

Automated Code Generation of Simulink Models as `ros_control` Controllers

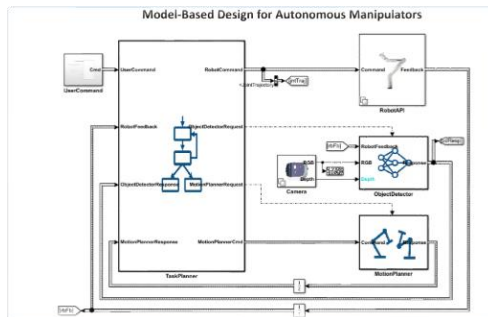
G.A. vd. Hoorn

Cognitive Robotics
Delft University of Technology
g.a.vanderhoorn@tudelft.nl

Murat Belge

Development Lead
MathWorks, Natick USA
mbelge@mathworks.com

Introduction



➔
Code
Generation
➔

```

#include "AutomatedParkingValetAlgorithm.h"
#include "AutomatedParkingValetAlgorithm_private.h"

int32_T div_s32_floor(int32_T numerator, int32_T denominator)
{
    int32_T quotient;
    uint32_T absNumerator;
    uint32_T absDenominator;
    uint32_T tempAbsQuotient;
    boolean_T quotientNeedsNegation;
    if (denominator == 0) {
        quotient = numerator >= 0 ? MAX_int32_T : MIN_int32_T;

        // Divide by zero handler
    } else {
        absNumerator = numerator < 0 ? ~static_cast<uint32_T>(numerator) + 1U :
            static_cast<uint32_T>(numerator);
        absDenominator = denominator < 0 ? ~static_cast<uint32_T>(denominator) + 1U :
            static_cast<uint32_T>(denominator);
        quotientNeedsNegation = ((numerator < 0) != (denominator < 0));
        tempAbsQuotient = absNumerator / absDenominator;
        if (quotientNeedsNegation) {
            absNumerator %= absDenominator;
            if (absNumerator > 0U) {
                tempAbsQuotient++;
            }
        }

        quotient = quotientNeedsNegation ? -static_cast<int32_T>(tempAbsQuotient) :
            static_cast<int32_T>(tempAbsQuotient);
    }

    return quotient;
}

void AutomatedParkingValetModelClass::APV_emxInit_real_T(emxArray_real_T_T
    **pEmxArray, int32_T numDimensions)
    
```

Motivation



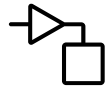
Idea: Design a controller in Simulink and use automated code generation to create a `ros_control` controller

