

# POLITECNICO DI TORINO

Master Degree  
in MECHATRONIC ENGINEERING

Master Degree Thesis

Model-based design of a test bench for electric  
traction powertrain development



**Supervisor**  
Prof. Stefano Carabelli

**Candidate**  
Gaetano Colombo

A.A.2020/2021



*A Dio,  
fonte di gioia e felicità;*

*A mio padre,  
pilastro portante e sorgente di ispirazione;*

*A mia madre,  
seme d'amore e maestra di vita;*

*A mio fratello,  
fonte di spensieratezza e compagno di avventure;*

*Alla mia amata Sicilia,  
terra di colori, sapori e odori  
che non vorrebbe lasciar scappare i propri figli;*

*A me stesso e alla mia tenacia,  
che mi hanno permesso di arrivare fin qui.*





## ACKNOWLEDGEMENTS

First, I would like to express my great gratitude to Prof. Stefano Carabelli for his constant support and guidance, for his constructive suggestions and his valuable knowledge during the development of this research work.

Also, I would like to thank PANDA G1 Electrical team's components to let me develop the final part of this thesis work and Panda 4WD-H PoliTO team's components to let me create the team and became the team leader.

Furthermore, I would like to thank all the people known in student residence Villa Claretta, roommates of Filadelfia 201 street, Jazz and BusyBox team friends, tennis friends, all colleagues known during master's degree and all old friends for having supported all my changes and enthusiasm, for showing that even those who are far away can be close, and for making me laugh during all difficult moments.

Finally, I would like to thank my family for constant and affectionate support of every day and for allowing me to reach this goal.

Thanks to all of you, this thesis work was developed with joy as well as passion.



## ABSTRACT

In electrified vehicles case, both in Battery Electric Vehicles (BEVs) case and in Full- Hybrid or Plug-in Hybrid Electric Vehicles (FHEVs or PHEVs) case, electric motors testing is a crucial phase of the entire final vehicles build-up process. This because tests allow to verify whether designed system works properly and to check whether desired requirements are satisfied. For testing electric motors, appropriate test benches, which can be used in two cases, have to be considered: to verify designed motors' performance, or to diagnose experienced problems during the entire electric motors' life. In the first case, test benches are used to evaluate motors' electrical and mechanical characteristics that can help designers into the choice of proper motor during electric vehicles development process. The choice is done taking into account costs, dimensions and other motor physical characteristics too. Instead, in the second case test benches are used to evaluate motors' electrical and mechanical characteristics by means causes of possible problems during operation can be understood.

Usually, electric motor test bench costs are very high due to expensive components used to build-up what it is called Human – Machine – Interface (HMI), the most important part of the entire system. This because its purpose is setting input data on the basis of test to be performed, managing system's status by means buttons and switches and storing interesting parameter values. In general, HMI is made-up of control systems and motor drives allowing to control and to manage brake motor's behavior and computers allowing to store data coming out from sensors.

In this thesis activities, the goal is designing a test bench for testing electric motors where a control law is implemented in a Vehicle Management Unit (VMU). This control law is responsible to evaluate actions to be taken for piloting both bench and test motor on the basis of user input data and data coming from feedback path (values sensed by sensors). VMU is adopted to reduce costs to be addressed by companies for developing HMI's control system. Moreover, test bench is implemented in such a way that power recirculation effect is always verified. This because the goal is designed a test bench for testing also electric motors that require huge power (30 – 50 kW) by using small power supplies that are able to provide only the total dissipated power. To accomplish these goals, test bench's design is done adopting System Engineering methodology which integrates Model-based Design approach which provides V-shaped model and Modular-Technical-Model tools. The first tool allows to develop the final physical system performing sequentially different phases which involve different tests that are intended to verify whether initial requirements are satisfied and to understand whether user's idea is feasible. On the other hand, the second tool allows to subdivide system's development in modules in order to reduce development time and to improve reusability.



# TABLE OF CONTENTS

<b>LIST OF FIGURE .....</b>	<b>III</b>
<b>CHAPTER 1: INTRODUCTION .....</b>	<b>1</b>
<b>CHAPTER 2: SYSTEM ENGINEERING APPROACH .....</b>	<b>3</b>
2.1 V-shaped model.....	4
2.1.1 Hybrid V-shaped model.....	10
2.2 Modular-Technical Model (MTM) .....	13
<b>CHAPTER 3: TEST BENCH CONCEPT MODEL .....</b>	<b>17</b>
3.1 User requirements .....	25
3.2 Concept model simulation .....	27
3.3 Concept model simulation results .....	30
<b>CHAPTER 4: SYSTEM DESIGN PHASE: TECHNICAL MODEL.....</b>	<b>39</b>
4.1 Preliminary technical model .....	39
4.1.1 Some preliminary modification of concept model .....	39
4.1.2 Tabulated bench motor .....	53
4.1.3 Tabulated bench and test motor .....	77
4.2 Modular Technical model .....	97
4.2.1 Environment reference model.....	98
4.2.2 Plant reference model .....	105
4.2.3 Control interface refence model .....	114
4.2.3.1 <b>Control Logic reference model</b> . . . . .	116
4.2.4 Human-Machine-Interface subsystem .....	124
4.2.5 User subsystem .....	145
4.3 Dashboard and control panel .....	147
<b>CHAPTER 5: System design: model-based testing .....</b>	<b>149</b>
5.1 MIL testing .....	150
5.1.1 Single working point MIL testing .....	156
5.1.2 Nominal power working point MIL testing .....	161
5.1.3 Torque curves MIL testing.....	163
5.1.4 3D maps MIL testing .....	166

5.2	SIL testing .....	171
<b>CHAPTER 6: Rapid-Control-Prototyping code production ..</b>		<b>183</b>
6.1	Rapid-Control-Prototyping MIL testing .....	189
6.1.1	RCP Single working point MIL testing .....	189
6.1.2	RCP Nominal power working point MIL testing.....	191
6.1.3	RCP Torque curves MIL testing .....	193
6.1.4	RCP 3D maps MIL testing .....	195
6.2	Rapid-Control-Prototyping SIL testing .....	198
<b>Bibliography .....</b>		<b>203</b>

# LIST OF FIGURE

2.1	Generic V-shaped model . . . . .	4
2.2	Model-in-the-loop testing in V-shaped model . . . . .	6
2.3	Software-in-the-loop testing in V-shaped model . . . . .	7
2.4	Processor-in-the-loop testing in V-shaped model . . . . .	8
2.5	Hardware-in-the-loop testing in V-shaped model . . . . .	9
2.6	Hybrid V-shaped model . . . . .	10
2.7	Modular-Technical-Model general architecture . . . . .	14
3.1	Static Test Bench concept model . . . . .	17
3.2	Bench mapped motor Simulink block in concept model . . . .	18
3.3	Test mapped motor Simulink block in concept model . . . . .	18
3.4	Motor shaft Simulink block in concept model . . . . .	19
3.5	PID controller Simulink block in concept model . . . . .	19
3.6	Power supply in concept model . . . . .	21
3.7	Control monitor for motors' info output port . . . . .	21
3.8	Viewer for battery current motors' output port . . . . .	23
3.9	Viewer for bench motor's motor torque output port . . . . .	23
3.10	Viewer for test motor's speed command . . . . .	24
3.11	Viewer for bench motor's torque command . . . . .	24
3.12	Viewer for test motor's torque command . . . . .	25
3.13	Scope for supply power and current in concept model . . . .	25
3.14	Concept model's MATLAB script . . . . .	27
3.15	Mapped motor Simulink block parameters window in concept model . . . . .	28
3.16	Total absorbed power in concept model . . . . .	31
3.17	Total demanded current in concept model . . . . .	32
3.18	Bench motor dissipated power in concept model . . . . .	32
3.19	Test motor dissipated power in concept model . . . . .	33
3.20	Bench motor torque command in concept model . . . . .	34
3.21	Test motor torque command in concept model . . . . .	34
3.22	Test motor speed command in concept model . . . . .	35
3.23	Bench and test motor battery currents in concept model . . .	35
3.24	Bench and test motor generated torques in concept model . .	37
3.25	Bench motor mechanical and electrical parameters in concept model . . . . .	37

3.26	Test motor mechanical and electrical parameters in concept model . . . . .	38
4.1	Preliminary technical model . . . . .	40
4.2	Preliminary technical model's control monitors . . . . .	41
4.3	Preliminary technical model's MATLAB script . . . . .	43
4.4	$P_{d_{bm}}$ with both motors set in single efficiency measurement .	44
4.5	$P_{a_{bm}}$ with both motors set in single efficiency measurement .	44
4.6	$I_{bm}$ with both motors set in single efficiency measurement .	46
4.7	$P_{d_{tm}}$ with both motors set in single efficiency measurement .	46
4.8	$P_{a_{tm}}$ with both motors set in single efficiency measurement .	48
4.9	$I_{tm}$ with both motors set in single efficiency measurement .	48
4.10	$\omega$ with both motors set in single efficiency measurement . . .	50
4.11	$T_{tm}$ with both motors set in single efficiency measurement . .	50
4.12	$P_a, P_d$ with both motors set in single efficiency measurement	52
4.13	$I$ with both motors set in single efficiency measurement . . .	53
4.14	Electrical torque section of bench motor block parameters'window	54
4.15	Bench motor maximum torque and power curves . . . . .	55
4.16	Bench motor electrical torque section's MATLAB script . . .	56
4.17	Bench motor electrical losses section in tabulated loss data way	57
4.18	Bench motor's dissipated power map . . . . .	58
4.19	MATLAB script with bench motor's dissipated power map .	58
4.20	Bench motor's electrical losses section in tabulated efficiency data way . . . . .	60
4.21	Bench motor's efficiency map . . . . .	61
4.22	MATLAB script with bench motor's efficiency map . . . . .	61
4.23	$P_{d_{bm}}$ with bench motor set in tabulated loss data . . . . .	63
4.24	$P_{a_{bm}}$ with bench motor set in tabulated loss data . . . . .	63
4.25	$I_{bm}$ with bench motor set in tabulated loss data . . . . .	65
4.26	$P_{d_{tm}}$ with bench motor set in tabulated loss data . . . . .	65
4.27	$P_{a_{tm}}$ with bench motor set in tabulated loss data . . . . .	66
4.28	$I_{tm}$ with bench motor set in tabulated loss data . . . . .	67
4.29	$\omega$ with bench motor set in tabulated loss data . . . . .	67
4.30	$T_{tm}$ with bench motor set in tabulated loss data . . . . .	68
4.31	$P_a, P_d$ with bench motor set in tabulated loss data . . . . .	69
4.32	$I$ with bench motor set in tabulated loss data . . . . .	69
4.33	$P_{d_{bm}}$ with bench motor set in tabulated efficiency data . . .	70
4.34	$P_{a_{bm}}$ with bench motor set in tabulated efficiency data . . .	71
4.35	$I_{bm}$ with bench motor set in tabulated efficiency data . . . .	71
4.36	$P_{d_{tm}}$ with bench motor set in tabulated efficiency data . . .	72
4.37	$P_{a_{tm}}$ with bench motor set in tabulated efficiency data . . .	72
4.38	$I_{tm}$ with bench motor set in tabulated efficiency data . . . .	74
4.39	$\omega$ with bench motor set in tabulated efficiency data . . . . .	74
4.40	$T_{tm}$ with bench motor set in tabulated efficiency data . . . .	76
4.41	$P_a, P_d$ with bench motor set in tabulated efficiency data . . .	76



4.42	$I$ with bench motor set in tabulated efficiency data . . . . .	77
4.43	MATLAB script with tabulated bench and test motor . . . . .	78
4.44	$P_{d_{bm}}$ with both motors set in tabulated loss data . . . . .	80
4.45	$P_{a_{bm}}$ with both motors set in tabulated loss data . . . . .	81
4.46	$I_{bm}$ with both motors set in tabulated loss data . . . . .	81
4.47	$P_{d_{tm}}$ with both motors set in tabulated loss data . . . . .	83
4.48	$P_{a_{tm}}$ with both motors set in tabulated loss data . . . . .	83
4.49	$I_{tm}$ with both motors set in tabulated loss data . . . . .	85
4.50	$\omega$ with both motors set in tabulated loss data . . . . .	85
4.51	$T_{tm}$ with both motors set in tabulated loss data . . . . .	87
4.52	$P_a, P_d$ with both motors set in tabulated loss data . . . . .	87
4.53	$I$ with both motors set in tabulated loss data . . . . .	88
4.54	$P_{d_{bm}}$ with both motors set in tabulated efficiency data . . . . .	89
4.55	$P_{a_{bm}}$ with both motors set in tabulated efficiency data . . . . .	89
4.56	$I_{bm}$ with both motors set in tabulated efficiency data . . . . .	90
4.57	$P_{d_{tm}}$ with both motors set in tabulated efficiency data . . . . .	91
4.58	$P_{a_{tm}}$ with both motors set in tabulated efficiency data . . . . .	91
4.59	$I_{tm}$ with both motors set in tabulated efficiency data . . . . .	93
4.60	$\omega$ with both motors set in tabulated efficiency data . . . . .	93
4.61	$T_{tm}$ with both motors set in tabulated efficiency data . . . . .	95
4.62	$P_a, P_d$ with both motors set in tabulated efficiency data . . . . .	95
4.63	$I$ with both motors set in tabulated efficiency data . . . . .	97
4.64	MTM Simulink template . . . . .	98
4.65	Environment's reference model . . . . .	99
4.66	Torque sensor noise's block parameters window . . . . .	99
4.67	Torque sensor HGM T40B datasheet . . . . .	101
4.68	Current sensor DHR 300 C420 datasheet . . . . .	102
4.69	Round per minute measurement system datasheet . . . . .	104
4.70	Environment reference model's MATLAB script . . . . .	105
4.71	Plant reference model . . . . .	105
4.72	Sensor models subsystem . . . . .	108
4.73	Torque sensor Simulink model . . . . .	108
4.74	Current sensor's Simulink model . . . . .	110
4.75	Current sensor's transfer characteristic . . . . .	110
4.76	APICOM FR 250 RPM measurement system . . . . .	111
4.77	RPM measurement system's Simulink model . . . . .	112
4.78	Plant reference model's MATLAB script . . . . .	113
4.79	Control reference model . . . . .	114
4.80	Control reference model's MATLAB script . . . . .	116
4.81	Control logic reference Simulink model . . . . .	117
4.82	Emergency task's Simulink Stateflow . . . . .	118
4.83	Supervisor task's Simulink Stateflow . . . . .	118
4.84	BM_TrqCmd_task's Simulink stateflow . . . . .	120
4.85	BM_CONTROLLER_ON state's Simulink model . . . . .	121

4.86	TM_TrqCmd_task's Simulink model . . . . .	123
4.87	TM_CONTROLLER_ON state's Simulink . . . . .	123
4.88	Control Logic reference model's MATLAB script . . . . .	124
4.89	HMI's Simulink model . . . . .	125
4.90	Signal_to_parameters subsystem Simulink model . . . . .	126
4.91	Cmd_to_signal subsystem's Simulink model . . . . .	127
4.92	$\omega - T$ plane . . . . .	128
4.93	Single_working_point subsystem's Simulink model . . . . .	129
4.94	bm_cmd3 subsystem's Simulink model . . . . .	129
4.95	tm_cmd3 subsystem's Simulink model . . . . .	130
4.96	Single working point's MATLAB script . . . . .	130
4.97	Nominal_power_working_point subsystem's Simulink model . . . . .	132
4.98	bm_cmd2 subsystem's Simulink model . . . . .	133
4.99	tm_cmd2 subsystem's Simulink model . . . . .	134
4.100	Nominal power working point's MATLAB script . . . . .	135
4.101	Torque curves subsystem's Simulink model . . . . .	136
4.102	bm_cmd1 subsystem's Simulink model . . . . .	137
4.103	tm_cmd1 subsystem's Simulink model . . . . .	139
4.104	Torque curves' MATLAB script . . . . .	140
4.105	3D_maps subsystem's Simulink model . . . . .	141
4.106	bm_cmd0 subsystem's Simulink model . . . . .	142
4.107	tm_cmd0 subsystem's Simulink model . . . . .	144
4.108	3D maps' MATLAB script . . . . .	145
4.109	User subsystem Simulink model . . . . .	147
4.110	User subsystem's MATLAB script . . . . .	147
4.111	Dashboard and Control panel in MTM template . . . . .	148
5.1	MTM's MATLAB script . . . . .	154
5.2	Single working point MIL testing path . . . . .	157
5.3	Single_working_point.m's MATLAB script . . . . .	158
5.4	Single working point results in system design phase MIL testing	160
5.5	Nominal_powe_working_point.m's MATLAB script . . . . .	162
5.6	Nominal power working point results in system design phase MIL testing . . . . .	163
5.7	Torque_curves.m's MATLAB script . . . . .	164
5.8	Torque curves results in system design phase MIL testing . . . . .	166
5.9	3D_maps.m's MATLAB script . . . . .	168
5.10	Dissipated power 3D map result in system design phase MIL testing . . . . .	169
5.11	Efficiency 3D map result in system design phase MIL testing	169
5.12	Test motor parameter trends in system design phase MIL testing	170
5.13	Control Logic block parameters setting in SIL testing . . . . .	171
5.14	Diagnostic viewer window in SIL testing . . . . .	174
5.15	Dissipated power 3D map in SIL testing . . . . .	175
5.16	Efficiency 3D map in SIL testing . . . . .	176

5.17	Control Logic reference model setting for test harness generation	177
5.18	Control Logic's test harness model . . . . .	178
5.19	<i>green_comparison.mlx</i> 's MATLAB script . . . . .	179
5.20	MIL and SIL Control Logic green signal comparison . . . . .	179
5.21	<i>red_comparison.mlx</i> MATLAB script . . . . .	180
5.22	MIL and SIL Control Logic red signal comparison . . . . .	180
5.23	<i>bm_trqcmd_comparison.mlx</i> 's MATLAB script . . . . .	181
5.24	MIL and SIL Control Logic <i>bm_trqcmd</i> signal comparison . . . . .	181
5.25	<i>tm_trqcmd_comparison.mlx</i> 's MATLAB script . . . . .	182
5.26	MIL and SIL Control Logic <i>tm_trqcmd</i> signal comparison . . . . .	182
6.1	Control interface reference model in RCP phase . . . . .	184
6.2	dSpace ADC block model scheme . . . . .	184
6.3	dSpace DAC block model scheme . . . . .	185
6.4	Signal conditioning sections in Control Interface reference model	185
6.5	slblocks.m MATLAB script . . . . .	186
6.6	slblocks.m's MATLAB script . . . . .	187
6.7	Single working point results in RCP MIL testing . . . . .	190
6.8	Nominal power working point results in RCP MIL testing . . . . .	191
6.9	Torque curves results in RCP MIL testing . . . . .	195
6.10	Efficiency map result in RCP MIL testing . . . . .	196
6.11	Dissipated power 3D map result in RCP MIL testing . . . . .	196
6.12	Test motor characteristics results in RCP MIL testing . . . . .	197
6.13	Control Logic reference model setting in RCP SIL simulation	199
6.14	Efficiency 3D map result in RCP SIL testing . . . . .	200
6.15	Dissipated power 3D map result in RCP SIL testing . . . . .	200
6.16	Test motor characteristics results in RCP SIL testing . . . . .	201



# Chapter 1

## Introduction

The rapid development and spread of electric motors in automotive industry and other sectors has involved the need to develop parallelly equipment that allows to perform tests on them and to evaluate their characteristics in order to be able to choose the best motor on the basis of requirements to be satisfied. Moreover, this development has required equipment's development to perform quality control and to diagnose possible problems while they are working.

A way for obtaining experimentally electrical and mechanical characteristics of an electric motor is based on the use of a test bench which is made up of a known electric motor, called also *brake motor*, connected to the motor to be tested in order to perform all tests that allow to characterize the electric motor under test. From this, it's possible to understand that it's very important to have under control the brake motor because in that way specific inputs can be imposed to the electric motor under test and all needed output data can be obtained. Usually, brake motors are piloted by a motor driver connected to a PC by means users is able to set input parameters on the basis of test to be performed. In addition, a control system made up of a computer is needed to control brake motor's behavior piloting its motor driver and to be able to acquire and store interesting data needed for characterizing the electric motor under test.

The goal of these thesis activities is implementing a *regenerative active test bench for electric motor* using a Vehicle Management Unit (VMU), electronic control unit used to elaborate actions to be taken on the basis of data coming from sensors used in automotive industry, in order to reduce test bench's costs that usually have to be addressed by companies for developing a new test bench. *Regenerative test bench* means that mechanical energy of under test electric motor has to be converted in electrical energy to be fed into national electricity grid. This, for sure, allows to have significant energy saving and a small power supply even if electric motors to be tested require a very huge power (30 -50 kW). This because, in that way power supply must be able to provide a power more or less equal to dissipated power by two

motors. While, *active test bench* means that brake motor is not moved by rotational movement imposed by under test electric motor, but it's piloted by user through its motor driver and its inverter. This, of course, allows to obtain more precise measurements.

In the cases where brake motor is put in rotation by rotational movement imposed by the under test electric motor, test bench is called *passive test bench*. In that cases, the under test electric motor is piloted by an inverter which manages voltage and frequency values to impose desired rotational speed values to test motor's shaft. Once test motor's shaft starts to rotate, also passive breaking motor starts rotating and what is called *electromagnetic brakes* happens. This because, once the test motor's shaft is put in rotation there will be electromagnetic induction that will generate eddy currents that are responsible of a flux variation that implies a breaking force. This breaking force is contrasted by load cell that allows to obtain force and torque values through which test motor's shaft power values can be computed: multiplying each other rotational speed and torque values.

Instead, in *active test bench cases* brake motor is used to generate a load connected to test motor. Setting bench motor parameters, load dynamic can be controlled so in that case an additional inverter with respect to passive test bench cases is needed to pilot bench motor and generating desired loads. From this, it's possible to understand that in active test bench cases higher costs have to be addressed by companies and more complicated system has to be developed. This, due to the need of two different inverters: one for piloting bench motor and another one for managing test motor. On the other hand, this kind of test bench allows to improve measurements' repeatability and accuracy.

Moreover, in active test bench a dynamometer and a speed sensor connected to shaft interconnecting two motors must be used to measure torque and rotational speed values. Other specific sensors are adopted to measure battery voltage and motor phase current values.

The following chapters allow to understand the procedure followed to develop a regenerative active test bench's model and to obtain corresponding C code, implemented into VMU, to be used to build up final physical system.

# Chapter 2

## System Engineering approach

To develop regenerative active test bench introduced in previous chapter, *system engineering approach* is followed in order to have standard procedure that allows to build-up a prototype with all user's required characteristics starting from an initial concept model. This is also done because thanks to this approach, different disciplines and tools needed to develop system can be included. Moreover, it allows to reduce time and costs required for developing the final system. In this thesis work two different system engineering tools have been adopted in order to develop desired test bench for electric motors: *V-shaped model and Modular Technical Model (MTM)*.

The first one allows to define a procedure to be followed starting from user's concept for obtaining final system prototype with all desired characteristics. So, V-shaped model tool defines all phases to deal with to develop the final physical system in a way that guarantees user safety. Indeed, it implies many tests during all development procedure.

Instead, the second tool allows to subdivide the system to be developed in *modules*, different parts, such that development's efficiency and quality are improved. This because each module can be developed by a different team having specific skills. Moreover, this tool allows to improve reusability since developed modules in a specific project can be also adopted in other project's development. Obviously, this implies a reduction of development time and costs to be held by companies, but on the other hand development's complexity is increased due to the fact that all modules must be able to communicate each other so designers must develop them in such a way that can be possible.

### 2.1 V-shaped model

V-shaped model allows to define development process for a system to be developed with specific characteristics and requirements defined by the user. This tool subdivides development process in phases that must be performed on succession to proceed in schematic way and obtaining final build-up

prototype. This, for sure, allows to improve user safety since in each phase different tests are required in order to verify whether user's requirements are satisfied and to proceed with the V-shaped model phases.

From all this, it is possible to understand that V-shaped model tool allows to organize in a better and more efficient way system's development in which debugging errors and problems is simpler.

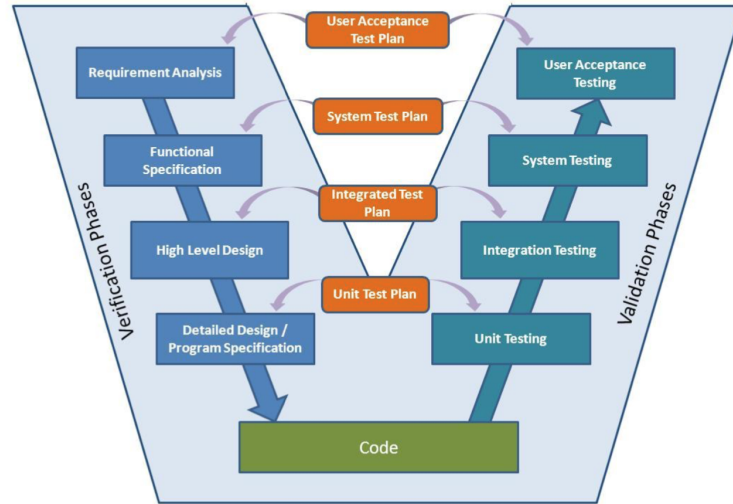


Figure 2.1: Generic V-shaped model

Observing the V-shaped model reported above, three different branches can be distinguished: *descendent branch* that represents *verification phase* that starts with requirements analysis and ends with detailed design and program specification; *horizontal branch* (the bottom part) that represents *coding phase* during which corresponding model's C code is obtained; *ascendant branch* that represents *validation phase* where modules previously designed are put together and tested to understand whether everything is working fine or not on the base of initial user's requirements. At the end of each phase, specific tests are performed. Below, a brief description of all performed tests is reported.



- *Model-in-the-loop testing*

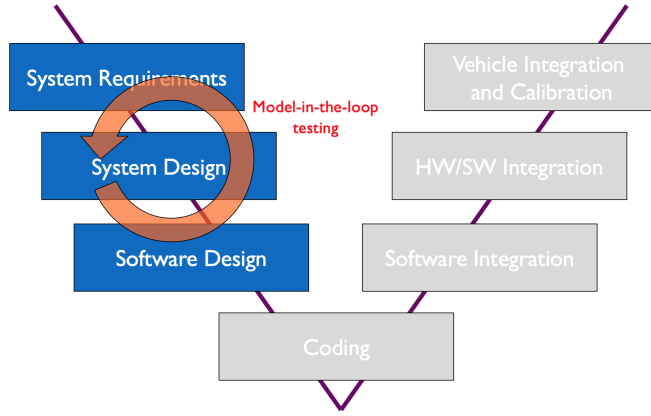


Figure 2.2: Model-in-the-loop testing in V-shaped model

It represents the first test performed during V-shaped model phases and includes all verification phases in order to test designed controller and plant. To perform it, both plant and controller are modelled in native simulation tool and they are run in development machine where they are fed with inputs to obtain corresponding outputs through which designed system's behavior can be evaluated in order to understand whether it must be changed or not on the basis of user's requirements. Moreover, to perform this test, both controller and plant models are run in development machine in no real- time.

- *Software-in-the-loop testing*

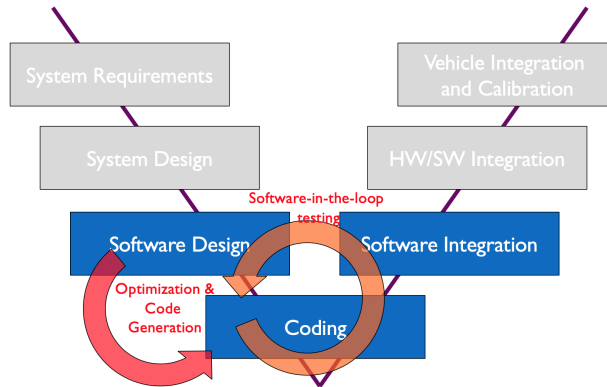


Figure 2.3: Software-in-the-loop testing in V-shaped model

It represents the second test performed during V-shaped model phases. It is performed during the last step of verification phase (descendant branch of V-shaped model), during code V-cycle phase (horizontal branch of V-shaped model) and during first step of validation phase (ascendant part of V-shaped model). Thanks to it, obtained Control Logic's C-code is tested in order to verify whether it has the same behavior of corresponding model before designed and tested. Also, in this case, like MIL testing case, both plant model and controller C-code are run in development machine and they are fed with inputs to obtain corresponding outputs to be compared to those obtained in MIL testing case to verify C-code's behavior: it works properly if obtained results are equal to each other.

- *Processor-in-the-loop testing*

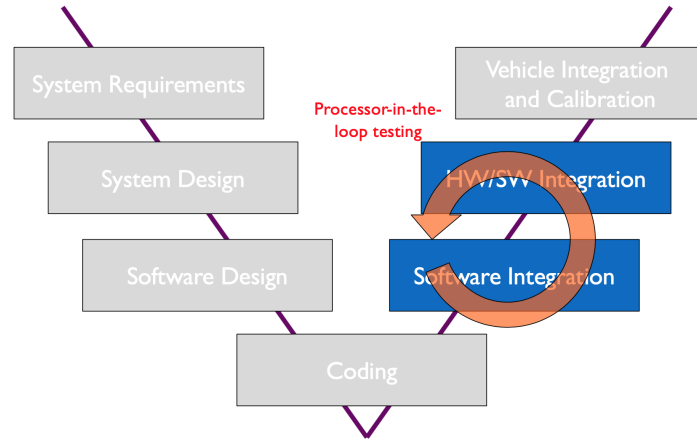


Figure 2.4: Processor-in-the-loop testing in V-shaped model

It represents the third test performed during V-shaped model phases. In particular, it is performed during the V-shaped model's first two steps of validation phase (ascendant branch). To perform it, plant's model is run in development machine while Control Logic's C-code, obtained in previous phase, is run in target or rapid prototype hardware to verify its behavior.

- *Hardware-in-the-loop testing*

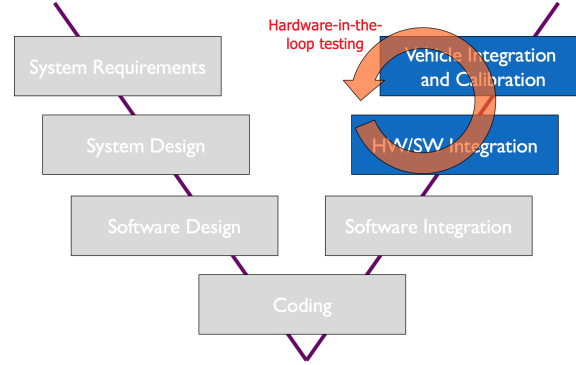


Figure 2.5: Hardware-in-the-loop testing in V-shaped model

It represents the last test performed during V-shaped model phases. In particular, it is performed during last two steps of validation phase (ascendant branch). To perform it, obtained and tested Control Logic's C-code is run in target or rapid prototyping hardware while plant is simulated by real time simulator. This test allows to verify plant's behavior that is managed by Control Logic's C-code implemented into the hardware.

