参加 MathWorks 对车队培训的收获与分享

作者：辽宁工业大学万得电车队

首先，感谢 MathWorks 对大学生方程式大赛的关注，感谢董淑成老师、郭文彦老师对于参与大学生方程式大赛的学生发展的付出，也感谢两位老师在今年 4 月底来我校辽宁工业大学就北方地区参赛队进行了专业的培训。

在对今年控制策略开发过程讲述之前，我想对去年我们存在的问题做一个简单的总结。在 2017 赛季，我们搭建了整车的控制模型，但是没有完整的一个控制策略开发和代码生成的概念，仅做了模型的搭建并未对模型进行各种测试，模型整体比较庞大、结构不清晰，这导致了后期实车调试过程中程序出现了死循环、状态混乱、除零等现象。提交的作品包含了车队使用的快速原型控制器自带的库模块，导致模型在其他电脑上运行不了，这应该也是大部分车队存在的问题。

为此，我们在 2018 赛季做了调整和规划。今年我们有幸参与了这次培训，并从这次培训中学到了很多东西，这也为后面我们做控制策略开发和代码生成奠定了良好的基础。

针对于控制策略开发与代码生成，董老师提到了‘V’模型的开发流程，并强调了在代码生成前，模型需要进行充分的验证保证其正确性，为此，我们车队今年做了以下的一些工作：1、控制算法设计；2、编写需求文档；3、建立团队开发的 Project；4、单元模型复杂度分析；5、模型与需求文档的链接；6、建模标准检查；7、设计错误检查；8、编写测试用例并进行功能性测试；9、建立数据字典；10、代码生成；11、代码验证。

下面我将通过以下几个方面对这次培训的收获进行总结：

1. 控制算法的设计

在控制算法设计中，我们考虑了 17 赛季赛车出现打滑、过多或不足转向的现象，加入了 TCS 牵引力控制和 DYC 横摆力矩控制，为了让模型保证实时性，对算法的理论模型进行了简化，采用最基本的车辆二自由度模型。

2. 需求文档的编写

对于需求文档，包含了整体需求、结构需求（输入、输出）、功能需求（算法求解）三大部分。在培训过程中，董老师提到信号的数据类型与模型的运行和代码生成息息相关，因此我们细化了每个输入输出信号的数据范围、单位以及数据类型。同时定义了团队开发的建模要求，符合 MAAB 的建模标准、单元模型的圈复杂度。

2. 结构需求

2.1 输入

2.1.1 目标驱动力矩输入

控制器应该具有目标转矩输入接口（通过采集油门踏板数据进行匹配计算得到，根据去年调试经验，拟定目标驱动力矩范围 0~80，单位（Nm），数据类型 Uint16）。

2.1.2 左前轮轮速输入

控制器应该具有左侧前轮轮速信号输入接口(通过采集左侧车轮轮速传感器信号并计算处理得到,根据去年调试经验,拟定左前轮轮速范围 0~60,单位(r/s),数据类型 Uint8)。

4. 基本参数表

项目	尺寸
车轮直径（d）	0.457m
轴距（L）	1.55m
后轮距(B)	1.20m
传动比(r)	4

5. 建模规范要求

5.1 单元模型圈复杂度小于 20

5.2 整体模型符合 MAAB 建模规范要求

图 2.1 需求文档

3. 单元模型复杂度分析

在 MathWorks 公众号上推送过一篇董老师写的关于圈复杂度的文章，其中提到复杂度过高会给单元测试带来很大的麻烦，可设计性也会大打折扣，同时模型的可维护性降低，容易出现“牵一发而动全身”的局面。因此，我们在建模之前考虑了算法的复杂程度，在需求文档中定义了团队建模标准：单元模型圈复杂度低于 20。按着这个标准建模之后使用 Coverage 覆盖度测试工具测试后，测试报告显示我们的三个单元模型的复杂度分别为 5、7、10，都满足需求文档要求。