- 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



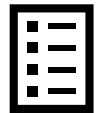
ros_control overview



Simulink C/C++ code generation



ros_control as a codegen target for Simulink models



Summary

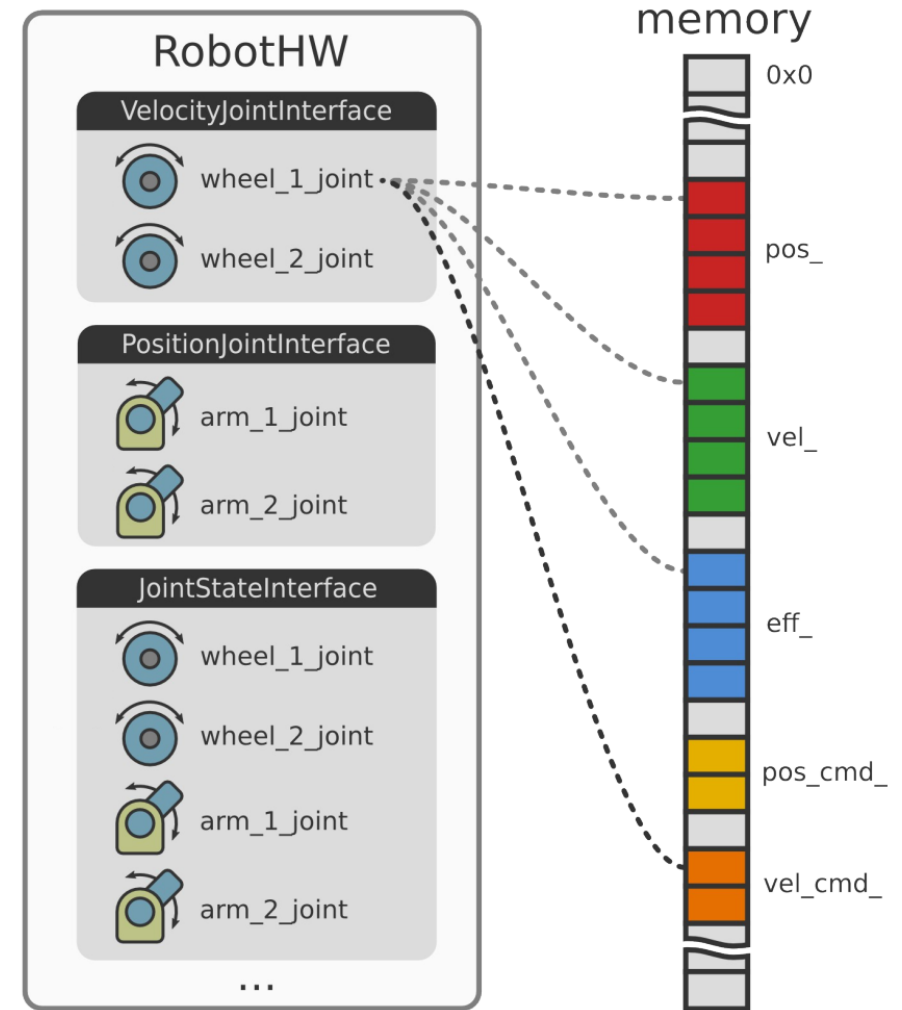
What is ros_control

- 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

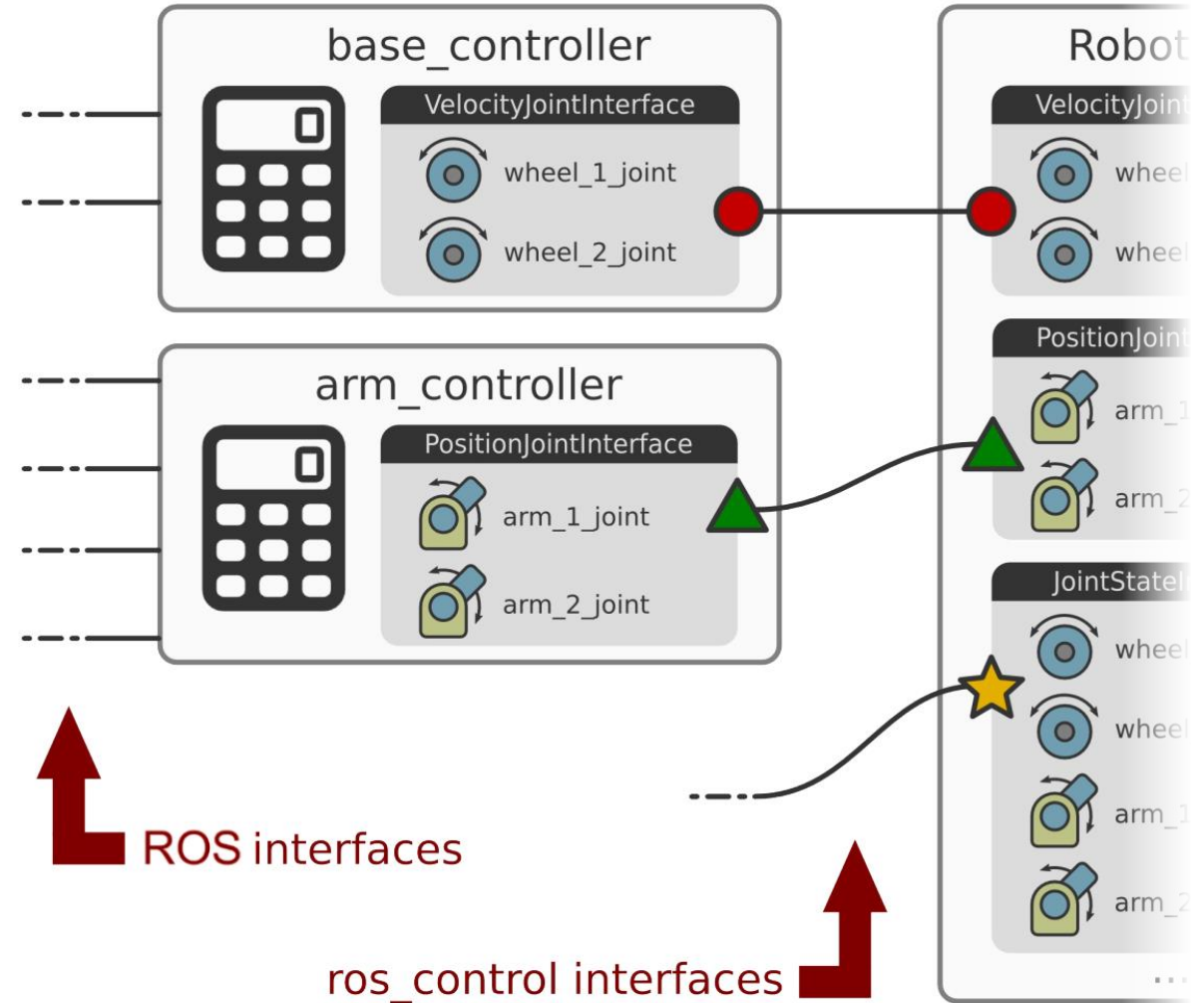
- No direct access to hardware
- RobotHW class wraps:
 - Registers / memory
 - OEM API / SDK
 - Fieldbus
 - TCP or UDP
 - ROS topics / services / actions
- Exposes abstract resources (*handles*):
 - OotB: position, velocity, effort, IMU, F/T
 - Plugin based (custom resources)
- Conflict detection (sometimes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



Original diagram: Adolfo Rodríguez Tsouroukdissian

ros_control: controllers

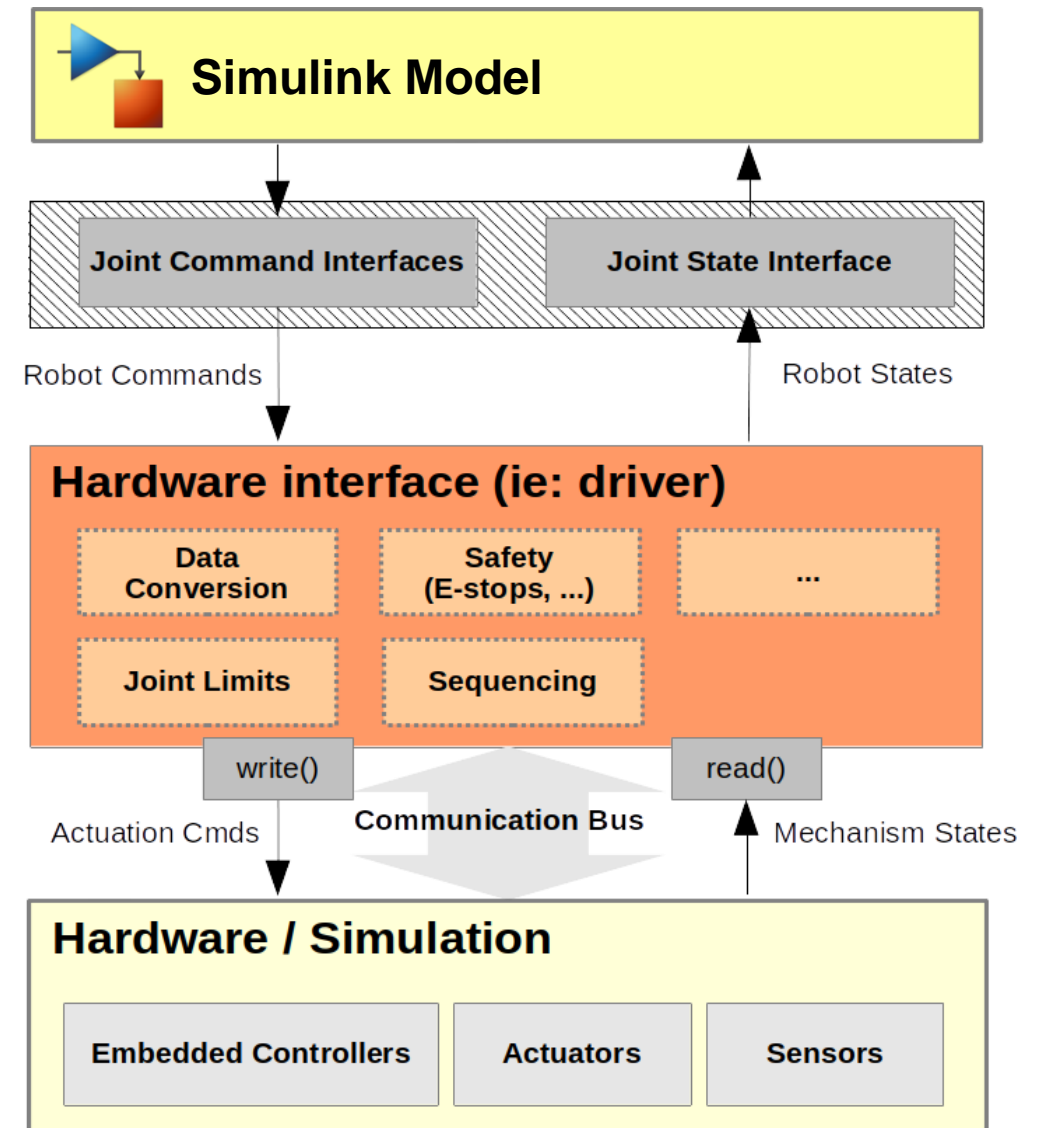
- Claim *resources*, not hardware
- Expose ROS interfaces
- Runtime load, init, (re)start, stop
- Runtime reconfigurable & switchable
- OotB: position, velocity, effort, trajectory execution
- Plugin based: custom controllers:
 - Whole-body (NASA Valkyrie, PAL)
 - Affordance
 - Cartesian F/T
 - Visual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



