - Fieldbus
  - TCP or UDP
  - ROS topics / services / actions
- Exposes abstract resources (*handles*):
  - OotB: position, velocity, effort, IMU, F/T
  - Plugin based (custom resources)
- Conflict detection (sometimes resolution)

Original diagram: Adolfo Rodríguez Tsouroukdissian
ros_control: controllers

- Claim *resources*, not hardware
- Expose ROS interfaces
- Runtime load, init, (re)start, stop
- Runtime reconfigurable & switchable
- OotB: position, velocity, effort, trajectory execution
- Plugin based: custom controllers:
  - Whole-body (NASA Valkyrie, PAL)
  - Affordance
  - Cartesian F/T
  - Visual Servoing
  - Etc

Original diagram: Adolfo Rodríguez Tsourouchissian
ros_control: architecture

- Layered architecture
- Single process (multi-threaded)
- Determinism within node (execution)
- OEM provides up to the interfaces layer
- RobotHW transforms data:
  - From HW to ROS (ex: enc ticks → rad)
  - From ROS to HW (ex: rad → enc ticks)
- Combine RobotHW (OEM1, OEM2, …)
- Controllers are user-facing
Agenda

- ros_control Recap
- Simulink C/C++ Code Generation
- ros_control as a codegen target for Simulink models
- Summary
Simulink Code Generation

Generate *efficient* C++ code that *plugs easily* into your C++ architectures.
Roles and Goals of Code Generation

Roles
- Acceleration
- Prototyping
- Production
- Verification

Goals
- Correct
- Efficient
- Customizable
- Certifiable
- Readable
- Scalable
- …

User design in high level / domain specific languages

General purpose programming languages

C code
C++ code
HDL code
PLC code
GPU code
Success Stories

NASA Orion Program

“All of our GN&C flight software is automatically coded into C++ for deployment onto the vehicle.”

Voyage

“Simulink + ROS allowed us to deploy a Level 3 Autonomous vehicle in less than 3 months”
Agenda

- ros_control Recap
- Simulink C/C++ Code Generation
  - ros_control as a codegen target for Simulink models
- Summary
Connect MATLAB and Simulink with ROS

1. rosbag import
   Offline analysis

2. ROS Connection
   Prototyping

3. ROS node generation
   Standalone implementation (C++ node generation)

MATLAB
Perception
Planning & decision
Control
Windows/Mac/Linux

Simulink

ROS
ROS 2
Network

ROS/ROS2 nodes

robag

Linux

Simulator
Real hardware

MATLAB/Simulink

ROS PC

ROS 2
Connect MATLAB and Simulink with ROS

- MATLAB
  - Perception
  - Planning & decision
  - Control

- Simulink
  - ros_control controller generation

- Windows/Mac/Linux
  - MATLAB/Simulink
  - ROS node generation
    - Standalone implementation (C++ node generation)
  - rosbag import
    - Offline analysis

- ROS PC
  - ROS
  - ROS 2
**Proof-of-Concept:** Differential Drive Proportional Controller

**Proportional Controller**

- Desired position
- Position Y
- Position X
- Orientation (θ)

Basic algorithm:
- Calculate difference between current and desired locations
- If distance is below a threshold, stop the robot.
- If distance is above the threshold, set yaw angle to orient robot toward desired location, and drive towards it.

**Colored blocks indicate tuneable parameters**
Proof-of-Concept: Differential Drive Proportional Controller

- Test & validate controller in Simulink environment
Steps to Deploy Simulink Controller as a `ros_control` Controller

1. Create interface model
2. Adjust model settings
3. Generate code
4. Integrate generated code into `ros_control` framework
Create Interface Model

Set Point

Inputs
- IsNew
- /command
- Msg
- Current X
- Current Y
- Current Theta

Outputs
- Desired position
- Position X
- Position Y
- Orientation (theta)
- leftWheelSpeed
- rightWheelSpeed
- LeftWheelSpeed
- RightWheelSpeed

Root I/O Ports
2 Adjust Model Settings

- **Set target**
  - Hardware board: Robot Operating System (ROS)
- **Set build process**
  - Generate code only
- **Configure codegen interface**
  - Code interface packaging: C++ class
Generate Code
Integrate Generated Code into `ros_control` Framework

```cpp
namespace SL {
    class modelClass {
        public:
            void initialize();
            void step();
            const ExtY &getExternalOutputs() const;
            void terminate();
        protected:
            // protected data
        private:
            // private member methods
            // private data
    };
}
```

```cpp
namespace ros {
    class controller {
        public:
            void init(...);
            void update(...);
            void starting(...);
            void stop();
            ~controller();
        private:
            SL::modelClass slModel;
    };
}
```

Simulink Model Class  
ros_control Class
Demo: Differential Drive Robot Control in Gazebo
Future: Extend to Manipulator Controllers
Future: Generate ros_control Package Automatically

Robot Hardware Interface Blocks
Key Takeaways

- Automated code generation from Simulink to create `ros_control` controllers
  - Simulink controllers can easily be incorporated into `ros_control` framework for real-time controller implementation

- Same automated code generation infrastructure can be used for `ros2_control`

- Call-To-Action:
  - Reach out to us to work on real-world industrial applications
% Thank you!

Gijs van der Hoorn
Robot Dynamics, Cognitive Robotics
Delft University of Technology
the Netherlands

GitHub : gavanderhoorn
Email : g.a.vanderhoorn@tudelft.nl

Murat Belge
Development Lead of ROS Toolbox
MathWorks Inc.
Natick MA, USA

Email : mbelge@mathworks.com