### 2.1.1 Hybrid V-shaped model

Going through V-shaped model details, Hybrid V-shaped model must be considered. Its scheme is reported below.

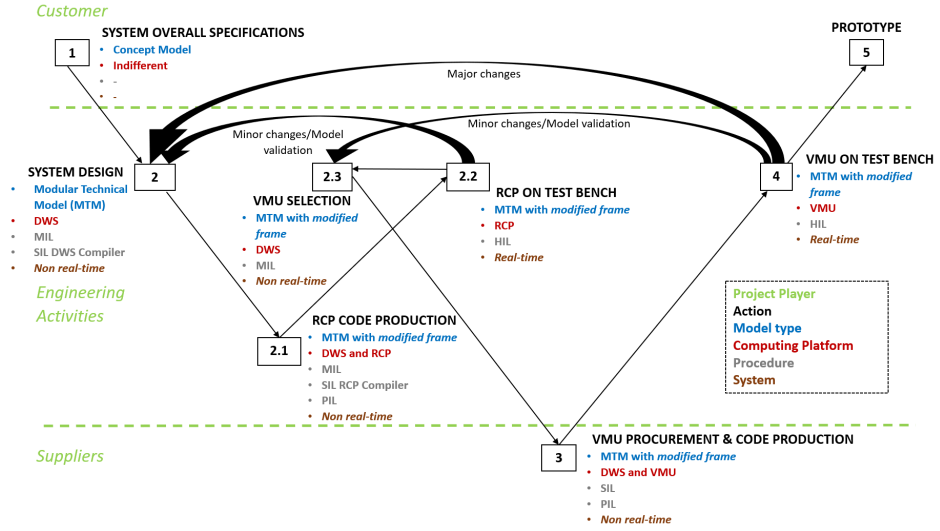


Figure 2.6: Hybrid V-shaped model

Here, phases to be followed for obtaining final build-up prototype are highlighted where several models and tools are used. Below, a brief description of each phase is reported.

- *System Overall specification*

It is the first phase during which user develops what is called *concept model* by means he presents his idea to design company that must find a way to implement it. To do this, user can adopt any native platform neglecting all related technical details. This because it is up to designers evaluating technical details to be considered for obtaining a system that allows to satisfy all user requirements.

- *System design*

It is the second phase during which *Development Workstation (DWS)* is used for designing and simulating system's model. Here, only no real-time concept is considered and at the beginning, system's design and simulation are done neglecting target hardware's characteristics. During this phase MIL and SIL testing are performed using *Modular Technical Model (MTM)*. Below all System design sub-phases are listed and described.

- *Project Environment*

It allows to organize system's development in a better way. Indeed, during this sub- phase what is called *project environment* is created that implies several folders: one for each performed step.

- *Model-in-the-loop*

It allows to evaluate system model's behavior once it is implemented starting from concept model upgraded considering all technical details and characteristics. Results of this test must be comparable with those obtained simulating concept model provided by user at the beginning. In this sub-phase, MTM frames consist of generic interfaces because they are not intended for *Rapid Control Prototyping Platform (RCPP)*.

- *Software-in-the-loop*

It is performed once Control Logic's C-code is automatically generated in order to verify its behavior: to proceed its behavior must be equal to that obtained in the previous case where model is used. Here control law is run in DWS and on the basis of tool to be used for obtaining code, DWS system target file is selected.

- *Rapid control prototyping (RCP) code production*

Here, both DWS and RCP platform are used to perform SIL, MIL and PIL testing. To do this MTM standard architecture is used and system is considered to be non-real-time. In this case, MIL testing, unlike MIL testing performed in previous phase, is performed considering RCPP interfaces which must be modelled. So, before to perform MIL testing in this case MTM interfaces must be modified with respect to those used in System Design MIL testing case. This allows to verify system model's behavior in presence of RCPP modelled interfaces. To do this, simulation results obtained in this case must be compared to those obtained in the previous phase during MIL and SIL testing. Once MIL testing is done and verified that system's model works properly Control Logic's C-code must be obtained and tested. For this reason, SIL testing has to be performed setting rapid control prototyping system target file and running everything in DWS. Also in this case, to understand whether obtained C-code works properly or not, simulation results must be compared to those obtained in previous MIL testing. Then, PIL testing can be performed using RCP platform connected to DWS. This is done using RCP tool of specific chosen platform which allows communication between rapid prototyping hardware where C-code is run and plant's model which is run in DWS.

- *Rapid control prototyping (RCP) on test bench*

In this case, like in PIL testing case of previous phase, RCP platform is considered and connected to DWS by means the specific RCP tool. But here, unlike what happens in PIL testing of previous phase, system is considered being in real-time. This because interfaces are represented by test bench so, for sure, they are real-time while control law is taken from MTM and run in DWS. Even this phase is subdivided in sub-phases, listed below.

- *Build*

Here, control law C-code is automatically generated selecting RCP platform System Target File and compiled so that C-code, to be implemented inside the chosen platform, is built.

- *Hardware-in-the-loop*

Here, control law C-code obtained in previous sub-phase is deployed on RCP platform such that the entire system can be tested using test bench which represents the real interface. To manage input/output signals of RCP platform a proper RCP software is used.

- *VMU Selection*

During this phase, MIL testing is performed to analyze the characteristics that VMU must have to build-up the final physical system.

- *VMU Procurement and Code Production*

During this phase SIL and PIL testing are performed considering the model developed during the previous phase. This procedure is similar to what is done in RCP Code Production phase but this time instead of considering RCP platform's model, VMU's model is considered.

- *VMU on Test Bench*

Here, VMU is tested in real-time since it is connected to DWS where control law is developed and interfaces, represented by test bench, are real. So, during this phase an HIL testing is performed.

- *Prototype*

This is the last phase of V-shaped model in which final system prototype is built-up. This prototype has real-time interfaces.

## 2.2 Modular-Technical Model (MTM)

*Modular Technical Model (MTM)*, like V-shaped model, is a tool used in System Engineering approach. It allows to subdivide system to be developed into what is called *modules* such that more than one team with different

skills can be engaged to develop all modules and obtain the final physical system's model. This, for sure, allows designers to improve reusability since developed modules can be adopted in different projects. At the same time, this allows to reduce development time because the different system parts (modules) are developed concurrently by more teams who at the end must be put together for obtaining the final physical system.

This tool is called *Modular* since it provides modules that represent the different parts on which the system is subdivided. So, each module is developed independently from other modules and it must be able to work without considering all system parts. Then, it is called *Technical* because the standard structure, provided by this tool, contains all technical details needed to be taken into care for developing desired system.

Standard architecture, shown in figure 2.7, never changes and it provides always the same modules that must be filled on the basis of details and characteristics of considered system.

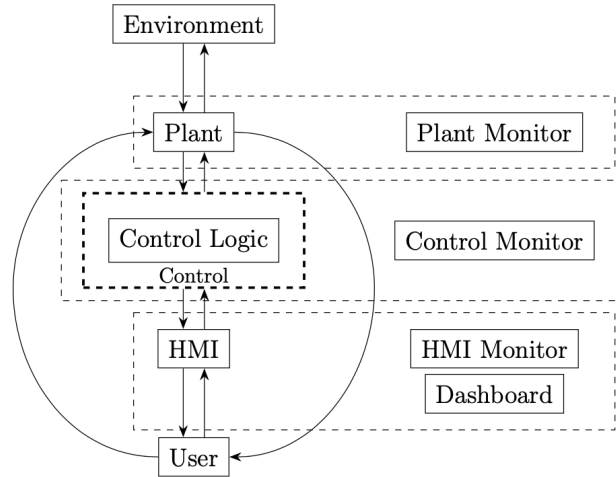


Figure 2.7: Modular-Technical-Model general architecture

From this scheme, the different modules below listed and described can be distinguished.

- *Environment*

This module includes all disturbances and noises coming from ambient in which system works that can impact system's behavior.

- *Plant*

This module contains system's model whose behavior must be controlled by control law. Moreover, it also includes monitors by means user is able to observe measured parameter trends.

- *Control*

Inside this modules *Control Logic module* is implemented that represents application code to be implemented for managing plant's behavior. Here, besides to find Control Logic module, layer allowing interaction between control law and target hardware is implemented. Moreover, a control scope is implemented to take under control input and output signals. So, Control module contains target hardware interface's models that are very important to be taken into care during Control Logic C-code production procedure in order to verify obtained C-code's behavior in presence of real target hardware's interfaces

- *Human-Machine-Interface*

This module is intended for translating user commands into signal commands to be transmitted to Control module. So, here, different push-buttons and LEDs could be implemented through which user can start/stop system and observe its system status. In other words, this module is intended to implement dashboard by means user can manage system's status. Also in this case, a scope can be implemented to take under control all interesting quantities coming out from plant and/or control.

- *User*

This module is intended to simulate all actions to be done by user to set system before to start it.

In addition, arrows that interconnect each other all different modules represent physical interconnections to be implemented for allowing communication among physical built-up system parts. These interconnections can be implemented by means busses or cables allowing signals exchanges. These signals can be either continuous or discrete: all modules work with continuous signals except Control module in which Control Logic module is implemented and works with only discrete signals. Layer modelled in Control module that represents interfaces between control law and target hardware can work both discrete and continuous signals.



## Chapter 3

# Test Bench Concept Model

As explained in previous chapters to develop and implement desired *electric motor test bench*, V-shaped model tool is adopted. So, following standard architecture provided by this tool the first phase is *System Overall Specification* by means user presents to designers his idea using what is called *concept model* where idea is modelled neglecting all technical details to be considered for system's development. Concept model presented by user in this case is shown in figure 3.1.

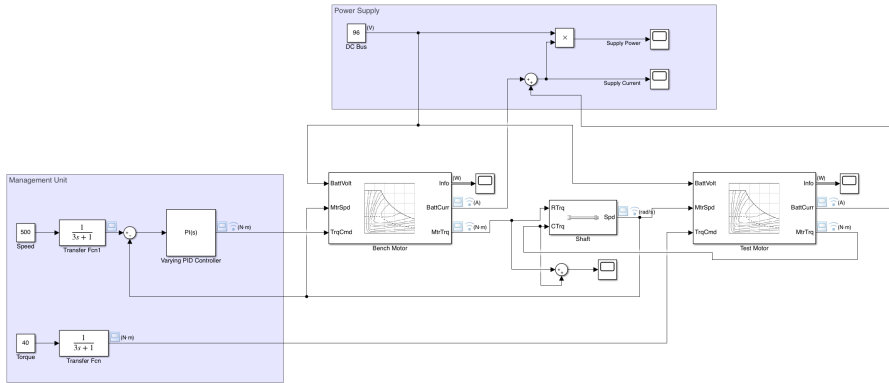


Figure 3.1: Static Test Bench concept model

From this model, it is possible to understand that user presents his idea by means a Simulink model where all specifications to be satisfied are highlighted. Here, the different components to be designed and implemented can be distinguished: *bench motor*, *test motor*, *shaft*, *controller*, *inverter* and *control monitors*. Below a brief description of each element is reported.

- *Bench motor*

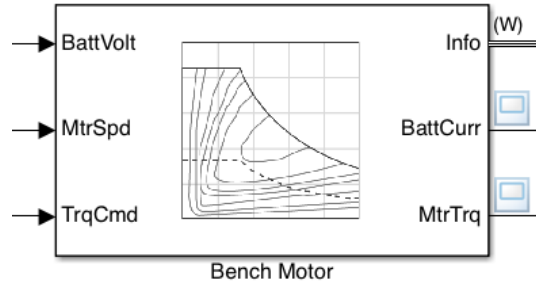


Figure 3.2: Bench mapped motor Simulink block in concept model

This, called also brake motor, is responsible to impose rotational speed to electric motor to be tested. It is piloted by user who provides rotational speed values that is compared to rotational speed values coming from feedback path in order to evaluate corresponding error to be provided to PID controller which is able to decide action to be performed so that measured motor shaft's rotational speed is equal to rotational speed value desired by user. Moreover, it must be provided by its own inverter which needs a specific input voltage value and by means a motor shaft it must be interconnected to electric motor to be tested in order to be able to impose desired rotational speed values.

- *Test motor*

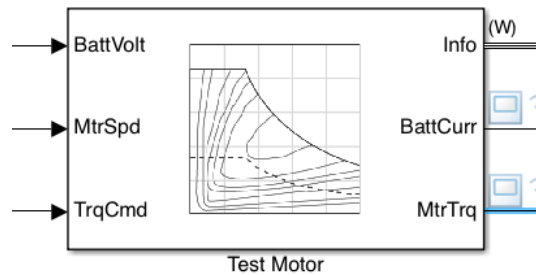


Figure 3.3: Test mapped motor Simulink block in concept model

It represents the electric motor to be tested for obtaining its mechanical and electrical characteristics. This electric motor, unlike bench motor, is piloted by user through torque command. It, like brake motor, must be provided by its own inverter which requires a constant input voltage value and connected to motor shaft in order to be interconnected to bench motor.

- *Motor shaft*

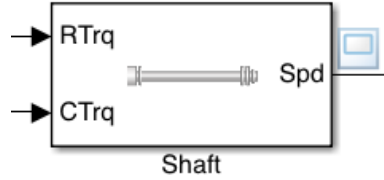


Figure 3.4: Motor shaft Simulink block in concept model

It represents interconnection elements between the two motors. It must be considered because, for sure, it introduces mechanical loss that cannot be neglected.

- *PID controller*

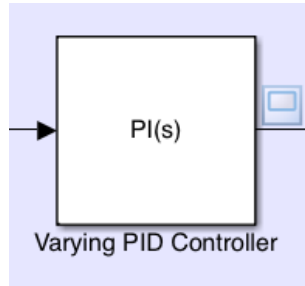


Figure 3.5: PID controller Simulink block in concept model

PID controller stands for *Proportional, Integral, Derivative controller* that represents a controller that is able to provide continuous variation of output. It implements a control loop based on feedback in order to control accurately system, increasing efficiency and removing oscillations. Thanks to feedback mechanism it is able to continuously compute an error value between what is called *setpoint* (rotational speed desired by user) and what is called *measured process variable* (bench motor out rotational speed) and apply a correction on three different parts: *proportional part* responsible to set a proportional correction on the basis of computed error (if error is equal to zero no correction occurs); *integral part* responsible for considering past error values and integrating them over time to evaluate *integral term (I)* to be multiplied for current error value; derivative parts responsible to implement a sort of anticipatory control because on the base of its change rate it is able to estimate further signal trend. *Integral term (I)* decreases once error is eliminated while *derivative term (D)* decreases once error variations become less rapid.

- *Power supply*

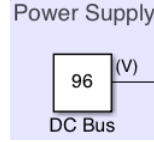


Figure 3.6: Power supply in concept model

It is modelled by means a constant because it represents constant voltage provided by a battery that must be connected to bench and test motor's power supply port.

- *Control monitors*

Monitors are used to analyze interesting parameter trends. Simulink model allows to analyze different parameter trends such as those coming out from bench and test motor's *info output port*. This output port allows to analyze motors' mechanical and electrical quantity trends such as: *mechanical power*, *power loss*, *motor power*, *electric power*, *motor power loss* and *motor power stored*. For this reason, user connects a scope to info output port of both mapped motor Simulink blocks (motif with a red rectangle in figure 3.7).

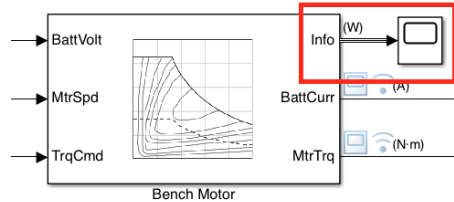


Figure 3.7: Control monitor for motors' info output port

Below a brief description of all parameters coming out from info output port is reported.

- *Mechanical power*

It is a signal coming out from bench and test motors' info output port that carries information about mechanical power produced by motors starting from electrical power provided by power supply and inverter. It is measured in rad.

- *Power loss*

This is another signal coming out from both motor blocks' info output port. It carries information about internal inverter and motor's power loss. It is measure in Nm.

– *Power info*

It represents another signal coming out from both motor blocks' info output that carries information about four different parameters such as: *motor power*, *electric power*, *motor power loss* and *motor power stored*. The first parameter refers to mechanical power produced by motor and it is measured in W while the second one refers to electrical power produced by motor measured in W. Instead, the third parameter refers to power loss due to friction presence (mechanical loss), iron losses because of hysteresis and eddy currents, copper losses because of conduction, additional losses in metal masses near windings because of eddy currents and in lamination because of imperfect insulation, losses in insulator that usually can be overlooked.

Besides scope above described, user implements other scopes to verify other parameter trends such as those below listed.

– *Battery current*

It represents another bench and test motor blocks' output port from which a signal carrying information about draw or demand current by motors comes out. It represents demand current if its value is negative or draw current if its value is positive. It is measured in A and its trend is taken under control by means a viewer connected directly on bus as highlighted in figure 3.8.

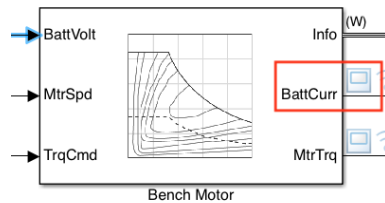


Figure 3.8: Viewer for battery current motors' output port

This viewer allows to control both motors' battery current trend and comparing them to each other.

– *Motor torque*

It represents another mapped motor Simulink blocks' output port from which a signal carrying information about motor output shaft torque. It is measured in Nm. This signal, like battery current case, is taken under control by means a viewer directly connected on bus as shown in figure 3.9.

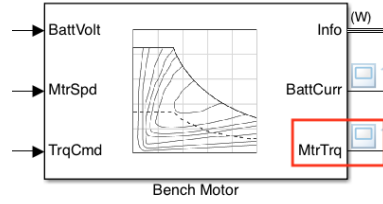


Figure 3.9: Viewer for bench motor's motor torque output port

This viewer allows to control motor torque signal trend of both motors and compare them to each other.

– *Motor speed*

It represents signal coming out from motor shaft's speed output port that carries information about speed control signal for test motor. It is measured in rad/s and taken under control by means a viewer directly connected to bus as shown in figure 3.10.

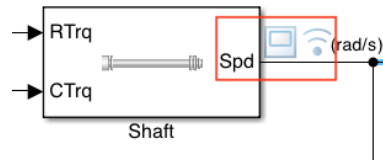


Figure 3.10: Viewer for test motor's speed command

– *Bench motor torque command*

It represents signal coming out from PID controller that carries information about torque command used to pilot bench motor and obtaining user's desired rotational speed at motor shaft's output port. It is measured in Nm and taken under control by means a viewer directly connected to bus as shown in figure 3.11.

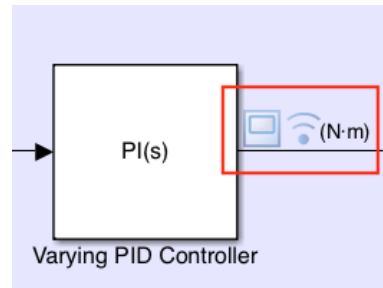


Figure 3.11: Viewer for bench motor's torque command

– *Test motor torque command*

It represents a signal coming out from transfer function Simulink block used to pilot test motor. It is measured in Nm and taken under control by means a viewer directly connected to bus as shown on figure below.

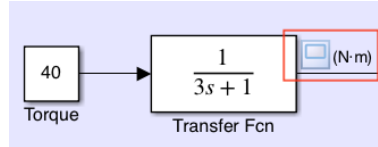


Figure 3.12: Viewer for test motor's torque command

– *Supply power and current*

They represent parameters computed starting from signals coming out from some mapped motor Simulink blocks' output ports. Indeed, supply current parameter is computed starting from battery current signals coming out from “*BattCurr*” output ports that must be added together while supply power is computed starting from supply current, before described, that has to be multiplied by power supply voltage value.

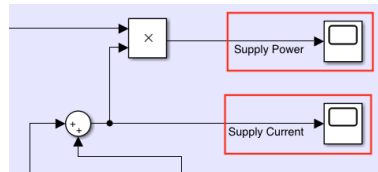


Figure 3.13: Scope for supply power and current in concept model

### 3.1 User requirements

User asks for different requirements to be satisfy for test bench to be designed and implemented. All these requirements can be summarized in: *static active regenerative test bench*. Below a detailed explanation is reported.

- *Regenerative*

This means that user requires an electric motor test bench in which total absorbed power is more or less equal to the sum of bench and test motor's dissipated power. This is required by user because it requires a test bench that is able to test motors that require very huge power (30 – 50 kW) using a small power supply. For this reason, in concept model user implements a control monitor allowing to take under control absorbed power and demanded current trends.

- *Active*

This means that user requires a test bench in which both motors (bench and test motor) are piloted by an own inverter. This because user wants an electric motor test bench with a very high accuracy into evaluating test motor characteristics. This can be reach only if bench motor is piloted by its own inverter, like test motor, because in that way user is able to control bench motor's behavior on the basis of test to be performed. This is very important since in those cases bench motor represents the load connected to test motor.

- *Static*

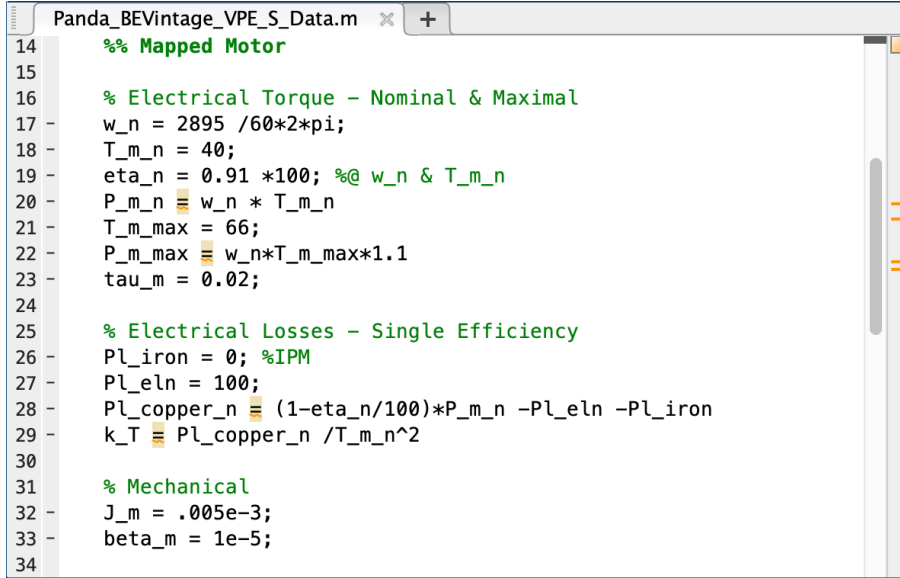
This means that user is asking for a test bench where dynamic that can affect test motor's behavior has not to be considered. This, in turns, means that interesting parameters must be measured in steady state condition.

Besides all these requirements, another very important user's request is the fact that electric motor test bench must be designed and implemented in such a way that both motors are piloted by using a *Vehicle Management Unit (VMU)* in order to reduce very high costs to be addressed by companies in standard situation. Indeed, in standard situation companies must address high costs for building up what is called *Human Machine Interface (HMI)* that is constituted of two different drivers, one for each motor, different computers for storing data and setting desired parameters on the basis of test to be performed. In addition, VMU usage must allow to automatize test bench in order to be able to obtain 3D efficiency and dissipated power map of electric motor under test.



### 3.2 Concept model simulation

To simulate concept model shown at beginning of this chapter, a MATLAB script must be run before to start Simulink simulation. This MATLAB script (figure 3.14) must contain all needed parameters for parametrizing mapped motor Simulink blocks used to implement model.



```

Panda_BEVintage_VPE_S_Data.m
14  %% Mapped Motor
15
16  % Electrical Torque - Nominal & Maximal
17  w_n = 2895 / 60 * 2 * pi;
18  T_m_n = 40;
19  eta_n = 0.91 * 100; % @ w_n & T_m_n
20  P_m_n = w_n * T_m_n;
21  T_m_max = 66;
22  P_m_max = w_n * T_m_max * 1.1;
23  tau_m = 0.02;
24
25  % Electrical Losses - Single Efficiency
26  Pl_iron = 0; % IPM
27  Pl_eln = 100;
28  Pl_copper_n = (1 - eta_n / 100) * P_m_n - Pl_eln - Pl_iron;
29  k_T = Pl_copper_n / T_m_n^2;
30
31  % Mechanical
32  J_m = .005e-3;
33  beta_m = 1e-5;
34

```

Figure 3.14: Concept model's MATLAB script

Here, a mapped motor section can be distinguished where all parameters needed to parametrize bench and test mapped motor Simulink blocks are defined. To do this user consider the same electric motor to model both brake and test motor. Indeed, in MATLAB script above reported there is no distinction between two motors. This can be also verified from Simulink model opening block parameters' window of each mapped motor Simulink block where the same variables' name is defined.

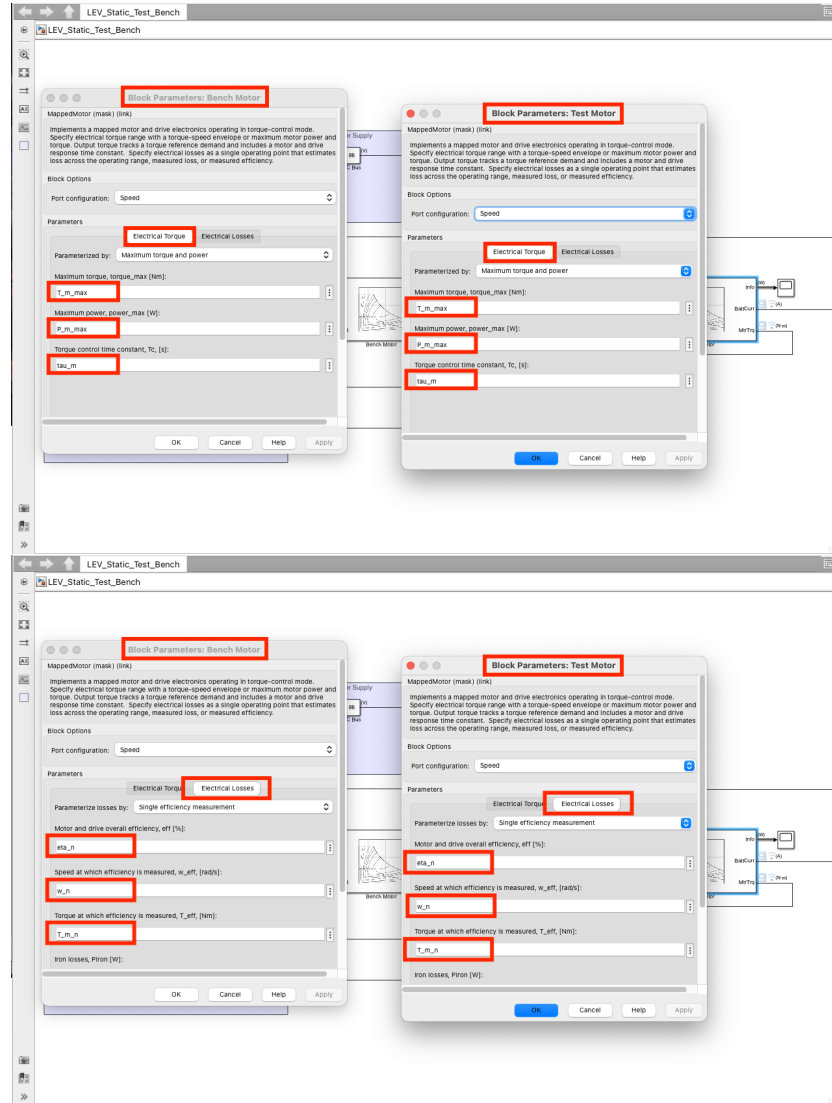


Figure 3.15: Mapped motor Simulink block parameters window in concept model

Below all needed parameters to be defined in MATLAB script are listed and described.

- *Nominal rotational speed ( $w_n$ )*

It represents rotational speed value assumed by motors in nominal conditions. So, it represents the rotational speed value for which electric motor can work for an “infinite period” guaranteeing always performances declared in rating plate by designer. It is measured in rad/s.

- *Nominal mechanical torque ( $T_{m_n}$ )*

It represents torque value provided by electrical motor in nominal conditions. So, it represents torque value for which electric motor can work for an “infinite period” without problems guaranteeing performances declared in rating plane. It is measured in Nm.

- *Nominal efficiency ( $\eta_n$ )*

It represents electric motor efficiency’s value in nominal conditions. So, it represents efficiency’s value guaranteed by electric motor when it works at nominal rotational speed and nominal mechanical torque.

- *Nominal mechanical power ( $P_{m_n}$ )*

It represents electric motor’s mechanical power at crankshaft when motor works in nominal conditions. Indeed, it can be computed as the product between nominal mechanical torque and nominal rotational speed. It is measured in W.

- *Maximum mechanical torque ( $T_{m_{max}}$ )*

It represents the maximum torque value that can be provided to electric motor without causing problems to electric and mechanical circuit inside motor. It is measured in Nm.

- *Maximum mechanical power ( $P_{m_{max}}$ )*

It represents the maximum mechanical power that can be provided by electric motor without causing problems to electric motor’s electrical and mechanical parts. It can be computed as product between maximum mechanical torque and maximum rotational speed. It is measured in W.

- *Mechanical time constant ( $\tau_m$ )*

It represents time constant measuring time needed by electric motor to reach about 63measures electric motor’s responsiveness.

- *Iron power loss ( $Pl_{iron}$ )*

This variable defines all power loss in the iron. So, it defines power loss due to Joule effect which depends on absorbed current by load, power loss due to ferromagnetic material’s hysteresis, power loss due to eddy currents’ presence. It is measured in W.

- *Nominal electric power loss ( $Pl_{eln}$ )*

This variable defines all power loss that are independent from torque and rotational speed. It is measured in W.

- *Torque constant ( $k_T$ )*

This constant is specified by motor's designer. It depends on magnetic strength and on wire's number. Moreover, it determines the slope of torque-current curve.

- *Mechanical moment of inertia ( $J_m$ )*

This variable defines moment of inertia generated by mechanical masses in rotation with respect to rotational axis. It is measured in  $\text{kgm}^2$ .

- *Mechanical constant ( $\beta_m$ )*

This variable defines friction generated between electric motor's rotating part. It allows to compute resistant torque applied to motor shaft.

### 3.3 Concept model simulation results

Running MATLAB script, shown in figure 3.14, all defined variables will be loaded in RAM memory so that Simulink model blocks can refer to them during simulation. Before to start Simulink simulation, simulation time must be set so that when it is expired simulation stops. Considering a simulation time of 50 seconds and running Simulink concept model results below reported are obtained.