Original diagram: Adolfo Rodríguez Tsouroukdissian

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

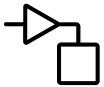


Original diagram: Dave Coleman

Agenda



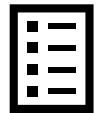
ros_control Recap



Simulink C/C++ Code Generation



ros_control as a codegen target for Simulink models

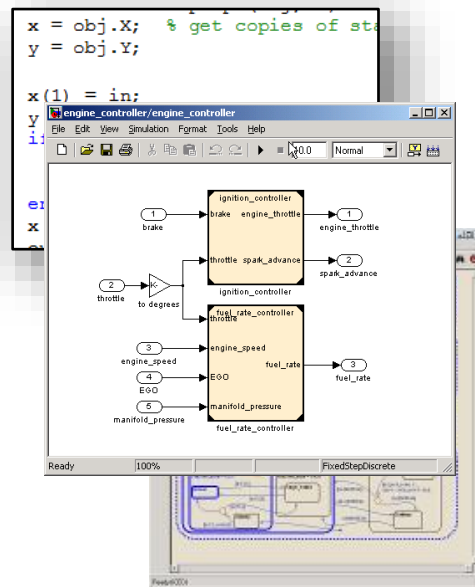


Summary

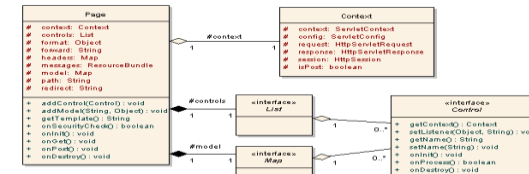
Simulink Code Generation

Generate *efficient C++* code that *plugs easily* into your C++ architectures

Simulink components



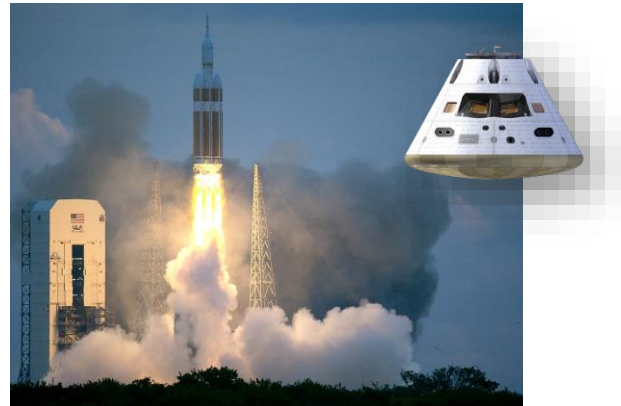
C++ Software Architecture



Success Stories

NASA Orion Program

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



Voyage

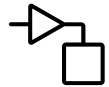


“Simulink + ROS allowed us to deploy a Level 3 Autonomous vehicle in less than 3 months”