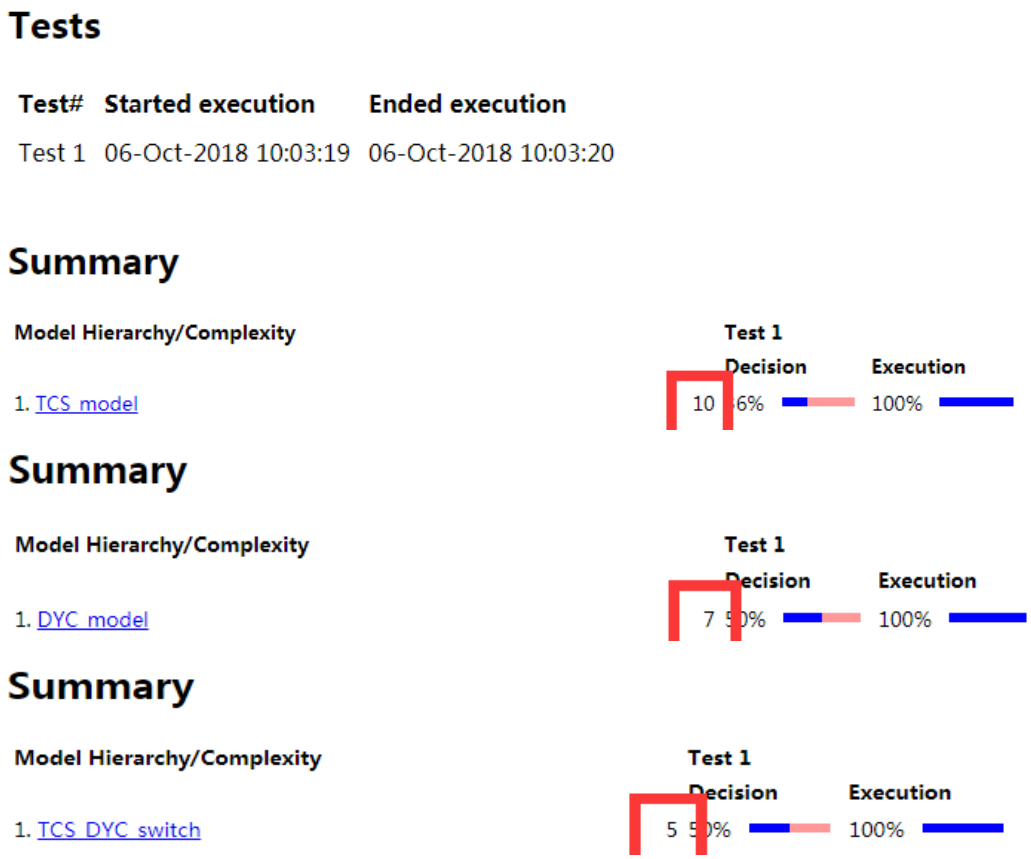


图 3.1 测试结果

4. 建模标准检查

对于模型标准检查，我们参考的是 MAAB 建模标准，使用的是 Model Advisor 工具。在修改之前模型存在的主要是命名的规范性、数据范围等一些问题，根据警告提示和修改参考对模型进行相应的修改后通过了警告。对于一些不影响的警告我们就不再做相应的修改。

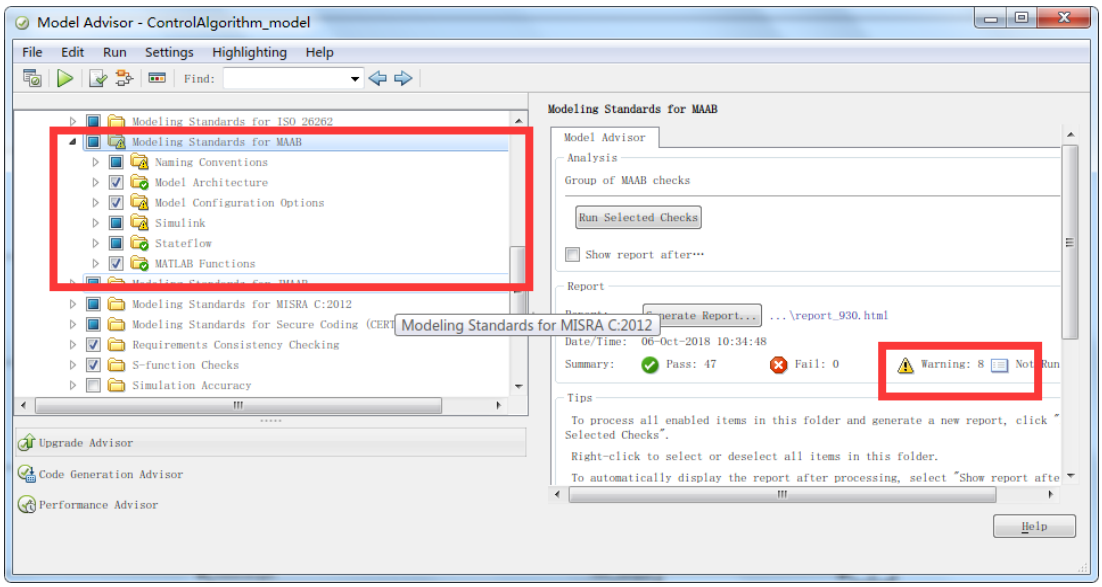


图 4.1 检查结果

5. 设计错误检查

考虑到上个赛季存在的问题，我们对模型进行了设计错误检查，使用的是 Design Verifier 中的 Detect Design Error 工具，配置了相应的参数，主要是对死循环、数据溢出、除零这些问题的检查。