- *Total absorbed power*

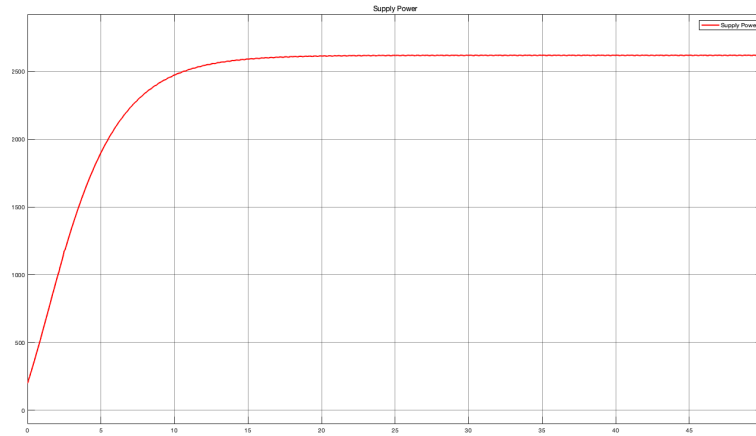


Figure 3.16: Total absorbed power in concept model

Here, total absorbed power's steady state value is about  $2.621\text{e}+03$  W.

- *Total demanded current*

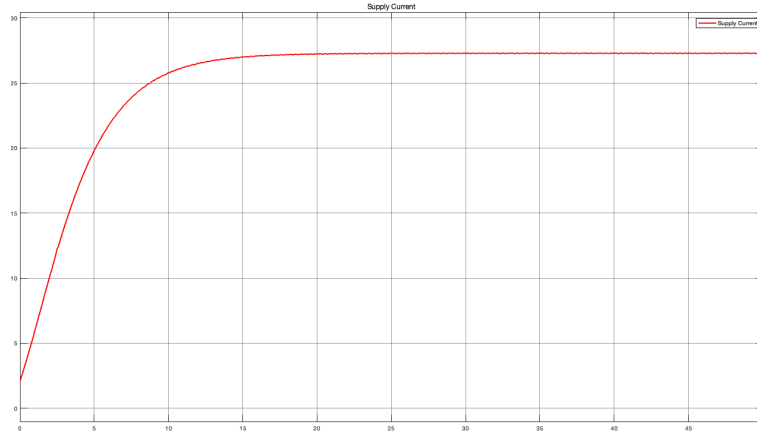


Figure 3.17: Total demanded current in concept model

Here, total demanded current steady-state value is about  $2.730 \times 10^1$  A.

- *Bench motor dissipated power*

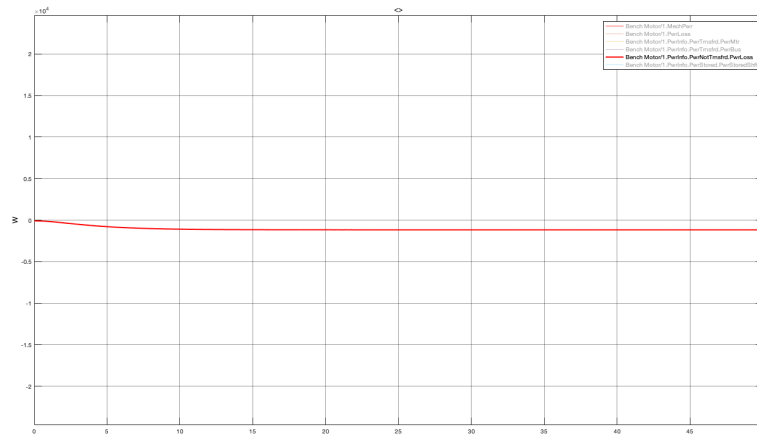


Figure 3.18: Bench motor dissipated power in concept model

Here, bench motor's dissipated power steady-state value is about -1.172 kW.

- *Test motor dissipated power*

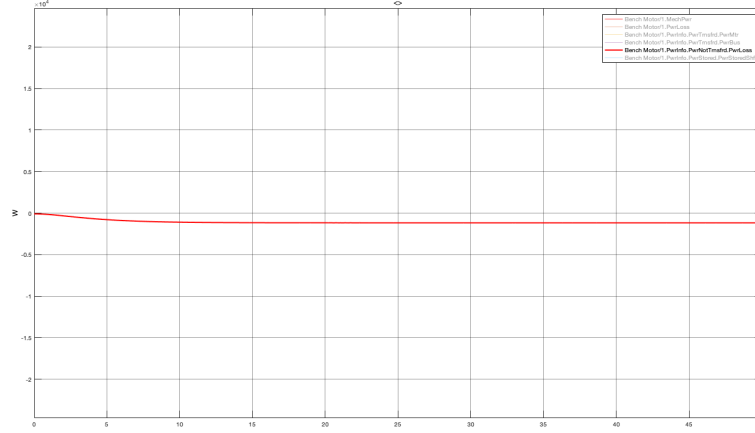


Figure 3.19: Test motor dissipated power in concept model

Here, test motor's dissipated power steady-state value is about -1.199 kW.

Summing bench and test motor's dissipated power, total dissipated power is obtained that at steady-state assumes a value equal to  $-2.371e+03$  W. Comparing this value with total absorbed power value, user's regenerating requirement is demonstrated. Absorbed power, of course, is a little bit bigger due to motor shaft presence that, for sure, introduce mechanical losses that are not considered in summing bench and test motor's dissipated power.

Besides above parameter trends, other parameters trends can be obtained from concept model such as: *bench motor torque command*, *test motor torque command*, *test motor speed command*, *battery currents*, *generated motor torques* and *all bench and test motor parameters coming out from their info output ports*.

- *Bench motor torque command*

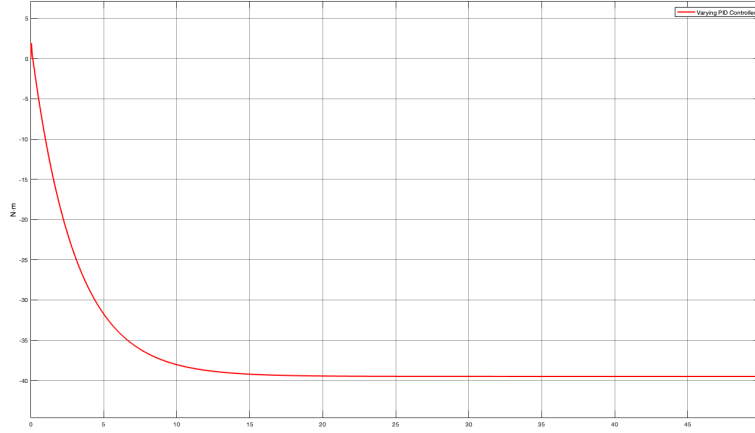


Figure 3.20: Bench motor torque command in concept model

Here, bench motor torque command's steady-state value coming out from PID controller is about -39.5 Nm.

- *Test motor torque command*

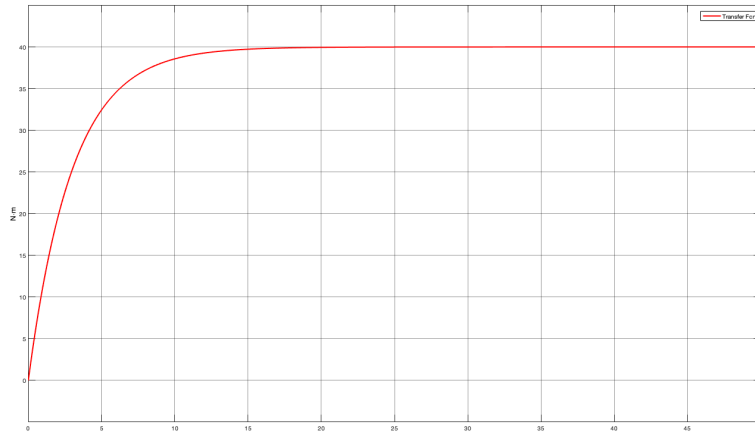


Figure 3.21: Test motor torque command in concept model

Here, test motor torque command's steady-state value coming out from transfer function is about 40 Nm as desired since user set 40 Nm as torque value to pilot test motor.

- *Test motor speed command*

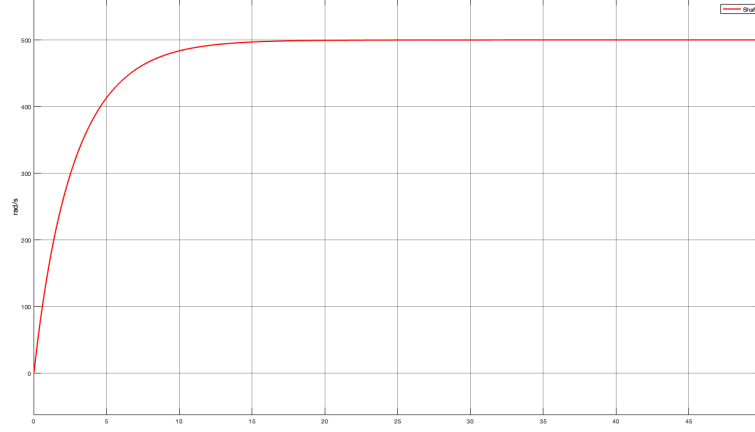


Figure 3.22: Test motor speed command in concept model

Here, test motor speed command's steady-state value coming out from motor shaft is about 500 rad/s as desired since user set 500 rad/s as speed value to pilot bench motor.

- *Bench and test motor battery currents*

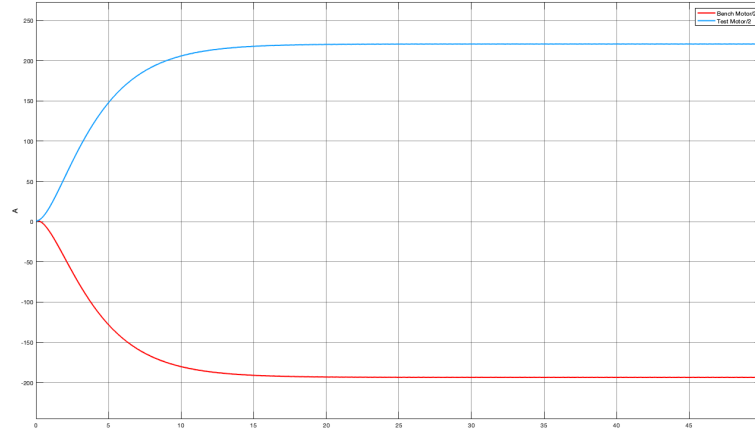


Figure 3.23: Bench and test motor battery currents in concept model

Here, bench motor battery current's steady-state value is about -193.485 A (demanded current) while test motor battery current's steady-state value is about 220.787 A (draw current).



- *Generated motor torques*

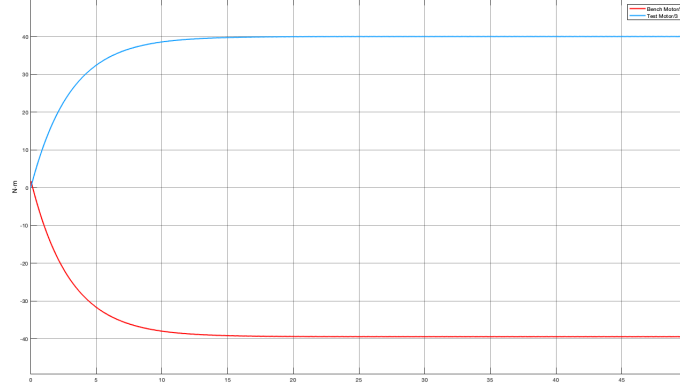


Figure 3.24: Bench and test motor generated torques in concept model

Here, generated bench motor torque's steady-state value is about -39.492 Nm (negative because bench motor is braked) while generated test motor torque's steady-state value is about 39.992 Nm. Test motor generated torque is less than torque values set by user (40 Nm) due to mechanical losses inside motor, while bench motor's generated torque is less than test motor's generated torque since part of torque is used to put in rotation motor shaft interconnected to both motors.

- *Bench motor mechanical and electrical parameters*

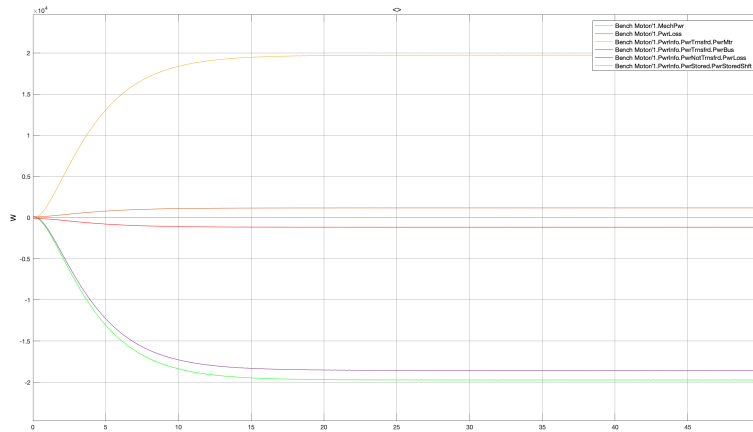


Figure 3.25: Bench motor mechanical and electrical parameters in concept model

Here, bench motor's mechanical and electrical parameter trends such as *mechanical power*, *internal invert* and *motor power loss*, *mechanical*

*power, electrical power, motor power loss and motor power stored* can be seen.

- *Test motor mechanical and electrical parameters*

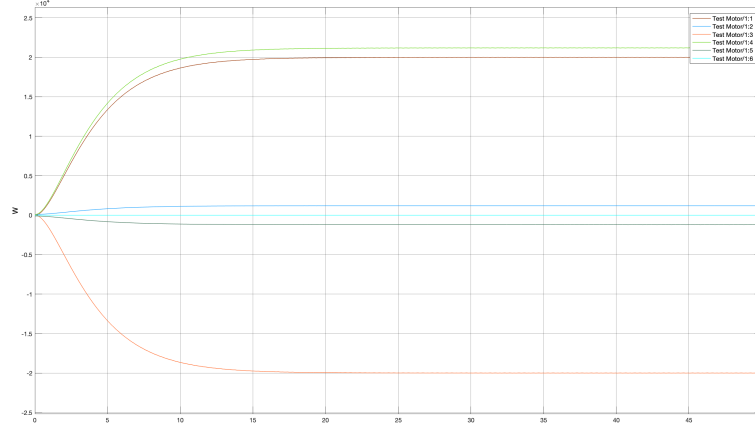


Figure 3.26: Test motor mechanical and electrical parameters in concept model

Here, test motor's mechanical and electrical parameter trends such as *mechanical power, internal invert and motor power loss, mechanical power, electrical power, motor power loss and motor power stored* can be seen.

# Chapter 4

## System design phase: technical model

This is the first phase performed by designer during which user requirements must be carefully analyzed in order to find the best solution to implement final physical system that is able to satisfy them. Here, designer must consider technical details that are neglected by user during concept model's production, described in previous chapter, and found the most efficient solution taking also into account costs to be addressed by company. But in doing so, designer neglects details about target hardware to be used. So, during this phase, the designer, starting from concept model given by user, must build-up a technical model that is able to satisfy all user's requirements considering all technical details. For this reason, during this phase often *Technical model term* is used and *Modular Technical Model tool* is considered to realize it.

### 4.1 Preliminary technical model

In this thesis work, before to consider Modular-Technical-Model, a preliminary technical model has been created to understand whether what is asked by the user is feasible or not and whether all his requirements can be satisfied.

#### 4.1.1 Some preliminary modification of concept model

First of all, concept model has been modified in such a way that what is asked by user is clearly visible. For this reason, modifications shown in figure 4.1 have been done in concept model provided by user.

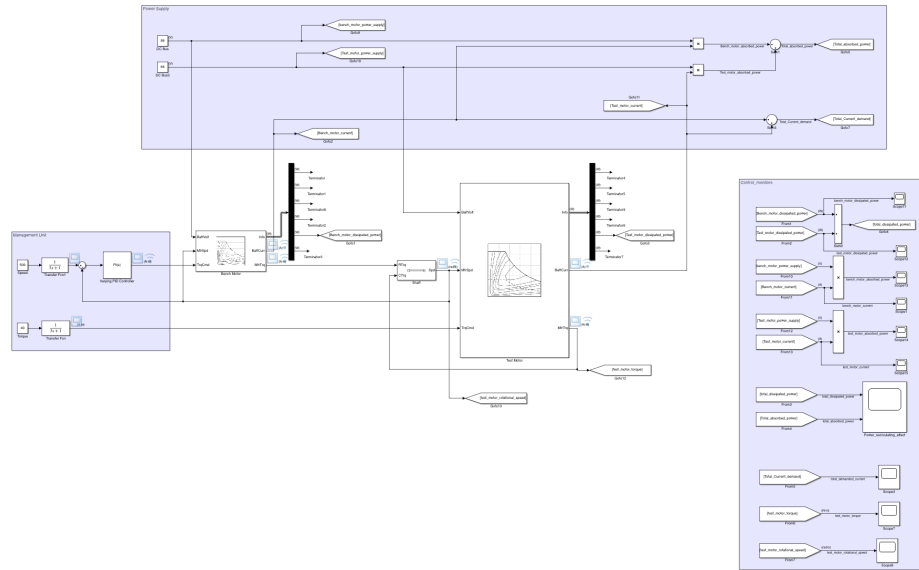


Figure 4.1: Preliminary technical model

Here, part highlighted in violet on the right part of model (figure 4.2) is added and named *Control Monitors*.

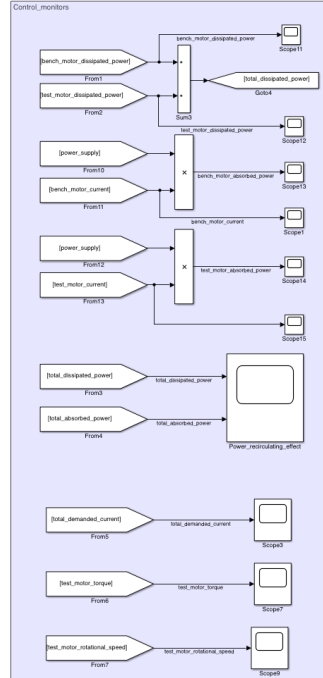


Figure 4.2: Preliminary technical model's control monitors

It contains all scopes that allow to take under control all relevant bench motor parameters such as *dissipated power, current demanded or draw and absorbed power*; all relevant test motor parameters such as *dissipated power, demanded or drawn current, absorbed power, rotational speed and torque*. By means the biggest scope, reported in this new part, *power recirculating effect* can be verified since it shows total absorbed and total dissipated power trends so comparison between them is done. Moreover, an additional scope shows total demanded current.

Then, a new power supply with respect to concept model has been implemented. This because, instead of considering the same electric motor both in bench and test motor case, like happens in concept model, two different electric motor have been considered which are characterized by an own voltage power supply value. This time in fact, *Sitem motor* has been considered for bench motor which is provided by its own inverter requiring a 96 V DC voltage supply while *SME motor* has been considered for test motor that, like Sitem motor, is provided by its own inverter requiring a 66 V DC voltage supply.

This, for sure, implies that also MATLAB script, provided by user, needs to be upgraded defining all SME motor's mechanical and electrical data. In figure 4.3 upgraded MATLAB scrip is shown.

```

5 ~ close all
6
7 ~%% Bench motor
8 ~% Electrical Torque - Nominal & Maximal
9 ~ w_n_bench= 4000 /60*2*pi;
10 ~ T_m_n_bench = 28;
11 ~ eta_n_bench = 0.91 *100; % @ w_n & T_m_n
12 ~ P_m_n_bench = w_n_bench * T_m_n_bench
13 ~ T_m_max_bench = 55;
14 ~ P_m_max_bench = w_n_bench * T_m_max_bench *1.1
15 ~ w_max_bench=w_n_bench*1.1;
16 ~ tau_m_bench= 0.02;
17 ~% Electrical Losses - Single Efficiency
18 ~ PL_iron_bench= 0; %IPM
19 ~ PL_ein_bench= 100;
20 ~ PL_copper_n_bench= (1-eta_n_bench/100)*P_m_n_bench -PL_ein_bench -PL_iron_bench
21 ~ k_T_bench= PL_copper_n_bench /T_m_n_bench^2
22 ~% Mechanical
23 ~ J_m_bench= .005e-3;
24 ~ beta_m_bench= 1e-5;
25
26 ~ i_max_bench=T_m_max_bench/k_T_bench;
27
28 ~%% Test motor
29
30 ~%Electrical Torque - Nominal & Maximal
31 ~ w_n_test = 5000 /60*2*pi;
32 ~ P_m_n_test=10500;
33 ~ T_m_n_test= P_m_n_test/w_n_test;
34 ~ V_n_test=66; %V
35 ~ I_n_test=115; %A
36 ~ cos_phi_test=0.86;
37 ~ eta_n_test=(P_m_n_test/(sqrt(3)*V_n_test*I_n_test*cos_phi_test))*100;
38 ~ w_max_test=7500/60*2*pi;
39 ~ T_m_max_test = 83;
40 ~ P_m_max_test = w_max_test * T_m_max_test
41 ~ tau_m_test = 0.02;
42
43 ~%Electrical Losses - Single Efficiency
44 ~ PL_iron_test = 0;
45 ~ PL_ein_test = 100;
46 ~ PL_copper_n_test= (1-eta_n_test/100)*P_m_n_test -PL_ein_test -PL_iron_test
47 ~ k_T_test= PL_copper_n_test /T_m_n_test^2
48
49 ~ i_max_test=T_m_max_test/k_T_test;
50
    
```

Figure 4.3: Preliminary technical model's MATLAB script

For obtaining desired parameter trends, MATLAB script shown in previous figure must be run before to run Simulink model. Once MATLAB script is run, a simulation time must be defined in Simulink model in order to decide how many time simulation must last. Below results are listed.

- *Bench motor dissipated power  $P_{d_{bm}}$*

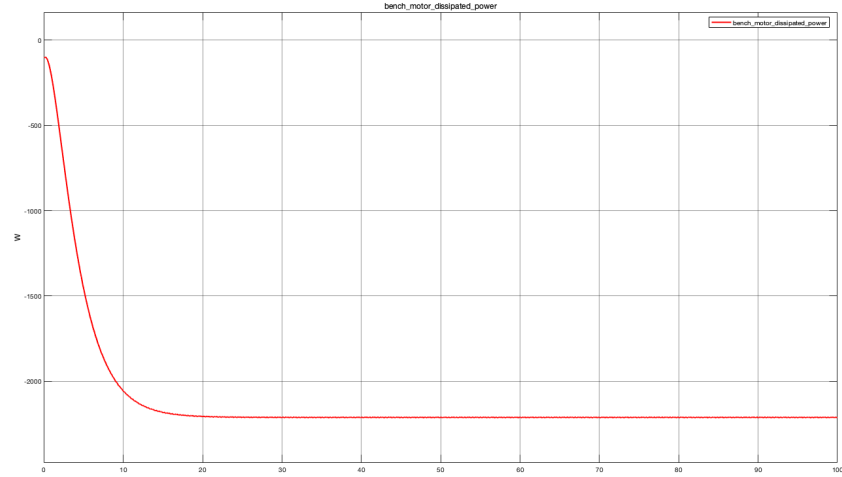


Figure 4.4:  $P_{d_{bm}}$  with both motors set in single efficiency measurement

This variable, at steady-state, is approximately -2.211 kW. Its value is negative since it represents a lost power.

- *Bench motor absorbed power  $P_{a_{bm}}$*

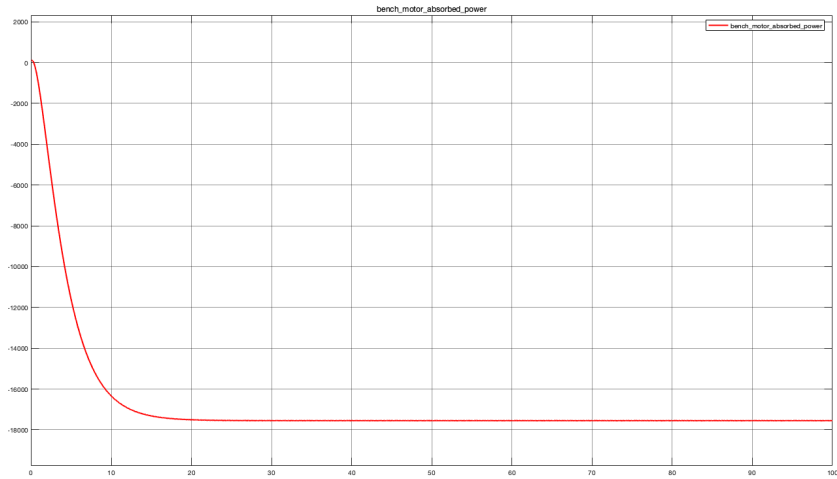


Figure 4.5:  $P_{a_{bm}}$  with both motors set in single efficiency measurement

This variable, at steady-state, is approximately -17.55 kW. It assumes a negative value since it represents provided power.

- *Bench motor demanded or drawn current  $I_{bm}$*

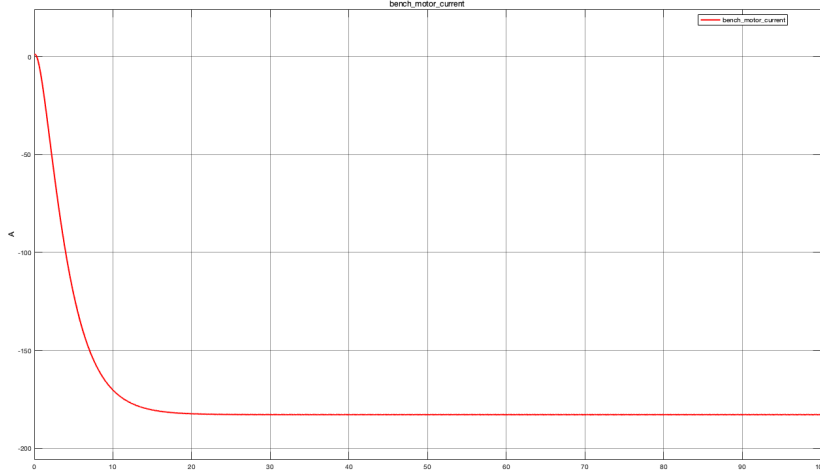


Figure 4.6:  $I_{bm}$  with both motors set in single efficiency measurement

At steady-state, this variable is approximately -182.785 A. Its value is negative since it represents a demanded current.

- *Test motor dissipated power  $P_{d_{tm}}$*

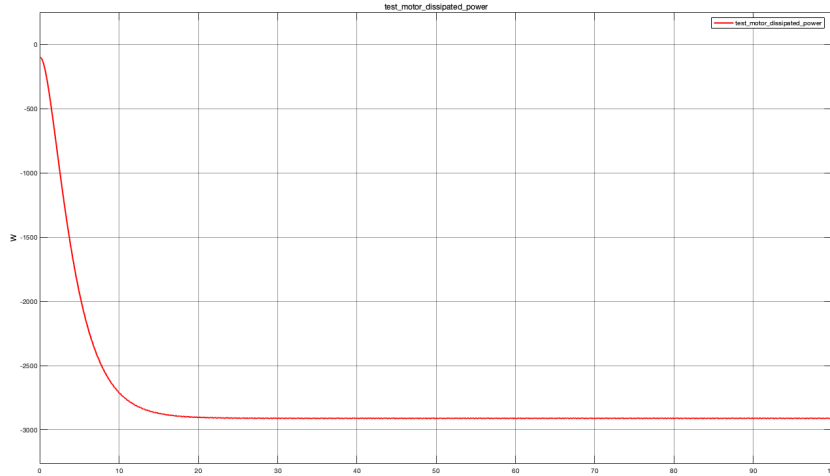


Figure 4.7:  $P_{d_{tm}}$  with both motors set in single efficiency measurement

This variable, at steady-state, is approximately -2.911 kW. Its value is negative since it represents a lost power.

- *Test motor absorbed power  $P_{atm}$*

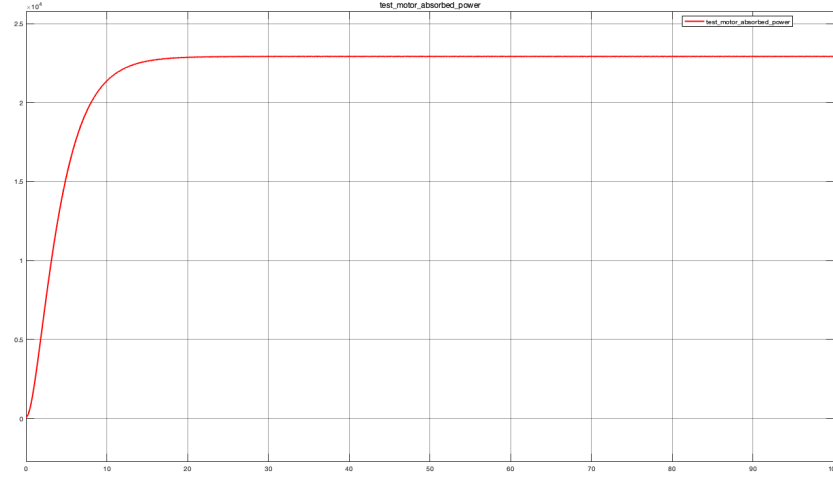


Figure 4.8:  $P_{atm}$  with both motors set in single efficiency measurement

At steady-state it results to be approximately 22.92 kW. In this case, unlike bench motor absorbed power, a positive value is reached since it represents an absorbed power.

- *Test motor demanded or drawn current  $I_{tm}$*

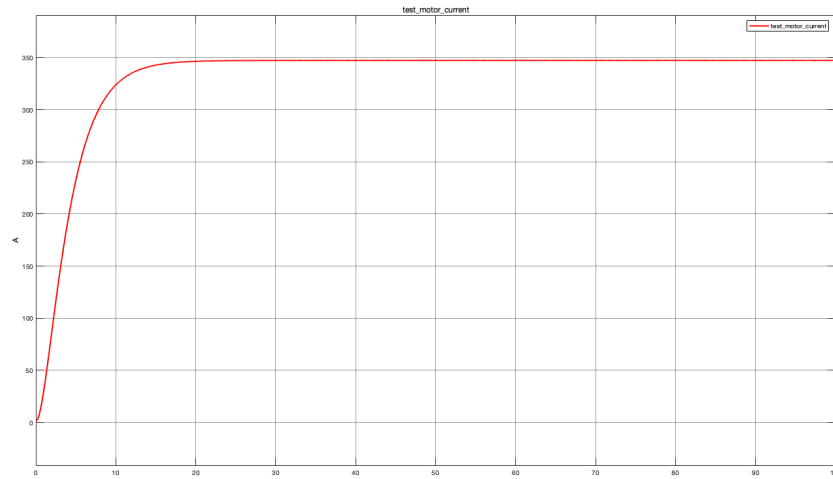


Figure 4.9:  $I_{tm}$  with both motors set in single efficiency measurement

At steady-state, it is approximately since it is a drawn 347.262 A. In this case current value assumes a positive value current.