Agenda



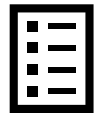
ros_control Recap



Simulink C/C++ Code Generation

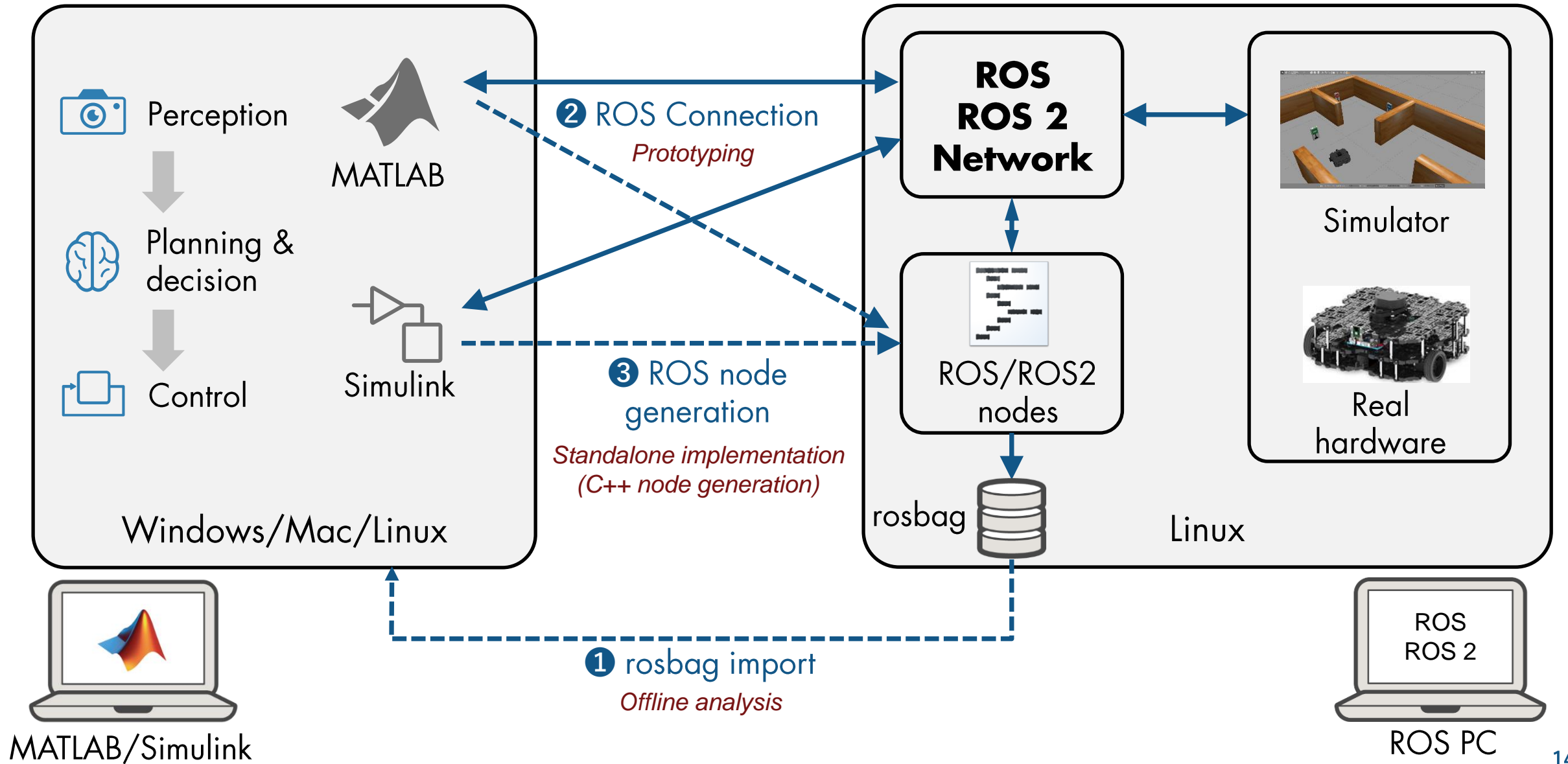


ros_control as a codegen target for Simulink models

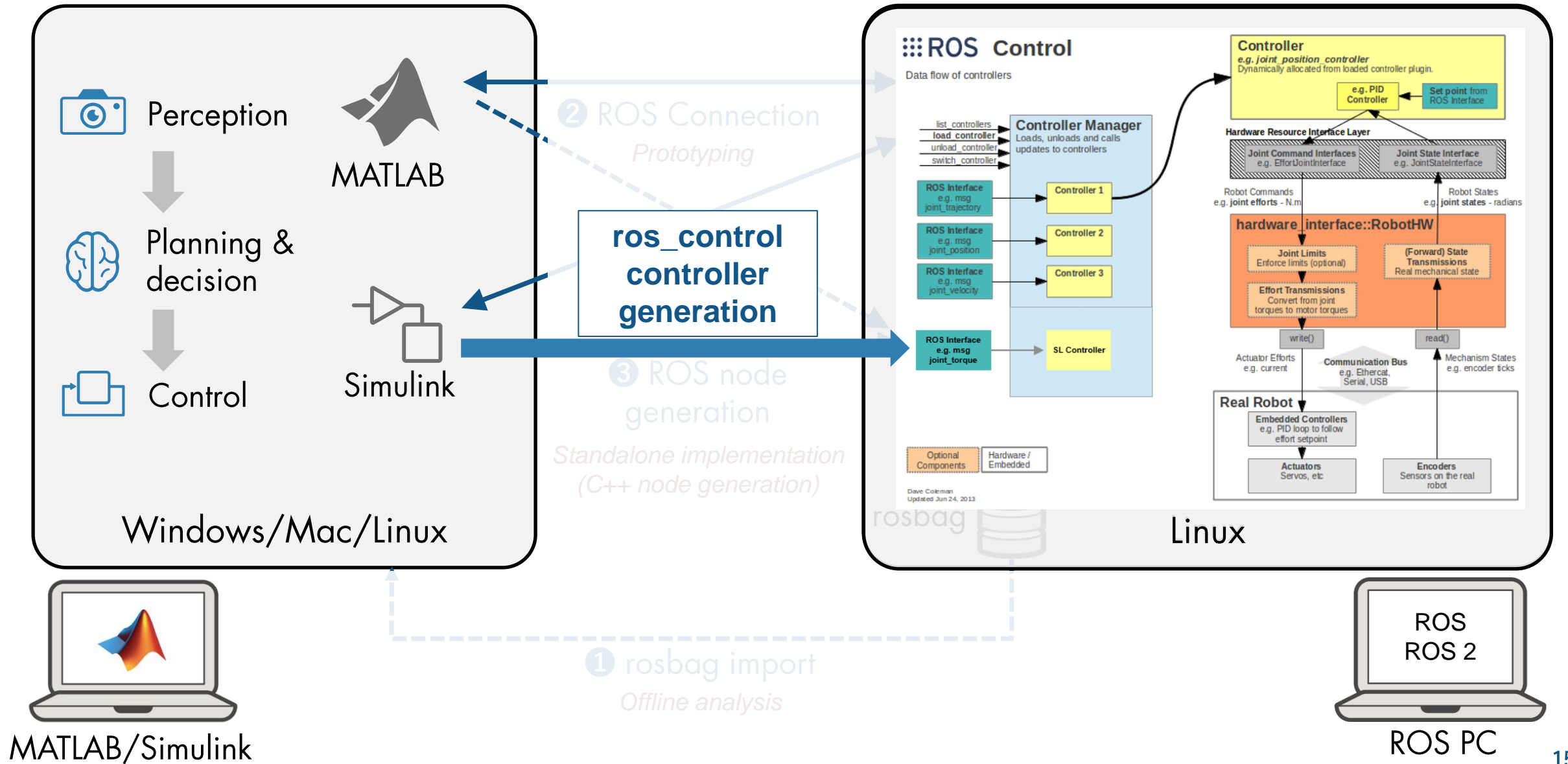


Summary

Connect MATLAB and Simulink with ROS



Connect MATLAB and Simulink with ROS

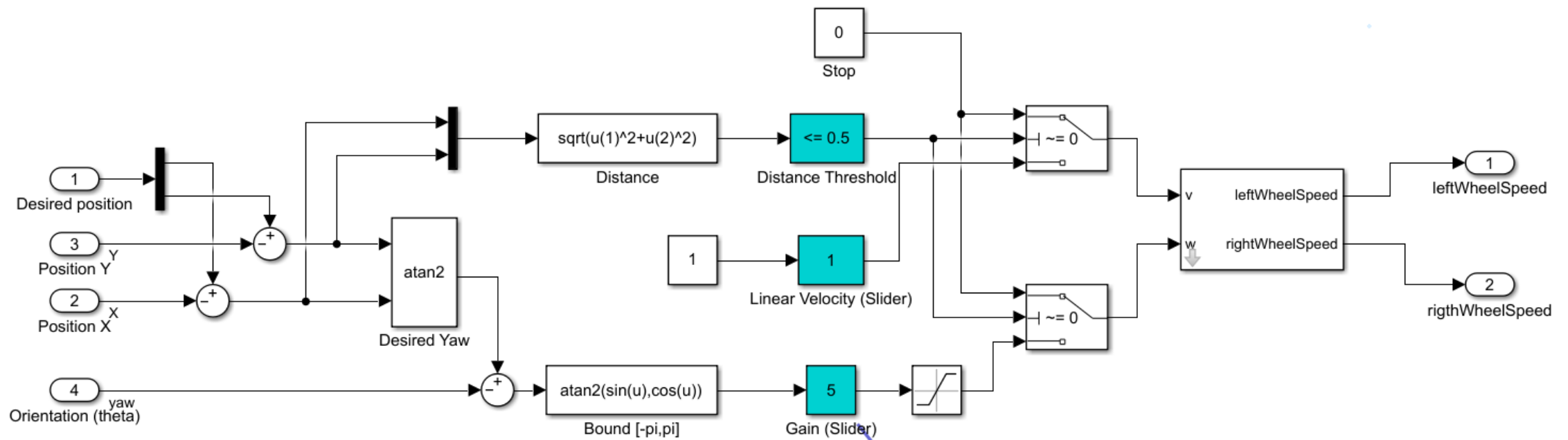