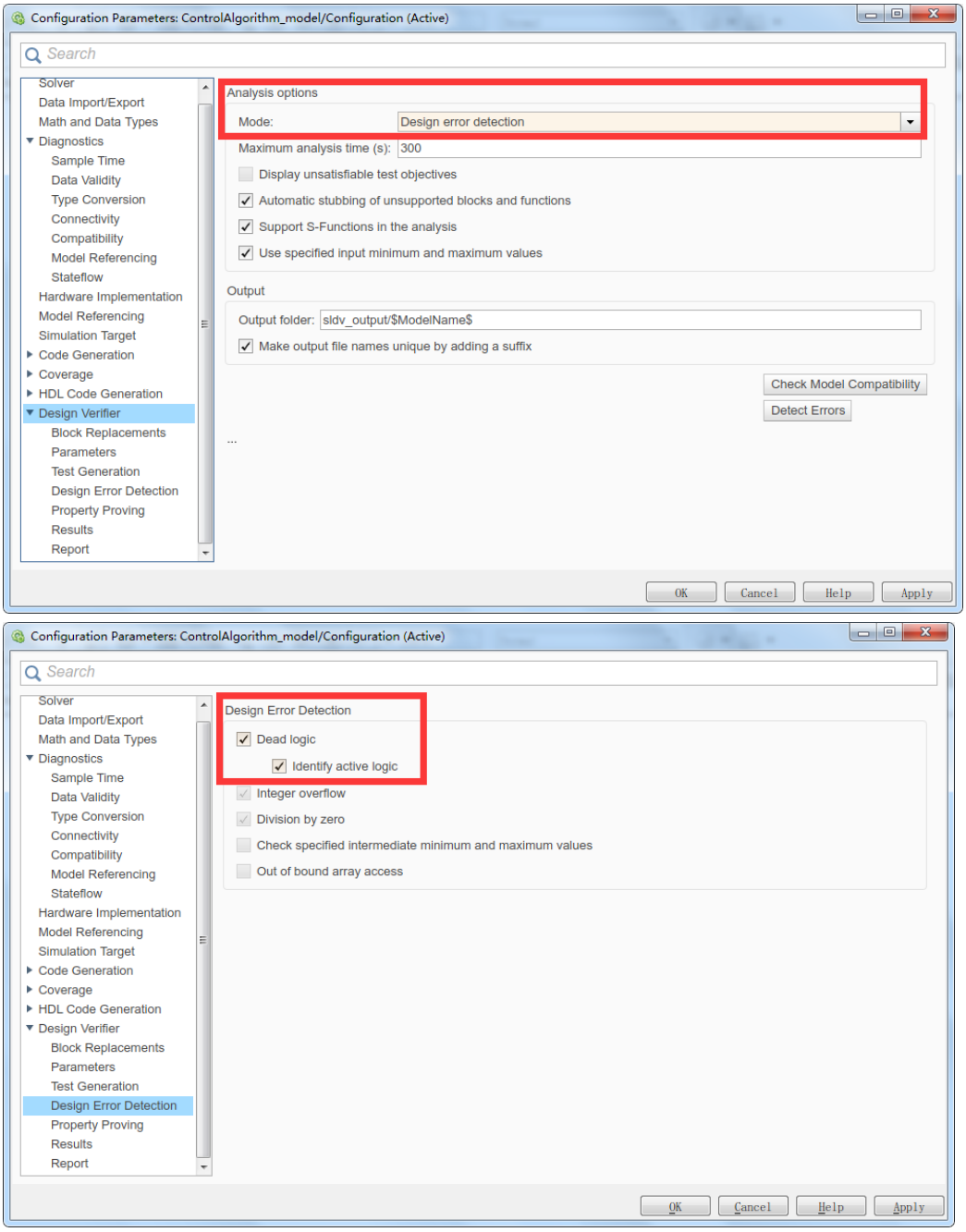


图 5.1 设计错误检查配置

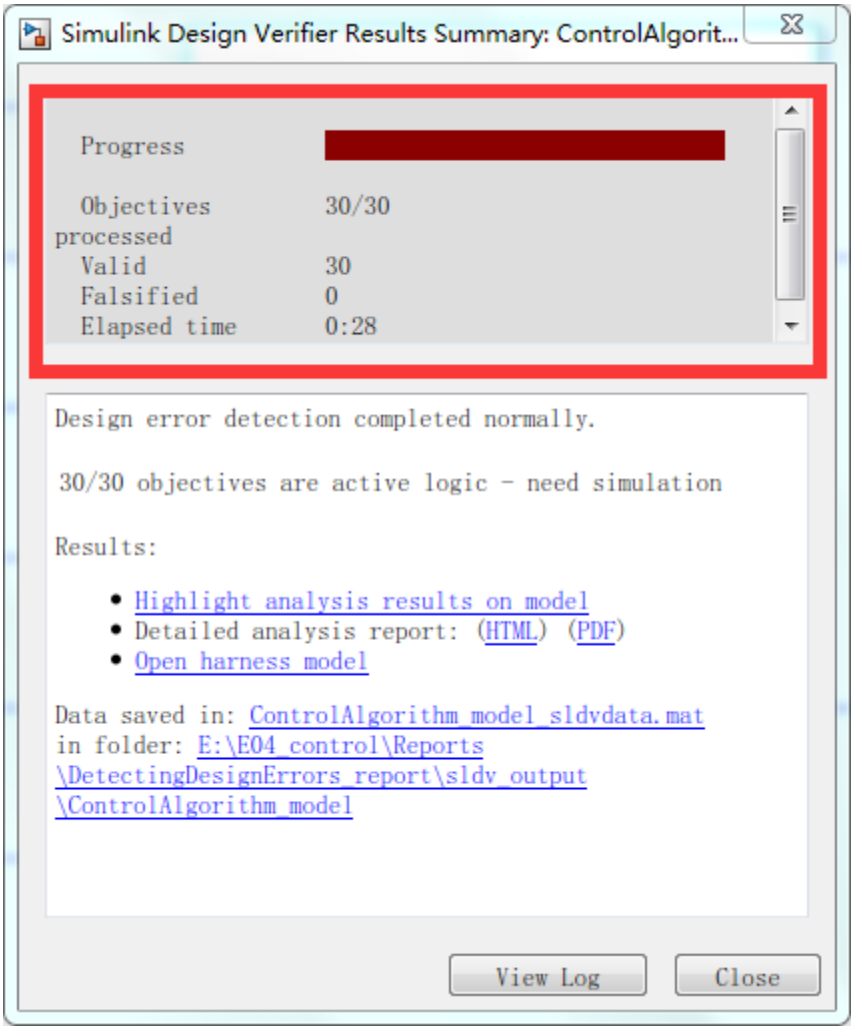


图 5.1 设计错误检查结果

检测结果显示全部通过，可以看出按着标准进行建模可以减少后期出现的一系列问题，这节省了我们后期花在解决问题上的时间。