- *Motor shaft rotational speed  $\omega$*

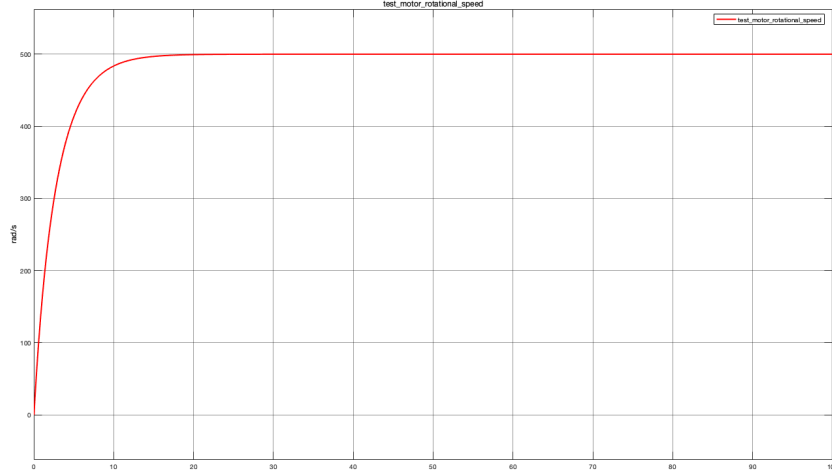


Figure 4.10:  $\omega$  with both motors set in single efficiency measurement

At steady-state, it results to be approximately 500 rad/s as expected since at beginning a rotational speed command of 500 rad/s has been set for piloting bench motor which is responsible to impose rotational speed to test motor.

- *Test motor torque  $T_{tm}$*

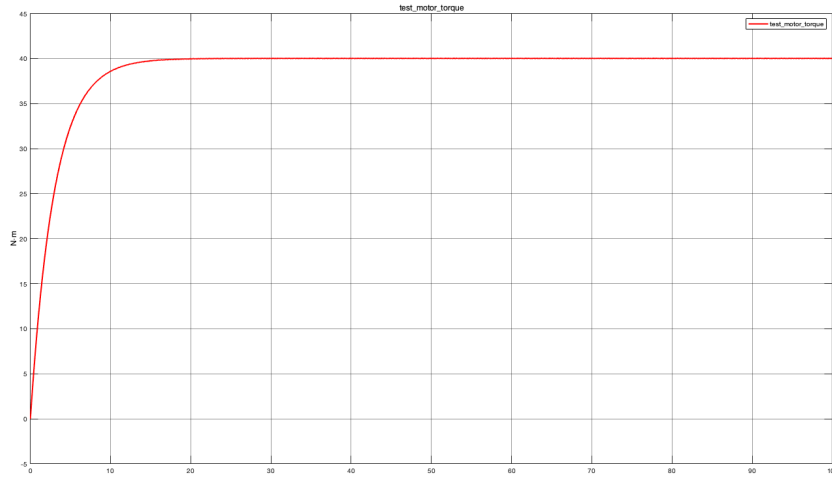


Figure 4.11:  $T_{tm}$  with both motors set in single efficiency measurement

At steady-state, it results to be approximately 40 Nm as expected since at beginning a torque command of 40 Nm has been set to pilot test motor.

- *Power recirculating effect*

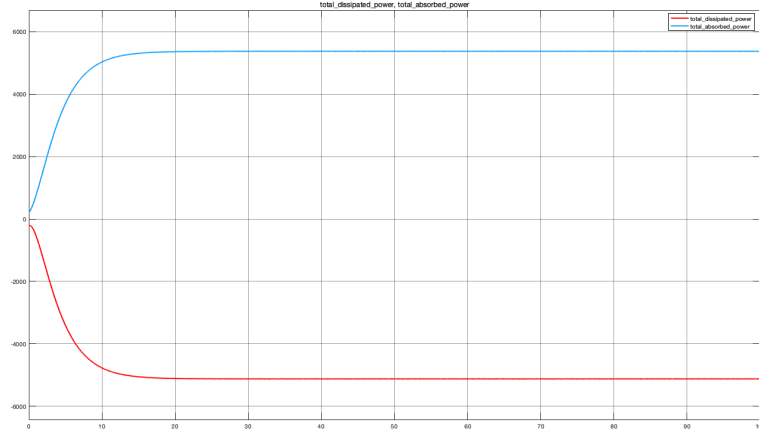


Figure 4.12:  $P_a$ ,  $P_d$  with both motors set in single efficiency measurement

Here, absorbed and dissipated power are not perfectly equal to each other as expected for verifying power recirculating effect. This does not mean that effect is not verified since difference between two powers is due to motor shaft presence that, of course, requires some power that must be added to absorbed power. Indeed, absorbed power at steady-state is more or less equal to 5.372 kW while dissipated power is more or less equal to -5.122 kW.

- *Total demanded current ( $I$ )*

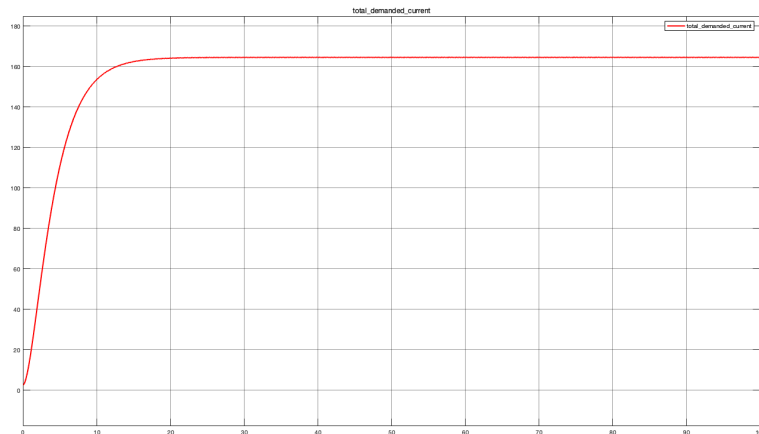


Figure 4.13:  $I$  with both motors set in single efficiency measurement

Here, the sum between bench and test motor current ( $I_{bm}$  and  $I_{tm}$ ) is shown. At steady-state it assumes a value equal to 164.5A.

#### 4.1.2 Tabulated bench motor

After checking that modifications above described work properly, preliminary technical model can be modified again. This time modifications concern *bench mapped motor Simulink block* that must be tabulated. This means that bench motor must be parametrized considering its efficiency and/or dissipated power 3D maps in order to obtain interesting parameter values as close as possible to those that can be obtained in practice. This because, tabulating bench mapped motor Simulink block, it is possible to obtain a more accurate bench motor behavior during simulations. So, this modification is very important to understand technical details to be considered for implementing final physical system.

First of all, only bench mapped motor Simulink block has been tabulated while test mapped motor Simulink block has been set in single efficiency measurements. To tabulate bench mapped motor Simulink block "ICE Test Bench Upgrading For Hybrid and Electrical Powertrain" thesis is considered where all needed data are reported. Below list of all needed variables is reported.

- *Vector of rotational speeds  $w\_t$  [rad/s]*
- *Vector of maximum torque values  $T\_t$  [Nm]*
- *Torque control time constant  $Tc$  [s]*
- *Vector of speed for tabulated losses  $w\_eff\_bp$  [rad/s]*
- *Vector of torques for tabulated losses  $T\_eff\_bp$  [Nm]*
- *Corresponding efficiency  $eff\_table$  [%]*
- *Corresponding losses  $losses\_table$  [W]*

To find these parameters, bench mapped motor Simulink block's block parameters window has to be opened by clicking two times on block. Once, window is opened two possible subsections can be selected: *Electrical Torque and Electrical Losses*.

Considering Electrical Torque section, "*Tabulated torque-speed envelope*" from drop-down menu can be selected in order to define parametrization way for bench mapped motor Simulink block. Doing this, the three above listed parameters must be defined.

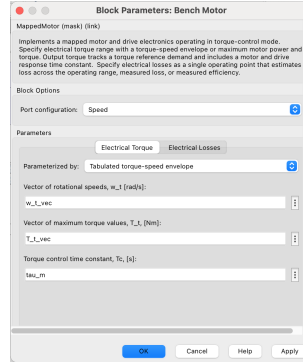


Figure 4.14: Electrical torque section of bench motor block parameters' window

- *Vector of rotational speeds  $w_t$  [rad/s]*

It is a vector to be defined in MATLAB script considering rotational speed values that can be used for permissible steady-state operation. Values reported in this vector have to be defined in rad/s.

- *Vector of maximum torque values  $T_t$  [Nm]*

It is a vector to be defined in MATLAB script considering maximum torque values that can be used for permissible steady-state operation. Here, torque value must be defined in Nm.

- *Torque control time constant  $T_c$  [s]*

It represents time constant defining seconds used by motor driver for tracking a torque demand.

To define  $w_t$  and  $T_t$  vectors, maximum torque and maximum power's graph must be considered from which rotational speed and torque values reported in table 4.1 can be read.

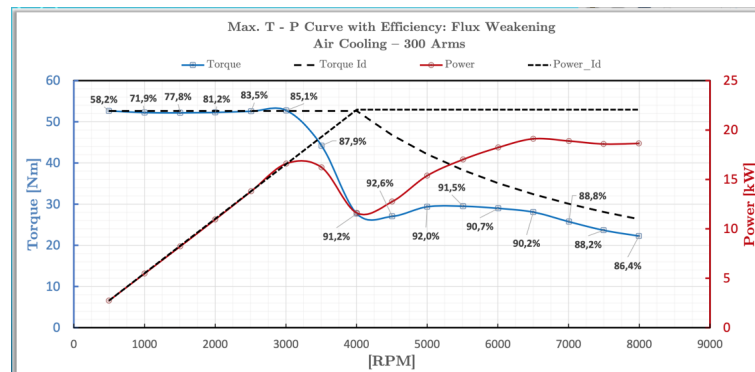


Figure 4.15: Bench motor maximum torque and power curves

Speed [ $rad/s$ ]	Torque [ $Nm$ ]
52.4	52.5
104.7	52.5
157.1	52.5
209.4	52.5
261.8	52.5
314.2	52.5
366.5	52.5
418.9	52.5
471.2	47
523.6	42
575.9	38
628.3	35

Table 4.1: Bench motor maximum torque values reachable at steady-state

At this point two vectors can be defined on MATLAB script, shown in figure 4.16, where  $w\_t$  vector is defined as  $w\_t\_vec\_bench$  while  $T\_t$  vector is defined as  $T\_t\_vec\_bench$ . In addition among all other bench motor parameters, *torque control time constant*  $T_c$  is defined as  $\tau_{m\_bench}$ .

```

1 | %Technical_model
2 | clc
3 | clear all
4 | close all
5 |
6 | %% Bench_motor
7 | % Electrical Torque - Nominal & Maximal
8 | w_n_bench= 4000 /60*2*pi;
9 | T_m_n_bench = 28;
10 | eta_n_bench = 0.91 *100; % @ w_n & T_m_n
11 | P_m_n_bench = w_n_bench * T_m_n_bench
12 | T_m_max_bench = 55;
13 | P_m_max_bench = w_n_bench * T_m_max_bench *1.1
14 | w_max_bench=w_n_bench*1.1;
15 | tau_m_bench= 0.02;
16 | % Electrical Losses - Single Efficiency
17 | P_l_iron_bench= 0; %IPM
18 | P_l_ein_bench= 100;
19 | P_l_copper_n_bench= (1-eta_n_bench/100)*P_m_n_bench -P_l_ein_bench -P_l_iron_bench
20 | k_T_bench= P_l_copper_n_bench /T_m_n_bench^2
21 | % Mechanical
22 | J_m_bench= .005e-3;
23 | beta_m_bench= 1e-5;
24 |
25 | i_max_bench=T_m_max_bench/k_T_bench;
26 |
27 | %% Tabulated_bench_motor
28 |
29 | %Electrical torque section
30 | w_t_vec_bench=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593 366.5191 418.8790 471.2389 523.5988 575.9587 628.3185]; %rad/s
31 | T_t_vec_bench=[52.5 52.5 52.5 52.5 52.5 52.5 52.5 47 42 38 35]; %Nm
32 |

```

Figure 4.16: Bench motor electrical torque section's MATLAB script

Instead considering *Electrical Losses section* of block parameters window, two different ways for parameterizing losses can be chosen: *Tabulated loss data* and *Tabulated efficiency data*. The first way is used when bench motor's dissipated power 3D map is considered to parametrize mapped motor Simulink block while the second way is used when bench motor's efficiency 3D map is considered.

Selecting *Tabulated loss data* way, only three of previous listed parameters must be defined in MATLAB script.

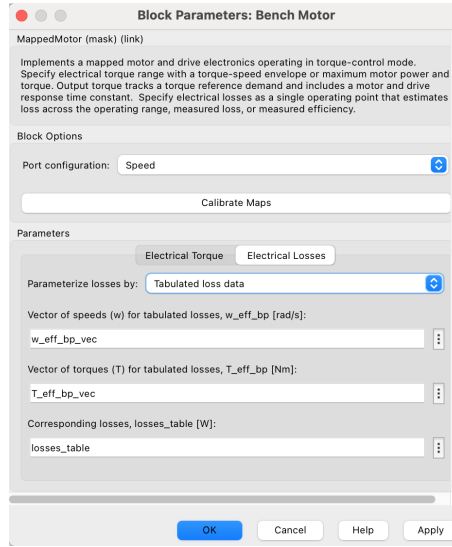


Figure 4.17: Bench motor electrical losses section in tabulated loss data way

- *Vector of speed for tabulated losses  $w\_eff\_bp$  [rad/s]*

It is a vector to be defined in MATLAB script considering rotational speed values that must be taken into care as rotational speed breakpoints for lookup table used for calculating losses. Here, rotational speed values must be defined in rad/s.

- *Vector of torques for tabulated losses  $T\_eff\_bp$  [Nm]*

It is a vector to be defined in MATLAB script considering torque values that must be taken into care as torque breakpoints for lookup table used for determining losses. Here, torque values must be defined in Nm.

- *Corresponding losses  $losses\_table$  [W]*

It is a matrix containing electrical losses values as function of rotational speed values defined in  $w\_eff\_bp$  vector and torque values defined in  $T\_eff\_bp$  vector. This matrix must be constituted of a number of raw equal to  $w\_eff\_bp$  vector elements and a number of columns equal to  $T\_eff\_bp$  vector elements. Electrical losses values in this matrix must be defined in W.

For defining these three variables, dissipated power map shown in figure 4.18 must be considered.

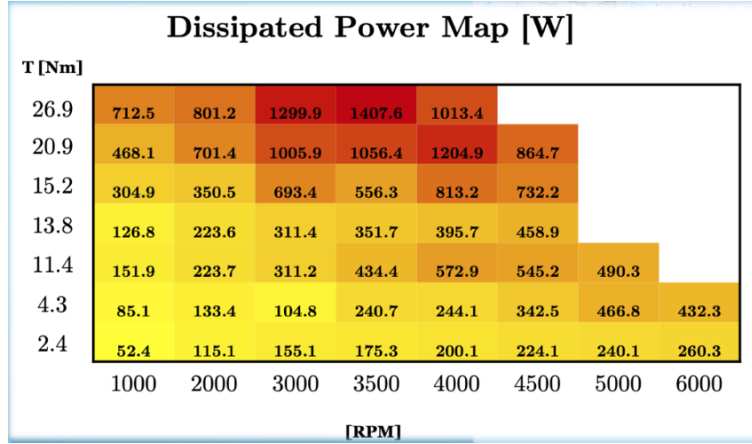


Figure 4.18: Bench motor's dissipated power map

Considering this map, MATLAB script must be upgraded as below shown where  $w\_eff\_bp$  vector is defined as  $w\_eff\_bp\_vec\_bench$ ,  $T\_eff\_bp$  vector is defined as  $T\_eff\_bp\_vec\_bench$  and  $losses\_table$  is defined as  $losses\_table\_bench$ .

```

28 %% Tabulated_bench_motor
29
30 %Electrical torque_section
31 w_t_vec_bench=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593 366.5191 418.8790 471.2389 523.5988 575.9587 628.3185]; %rad/s
32 T_t_vec_bench=[52.5 52.5 52.5 52.5 52.5 52.5 52.5 47 42 38 35]; %Nm
33
34 %Electrical losses_section
35 %%Tabulated_loss_data
36 w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6 628.32];
37 T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 20.9 26.9];
38
39 b=2000;
40 losses_table_bench=[52.4 85.1 151.9 126.8 304.9 468.1 712.5;
41 115.1 133.4 223.7 223.6 350.5 701.4 801.2;
42 155.1 104.8 311.2 311.4 693.4 1005.9 1299.9;
43 175.3 240.7 434.4 351.7 556.3 1056.4 1407.6;
44 200.1 244.1 572.9 395.7 813.2 1204.9 1013.4;
45 224.1 342.5 545.2 458.9 732.2 864.7 b;
46 240.1 466.8 490.3 b b b b;
47 260.3 432.3 b b b b b];
48
49 %%Tabulated_efficiency_data
50 d=85;
51 efficiency_table_bench=[82.8 85.8 88 91.9 84.5 82.6 84.5;
52 79.6 88.4 90.5 92.2 90.2 86.2 89.2;
53 80.2 93.1 90.9 93.3 87.3 86.7 91.2;
54 80.3 87.4 89.3 93.5 90.9 87.8 91.6;
55 80.2 80.8 88.1 93.5 88.7 88.1 91.9;
56 78.9 88.1 89.2 93.2 90.3 92.4 d;
57 79.5 87.1 91.8 d d d d;
58 78.7 85.2 d d d d d];
59
    
```

Figure 4.19: MATLAB script with bench motor's dissipated power map

Selecting *Tabulated efficiency data* way, the only thing that changes with respect to previous case is related to the fact that instead of considering bench motor's power losses matrix, efficiency matrix is considered. In this case, block parameters' window is the one shown in figure 4.20.

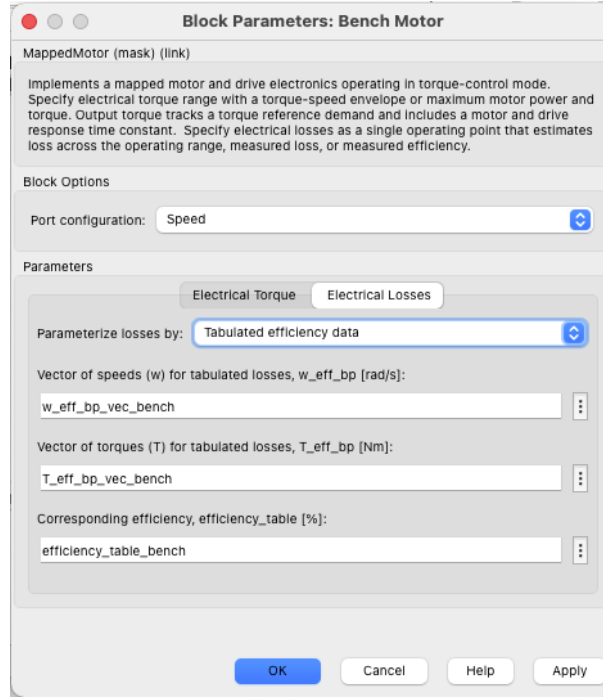


Figure 4.20: Bench motor's electrical losses section in tabulated efficiency data way

Here, the same variables described in previous case appears ( $w\_eff\_bp$  and  $T\_eff\_bp$ ) but instead of having `losses_table`, *efficiency table* must be defined.

- *Corresponding efficiency ( $efficiency\_table$ ) [%]*

It is a matrix containing efficiency values as function of rotational speed values defined in  $w\_eff\_bp$  vector and torque values defined in  $T\_eff\_bp$  vector. This matrix must be constituted of a number of raw equal to number of  $w\_eff\_bp$  vector elements and a number of columns equal to  $T\_eff\_bp$  vector elements. Efficiency values in this matrix must be defined in %.

$T\_eff\_bp$  and  $w\_eff\_bp$  vectors are defined as described in previous case while efficiency matrix is defined considering bench motor's efficiency map (figure 4.21). Taking into care this map, MATLAB script must be upgraded defining *efficiency\_table* variable. This is done as shown in figure 4.22 where that variable is defined as *efficiency\_table\_bench*.



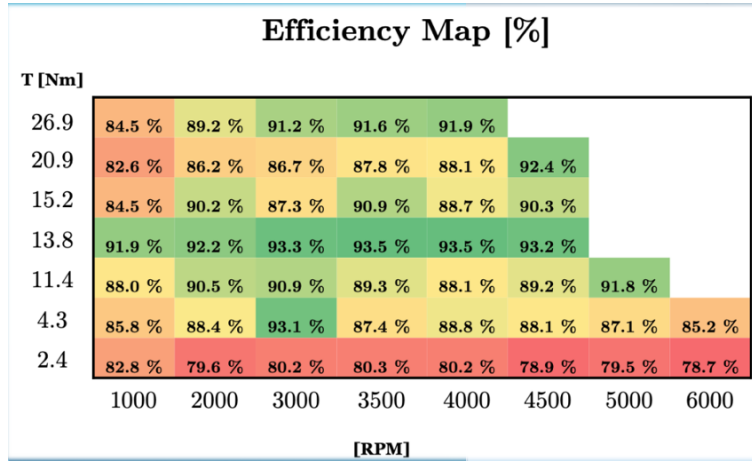


Figure 4.21: Bench motor's efficiency map

```

3 - clc
4 - clear all
5 - close all
6
7 %% Bench motor
8 % Electrical Torque - Nominal & Maximal
9 - w_n_bench= 4000 /60*2*pi;
10 - T_m_n_bench = 28;
11 - eta_n_bench = 0.91 *100; % w_n & T_m_n
12 - P_m_n_bench = w_n_bench * T_m_n_bench
13 - T_m_max_bench = 55;
14 - P_m_max_bench = w_n_bench * T_m_max_bench *1.1
15 - w_max_bench=w_n_bench*1.1;
16 - tau_m_bench= 0.02;
17 % Electrical Losses - Single Efficiency
18 - PL_iron_bench= 0; %IPM
19 - PL_ein_bench= 100;
20 - PL_copper_n_bench= (1-eta_n_bench/100)*P_m_n_bench -PL_ein_bench -P
21 - k_T_bench= PL_copper_n_bench /T_m_n_bench^2
22 % Mechanical
23 - J_m_bench= .005e-3;
24 - beta_m_bench= 1e-5;
25
26 - i_max_bench=T_m_max_bench/k_T_bench;
27
28 %% Tabulated_bench_motor
29
30 %Electrical torque_section
31 - w_t_vec_bench=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593
32 - T_t_vec_bench=[52.5 52.5 52.5 52.5 52.5 52.5 47 42 38 35]
33
34 %Electrical_losses_section
35 %%Tabulated_loss_data
36 - w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6
37 - T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 20.9 26.9];
38
39 - b=2000;
40 - losses_table_bench=[52.4 85.1 151.9 126.8 304.9 468.1 712.5;
41 - 115.1 133.4 223.7 223.6 350.5 701.4 801.2;
42 - 155.1 104.8 311.2 311.4 693.4 1005.9 1299.9;
43 - 175.3 240.7 434.4 351.7 556.3 1056.4 1407.6;
44 - 200.1 244.1 572.9 395.7 813.2 1204.9 1013.4;
45 - 224.1 342.5 545.2 458.9 732.2 864.7 b;
46
47 260.3 432.3 b b b b b];
48
49 %%Tabulated_efficiency_data
50 - d=85;
51 - efficiency_table_bench=[82.8 85.8 88 91.9 84.5 82.6 84.5;
52 - 79.6 88.4 90.5 92.2 90.2 86.2 89.2;
53 - 80.2 93.1 90.9 93.3 87.3 86.7 91.2;
54 - 80.3 87.4 89.3 93.5 90.9 87.8 91.6;
55 - 80.2 88.8 88.1 93.5 88.7 88.1 91.9;
56 - 78.9 88.1 89.2 93.2 90.3 92.4 d;
57 - 79.5 87.1 91.8 d d d d;
58 - 78.7 85.2 d d d d d];
59
60 %% Test motor
61
62 %Electrical Torque - Nominal & Maximal
63 - w_n_test = 5000 /60*2*pi;
64 - P_m_n_test=10500;
65 - T_m_n_test = P_m_n_test/w_n_test;
66 - V_n_test=66; %V
67 - I_n_test=115; %A
68 - cos_phi_test=0.86;
69 - eta_n_test =(P_m_n_test/(sqrt(3)*V_n_test*I_n_test*cos_phi_test))*1
70 - w_max_test=7500/60*2*pi;
71 - T_m_max_test = 63;
72 - P_m_max_test = w_max_test * T_m_max_test
73 - tau_m_test = 0.02;
74
75 %Electrical Losses - Single Efficiency
76 - PL_iron_test = 0;
77 - PL_ein_test = 100;
78 - PL_copper_n_test= (1-eta_n_test/100)*P_m_n_test -PL_ein_test -PL_i
79 - k_T_test= PL_copper_n_test /T_m_n_test^2
80
81 - i_max_test=T_m_max_test/k_T_test;
82
83 %%Tabulated_test_motor
84
85 %Electrical torque_section
86 - w_t_vec_test=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593
87 - T_t_vec_test=[83 82.5 82 81.5 80.5 80.5 77.5 68 56 46.5 39.5 34.5]
88 - tau_m_test = 0.02;
89
    
```

Figure 4.22: MATLAB script with bench motor's efficiency map

Here, on the right bottom part also SME motor's mechanical and electrical parameters are defined. Once it is run, simulation time must be set in Simulink model and bench motor's parameterization way both for electrical torque and electrical losses section must be selected as before described. Selecting *Tabulated loss data* as losses' parametrization way in *Electrical Losses section* of bench motor parameters' window, results below shown have been obtained.

- *Bench motor dissipated power ( $P_{d_{bm}}$ )*

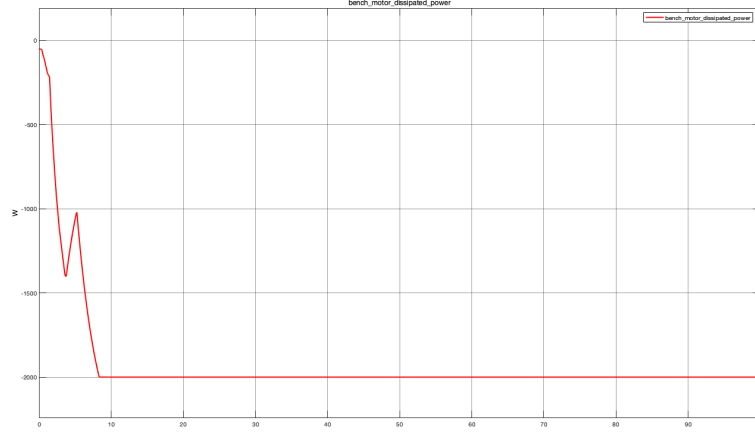


Figure 4.23:  $P_{d_{bm}}$  with bench motor set in tabulated loss data

At steady-state, it is approximately -2 kW. Its value has negative sign since it represents lost power and it assumes smaller value with respect to the case where both mapped motor Simulink blocks are set in single efficiency measurements (-2.211 kW as shown in figure 4.4)

- *Bench motor absorbed power ( $P_{a_{bm}}$ )*

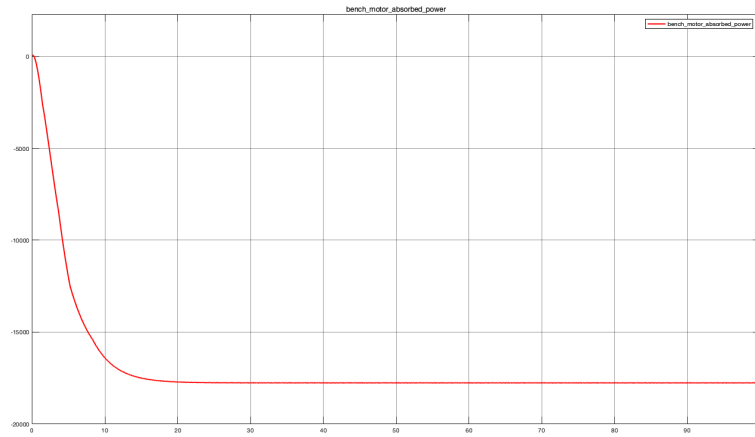


Figure 4.24:  $P_{a_{bm}}$  with bench motor set in tabulated loss data

At steady-state, it results to be approximately -17.75 kW. It assumes exactly the same value obtained in case where both mapped motor Simulink blocks are set in single efficiency measurements and its negative sign means that bench motor provides power.

- *Bench motor demanded or drawn current ( $I_{bm}$ )*

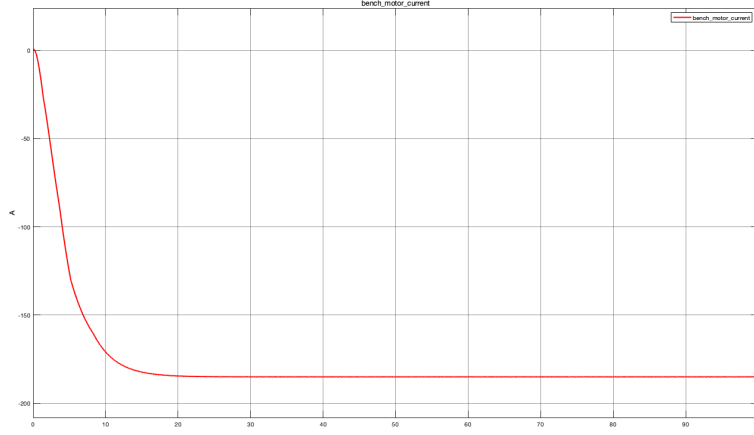


Figure 4.25:  $I_{bm}$  with bench motor set in tabulated loss data

At steady-state, this variable is approximately -184.941 A. Its value is negative since it represents a demanded current and it is bigger than the value obtained in case where both mapped motor Simulink blocks are set in single efficiency measurements (-182.785 A as shown in figure 4.6).

- *Test motor dissipated power ( $P_{d_{tm}}$ )*

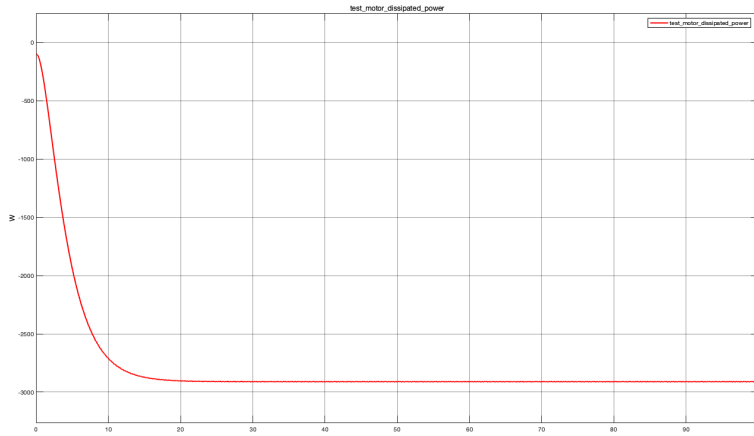


Figure 4.26:  $P_{d_{tm}}$  with bench motor set in tabulated loss data

At steady-state, it results to be approximately -2.901 kW. Its value is negative since it represents lost power and it assumes smaller value than that obtained in case where both mapped motor Simulink blocks are set in single efficiency measurements (-2.911 kW as shown in figure 4.7)

- *Test motor absorbed power ( $P_{atm}$ )*

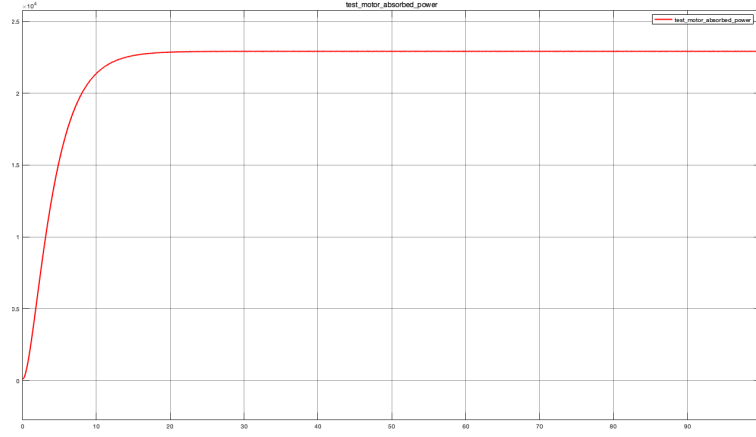


Figure 4.27:  $P_{atm}$  with bench motor set in tabulated loss data

At steady-state, it assumes exactly the same value measured in case where both mapped motor Simulink blocks are set in single efficiency measurements, 22.92 kW. In this case, unlike bench motor absorbed power, it assumes a positive value since it represents absorbed power instead of provided power.