MATLAB/Simulink

ROS PC

Proof-of-Concept: Differential Drive Proportional Controller

Proportional Controller



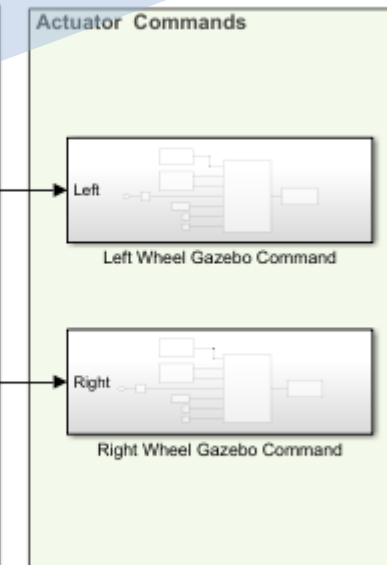
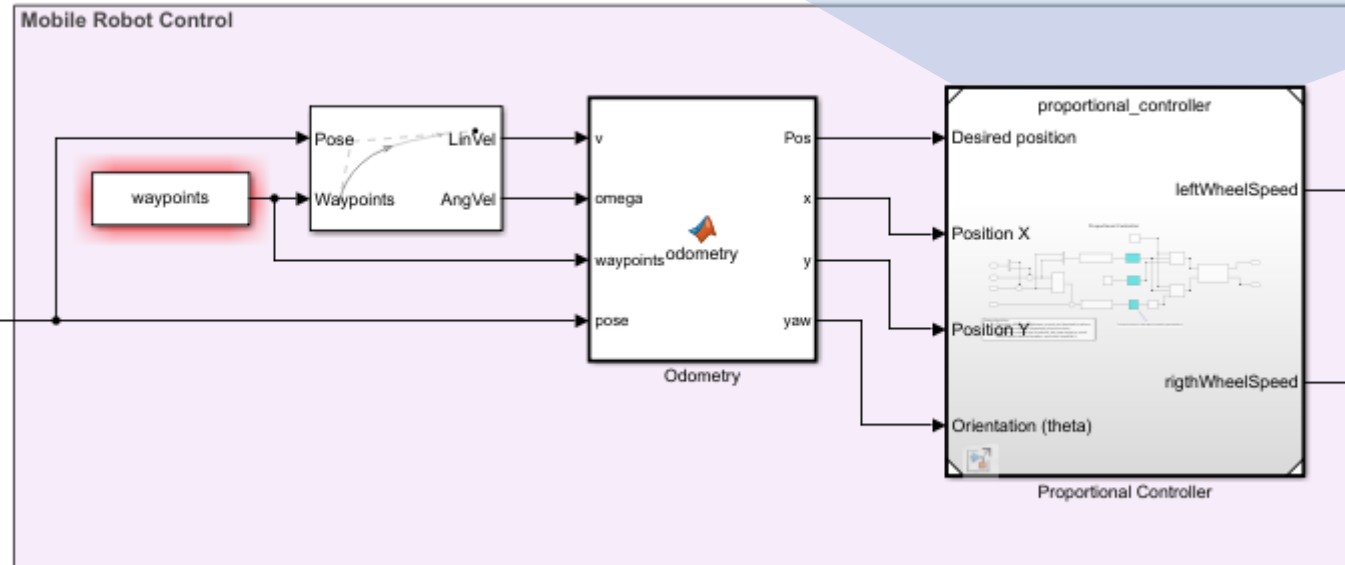
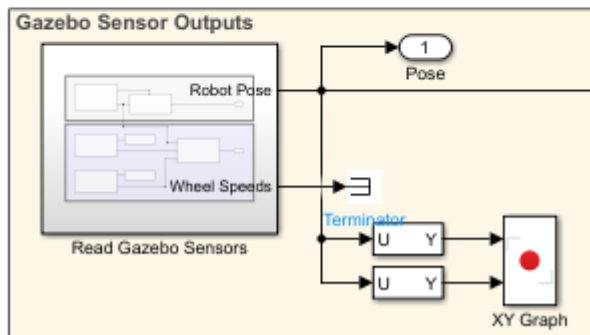
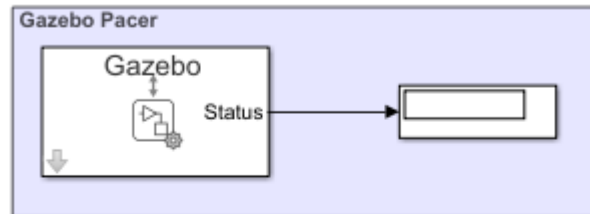
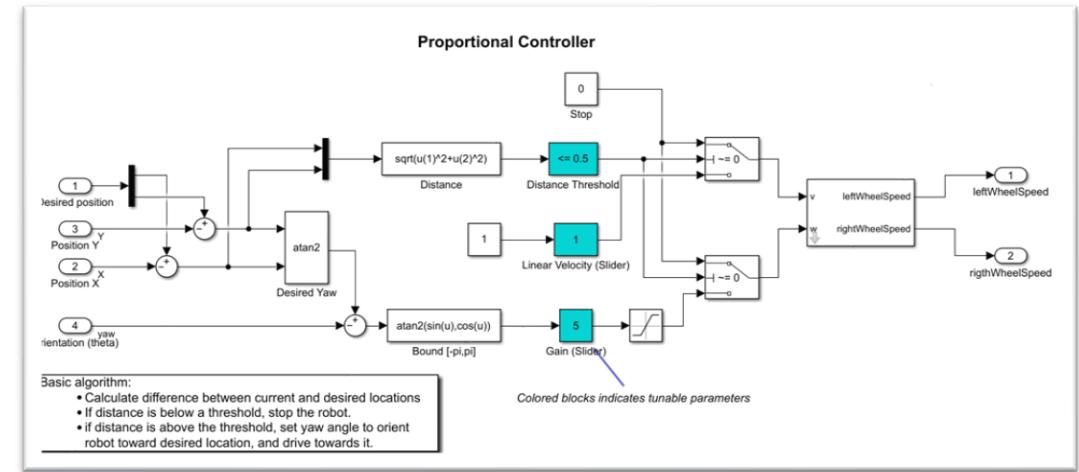
Basic algorithm:

- 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 indicates tunable parameters

Proof-of-Concept: Differential Drive Proportional Controller

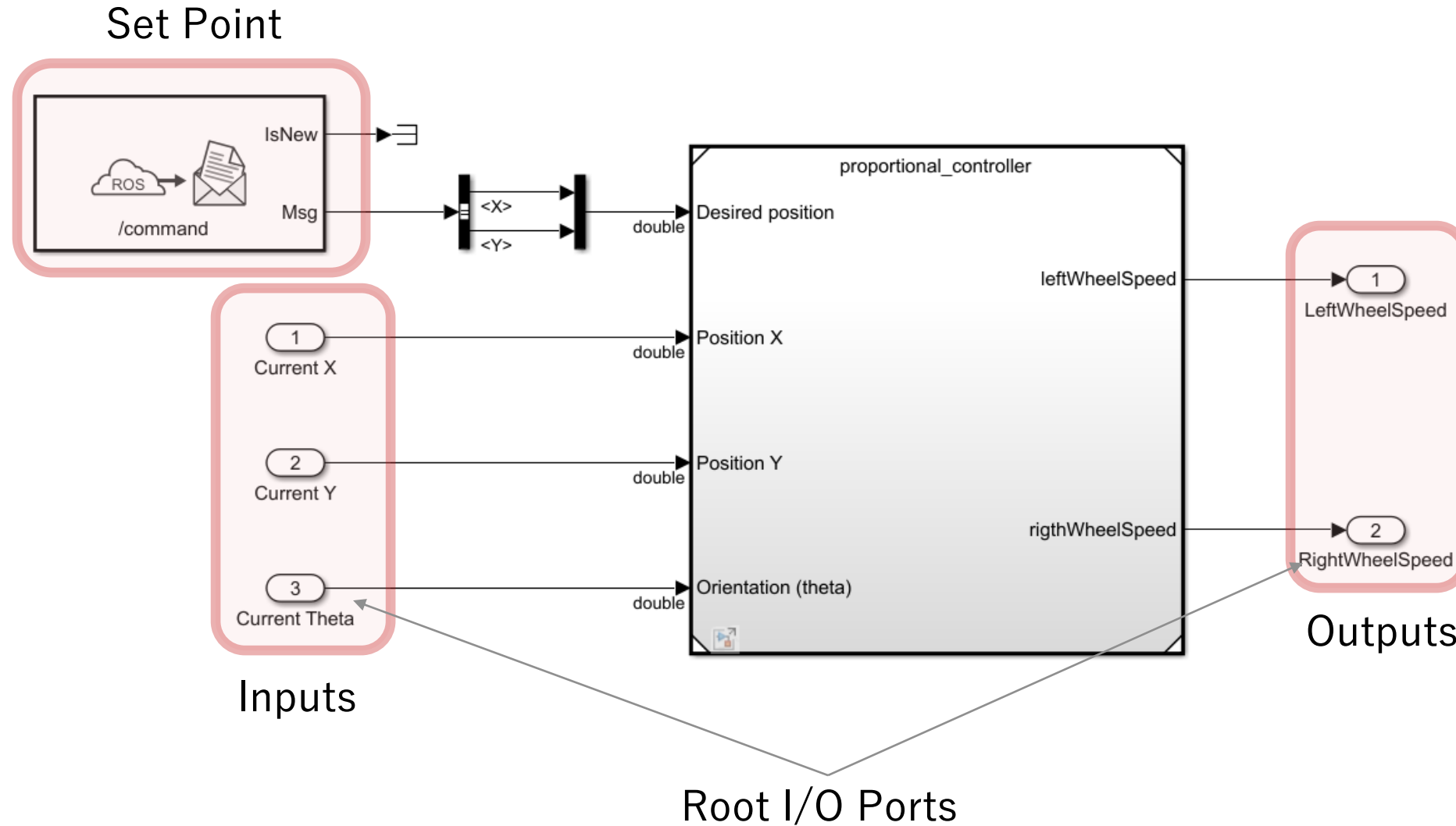
- Test & validate controller in Simulink environment



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

1 Create Interface Model



2 Adjust Model Settings

a :Set target

Hardware board: Robot Operating System (ROS)

Code Generation system target file: [ert.tlc](#)

Device vendor: Intel

Device details

Hardware board settings

- ▶ Operating system/schedule
- ▶ Target hardware resources

b :Set build process

Target selection

Description: Embedded Coder

Build process

- Generate code only
- Package code and artifacts

Toolchain settings

Toolchain: Catkin

Build configuration: Faster Runs

▶ Toolchain details

Code generation objectives

Prioritized objectives: Unspecified

Check model before generating code: Off

c :Configure codegen interface

Code interface packaging: C++ class

Multi-instance code error d

Remove error status field in real-time model data structure

Include model types in model class

Data exchange interface

Generate C API for:

- signals
- parameters
- states
- root-level I/O

3 Generate Code

The screenshot shows the Simulink software interface for a ROS-based controller. The top toolbar includes tabs for SIMULATION, DEBUG, MODELING, FORMAT, ROS, and APPS. The ROS tab is active, showing options for ROS Network (Robot Operating System (ROS)), Deploy to (Remote Device (192.168.128...)), and various actions like Hardware Settings, Test Point, and Control Panel. A 'Build Model' button is highlighted with a red box in the top right corner, with a tooltip that reads 'Generate code for model'. The main workspace contains a Simulink diagram for 'controller_interface'. It features a 'Msg' block connected to a 'proportional_controller' block. The 'Msg' block has an input from a cloud icon labeled '/command' and an output labeled 'Msg'. The 'proportional_controller' block has three inputs: 'Desired position' (double), 'Position X' (double), and 'Position Y' (double). It also has an input for 'Orientation (theta)' (double). The controller block has two outputs: 'leftWheelSpeed' and 'righthWheelSpeed'. The 'leftWheelSpeed' output is connected to a scope block labeled '1 LeftWheelSpeed', and the 'righthWheelSpeed' output is connected to a scope block labeled '2 RightWheelSpeed'. The status bar at the bottom indicates 'Ready', '123%', and 'FixedStepAuto'.

4 Integrate Generated Code into `ros_control` Framework

```

namespace SL {

class modelClass {

public:
    void initialize();
    void step();
    const ExtY &getExternalOutputs() const;
    void terminate();

protected:
    // protected data

private:
    // private member methods
    // private data
};
}
    
```

Simulink Model Class

```

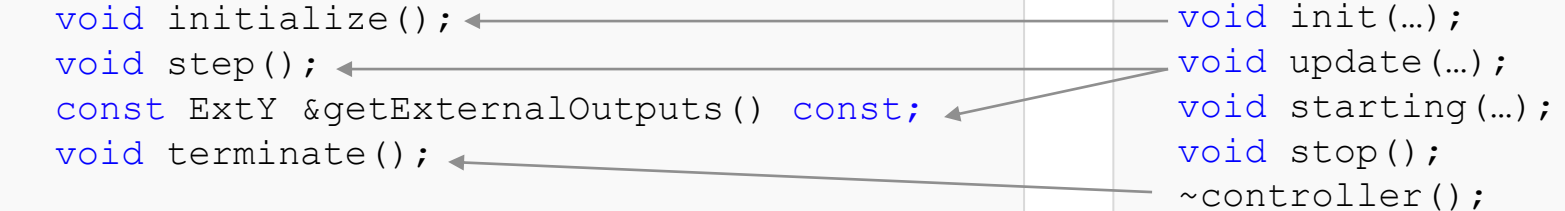
namespace ros {

class controller{

public:
    void init(...);
    void update(...);
    void starting(...);
    void stop();
    ~controller();

private:
    SL::modelClass slModel;
};
}
    
```

`ros_control` Class



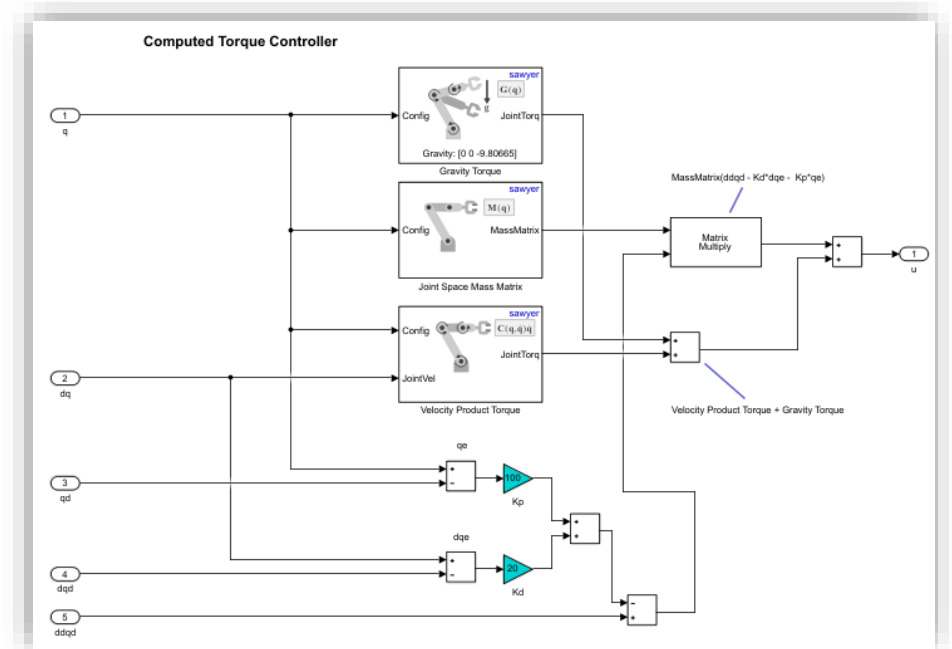
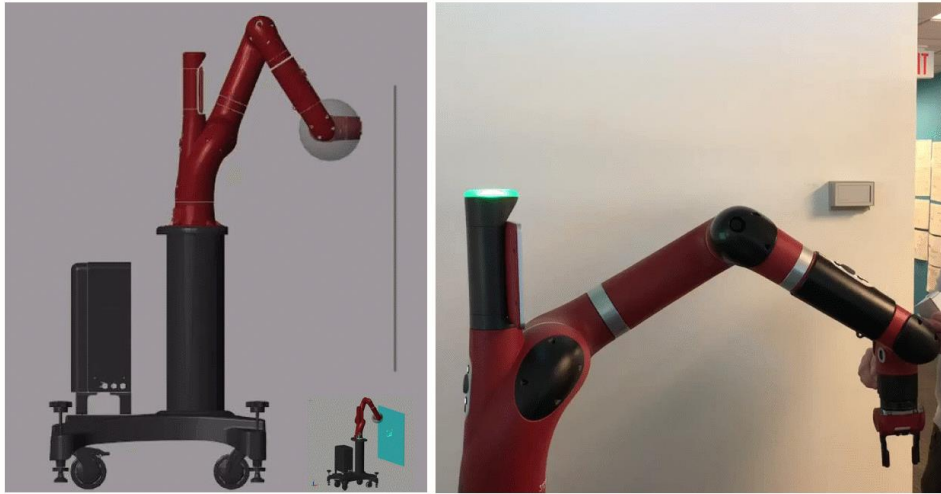
Demo: Differential Drive Robot Control in Gazebo

The screenshot displays the Gazebo simulation environment. A small red and black differential drive robot is positioned on a grey grid floor. Three colored lines (blue, green, and red) extend from the robot, representing sensor beams. The interface includes a menu bar (File, Edit, Camera, View, Window, Help), a toolbar with various simulation controls, and a left sidebar with 'World', 'Insert', and 'Layers' tabs. Below the sidebar is a 'Property Value' table. At the bottom of the Gazebo window, a status bar shows simulation metrics: Steps: 1, Real Time Factor: 0.99, Sim Time: 00 00:02:33.405, Real Time: 00 00:02:35.186, Iterations: 153405, FPS: 23.46, and a 'Reset Time' button.