6. 编写测试用例和功能性测试

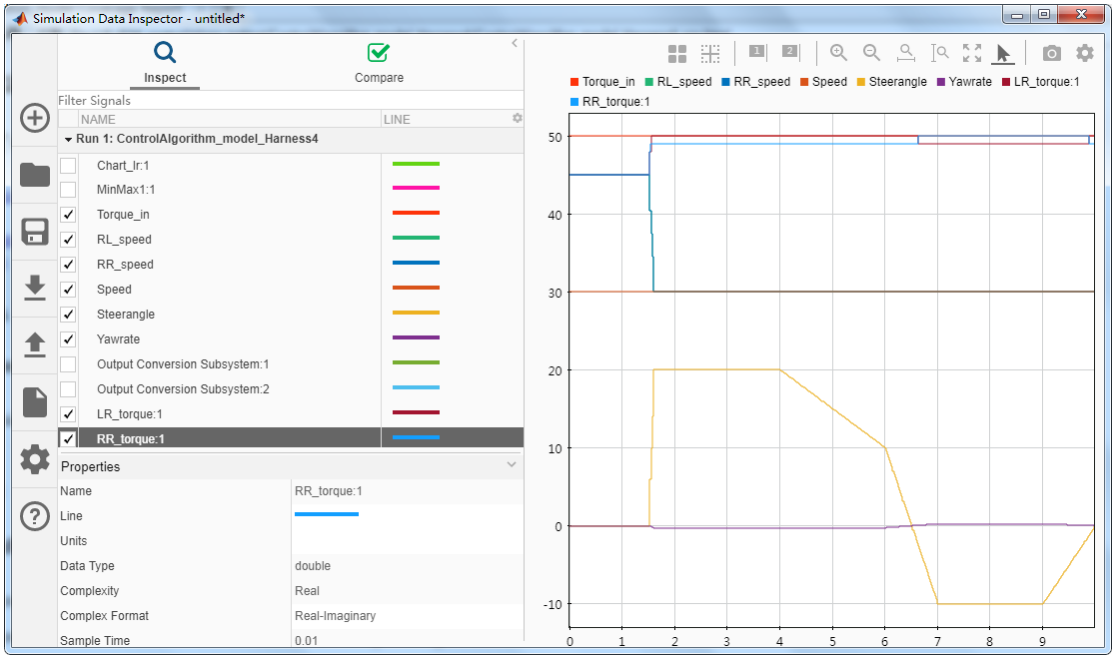
在前面几项工作完成后，需要对模型进行功能上的测试，因此我们根据大赛的动态赛项目编写了简单的测试用例，根据需求文档对于一些输入输出信号进行定义，模拟赛车在动态赛车中各种状态。