- *Test motor demanded or drawn current ( $I_{tm}$ )*

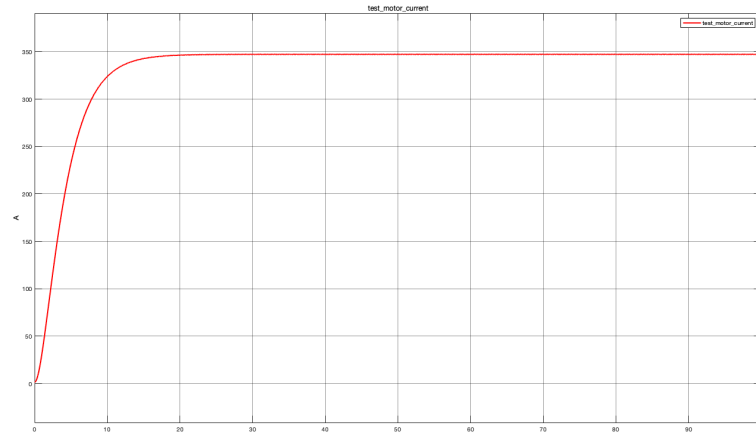


Figure 4.28:  $I_{tm}$  with bench motor set in tabulated loss data

At steady-state, this variable is approximately 347.178A. Here, unlike bench motor current's case, current value assumes a positive value since it represents drawn current and it is a bit smaller than the value obtained in case where both mapped motor Simulink blocks are in single efficiency measurements (347.262 A as shown in figure 4.9)

- *Motor shaft rotational speed  $\omega$*

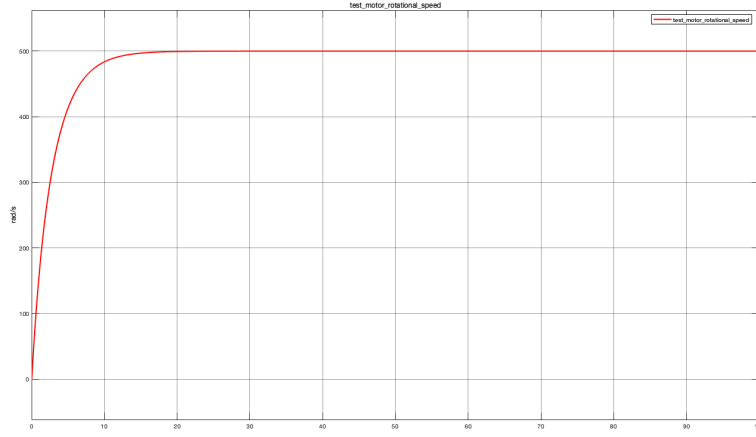


Figure 4.29:  $\omega$  with bench motor set in tabulated loss data

Here, like the case in which both mapped motor Simulink blocks are set in single efficiency measurements, at steady state rotational speed value assumes exactly the same value of rotational speed command, set at beginning, that is chosen to pilot bench motor (500 rad/s), responsible to brake and impose rotational speed to test motor connected to it

- *Test motor torque  $T_{tm}$*

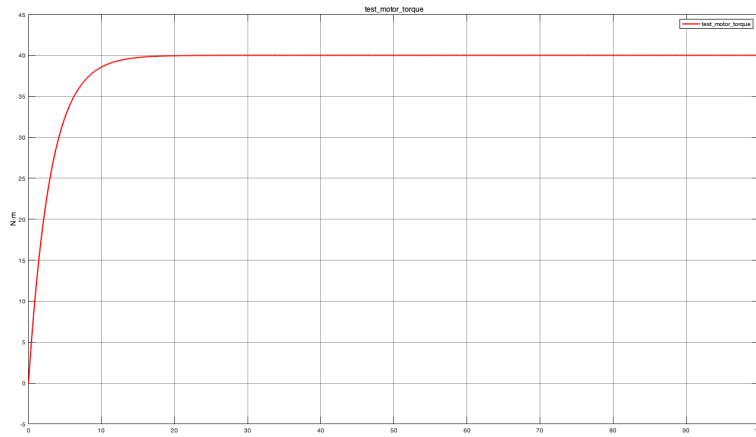


Figure 4.30:  $T_{tm}$  with bench motor set in tabulated loss data

Also in this case, like the case where both mapped motor Simulink blocks are set in single efficiency measurements, at steady state torque value measured at the output of test motor assumes exactly the same value of torque command set at beginning to pilot test motor, 40 Nm.

- *Power recirculating effect*

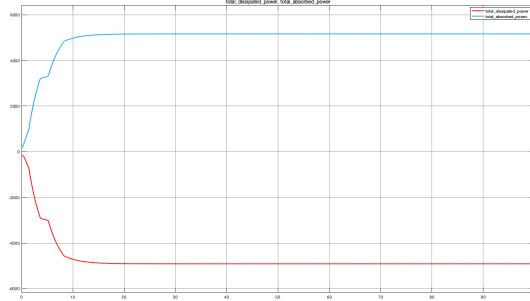


Figure 4.31:  $P_a$ ,  $P_d$  with bench motor set in tabulated loss data

Here, like the case where both mapped motor Simulink blocks are set in single efficiency measurement, absorbed and dissipated power are not perfectly equal to each other as expected. This, due to motor shaft presence that, of course, requires some power that must be added to motors' absorbed power. Indeed, at steady state absorbed power results to be about 5.159 kW while dissipated power is about -4.909 kW. In this case, both powers assume smaller value with respect to case where both mapped motor Simulink blocks are set in single efficiency measurements (5.371 kW and -5.121 kW as shown in figure 4.12).

- *Total demanded current ( $I$ )*

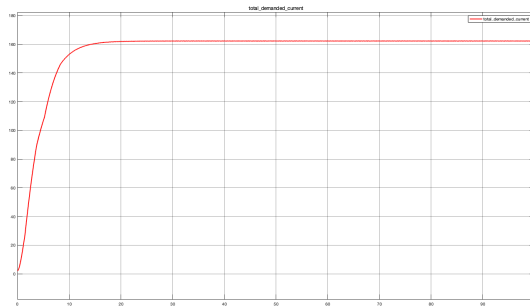


Figure 4.32:  $I$  with bench motor set in tabulated loss data

This graph shows the sum between bench and test motor's current from which its steady- state value can be read (about 162.2 A). Comparing this value with that obtained in case where both mapped motor Simulink blocks are set in single efficiency measurement a difference can be observed since this time total demanded current assumes smaller value with respect to previous case (about 164.5 A).

Instead, setting bench mapped motor Simulink block considering *Tabulated efficiency data* as losses parameterization way in parameters window's Electric Losses section and performing Simulink simulation choosing the same simulation time considered for performing simulation in case where only bench mapped motor Simulink blocks is set in tabulated losses (100 s), results shown in following figures have been obtained that can be compared to those obtained in previous case.

- *Bench motor dissipated power ( $P_{d_{bm}}$ )*

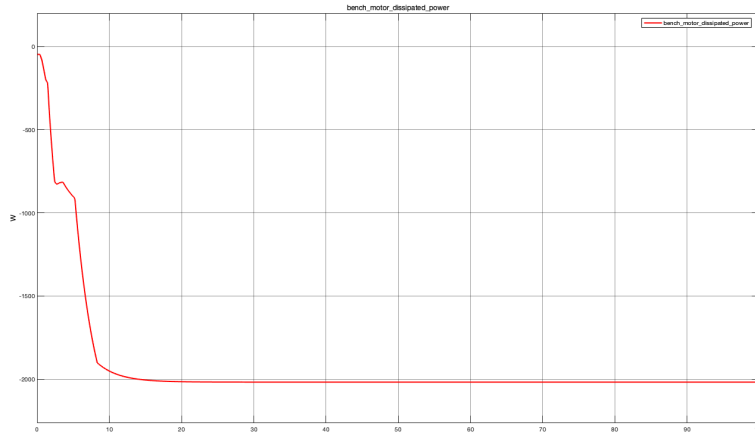


Figure 4.33:  $P_{d_{bm}}$  with bench motor set in tabulated efficiency data

At steady-state, it results to be approximately -2.017 kW. Also in this case, it assumes negative sign since it represents dissipated power and it is more or less equal to value obtained in case where only bench motor is set in tabulated loss data (-2 kW as shown in figure 4.23)

- *Bench motor absorbed power ( $P_{a_{bm}}$ )*

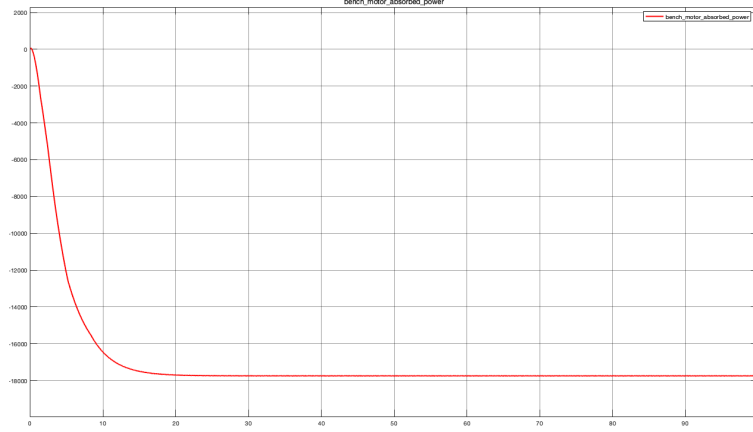


Figure 4.34:  $P_{a_{bm}}$  with bench motor set in tabulated efficiency data

At steady-state, it assumes a negative sign approximately equal to -17.74 kW which means it represents bench motor's provided power. Its value is more or less equal to that obtained in case where only bench mapped motor is set in tabulated loss data (-17.75 kW as shown in figure 4.24)

- *Bench motor demanded or drawn current ( $I_{bm}$ )*

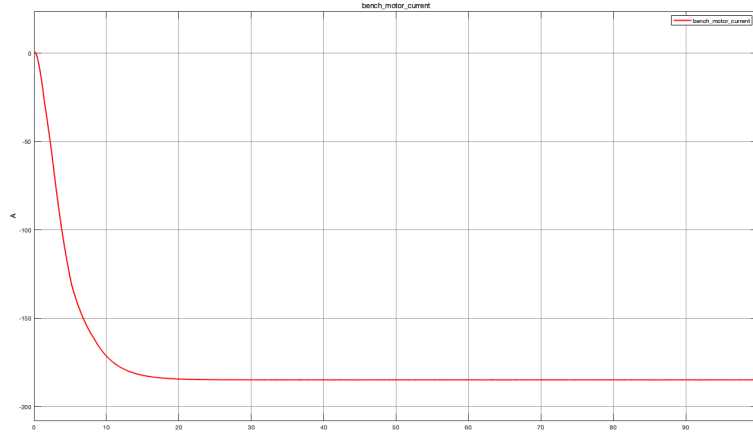


Figure 4.35:  $I_{bm}$  with bench motor set in tabulated efficiency data

At steady-state, it is approximately -184.759 A that is a bit smaller than value obtained in case where only bench mapped motor is set in tabulated loss data (-184.941 A as shown in figure 4.25). Its negative sign means that it represents demanded current.



- *Test motor dissipated power ( $P_{d_{tm}}$ )*

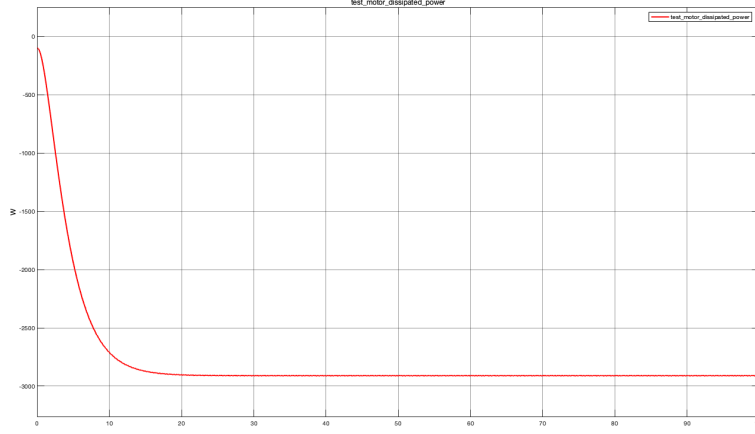


Figure 4.36:  $P_{d_{tm}}$  with bench motor set in tabulated efficiency data

Here, at steady-state variable results to be approximately -2.909 kW, so it is very similar to value obtained in case where only bench mapped motor Simulink blocks is set in tabulated loss data (-2.901 kW as shown in figure 4.26). Its negative sign means that it is lost power.

- *Test motor absorbed power ( $P_{a_{tm}}$ )*

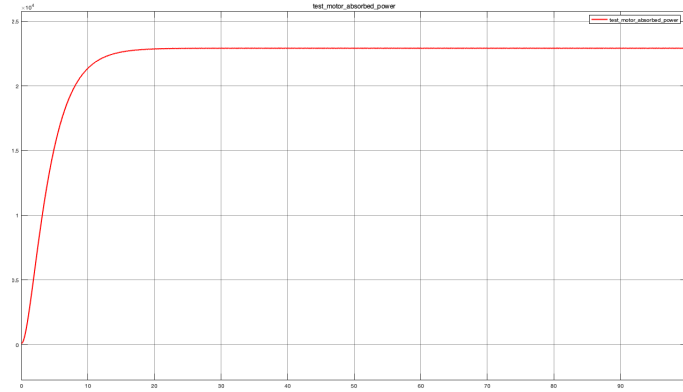


Figure 4.37:  $P_{a_{tm}}$  with bench motor set in tabulated efficiency data

At steady-state, it is approximately 22.91 kW, so it results to be equal to value obtained in case where only bench mapped motor Simulink block is set in tabulated loss data. In this case, unlike bench motor absorbed power, its value is positive since it represents absorbed power instead of provided power.

- *Test motor demanded or drawn current ( $I_{tm}$ )*

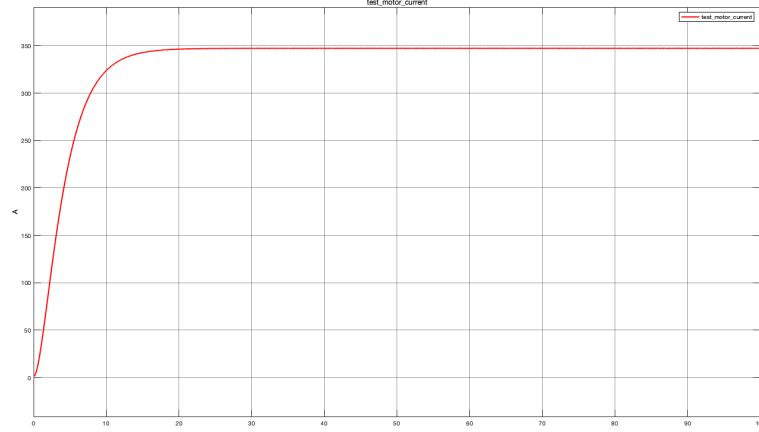


Figure 4.38:  $I_{tm}$  with bench motor set in tabulated efficiency data

At steady-state, it results to be approximately 347.178A like value obtained in case where only bench motor is set in tabulated loss data. In this case, unlike bench motor case, current value assumes a positive value since it represents drawn current instead of demanded current.

- *Motor shaft rotational speed  $\omega$*

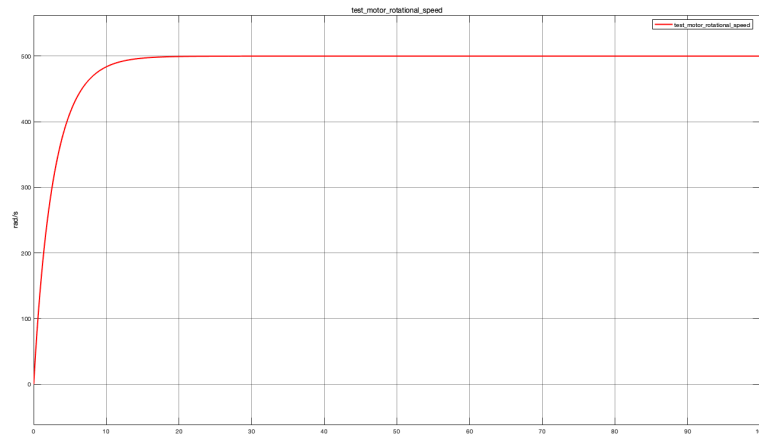


Figure 4.39:  $\omega$  with bench motor set in tabulated efficiency data

At steady-state, it assumes the expected value 500 rad/s that results to be equal to rotational speed command set at beginning to pilot bench motor that is responsible to impose rotational speed value to test motor.

- *Test motor torque  $T_{tm}$*

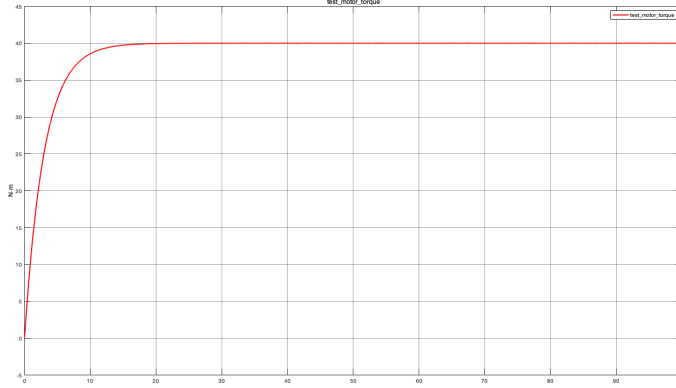


Figure 4.40:  $T_{tm}$  with bench motor set in tabulated efficiency data

At steady-state, it reaches desired value that is equal to torque command value set at beginning by user to pilot motor under test, 40 Nm.

- *Power recirculating effect*

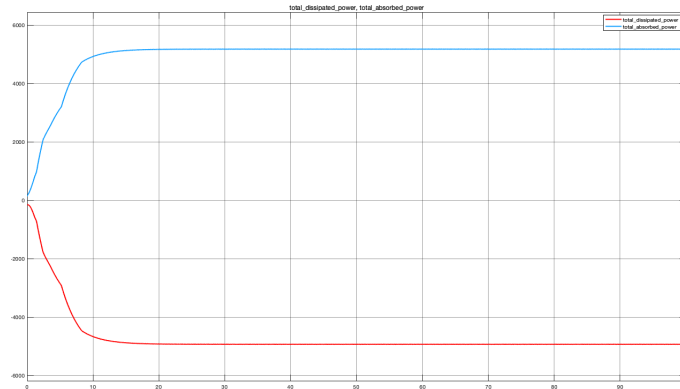


Figure 4.41:  $P_a$ ,  $P_d$  with bench motor set in tabulated efficiency data

Here, like previous case where bench mapped motor Simulink block is set in tabulated loss data, absorbed and dissipated power steady-state values are not perfectly equal between them as expected. This, due to motor shaft presence which requires power that must be added to motors' absorbed power. This time absorbed power at steady-state assumes is more or less equal to 5.177 kW, a bit bigger than value obtained in case where only bench mapped motor Simulink model is set in tabulated loss data (5.159 kW), like also happens in dissipated power case indeed here it assumes a value of -4.927 kW while in previous case it is more or less equal to 4.909 kW.

- *Total demanded current ( $I$ )*

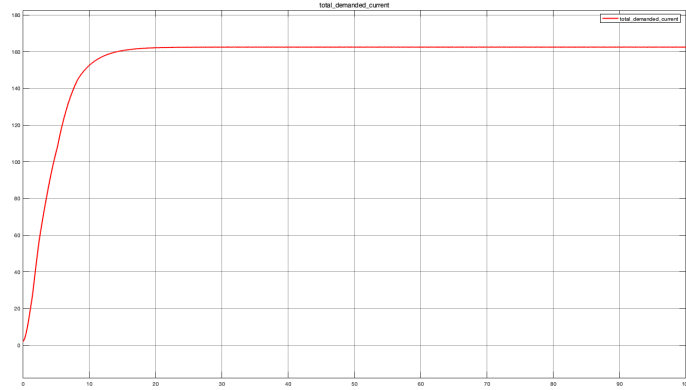
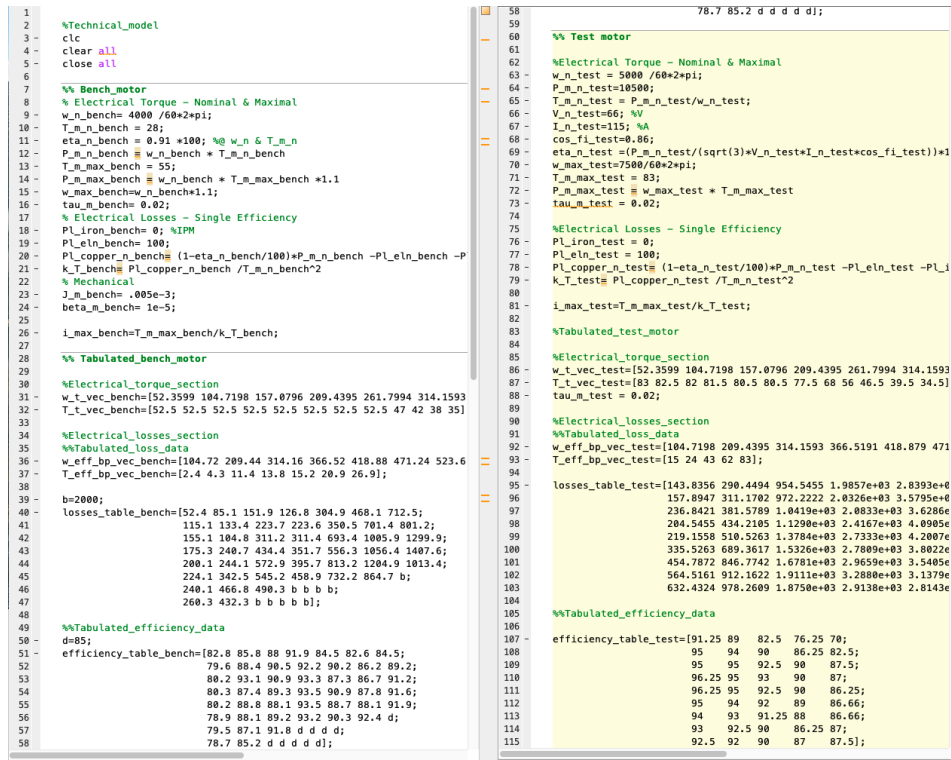


Figure 4.42:  $I$  with bench motor set in tabulated efficiency data

Here, the sum between bench and test motor phase current ( $I_{bm}$ ,  $I_{tm}$ ) is shown. At steady- state it assumes a value more or less equal to that obtained in case where only bench motor is set in tabulated loss data. Indeed, here it is approximately 162.4A while in the other case it is about 162.2 A.

#### 4.1.3 Tabulated bench and test motor

Once model with only bench mapped motor Simulink block sets in tabulated loss data is tested and validated, next step can be performed. Here, test mapped motor Simulink block has to be tabulated by using the same procedure adopted for bench mapped motor Simulink block. So, also in this case the two parameters window's section must be considered: *Electrical Torque and Electrical Losses*. In the first section only one parameterization way can be chosen while in the second section two different parameterization way can be chosen. Indeed, opening test mapped motor Simulink block and selecting Electrical Torque section, only *Tabulated torque-speed envelope* can be chosen as parameterization way from drop-down menu while considering Electrical Losses section two different options can be chosen as parameterization way: *Tabulated loss data* in case test motor's dissipated power map is chosen or *Tabulated efficiency data* in case test motor's efficiency map is chosen. Each section is characterized by different variables to be defined in MATLAB script as described in bench motor's tabulation case. So, corresponding MATLAB script to be run for starting Simulink simulation is the one shown in figure 4.43.



```

1  %Technical_model
2  c1c
3  clear all
4  close all
5
6  %% Bench_motor
7  % Electrical Torque - Nominal & Maximal
8  w_n_bench= 4000 /60*2*pi;
9  T_m_n_bench = 28;
10 eta_n_bench = 0.91 *100; %w_n & T_m_n
11 P_m_n_bench = w_n_bench * T_m_n_bench
12 P_m_max_bench = 55;
13 P_m_max_bench = w_n_bench * T_m_max_bench *1.1
14 w_max_bench=w_n_bench*1.1;
15 tau_m_bench= 0.02;
16
17 % Electrical Losses - Single Efficiency
18 PL_iron_bench= 0; %1PM
19 PL_ein_bench= 100;
20 PL_copper_n_bench= (1-eta_n_bench/100)*P_m_n_bench -PL_ein_bench -P
21 k_T_bench= PL_copper_n_bench /T_m_n_bench^2
22 % Mechanical
23 J_m_bench= .005e-3;
24 beta_n_bench= 1e-5;
25
26 i_max_bench=T_m_max_bench/k_T_bench;
27
28
29 %% Tabulated_bench_motor
30
31 %Electrical torque_section
32 w_t_vec_bench=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593
33 T_t_vec_bench=[52.5 52.5 52.5 52.5 52.5 52.5 52.5 47 42 38 35]
34
35 %Electrical_losses_section
36 %%Tabulated_loss_data
37 w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6
38 T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 20.9 26.9];
39
40 b=2000;
41 losses_table_bench=[52.4 85.1 151.9 126.8 304.9 468.1 712.5;
42 115.1 133.4 223.7 223.6 350.5 701.4 801.2;
43 155.1 104.8 311.2 311.4 693.4 1005.9 1299.9;
44 175.3 248.7 434.4 351.7 556.3 1056.4 1407.6;
45 200.1 244.1 572.9 395.7 613.2 1204.9 1013.4;
46 224.1 342.5 545.2 458.9 732.2 864.7 b;
47 240.1 466.8 490.3 b b b b;
48 260.3 432.3 b b b b b];
49
50 %%Tabulated_efficiency_data
51 d=85;
52 efficiency_table_bench=[82.8 85.8 88 91.9 84.5 82.6 84.5;
53 79.6 88.4 90.5 92.2 90.2 86.2 89.2;
54 80.2 93.1 90.9 93.3 87.3 86.7 91.2;
55 80.3 87.4 89.3 93.5 90.9 87.8 91.6;
56 80.2 88.8 88.1 93.5 88.7 88.1 91.9;
57 78.9 88.1 89.2 93.2 90.3 92.4 d;
58 79.5 87.1 91.8 d d d d;
59 78.7 85.2 d d d d d];
60
61 %% Test motor
62 %Electrical Torque - Nominal & Maximal
63 w_n_test = 5000 /60*2*pi;
64 P_m_n_test=10500;
65 T_m_n_test = P_m_n_test/w_n_test;
66 V_n_test=66; %V
67 I_n_test=115; %A
68 cos_phi_test=0.86;
69 eta_n_test =(P_m_n_test/(sqrt(3)*w_n_test*I_n_test*cos_phi_test))*1
70 w_max_test=7500/60*2*pi;
71 T_m_max_test = 83;
72 P_m_max_test = w_max_test * T_m_max_test
73 tau_m_test = 0.02;
74
75 %Electrical Losses - Single Efficiency
76 PL_iron_test = 0;
77 PL_ein_test = 100;
78 PL_copper_n_test= (1-eta_n_test/100)*P_m_n_test -PL_ein_test -PL_i
79 k_T_test= PL_copper_n_test /T_m_n_test^2
80
81 i_max_test=T_m_max_test/k_T_test;
82
83 %%Tabulated_test_motor
84
85 %Electrical torque_section
86 w_t_vec_test=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593
87 T_t_vec_test=[83 82.5 82 81.5 80.5 80.5 77.5 68 56 46.5 39.5 34.5]
88 tau_m_test = 0.02;
89
90 %Electrical_losses_section
91 %%Tabulated_loss_data
92 w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471
93 T_eff_bp_vec_test=[15 24 43 62 83];
94
95 losses_table_test=[143.8356 290.4494 954.5455 1.9857e+03 2.8393e+0
96 157.8947 311.1702 972.2222 2.0326e+03 3.5795e+0
97 236.8421 381.5789 1.0419e+03 2.0833e+03 3.6286e
98 204.5455 434.2105 1.1290e+03 2.4167e+03 4.0905e
99 219.1558 510.5263 1.3784e+03 2.7333e+03 4.2007e
100 335.5263 689.3617 1.5326e+03 2.7809e+03 3.8022e
101 454.7872 846.7742 1.6781e+03 2.9659e+03 3.5405e
102 564.5161 912.1622 1.9111e+03 3.2880e+03 3.1379e
103 632.4324 978.2609 1.8750e+03 2.9138e+03 2.8143e
104
105 %%Tabulated_efficiency_data
106
107 efficiency_table_test=[91.25 89 82.5 76.25 70;
108 95 94 90 86.25 82.5;
109 95 95 92.5 90 87.5;
110 96.25 95 93 90 87;
111 96.25 95 92.5 90 86.25;
112 95 94 92 89 86.66;
113 94 93 91.25 88 86.66;
114 93 92.5 90 86.25 87;
115 92.5 92 90 87 87.5];
    
```

Figure 4.43: MATLAB script with tabulated bench and test motor

Here, all needed variables to parametrize Simulink model blocks are defined so that, once it is run, they are loaded into workspace and Simulink blocks can use them during simulation. In this MATLAB script, besides to find needed variables to parametrize bench mapped motor Simulink block, all needed variables to parameterize test mapped motor Simulink block are defined. Indeed, variables below listed must be added.