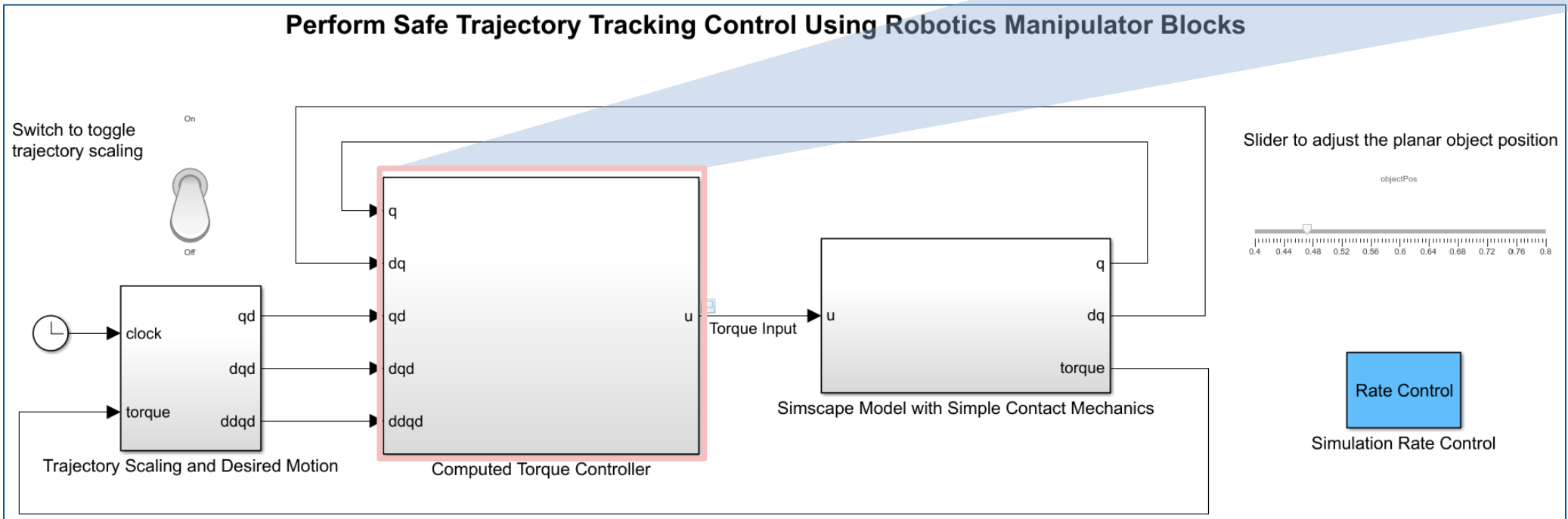
In the bottom right corner, a terminal window titled 'rqt_controller_manager__ControllerManager - rqt' is open, showing the ROS controller manager interface. The namespace is set to '/p3dx/controller_manager'. The following table shows the status of the controllers:

controller	state
joint_state_controller	running
diff_drive_controller	uninitialized
my_test_controller	uninitialized

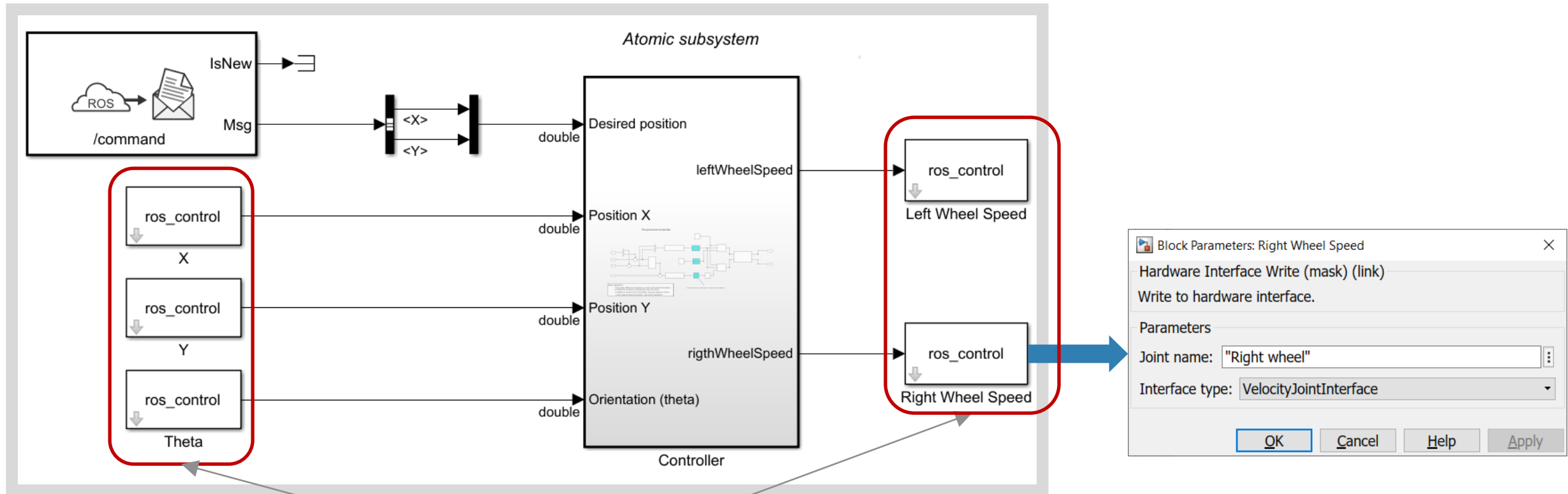
Future: Extend to Manipulator Controllers



Perform Safe Trajectory Tracking Control Using Robotics Manipulator Blocks



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