我们的测试用例使用 Test Harness 工具进行手动添加，生成相应的测试用例模型，将设计的测试用例导入到信号发生器中，并在信号线上设置相应的信号记录，使用 Simulation Data Inspector 工具进行数据查看和分析。图中的是高速避障的测试结果，在出现车轮打滑时使用 TCS 模块对输出转矩进行调整，在无车轮打滑的情况下使用 DYC 对输出转矩进行调整。最终生成测试报告，覆盖度能够达到 80%以上。

	A	B	C	D	E	F	G	H
1	Time	Torque_in	RL_speed	RR_speed	Speed	Steerangle	Yawrate	
2	0	50	30	30	30	0	0	
3	0.5	50	30	30	30	0	0	
4	0.5	50	30	30	30	0	0	
5	1	50	30	30	30	0	0	
6	1	50	30	30	30	0	0	
7	1.1	50	30	30	30	0	0	
8	1.1	50	30	30	30	0	0	
9	1.5	50	30	30	30	0	0	
10	1.5	50	30	30	30	0	0	
11	1.6	50	30	30	30	0	0	
12	1.6	50	30	30	30	0	0	
13	2	50	30	30	30	0	0	
14	2	50	30	30	30	0	0	
15	4	50	30	30	30	0	0	
16	4	50	30	30	30	0	0	
17	6	50	30	30	30	0	0	
18	6	50	30	30	30	0	0	
19	7	50	30	30	30	0	0	
20	7	50	30	30	30	0	0	
21	9	50	30	30	30	0	0	
22	9	50	30	30	30	0	0	
23	10	50	30	30	30	0	0	
24								

Acceleration | Acceleration_s | Skidpad | Endurance

图 6.1 测试用例



Summary

Model Hierarchy/Complexity

1. ControlAlgorithm_model

Test 1

	Decision	Execution
20	87%	100%

图 6.2 测试结果

7. 建立数据字典

在培训中，董老师对数据字典和面向对象中的类进行了很多强调，这是我们在以往建模中忽略的问题，而这对于模型数据的管理以及代码的生成有着很大的影响。就我们车队去年而言，前期没有对信号和参数做定义，导致后期模型运行时报错，需要对数据类型等进行多次的改动，浪费了太多时间。在今年我们继承了继承了 Simulink 的包+Simulink 里面的类，并进行相应的修改配置符合我们团队开发的需求，这也为后期的维护提供了很大的便捷。