- $w\_t\_vec\_test$ , added to define  $w\_t$  vector for test motor
- $T\_t\_vec\_test$ , added to define  $T\_t$  vector for test motor
- $tau\_m\_test$ , added to define  $T_c$  constant for test motor
- $w\_eff\_bp\_vec\_test$ , added to define  $w\_eff\_bp$  vector for test motor
- $T\_eff\_bp\_vec\_test$ , added to define  $T\_eff\_vec$  vector for test motor
- $loss\_table\_test$ , added to define test motor  $losses\_table$
- $efficiency\_table\_test$ , added to define test motor  $efficiency\_table$

Before to start Simulink simulation, simulation time must be defined in order to decide how many time simulation must last to evaluate parameter

trends. To obtain trend easily comparable with the ones obtained in previous cases, it is set equal to 100s, like previous cases. Moreover, bench and test mapped motor Simulink blocks must be set in proper way in order to tabulate them. For this reason, first of all *Tabulated torque-speed envelope* parameterization way from electrical torque section and *Tabulated loss data* or *Tabulated efficiency data* parameterization way from electrical losses section must be chosen in order to parametrize both bench and test mapped motor Simulink block considering either their dissipated power maps or their efficiency maps.

In case user decides to consider dissipated power maps to parametrize both mapped motor Simulink blocks, Tabulated loss data parameterization has to be selected from electrical losses section of block parameters' window. At that point running Simulink model, parameter trends below shown are obtained.

- *Bench motor dissipated power ( $P_{d_{bm}}$ )*

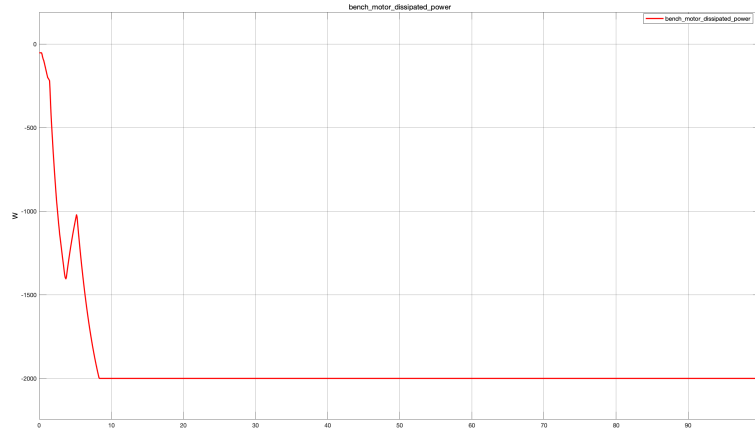


Figure 4.44:  $P_{d_{bm}}$  with both motors set in tabulated loss data

At steady-state, it results to be approximately -2 kW. It reaches that value assuming the same trend obtained in case where only bench mapped motor Simulink model is set in tabulated loss data (figure 4.23). Its value has negative sign since it represents dissipated power.

- *Bench motor absorbed power ( $P_{a_{bm}}$ )*

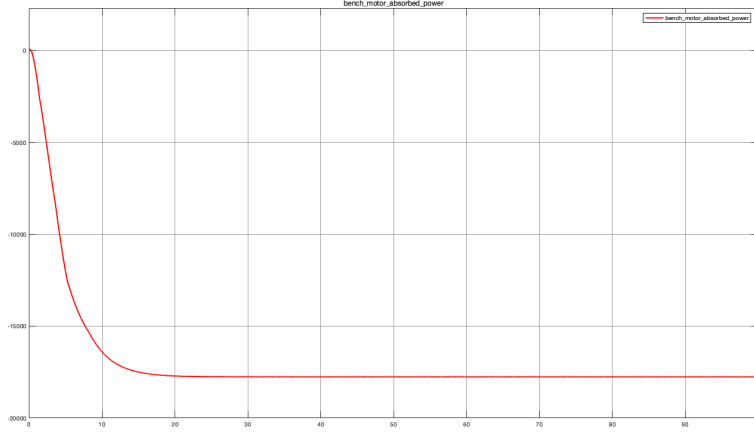


Figure 4.45:  $P_{a_{bm}}$  with both motors set in tabulated loss data

At steady-state, it assumes a value more or less equal to -17.75 kW that results to be exactly equal to value obtained in case where only bench mapped motor Simulink block is set in tabulated loss data (figure 4.24). Its negative value means that it represents provided power.

- *Bench motor demanded or drawn current ( $I_{bm}$ )*

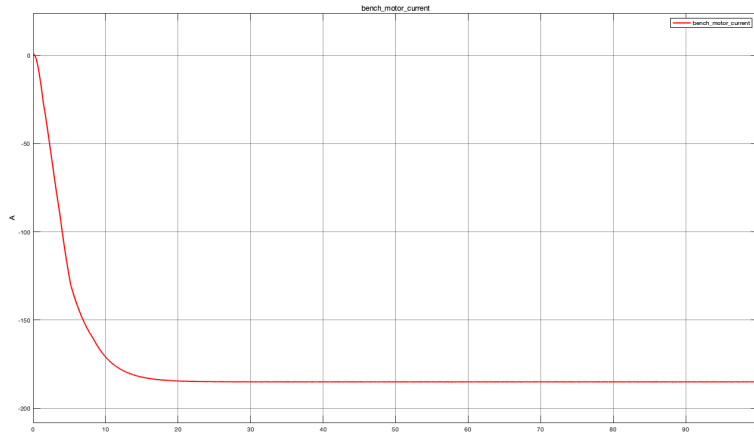


Figure 4.46:  $I_{bm}$  with both motors set in tabulated loss data

At steady-state, it is about -184.947 A. Its negative sign means that it represents demanded current and its value is very similar to that reached in case where only bench mapped motor Simulink block is set in tabulated loss data (-184.941 A as shown in figure 4.25).

- *Test motor dissipated power ( $P_{d_{tm}}$ )*

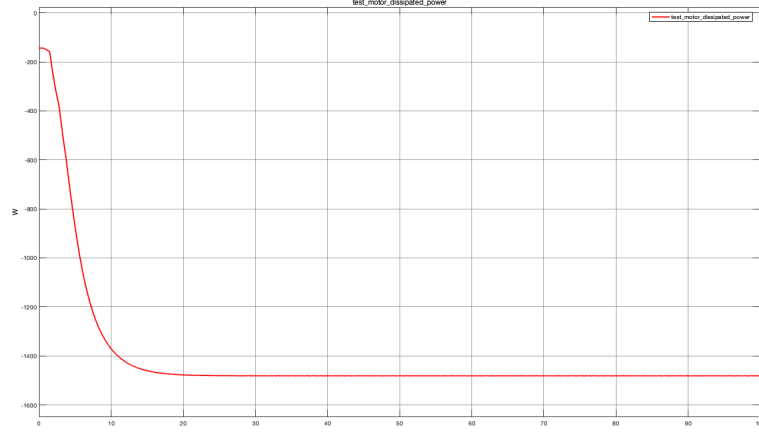


Figure 4.47:  $P_{d_{tm}}$  with both motors set in tabulated loss data

At steady-state, it results to be approximately -1.481 kW which means that it is smaller than value obtained in all other cases. Its negative sign means that it represents lost power.

- *Test motor absorbed power ( $P_{a_{tm}}$ )*

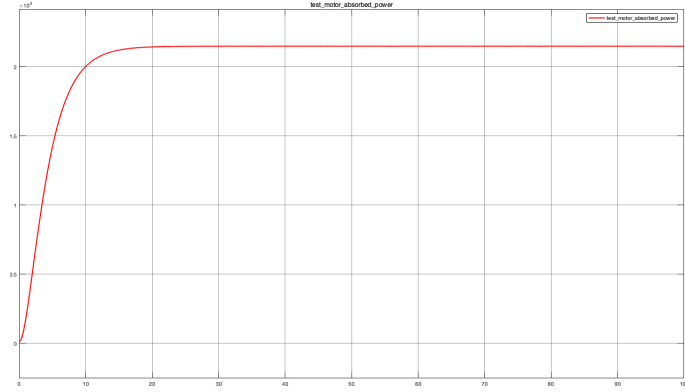


Figure 4.48:  $P_{a_{tm}}$  with both motors set in tabulated loss data

At steady-state, it assumes a value approximately equal to 21.49 kW, smaller than those obtained in previous cases. In this case, unlike bench motor absorbed power, it assumes a positive sign since it represents absorbed power instead of provided power.



- *Test motor demanded or drawn current ( $I_{tm}$ )*

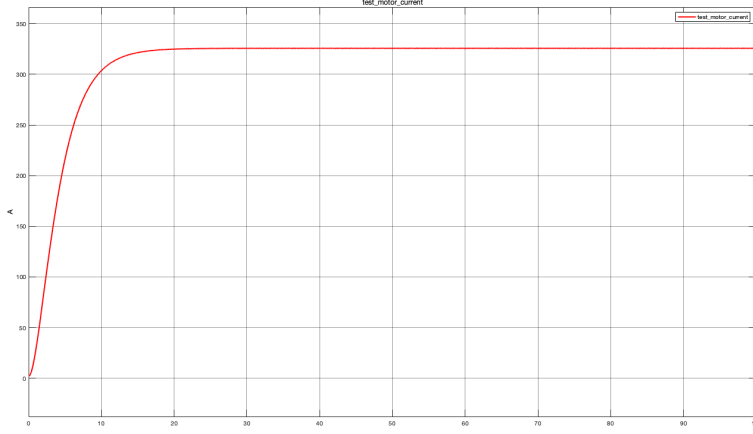


Figure 4.49:  $I_{tm}$  with both motors set in tabulated loss data

At steady-state, it assumes a value approximately equal to 325.542 A, smaller than steady-state values obtained in other cases. In this case, unlike bench motor case, current value assumes positive sign since it represents drawn current instead of demanded current.

- *Motor shaft rotational speed  $\omega$*

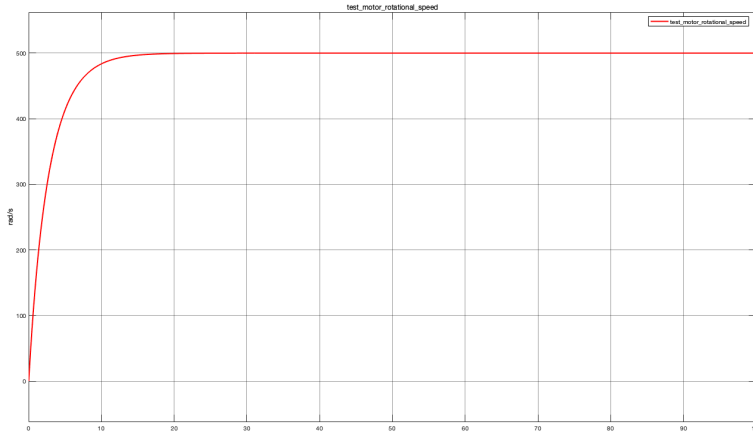


Figure 4.50:  $\omega$  with both motors set in tabulated loss data

At steady-state, is approximately equal to desired value (500 rad/s) that corresponds to rotational speed's command set at beginning by user to pilot bench motot which is responsible to impose rotational speed to test motor.

- *Test motor torque  $T_{tm}$*

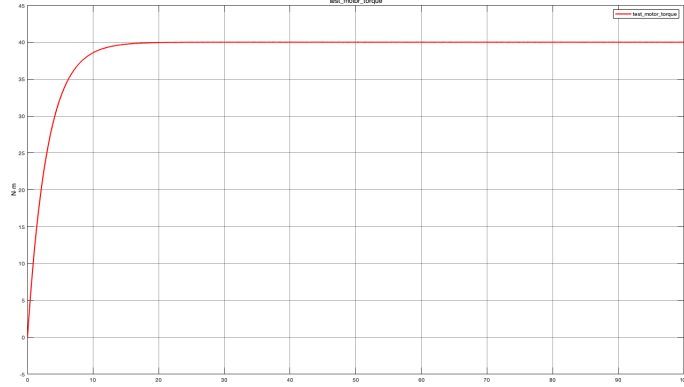


Figure 4.51:  $T_{tm}$  with both motors set in tabulated loss data

At steady-state, it assumes desired value (40 Nm) which corresponds to torque value's command set at beginning by user to pilot test motor is approximately 40 Nm.

- *Power recirculating effect*

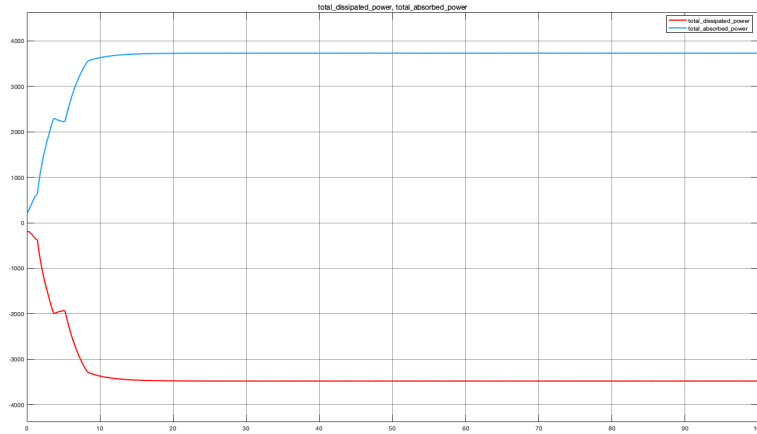


Figure 4.52:  $P_a$ ,  $P_d$  with both motors set in tabulated loss data

Here, like all cases previously analyzed, absorbed and dissipated power steady-state values are not perfectly equal between them, as expected. This due to motor shaft presence which absorbs power that must be added to motors' absorbed power. This time absorbed power at steady-state is about 3.731 kW, while dissipated power steady-state value is about -3.481 kW. Both are smaller than values obtained in previous cases.

- *Total demanded current ( $I$ )*

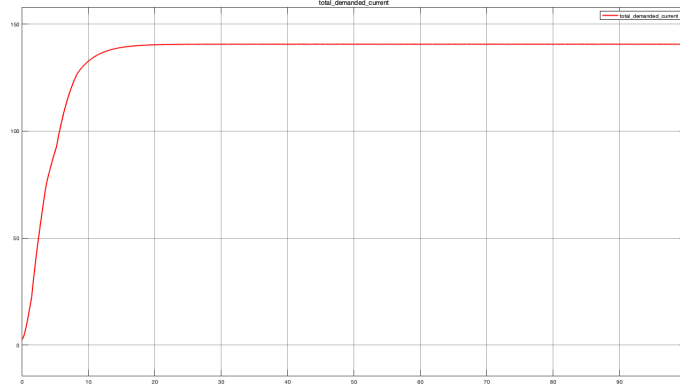


Figure 4.53:  $I$  with both motors set in tabulated loss data

Here, sum between bench and test motor current ( $I_{bm}$ ,  $I_{tm}$ ) is shown. At steady-state it is about 140.6A, smaller than steady-state values obtained in all other cases.

Instead, in case where user decides to parametrize bench and test mapped motor Simulink blocks considering their efficiency maps, Tabulated efficiency data parameterization way from electrical losses section must be chosen. Doing this and performing Simulink simulation setting the same simulation time considered in previous cases, parameters trends below shown have been obtained that can be compared to those obtained in previous cases.

- *Bench motor dissipated power ( $P_{d_{bm}}$ )*

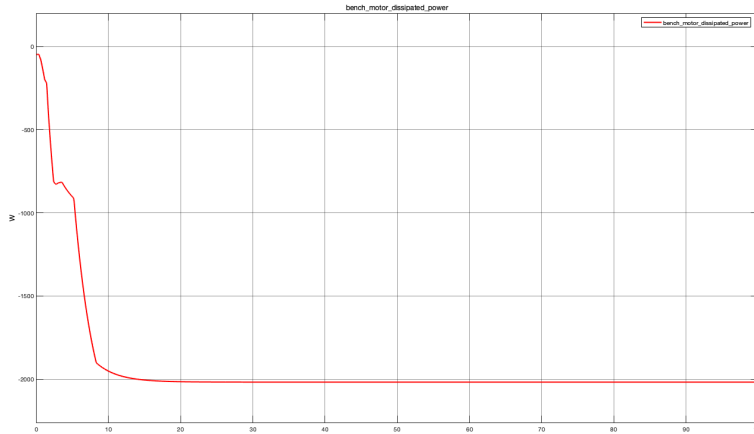


Figure 4.54:  $P_{d_{bm}}$  with both motors set in tabulated efficiency data

At steady-state, it results to be approximately -2.017 kW which is the

same value obtained in case where only bench mapped motor Simulink block is tabulated in efficiency data (figure 4.33). Its negative sign means that it represents dissipated power.

- *Bench motor absorbed power ( $P_{abm}$ )*

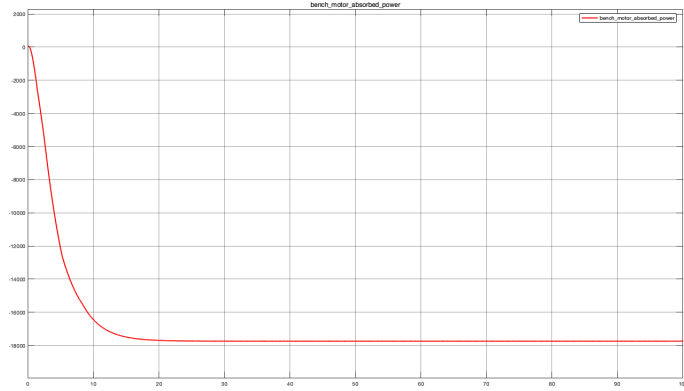


Figure 4.55:  $P_{abm}$  with both motors set in tabulated efficiency data

At steady-state, it assumes the same value obtained in case where only bench mapped motor Simulink block is tabulated in tabulated efficiency data (figure 4.34), about -17.74 kW. Its negative value means that it represents provided power.

- *Bench motor demanded or drawn current ( $I_{bm}$ )*

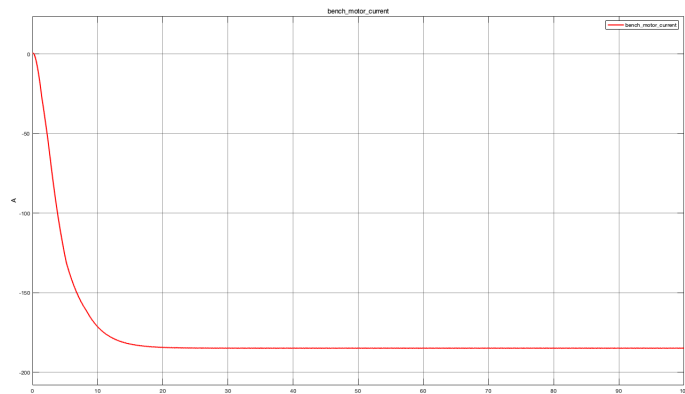


Figure 4.56:  $I_{bm}$  with both motors set in tabulated efficiency data

At steady-state, it is approximately equal to -184.765 A which means that it is perfectly equal to value obtained in case where only bench mapped Simulink block is set in tabulated efficiency data (figure 4.35). Its negative sign means that it represents demanded current.

- *Test motor dissipated power ( $P_{d_{tm}}$ )*

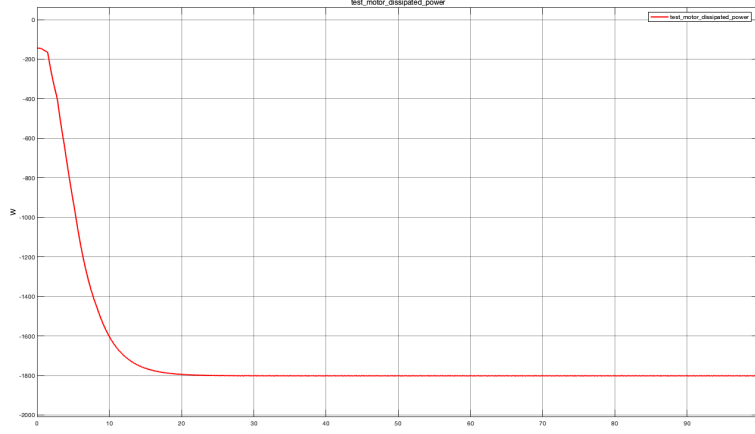


Figure 4.57:  $P_{d_{tm}}$  with both motors set in tabulated efficiency data

At steady-state, it assumes bigger value than those obtained in previous cases, about -1.801 kW. Its negative value means that it is lost power.

- *Test motor absorbed power ( $P_{a_{tm}}$ )*

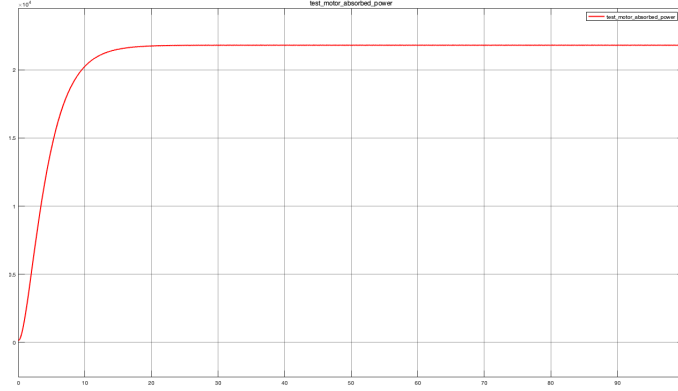


Figure 4.58:  $P_{a_{tm}}$  with both motors set in tabulated efficiency data

At steady-state, it reaches smaller value with respect to those obtained in previous cases but at the same time reached value is bigger than that obtained in case where both mapped motor Simulink blocks are set in tabulated loss data, about 21.81 kW. Here, unlike bench motor case, absorbed power assumes positive sign since it represents absorbed power and not provided power.

- *Test motor demanded or drawn current ( $I_{tm}$ )*

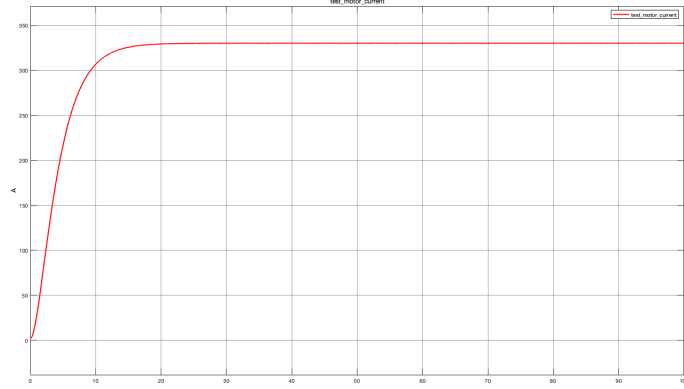


Figure 4.59:  $I_{tm}$  with both motors set in tabulated efficiency data

At steady-state, it is approximately equal to 330.399 A, smaller than those obtained in previous cases but at the same time it is bigger than value obtained in case where both motors are set in tabulated loss data. Here, unlike bench motor's phase current case, current value assumes positive sign since it represents drawn current.

- *Motor shaft rotational speed  $\omega$*

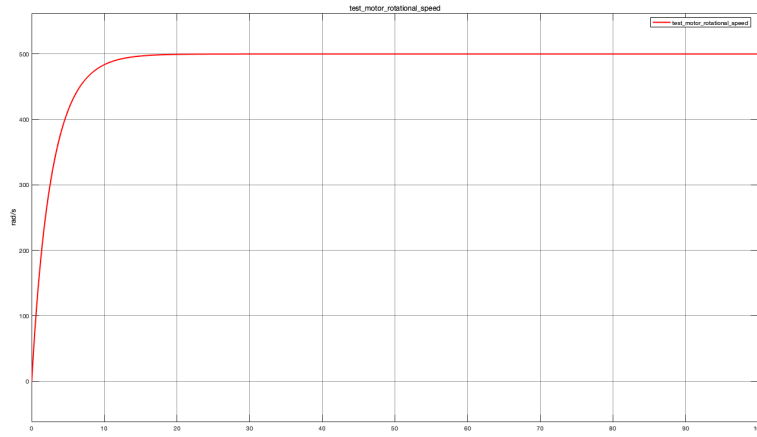


Figure 4.60:  $\omega$  with both motors set in tabulated efficiency data

At steady-state, it assumes desired rotational speed value, 500 rad/s which represents rotational speed command's value set at beginning by user to pilot bench motor that is responsible to impose rotational speed to test motor.

- *Test motor torque  $T_{tm}$*

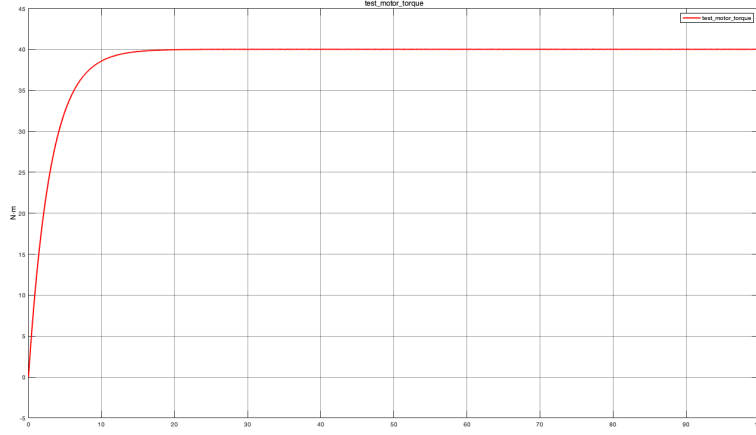


Figure 4.61:  $T_{tm}$  with both motors set in tabulated efficiency data

At steady-state, it assumes desired torque value since it results to be more or less equal to torque value command set at beginning by user to pilot motor under test, approximately 40 Nm.

- *Power recirculating effect*

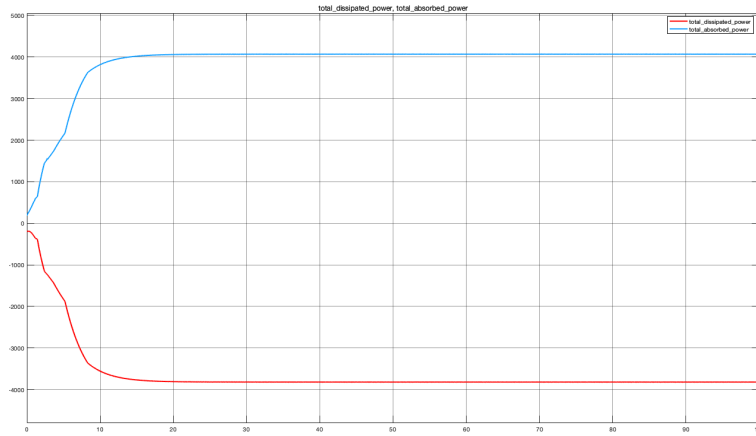


Figure 4.62:  $P_a$ ,  $P_d$  with both motors set in tabulated efficiency data

Here, the sum between bench and test motor's phase current ( $I_{bm}$ ,  $I_{tm}$ ) is shown. At steady-state its value is approximately equal to 145.6A, smaller than those obtained in all other cases but, at the same time it is bigger than value obtained in case where both mapped motor Simulink blocks are set in tabulated loss data.

- *Total demanded current ( $I$ )*

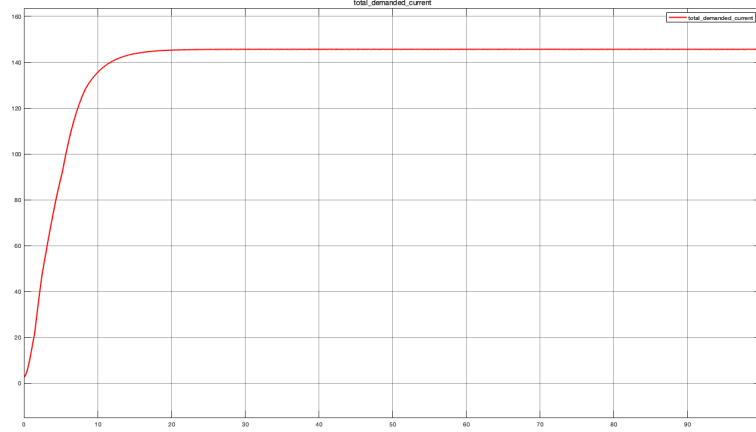


Figure 4.63:  $I$  with both motors set in tabulated efficiency data

Here, sum between bench and test motor phase current ( $I_{bm}$ ,  $I_{tm}$ ) is shown. At steady- state its value is approximately equal to 145.6A, smaller than steady-state values obtained in all other cases whose simulation results are reported in previous subsections but, at the same time, it is bigger than steady-state value obtained in case where both motors are set in tabulated loss data.

## 4.2 Modular Technical model

Once preliminary technical model, described in previous subsections, is completed in order to highlight in a better way user requirements, *modular technical model tool* can be adopted to simplify electrical motor's test bench development. This because, MTM tool allows to subdivide development in modules that can be developed by different designers and/or teams with different skills in order to reduce development time and in turns costs to be addressed. This is what happens from one hand but in the other hand MTM tool requires designers/teams' higher efforts since each module must be developed in such a way that it is able to communicate with other modules, so they need to be constituted of specific interfaces that allow communication to other modules. During this thesis work to build modular technical model, Simulink template shown in figure 4.64 has been adopted. Here, five modules can be distinguished: *Environment*, *Plant*, *Control*, *Human-Machine Interface*, *User*. Some of these modules such as Environment, Plant and Control must be implemented as *reference models*, models that are able to work independently from other ones, while other modules such as Human-Machine



Interface and User must be implemented as *subsystems*, models that allows to arrange in a better way different system parts in order to obtain a more intelligible model.

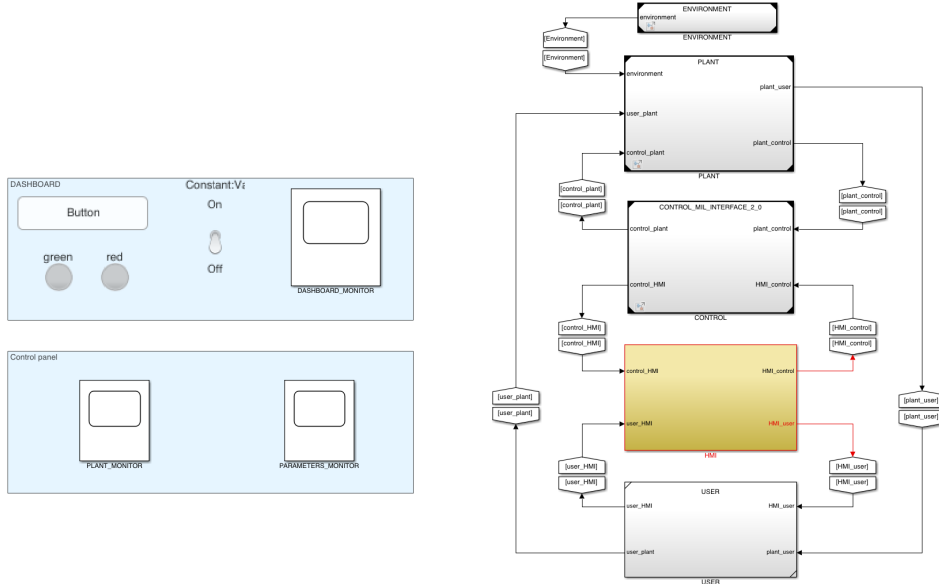


Figure 4.64: MTM Simulink template

#### 4.2.1 Environment reference model

This module allows to implement noises, disturbances and loads coming from external ambient which, for sure, influence plant's behavior so they have to be considered for obtaining more realistic results from Simulink simulation. In this case, this module includes noises' model that influence sensors used in plant to measure interesting parameters such as rotational speed, torque and current. In figure 4.65 model implemented inside this module is shown.

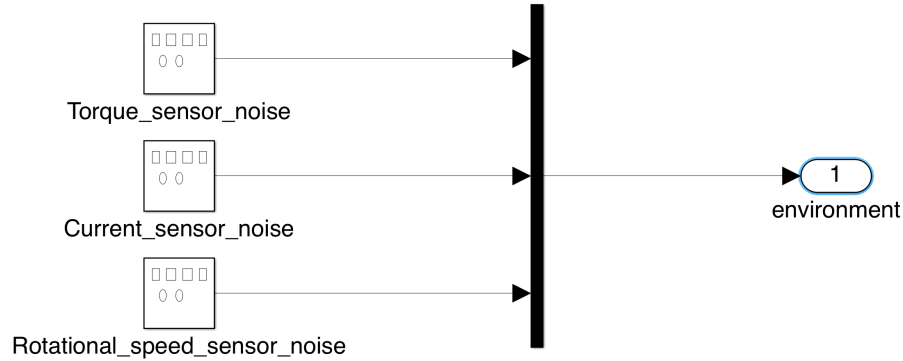


Figure 4.65: Environment's reference model

Here, noises are modeled using signal generator Simulink blocks that need different parameters to be parametrized. To know needed parameters to be defined in MATLAB script a double click on Simulink block is sufficient so that block parameters' window opens (figure 4.66).

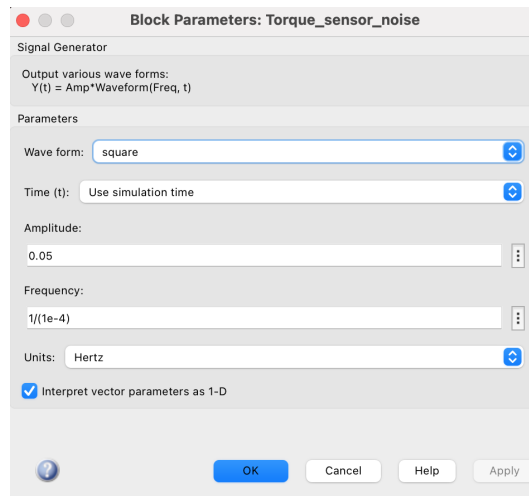


Figure 4.66: Torque sensor noise's block parameters window

In this case, *square* and *Use simulation time* from "wave form" and "time" drop-down menus are chosen such that signal generator signal Simulink blocks implement a square wave that lasts for all simulation time. Then, other parameters such as amplitude and frequency have to be defined in order to implement signals with desired characteristics. These parameters are defined considering sensor's datasheets below reported.



- Current sensor DHR 300 C420



Figure 4.68: Current sensor DHR 300 C420 datasheet

Here, accuracy sensor is declared to be smaller than  $\pm 1\%$  of rated current and since current sensor with 300A rated current has to be considered to build-up final physical system, accuracy's value to be considered for modelling sensor is  $\pm 3$  A. Instead, frequency is declared between 20 and 6000 Hz and since to model sensor it is better to consider the worst case a frequency of 6kHz has been considered.

- Round per minute measurement system



Foglio 11

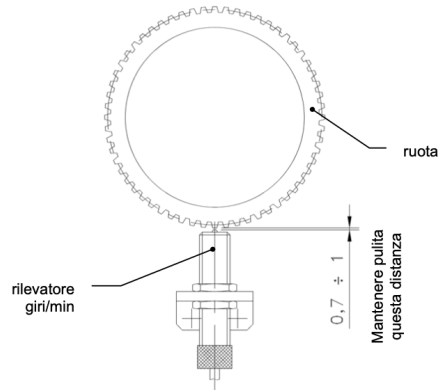
### DISPOSITIVO DI MISURA DEI GIRI/MINUTO

#### 1. STRUTTURA E FUNZIONAMENTO

Il misuratore del numero di giri consiste in una ruota con 60 denti posta sulla flangia posteriore del freno ed in un rilevatore del numero di giri. Gli impulsi del rilevatore vengono trasmessi tramite un convertitore di frequenza in tensione allo strumento di misura.

#### 2. DATI TECNICI DEL RILEVATORE DEL NUMERO DI GIRI TIPO MP 62 TA

Trafero sulla ruota dentata =  $0,7 \pm 1$  mm.  
Temperatura ambiente ammessa = da  $-50^{\circ}\text{C}$  a  $+120^{\circ}\text{C}$ .  
Campo di misurazione da  $50 \div 17.000$  giri.  
Dimensioni: lunghezza totale mm 76;  
filettatura 5/8 - 18 UNF - Lunghezza mm 60.



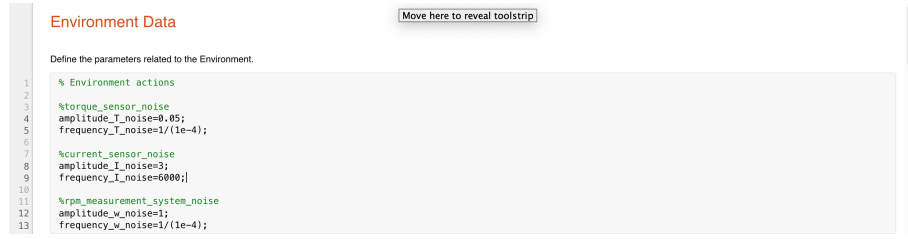
#### 3. ATTENZIONE

Verificare che il rilevatore dei giri sia ad una distanza di  $0,7 \pm 1$  mm. dal dente della ruota  $Z = 60$ .

Figure 4.69: Round per minute measurement system datasheet

In “ICE Test Bench Upgrading For Hybrid and Electrical Powertrain” thesis, accuracy of this system is declared to be equal to  $\pm 1$ rpm. For this reason, corresponding signal generator amplitude is set equal to 1. Instead, for frequency value 10 kHz is chosen in order to have a disturbance that changes its value quickly.

In figure 4.70 corresponding MATLAB script is shown where all needed variables to parametrize signal generator Simulink blocks implemented inside Environment reference model are defined.



```

1 % Environment actions
2
3 %torque_sensor_noise
4 amplitude_T_noise=0.05;
5 frequency_T_noise=1/(1e-4);
6
7 %current_sensor_noise
8 amplitude_I_noise=3;
9 frequency_I_noise=6000;|
10
11 %rpm_measurement_system_noise
12 amplitude_w_noise=1;
13 frequency_w_noise=1/(1e-4);

```

Figure 4.70: Environment reference model's MATLAB script

### 4.2.2 Plant reference model

This module is implemented as reference model so that it is able to work independently from rest parts. Here, plant's model to be controlled is implemented which is constituted of two mapped motor Simulink blocks connected each other by means motor shaft and sensors used to measure interesting parameters during testing. Implemented model inside Plant reference model is shown in figure 4.71.

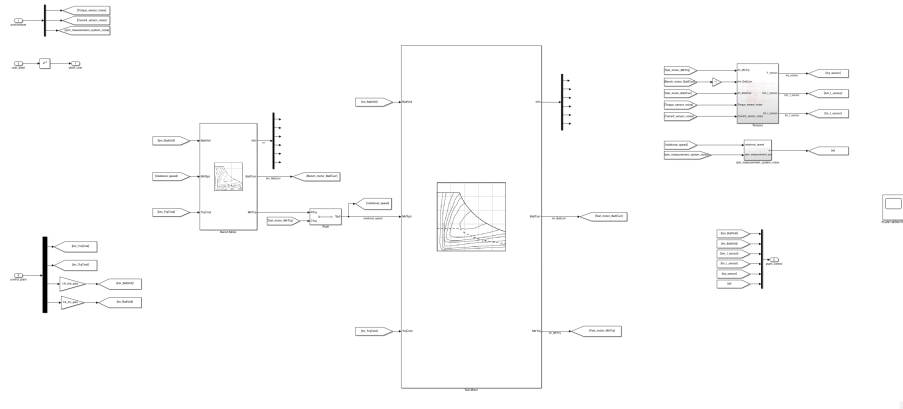


Figure 4.71: Plant reference model

On the left model's side, three different inputs are modelled that represent interface through which plant is able to communicate to other parts (reference model or subsystem) such as Environment, Control and User.

- *first input*

It represents interface by means plant is able to communicate to Environment reference model, before described. From it, plant receives noise signals characterizing adopted sensors whose models are implemented on the right part.

- *second input*

It represents interface by means plant is able to communicate to user subsystem but, since in this case user and plant do not exchange signals each other they are connected each other through which a discrete delay Simulink block that allows to avoid algebraic loop errors during simulations.

- *third input*

It represents interface by means plant is able to communicate to Control reference model. From it, plant receives below listed signals.

- *bench motor torque command signal*

It represents used signal to pilot bench motor coming out form control law implemented inside *Control Logic refence model* (analyzed in subsection 4.2.3.1) that is able to evaluate this signal on the basis of rotational speed value set at beginning by user and on the basis of rotational speed value coming from feedback path.

- *test motor torque command signal*

It represents used signal to pilot test motor coming out from control law implemented inside *Control Logic reference model* on the basis of initial torque value set by user.

- *bench motor battery voltage signal*

It represents used signal to set bench motor power supply value. Since it is a voltage signal coming out from *Control reference model* it has to be translated into voltage value to be provided to bench mapped motor Simulink block. This is done by a gain implementing a constant that multiplies voltage signal to obtain corresponding voltage value.

- *Test motor battery voltage signal*

It represents used signal to set test motor power supply value. Also in this case, like bench motor battery voltage signal, since it is a voltage signal coming out form *Control reference model* it has to be translated in voltage value to be provided to test mapped motor Simulink block. This is done by means a gain implemented to multiply voltage signal for obtaining corresponding voltage value.

In the middle bench and test mapped motor Simulink blocks, connected each other by means motor shaft, are implemented. They work exactly in the same way described in preliminary technical model section. Instead on the right part, two subsystems are implemented: the first one, on upper part, is dedicated to sensor models while the second one, on bottom part, is

dedicated to round per minutes measurements system's model.

Opening sensor models' subsystem, scheme shown in figure 4.72 can be observed where gains, constants and sum Simulink blocks are implemented to model torque and current sensors.

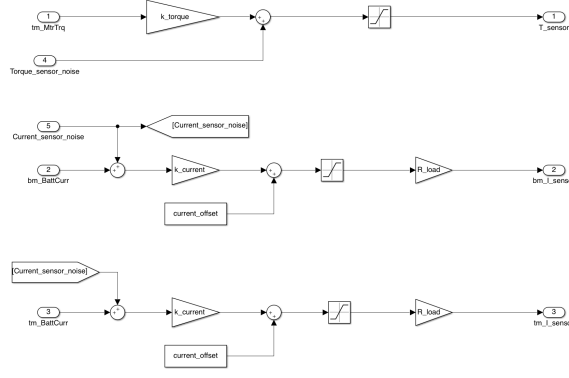


Figure 4.72: Sensor models subsystem

- *Torque sensor HBM T40B*

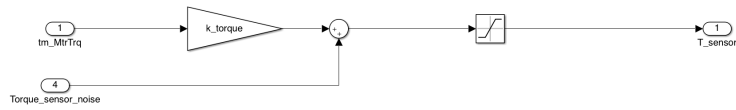


Figure 4.73: Torque sensor Simulink model

Gain allows to convert torque value measured in Nm into voltage value that has to be added to disturbance coming from Environment reference model. The result of the sum goes through a saturator Simulink block that allows to set upper and down limits for voltage signal coming out from sensor. To define gain value, sensor datasheet shown in figure 4.67 has to be considered where maximum output voltage value is reported (10 V). So,  $k\_torque$  gain is computed thinking that when the maximum torque value measurable by sensor is sensed the maximum voltage value is emitted. Indeed, it is computed as ratio between maximum voltage value and maximum torque value. Instead, for setting upper and lower limits by saturator Simulink blocks maximum and minimum voltage value that voltage signal can reach at the output of sensor (respectively 10V and -10V) have to be considered.



- *Current sensors DHR 300 C420*

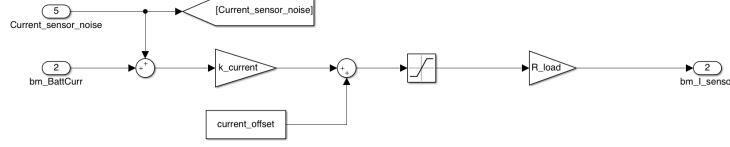


Figure 4.74: Current sensor's Simulink model

To implement this model, sensor datasheet has to be considered which contains implemented transfer function for measuring current. This transfer characteristic, shown in figure 4.75, allows to obtain below reported equation which must be considered to build-up sensor's model as before shown.

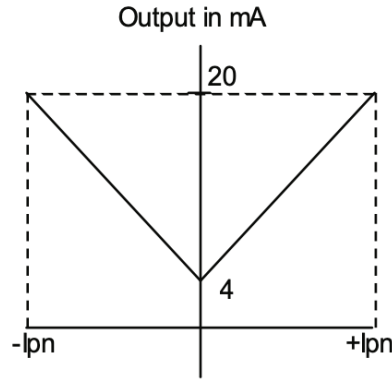


Figure 4.75: Current sensor's transfer characteristic

$$I_{out} = 4 + \frac{16}{300} I_{in} \quad (4.1)$$

So, *k\_current gain* in previous shown Simulink model is set equal to 16/300 while “*current offset*” constant is set equal to 4. Saturator block is implemented to limit current coming out from sensor indeed it is used to set upper limit equal to 20 mA which represents the maximum current value acceptable by sensor. The second gain (*R\_load*) is implemented to model resistance put at sensor's output that allows to convert current signal into voltage signals. Resistance value is taken from datasheet where it is declared smaller than 300 Ω so, in MATLAB script it is defined equal to 250 Ω.

Rotational speed is measured by APICOM FR 250 RPM Measurement System, shown in figure 4.76, which is constituted of a gear wheel having sixty cogs and an rpm detector that is able to send impulse such that rpm can be evaluated by means following relation.

$$RPM = \frac{\left( \frac{\text{Pulse Frequency in pulses}}{\text{sec}} \right) \times \left( 60 \frac{\text{sec}}{\text{min}} \right)}{\left( \frac{\text{Sensor pulses}}{\text{revolution}} \right)} \quad (4.2)$$

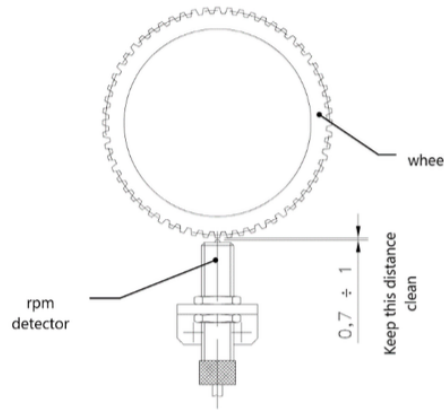


Figure 4.76: APICOM FR 250 RPM measurement system

In Simulink, this system is not modelled since rotational speed values are already given in rad/ s, so first of all they have to be converted in rpm and then in voltage signal. To do this, scheme shown in figure 4.77 is implemented which is constituted of two gains and a sum block: the first gain allows to convert rad/s into rpm, the second one allows to convert rpm into voltage values while sum block allows to consider noise coming from Environment reference model. The value of second gain is computed as ratio between maximum admitted voltage value and maximum measurable rotational speed value.

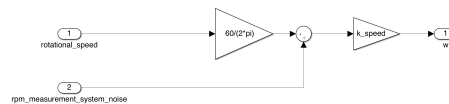


Figure 4.77: RPM measurement system's Simulink model

Always on the right part of plant reference model, besides the two before described subsystems an output port is implemented. It represents interface by means Plant reference model communicates to Control reference model. Indeed, thanks to it Plant sends to Control below listed signals.

- *Bench motor battery voltage signal*

It is a voltage signal used by Plant to communicate to Control reference model measured voltage value provided to bench motor.

- *Test motor battery voltage signal*

It represents voltage signal sent by Plant to communicate to Control reference model measured voltage value provided to test motor.

- *Bench motor phase current signal*

This is a voltage signal allowing Plant to communicate to Control reference model measured bench motor's phase current.

- *Test motor phase current signal*

It is a voltage signal by means Plant sends to Control reference model measured test motor's phase current.

- *Torque signal*

It is a voltage signal coming out from torque sensor that has to be sent to Control reference model.

- *Rotational speed signal*

It represents voltage signal coming out from round per minute measurement system that has to be sent to Control reference model.

Corresponding MATLAB script containing all needed variables to parametrize Simulink blocks implemented in Plant reference model is shown in figure 4.78.

```

14 Plant Data
15
16 %Define the characteristics of the system that has to be controlled
17
18 %Electrical Torque - Nominal & Maximal
19 w_n_bench= 4000 /60*2*pi;
20 T_m_n_bench = 28;
21 eta_n_bench = 0.91 *100; %0 w_n & T_m_n
22 P_m_n_bench = w_n_bench * T_m_n_bench;
23 T_m_max_bench = 55;
24 P_m_max_bench = w_n_bench * T_m_max_bench *1.1;
25 w_max_bench=w_n_bench*1.1;
26 tau_m_bench= 0.02;
27
28 %Electrical Losses - Single Efficiency
29 Pl_iron_bench= 0; %1PM
30 Pl_eln_bench= 100;
31 Pl_copper_n_bench= (1-eta_n_bench/100)*P_m_n_bench -Pl_eln_bench -Pl_iron_bench;
32 k_T_bench= Pl_copper_n_bench /T_m_n_bench^2;
33
34 %Mechanical
35 J_m_bench= .005e-3;
36 beta_m_bench= 1e-5;
37 i_max_bench=T_m_max_bench/k_T_bench;
38
39 %Tabulated_bench_motor
40
41 %Electrical torque section
42 w_t_vec_bench=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593 366.5191 418.8790 471.2389 523.5988 575.9587 628.3185]; %rad/s
43 T_t_vec_bench=[52.5 52.5 52.5 52.5 52.5 52.5 52.5 47 42 38 35]; %Nm
44
45 %Electrical losses section
46 %Tabulated_loss_data
47 w_eff_bp_vec_bench=[184.72 209.44 314.16 366.52 418.88 471.24 523.6 628.32];
48 T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 20.9 26.9];
49 T_m_min_bench=min(T_eff_bp_vec_bench);
50 T_m_max_bench=max(T_eff_bp_vec_bench);
51 w_max_bench=max(w_eff_bp_vec_bench);
52
53 b=2000;
54 losses_table_bench=[52.4 85.1 151.9 126.8 304.9 468.1 712.5;
55 115.1 133.4 223.7 223.6 350.5 701.4 801.2;
56 155.1 104.0 311.2 311.4 693.4 1005.9 1299.9;
57 175.3 240.7 434.4 351.7 556.3 1056.4 1407.6;
58 200.1 244.1 572.9 395.7 813.2 1204.9 1813.4;
59 224.1 342.5 545.2 458.9 732.2 864.7 b;
60 240.1 466.8 490.3 b b b b;
61 268.3 432.3 b b b b b];
62
63 %Tabulated_efficiency_data
64 d=85;
65 efficiency_table_bench=[82.8 85.8 80 91.9 84.5 82.6 84.5;
66 79.6 80.4 90.5 92.2 90.2 86.2 80.2;
67 80.2 93.1 90.9 93.3 87.3 86.7 91.2;
68 80.3 87.4 89.3 93.5 90.9 87.0 91.6;
69 80.2 80.8 88.1 93.5 88.7 88.1 91.9;
70 78.9 88.1 89.2 93.2 90.3 92.4 d;
71 79.5 87.1 81.8 d d d d;
72 78.7 85.2 d d d d d];
73
74 %Test motor
75 %Electrical Torque - Nominal & Maximal
76 w_n_test = 5000 /60*2*pi;
77 P_m_n_test=10500;
78 T_m_n_test = P_m_n_test/w_n_test;
79 V_n_test=66; %V
80 I_n_test=115; %A
81 cos_phi_test=0.86;
82 eta_n_test =(P_m_n_test/(sqrt(3)*V_n_test*I_n_test*cos_phi_test))*100;
83 w_max_test=7500/60*2*pi;
84 T_m_max_test = 63;
85 P_m_max_test = w_max_test * T_m_max_test;
86 tau_m_test = 0.02;
87
88 %Electrical Losses - Single Efficiency
89 Pl_iron_test = 0;
90 Pl_eln_test = 100;
91 Pl_copper_n_test= (1-eta_n_test/100)*P_m_n_test -Pl_eln_test -Pl_iron_test;
92 k_T_test= Pl_copper_n_test /T_m_n_test^2;
93 i_max_test=T_m_max_test/k_T_test;
94
95 %Electrical losses section
96 %Tabulated_loss_data
97 w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471.2389 523.5988 575.9587 628.3185];
98 T_eff_bp_vec_test=[15 24 43 62 83];
99 T_m_min_test=min(T_eff_bp_vec_test);
100 T_m_max_test=max(T_eff_bp_vec_test);
101 w_max_bench=max(w_eff_bp_vec_test);
102
103 losses_table_test=[143.8356 290.4494 954.5455 1.9857e+03 2.8393e+03
104 157.8947 311.1702 972.2222 2.8326e+03 3.5795e+03
105 236.8421 381.5789 1.0419e+03 2.0033e+03 3.6286e+03
106 284.5455 434.2105 1.1200e+03 2.4107e+03 4.0906e+03
107 219.1558 510.5263 1.3784e+03 2.7333e+03 4.2007e+03
108 335.5263 689.3617 1.5326e+03 2.7809e+03 3.8022e+03
109 454.7872 846.7742 1.6781e+03 2.9659e+03 3.5405e+03
110 564.5161 912.1622 1.9111e+03 3.2808e+03 3.1379e+03
111 632.4324 978.2609 1.8758e+03 2.9138e+03 2.8143e+03];
112
113 %Tabulated_efficiency_data
114 efficiency_table_test=[91.25 89 82.5 76.25 70;
115 95 94 90 86.25 82.5;
116 95 95 92.5 90 87.5;
117 96.25 95 93 90 87;
118 96.25 95 92.5 90 86.25;
119 95 94 92 89 86.66;
120 94 93 91.25 88 86.66;
121 93 92.5 90 86.25 87;
122 92.5 92 90 87 87.5];
123
124
125
126
127
128
129

```

Figure 4.78: Plant reference model's MATLAB script

### 4.2.3 Control interface refence model

It represents MTM's part allowing interactions between Control Logic (described in subsection 4.2.3.1) and target hardware. These interactions are allowed including device specifications without affecting control law implemented inside Control Logic reference model to control plant's behavior on the basis of what desired by user.

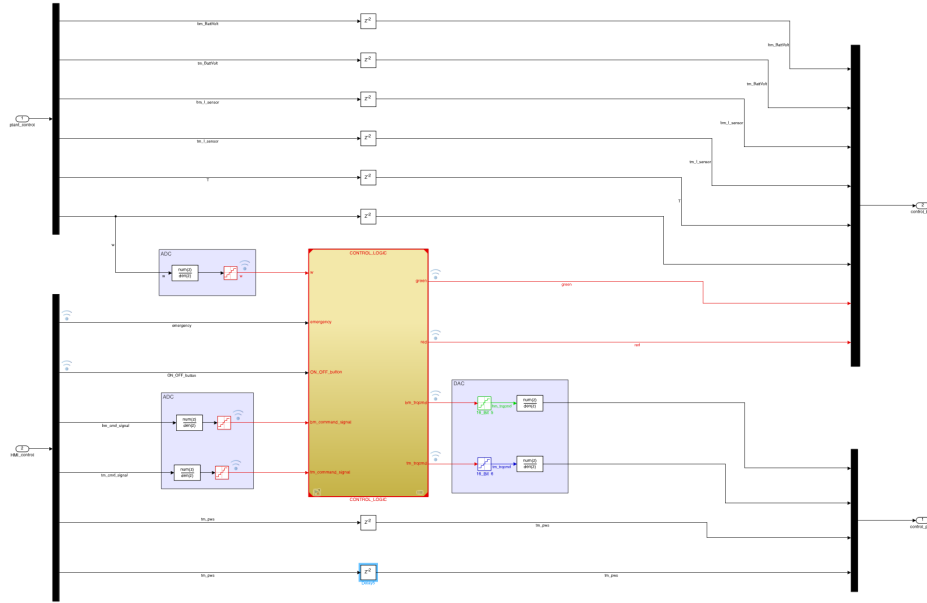


Figure 4.79: Control reference model

Here, on the left part two different inputs are implemented that represent interfaces allowing Control reference model to communicate to Plant and Human-Machine-Interface. Indeed, the first input allows Control to receive signals coming out from Plant such as bench and test motor battery voltage signals, bench and test motor phase current signals, torque and rotational speed signals, while the second one allows Control reference model to receive signals coming out from HMI such as emergency and on/off button signals, bench motor command signal used to pilot bench motor on the basis of rotational speed value decided by user, test motor command signal used to pilot motor under test that is evaluated on the basis of torque value decided by user, bench and test motor power supply signals that represent voltage values to be provided to mapped motor Simulink blocks. Among all these signals there are signals that enter to Control Logic reference model, positioned on the middle, and other that go directly to outputs positioned on the right part. Indeed, between all signals coming from Plant (input 1) only rotational speed signal goes to Control Logic reference model while all other go directly to output 1 that represents interface that allows Control reference model to send signals to HMI subsystem. Instead, between all signals coming from HMI (input 2) only bench and test motor power supply signals go directly to output 2 that represent interface that allows Control reference model to send signals to Plant while all other go to Control Logic reference model. In case of signals going directly to outputs discrete delay block is implemented to avoid algebraic loop errors during simulation due to the fact that signals constantly update their values. Instead in case of signals

that go to Control Logic reference model, Analog to Digital Converters (ADCs) are implemented to discretize signals since Control Logic works only with digital signals. They are implemented using discrete filter and quantizer: the first one allows data averaging since sensors produce high changes in output values that is unacceptable for Control Logic inputs while the second one allows to pass from large to small values' set and to set ADCs' quantization. Quantization parameters is set by means MATLAB script on the basis of maximum voltage values entering to ADCs. Instead in Control Logic outputs, Digital to Analog Converters (DACs) are implemented since an inverse conversion is needed to be performed. This because signals coming out from Control Logic are digital and they have to be sent to plant or HMI where only analog signals are acceptable, so digital to analog conversion is needed. Also in this case, quantizer and discrete filter blocks are used but this time discrete filter is used to implement what is called reconstruct filter whose goal is reconstructing original analog signal starting from digital signal while quantizer block is used for the same reason for which it is used in ADC case. The DACs are not implemented in cases of LED signals (green and red signal) because they are sent to dashboard LEDs that work with digital signal, so in that case conversion is not needed.

Before to run Simulink reference model for performing simulation, corresponding MATLAB script has to be run. This scrip, shown in figure 4.80, contains all needed variables to parametrize Simulink blocks used to implement model.



```

Control Data
Move here to reveal tooltip

157 % ADC
158 % LP filter
159 Ts1=1e-4;
160 wc1=23e3*2*pi;
161 s=tf('s');
162 z=tf('z',Ts1);
163 H_s1=wc1^2/(s+wc1)^2;
164 H_z1=zpk(c2d(H_s1,Ts1,'zoh'));
165 [NUM_LPF,DEN_LPF]=tfdata(H_z1,'v');
166
167 %rotational_speed_sensor
168 Vp_w=10;
169 N=16;
170 quant_w=Vp_w/(2*N-1);
171
172 %bm_trqcnd_signal
173 Vp_bm_trqcnd=5;
174 N=16;
175 quant_bm_trqcnd=Vp_bm_trqcnd/(2*N-1);
176
177 %tm_trqcnd_signal
178 Vp_tm_trqcnd=5;
179 N=16;
180 quant_tm_trqcnd=Vp_tm_trqcnd/(2*N-1);
181
182 %bm_cmd_signal
183 Vp_bm_command_signal=5;
184 N=16;
185 quant_bm_cmd_signal=Vp_bm_command_signal/(2*N-1);
186
187 %tm_cmd_signal
188 Vp_tm_cmd_signal=5;
189 N=16;
190 quant_tm_cmd_signal=Vp_tm_cmd_signal/(2*N-1);
191
192 % DAC
193 % LP filter
194 Ts2=1e-4;
195 wc2=500e3*2*pi;
196 s=tf('s');
197 z=tf('z',Ts2);
198 H_s2=wc2^2/(s+wc2)^2;
199 H_z2=zpk(c2d(H_s2,Ts2,'zoh'));
200 [NUM_LPF2,DEN_LPF2]=tfdata(H_z2,'v');

```

Figure 4.80: Control reference model's MATLAB script

#### 4.2.3.1 Control Logic reference model

Inside this reference model, control law that controls and manages plant's behavior is implemented in such a way that user requirements are satisfied. It works only with discrete signals and manages bench and test torque commands to pilot mapped motor Simulink blocks implemented inside Plant reference model. These commands must be evaluated on the basis of rotational speed and torque values set at beginning by user.

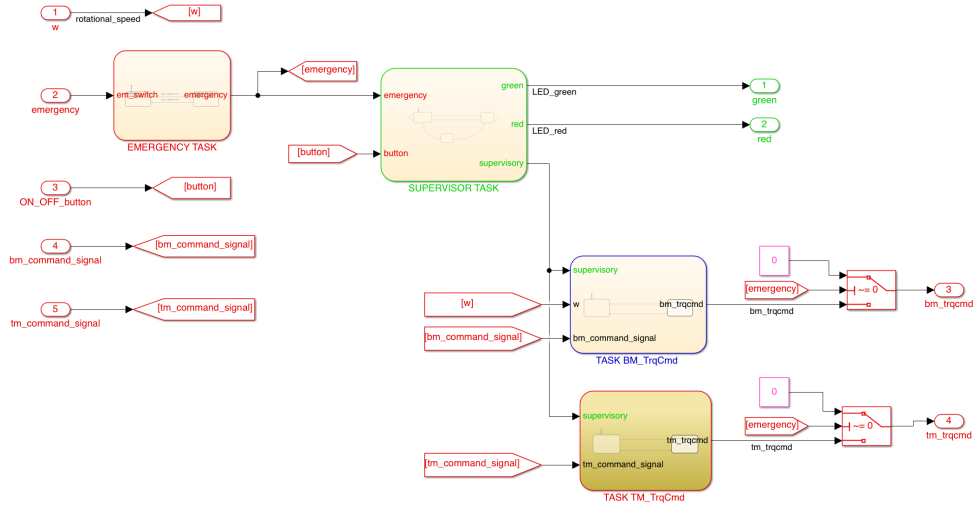


Figure 4.81: Control logic reference Simulink model

Here, on the left part five inputs are implemented which represent interfaces by means Control Logic communicates to Control reference. Indeed, by means these inputs Control Logic receives signals coming from Control interface reference model such as rotational speed signal, emergency and on/off button signals and bench and test motor torque command signals. Instead, on the right part four outputs are implemented that allow Control Logic to send signals to Control interface such as green and red signals that must be sent to dashboard LEDs and bench and test torque command that have to be sent to Plant. In the middle four below list and described tasks are implemented using *control logic Stateflow tool* to define control law to be used by ECU to control and manage plant's behavior on the basis of what desired by user.