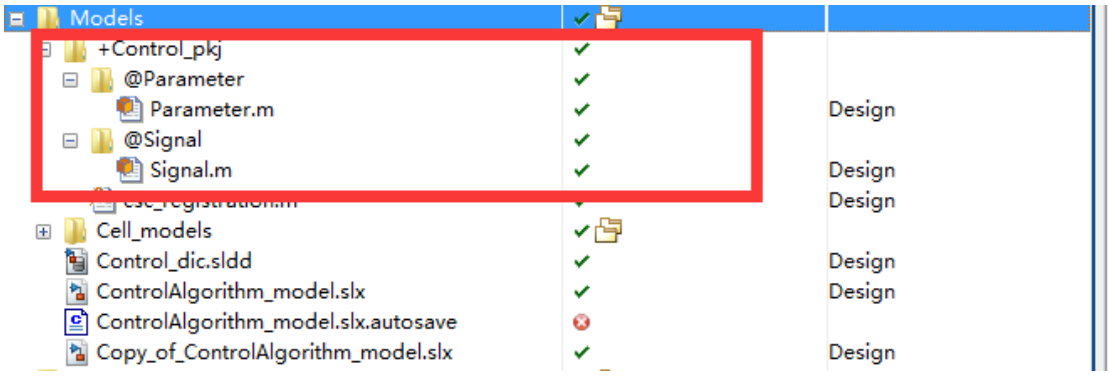


图 7.1 继承的类

8. 代码生成

在代码生成中，我们团队设置了需求到源代码的可追溯性，这样生成的代码就可以与需求文档能够很直观的一一对应上，提高了对代码的可读性。

```
/* End of Saturate: '<S13>/Saturation2' */

/* Product: '<S13>/Divide3' incorporates:
 *  Constant: '<S13>/Constant2'
 *  Inport: '<Root>/RR_speed'
 *  Inport: '<Root>/Speed'
 *  Sum: '<S13>/Add5'
 *  Sum: '<S13>/Add9'
 *
 * Block requirements for '<Root>/RR_speed':
 *  1. 右后轮轮速输入
 *
 * Block requirements for '<Root>/Speed':
 *  1. 车速输入
 */
rtb_Filter_j = ((real_T)RR_speed - ((real_T)Speed + 1.0)) / ((real_T)Speed +
  1.0);
```

图 8.1 生成的代码可追溯性体现

9. 代码的等效性测试

最后一步代码的等效性测试，在此之前观看过 MathWorks 网上研讨会视屏，视屏中董老师讲解了两个方法，一个是对比模型在 Normal 模式下与 SIL 模式下运行得到的输出是否一致；另外一个方法是将模型编译成 S-Function 模块，给模型与生成的 S-Function 模块输入相同的数据，然后对比输出是否一致。我们团队采用了第一种方法，其中做了一些小的变化：将原始模型拷贝一份，使用两个 Model Reference 来分别引原模型和拷贝的模型，把原模型设置在 Normal 模式下运行，拷贝模型设置在 SIL 模式下运行，然后输入相同的信号来对输出对应的做差，结果一直保持零，说明生成的代码与模型等效。

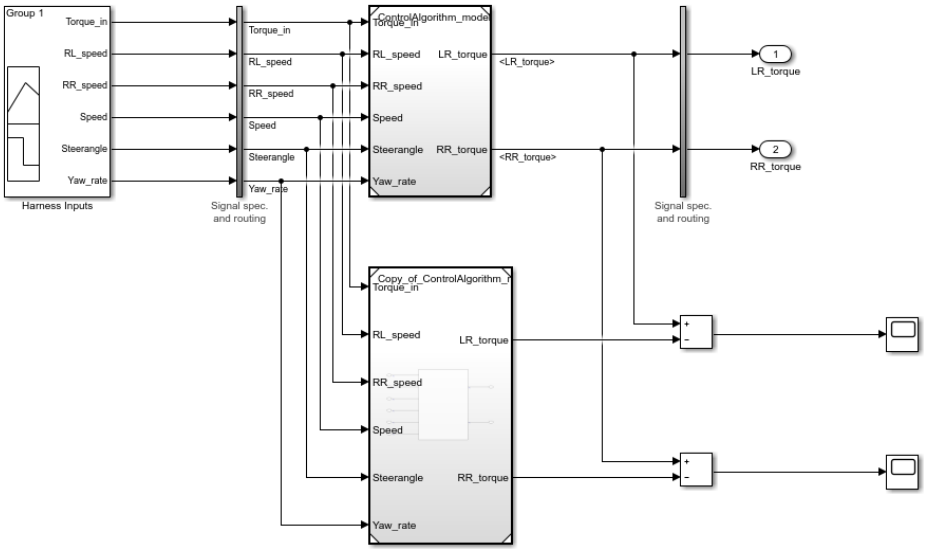


图 9.1 测试模型

通过一系列的验证和修改后，我们搭建的模型成功的在今年的赛车上稳定的运行。在实车调试过程中，相比于去年少了很多问题，即使出现了问题也能够及时的发现问题所在并进行合理修改，这节省了我们大部分的时间。从今年赛车在动态赛中的表现可以看出按照建模标准建立并通过多项测试后的模型在实时性和稳定性上有了很大的提高。