- *Emergency task*

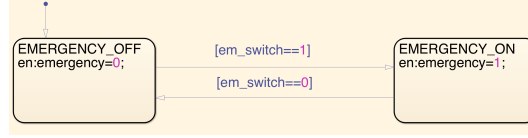


Figure 4.82: Emergency task's Simulink Stateflow

It allows user to stop system by an opportune dashboard switch in case something goes wrong during test. Here, two states are implemented: *EMERGENCY\_OFF* that represents initial state during which emergency variable is set equal to 0 and *EMERGENCY\_ON* that is responsible to set emergency variable equal to 1. To switch from initial to second state or vice versa, defined conditions over arrows connecting two states have to be verified:  $em\_switch == 1$  which means that switch from initial state to *EMERGENCY\_ON* state happens when emergency dashboard switch is toggled from 0 to 1 while inverse switch happens when  $em\_switch == 0$  condition is verified which means that to have switch from *EMERGENCY\_ON* to *EMERGENCY\_OFF* emergency dashboard switch has to be toggled from 1 to 0.

- *Supervisor task*

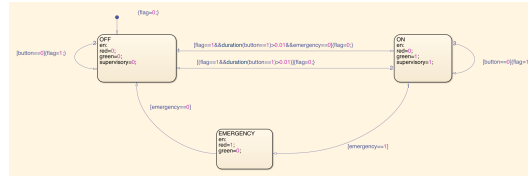


Figure 4.83: Supervisor task's Simulink Stateflow

It allows the user to pilot and check system's behavior by using dashboard pushbutton and LEDs. It is constituted of below listed states.

- *OFF state*

It represents initial state, set at beginning when system is not working and user is not using dashboard. Thanks to this state red, green and supervisor signals are set equal to 0 and nothing happen until  $[button == 0]\{flag = 1;\}$  condition is verified. This means that until on/off dashboard button is not pushed by user, task state does not change and no change happens ( $\{flag = 1;\}$ ). State's change happens when  $[flag == 1 \&\& duration(button == 1) > 0.01 \&\& emergency == 0]\{flag = 0;\}$  condition is verified which means that change happens when on/off pushbutton is



pushed by user for a duration greater than 0.01s and emergency switch state remains always 0. At this point some change is verified ( $\{flag = 0; \}$ ) and task state switches from OFF to ON.

– *ON state*

Once task state switch in ON, red signal remains set equal to 0 while green and supervisory signals will be set equal to 1 such that green dashboard LED will switch on and *TASK BM\_TrqCmd* and *TASK TM\_TrqCmd* are activated. This task state can change returning in OFF or switching in EMERGENCY on the basis of what happens. Indeed, if  $[(flag == 1 \&\&duration(button == 1) > 0.01)]\{flag = 0; \}$  condition is verified, task state return to OFF which means that if any variation happens and on/off dashboard button is pushed for a duration greater than 0.01s, task state switches from ON to OFF while if  $[emergency == 1]$  condition is verified, which means that emergency dashboard switch has to be toggled from 0 to 1, task state switches from ON to EMERGENCY. This state does not change if  $[button == 0]\{flag = 1; \}$  condition is verified which means that task state does not change if on/off dashboard button is not pushed.

– *EMERGENCY state*

Once  $[emergency == 1]$  condition is verified, task state switches from *ON* to *EMERGENCY* where red signal will be set equal to 1 in order to switch on red dashboard LED and green signal will be set equal to 0 in order to switch off green dashboard LED. This task state changes when  $[emergency == 0]$  condition is verified which means that emergency dashboard switch has to be toggled from 1 to 0. At that point task state return to be OFF in which red, green and supervisory signals return to be equal to 0.

• *Bench motor torque command task*

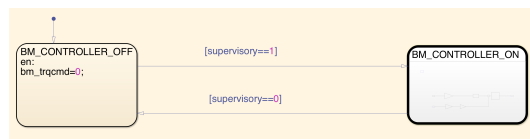


Figure 4.84: *BM\_TrqCmd\_task*'s Simulink stateflow

It allows to evaluate torque command to be used for piloting bench motor on the basis of user's desired rotational speed value. It is constituted of below listed states.

– *BM\_CONTROLLER\_OFF state*

It represents task state active when supervisor signal is set equal to 0, so when system does not work. In this state, bench motor torque command is set equal to 0 ( $bm\_trqcmd = 0$ ) such that bench motor is kept stopped. When  $[supervisory == 1]$  condition is verified, upervisor signal switch from 0 to 1, task state switches from *BM\_CONTROLLER\_OFF* to *BM\_CONTROLLER\_ON*.

– *BM\_CONTROLLER\_ON state*

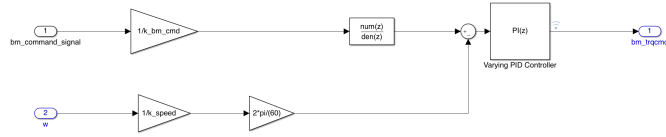


Figure 4.85: *BM\_CONTROLLER\_ON* state's Simulink model

This state is used once  $[supervisory == 1]$  condition is verified since at that point bench motor must be piloted by a torque command that has to be computed on the basis of user's desired rotational speed value. Here, on the left part two inputs are implemented to receive bench motor command signal ( $bm\_command\_signal$ ) carrying rotational speed value set by user and rotational speed signal ( $w$ ) carrying rotational speed value coming from feedback path.  $bm\_command\_signal$  goes through a gain in order to convert voltage signal into rad/s signal and then it goes through discrete filter to perform data averaging since PID controller does not accept high changes in input variable. Instead,  $w$  signal goes through two gains: the first one allows conversion of voltage signal, coming out from round per minute measurement system, into rpm signal while the second one allows conversion of rpm in rad/s. Once conversions are done, the two obtained results can be added each other to obtain error that has to enter to PI controller block. On the basis of this error PI controller is able to evaluate bench motor torque control signal needed to be provided for obtaining desired rotational speed value at motor shaft's output in order to have an error equal to zero. This task state changes once  $[supervisory == 0]$  condition is verified, supervisory signal switches from 1 to 0, and at that point task state returns to *BM\_CONTROLLER\_OFF* state.

- *Test motor torque command task*

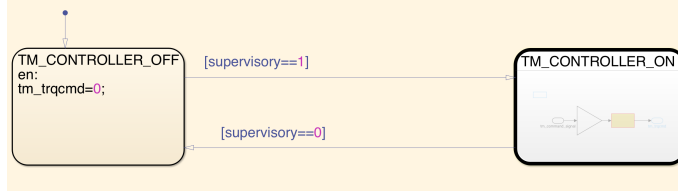


Figure 4.86: TM\_TrqCmd\_task's Simulink model

It allows to evaluate torque command to be used for piloting test motor on the basis of user's desired torque value. It is constituted of below listed states.

- *TM\_CONTROLLER\_OFF state*

It represents task state active when supervisor signal is set equal to 0, so when system does not work. This state sets test motor torque command equal to 0 ( $tm\_trqcmd=0$ ) so that test motor is kept stopped. When  $[supervisory == 1]$  condition is verified, supervisor signal switches from 0 to 1, task state switches from *TM\_CONTROLLER\_OFF* to *TM\_CONTROLLER\_ON*.

- *TM\_CONTROLLER\_ON state*

Action

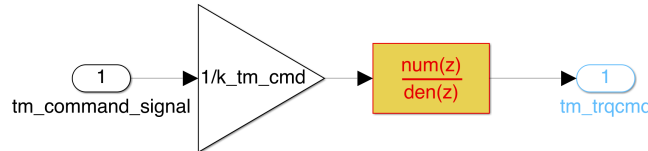


Figure 4.87: TM\_CONTROLLER\_ON state's Simulink

This state is considered once  $[supervisory == 1]$  condition is verified since at that point test motor has to be piloted by torque command that has to be evaluated on the basis of torque value set by user. Here, on left part a single input is implemented in order to receive test motor command signal ( $tm\_command\_signal$ ) carrying torque value set by user. This is a voltage signal going through a gain in order to be converted into Nm signal that represent discrete filter's input signal such that data averaging is performed and high changes in output signal ( $tm\_trqcmd$ ) are avoided. This task state changes when supervisory signal is switched from 1 to 0. At that point task state return to be *TM\_CONTROLLER\_OFF*.

Inside Control Logic model besides what before described, two different switches are implemented that allow to set equal to 0 bench and test motor torque command once emergency signal becomes equal to 1 so, when user toggles emergency dashboard switch from 0 to 1. This allows to stop system in cases something wrong happens, and it is better to stop system in order to avoid serious problem for plant elements.

To parametrize all block used for implementing this reference model, a MATLAB script containing all needed variables has to be implemented and run before to start running Simulink model.

**Control Logic Data**

The Control Logic must contain the control algorithm that is intended to be deployed on the target hardware.

```

196 sampling_time=1e-2;
197 s=tf('s');
198 z=tf('z',sampling_time);
199 H_s=1/(3*s+1);
200 H_z=zpk(c2d(H_s,sampling_time,'zoh'));
201 [NUM,DEN]=tfdata(H_z,'v');
202
203 %w_signal
204 Vp_w=10;
205 rpm_max=6000;
206 k_speed=Vp_w/rpm_max;
207
208 %bm_cmd_signal
209 Vp_bm_cmd_signal=5;
210 bm_cmd_signal_max=6000*2*pi/60; %rad/s
211 k_bm_cmd=Vp_bm_cmd_signal/bm_cmd_signal_max;
212
213 %tm_cmd_signal
214 Vp_tm_cmd=5; %V
215 tm_cmd_max=100; %Nm
216 k_tm_cmd=Vp_tm_cmd/tm_cmd_max;
217

```

Figure 4.88: Control Logic reference model's MATLAB script

#### 4.2.4 Human-Machine-Interface subsystem

It allows the user to send command to plant and monitoring what happens while system works. This part is implemented as subsystem that allows to have a better elements' arrangement such that a more intelligible model is obtained. This reference model, whose scheme is shown in figure 4.89, is constituted of two inputs that represent interfaces by means HMI is able to receive signals from Control reference model and from User subsystem. Indeed, the first input allows HMI to receive height signals coming from Control reference model such as bench and test motor battery voltage signals that are connected to displays that allow user to check whether two motors are provided by right voltage value (the one set at beginning by user), bench and test phase current signals coming from current sensors, torque signal coming from torque sensor, rotational speed signal coming from rounds per minute measurement system and green and red signals connected to displays in order to allow user to check their status (1 means LED on while 0 means

LED off). Instead, the second input allows HMI to receive signals from User subsystem such as bench and test motor power supply values that have to be sent to plant to set properly inverters that provide motors.

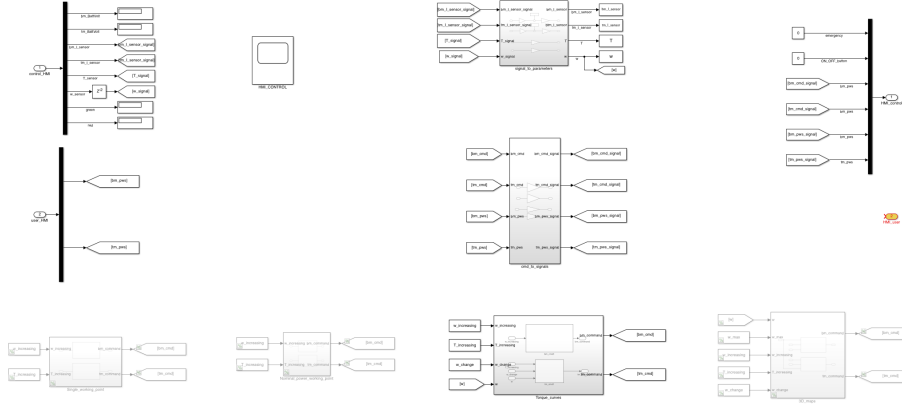


Figure 4.89: HMI's Simulink model

Here, *signal\_to\_parameters* subsystem is implemented to convert voltage signals coming out from Control reference model into user interesting parameters such as bench and test motor phase currents, torque and rotational speed. Inside this subsystem model shown in figure 4.90 is implemented where in the first two rows inverse relation considered in current sensor model case, below reported, is implemented that allows to compute current values starting from voltage signals coming out from current sensors. Before to apply this relation, voltage signals coming out from current sensors must be translated in current signals. For this reason, a gain whose value is defined as the inverse of current sensor's output resistance is implemented.

$$I_{in} = (I_{out} - 4) * \frac{300}{16} \quad (4.3)$$

The same thing happens in torque and rotational speed signals case since they are voltage signals coming out from torque sensor and rounds per minute measurement system that have to be converted into Nm and rpm values. These conversions are done by gains whose values are computed as ratio between maximum measurable torque and rpm values and the maximum voltage values that can be provided at the output of corresponding sensor. These values have to be defined in corresponding MATLAB script.

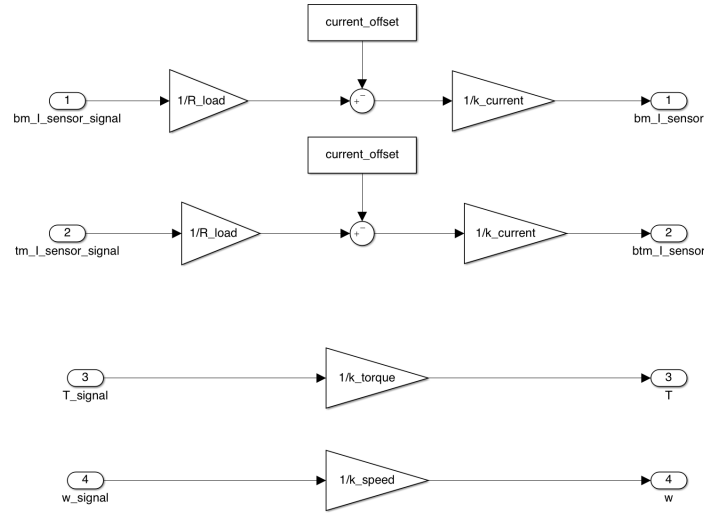


Figure 4.90: Signal\_to\_parameters subsystem Simulink model

Then, *cmd\_to\_signal subsystem* is implemented to convert bench and test motor commands, carrying rotational speed and torque values to be considered for piloting bench and test motors (computed by subsystems reported on bottom part of HMI's model on the basis of what is desired by user), in voltage signals to be sent to Control reference model. The same thing happens for bench and test motor power supply values. This, as shown in figure 4.91, is done by gains whose values are computed as ratio between voltage signals' peak value and the maximum reachable value of corresponding parameters (bench motor rotational speed command, test motor torque command, bench motor power supply, test motor power supply).

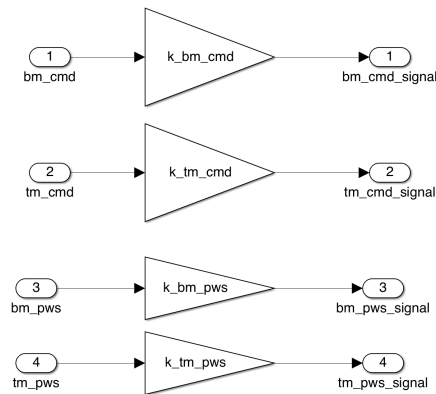


Figure 4.91: Cmd\_to\_signal subsystem's Simulink model

On right part (figure 4.89), two output ports are implemented: the first one represents interface by means HMI reference model sends signals to

Control interface reference model while the second one represents interface by means HMI is able to send signals to User subsystem (in this case it is not used). Among signals sent to Control reference model there is: emergency and on/off button signals that are digital signal because they can assume either 0 or 1 as value coming out from constant blocks linked to dashboard switch and push button, bench and test motor command signals and bench and test power supply signals. Instead, on bottom part, four other subsystems are implemented by means user is able to perform a different test to test motors. This because, user requires an electrical motors' test bench by means electric motors can be tested both for a specific selected working point and for obtaining their torque curves and 3D maps (efficiency 3D map and dissipated power 3D map).

Observing *Speed-Torque plane*, shown in figure 4.92, different working points can be selected that can be selected by means a pair of rotational speed and torque values. User would like to have the chance to test motors in different ways: selecting a single working point, selecting a single torque value and different rotational speed values in order to consider different working points and obtaining torque curves, selecting different torque and different rotational speed values in order to cover all admissible w-T plane and obtaining 3D maps, selecting different torque values and different rotational speed values for reaching working point where motor works at nominal power which is defined by nominal rotational speed value and nominal torque value.

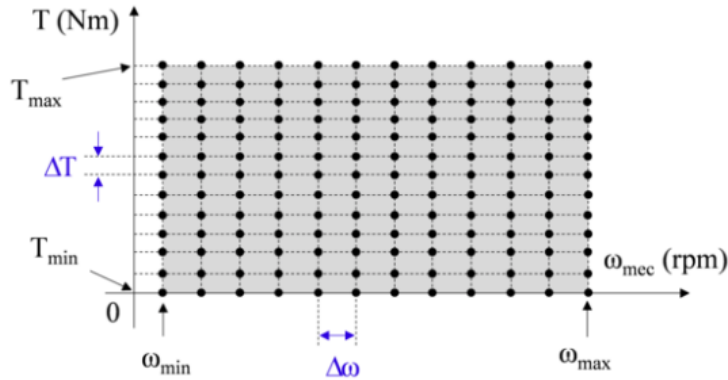


Figure 4.92:  $\omega - T$  plane

So, different tests can be performed to evaluate test motor characteristics and for this reason inside HMI subsystem, on bottom part, four subsystems are implemented by means user can choose the test to be performed.

- *Single working point test*

In this scenario, user must define a single torque and a single rotational speed value in order to select working point to be considered for evaluating test motor characteristics. To perform this kind of test, user must uncomment *Single\_working\_point subsystem* and comment out other three subsystems. This subsystem contains model shown in figure 4.93 that is constituted of two subsystems: *bm\_cmd3* and *tm\_cmd3*.

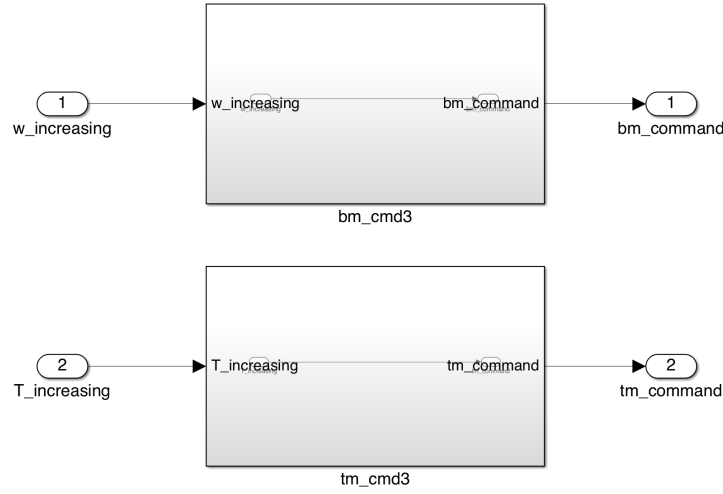


Figure 4.93: *Single\_working\_point* subsystem's Simulink model

*bm\_cmd3 subsystem* is intended to evaluate bench motor command and it is constituted of a single input and a single output. Input represents rotational speed value set by user to pilot bench motor and it is directly connected to the output since it does not need any operation to become a bench motor command.

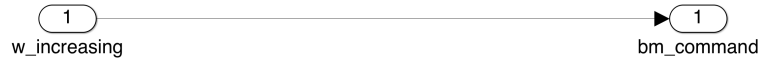


Figure 4.94: *bm\_cmd3* subsystem's Simulink model

Instead, *tm\_cmd3 subsystem* is intended to evaluate test motor command and it is constituted of a single input representing torque value defined by user to pilot test motor and a single output representing test motor command to be sent to Control reference model after conversion in voltage signal. They are connected directly each other since input does not need any transformation to become test motor command.



Figure 4.95: *tm\_cmd3* subsystem's Simulink model

To perform this test, user must define needed variables in corresponding MATLAB script and run it before to start simulation so that they are load on MATLAB workspace and Simulink blocks can use them to be parametrized during simulation. Before to run MATLAB script, user must define some parameters' value such as *w\_increasing* and *T\_increasing* that represents respectively rotational speed and torque values to be considered for selecting working point to be considered for performing test, *T\_sim* representing simulation time and *sample* that represents sample time to be considered for storing interesting variables to be considered for computing desired motor parameters and characteristics.

```

364
365
366
367
368
369
370

```

**Single\_working\_point**

```

% bm_cm3 & tm_cmd3 blocks
w_increasing=3500/60*2*pi; %rad/s
T_increasing=20; %Nm
T_sim=100; %s

sample=0.1; %sample time of variables to be stored

```

Figure 4.96: Single working point's MATLAB script

- *Nominal power working point*

In this case, the goal is reaching the working point in which motor works at nominal power. This means that working point to be reached is the one corresponding to nominal rotational speed and nominal torque values that are reached increasing time by time both rotational speed and torque values. To perform this kind of test, user must uncomment *Nominal\_power\_working\_point subsystem* and comment out other three subsystems. This subsystem contains model shown in figure 4.97 that is constituted of two subsystems: *bm\_cmd2* and *tm\_cmd2*.

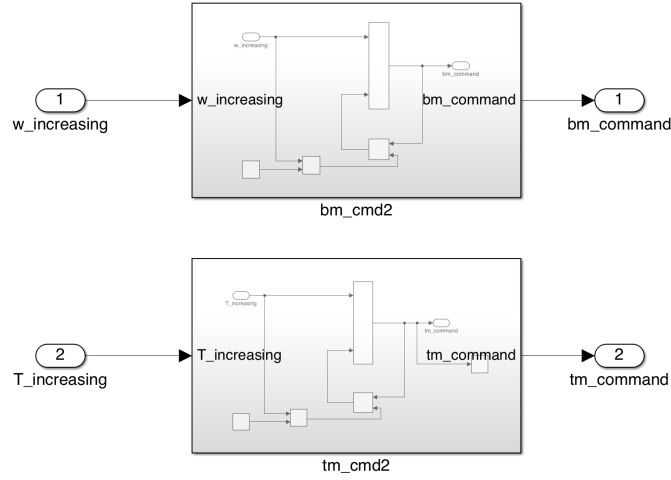
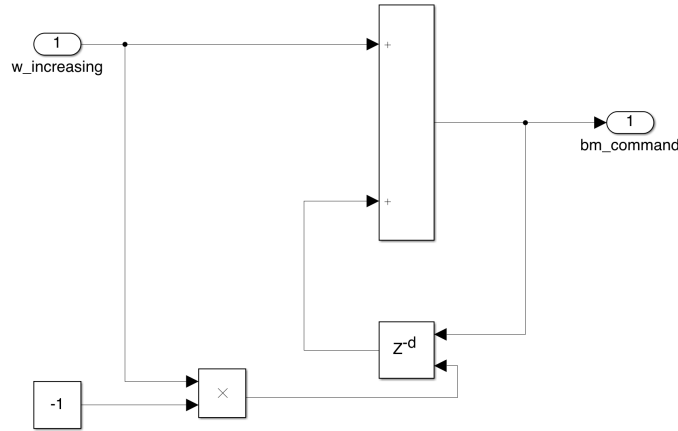
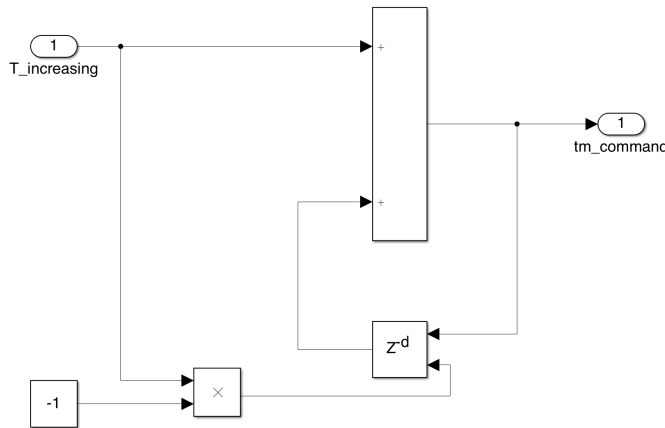


Figure 4.97: Nominal\_power\_working\_point subsystem's Simulink model

*bm\_cmd2 subsystem* is intended to evaluate bench motor command and it is constituted of a single output representing rotational speed value to be considered for piloting bench motor. Since minimum and maximum rotational speed values are known user can define an increment to be considered for moving along rotational speed axis and selecting new working point where evaluating test motor characteristics. This increment can be computed as ratio between nominal rotational speed value and rotational speed values' number to be considered to move from initial to final rotational speed value. It can be used to compute time by time new working point since it can be added to previous rotational speed for obtaining new rotational speed value to be considered for selecting new working point. In Simulink this logic is implemented by means model shown in figure 4.98 where a sum block is implemented with an input port connected to a constant block (computed increment) and other input connected to discrete delay's output. Discrete delay's output value changes each time a time delay is expired such that every time rotational speed value to be considered for computing bench motor command is computed as the sum between previous rotational speed value and rotational speed increment. Time delay is defined by user on the basis of transient phase that characterizes interesting parameters' trend (time needed to reach steady- state value). Sum Simulink block's output represents bench motor command that must be sent to Control reference model after being converted in voltage signal. Moreover, discrete delay's output is defined by means an external port which is connected to product block's output port that allows to compute increment opposite value such 0 rad/ s is the first rotational speed value considered to select a working point.


 Figure 4.98: *bm\_cmd2* subsystem's Simulink model

Instead, *tm\_cmd2 subsystem* is intended to evaluate test motor command and it is implemented in the same way of *bm\_cmd2 subsystem*. But, this time instead of considering rotational speed, torque is considered to evaluate test motor command that has to be sent to Control reference model after being converted in voltage signal. Indeed, also in this case an increment is defined to move along torque axis for selecting new working point. This increment is defined as ratio between nominal torque value and torque values' number to be considered to move from initial to final torque value. It can be used to compute time by time new working point since it can be added to previous torque value to obtain new torque value to be considered for selecting new working point. In Simulink, this is implemented by means model shown in figure 4.99 that is the same to that described in *bm\_cmd2* case.


 Figure 4.99: *tm\_cmd2* subsystem's Simulink model

To parametrize Simulink blocks implemented in `Nominal__power__working__point` subsystem a MATLAB script, containing all needed variables, has to be defined and run in order to load parameter values into MATLAB workspace. Among all these variables there is *w\_increasing* and *T\_increasing* which represent respectively rotational speed and torque increment defined as before described, *delay\_T* representing time delay to be considered for discrete delay implemented in `tm_cm2` subsystem whose value is decided by user, *delay\_w* that represents time delay to be considered for discrete delay implemented inside `bm_cmd2` subsystem whose value is defined in function of *delay\_T*, number of rotational speed and of torque values, *sample* defining sample time to be considered for ToFile Simulink blocks used to store interesting variables and *T\_sim* that represents simulation time duration computed as function of *delay\_T* and number of torque values considered for performing test.

**Nominal\_power\_working\_point**

```

348 w_n_test = 5000 / 60 * 2 * pi;
349 w_sampling = 10;
350 w_increasing = w_n_test / w_sampling;
351
352 P_m_n_test = 10500;
353 T_m_n_test = P_m_n_test / w_n_test;
354 T_sampling = 10;
355 T_increasing = T_m_n_test / T_sampling;
356
357 %bm_cm2 & tm_cmd2 blocks
358 delay_T = 25;
359 delay_w = (T_sampling * delay_T) / w_sampling;
360
361 sample = 0.1;
362 T_sim = delay_T + (T_sampling * delay_T)

```

Figure 4.100: Nominal power working point's MATLAB script

- *Torque curves test*

In this scenario, user must define a single torque value and a vector of rotational speed values in order to select different working points for which evaluating test motor behavior and characteristics. To perform this kind of test user must uncomment *Torque\_curves subsystem* and comment out other three subsystems. This subsystem contains model shown in figure 4.101 that is constituted of two different subsystems: *bm\_cmd1* and *tm\_cmd1*.

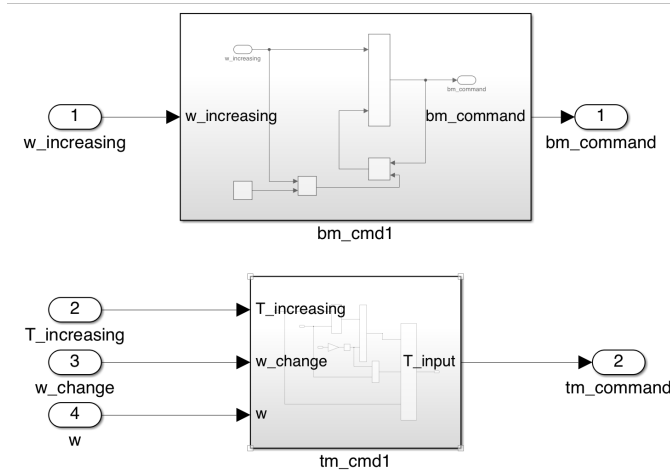
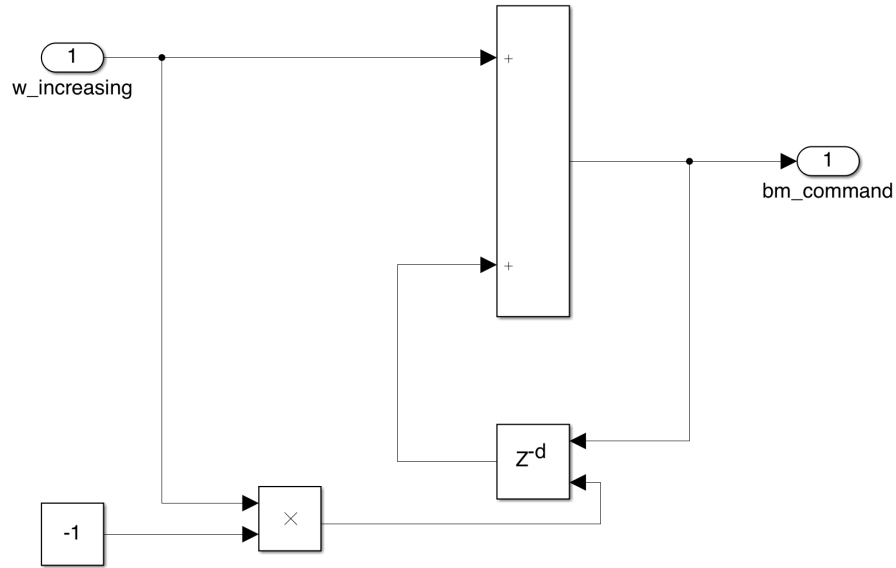
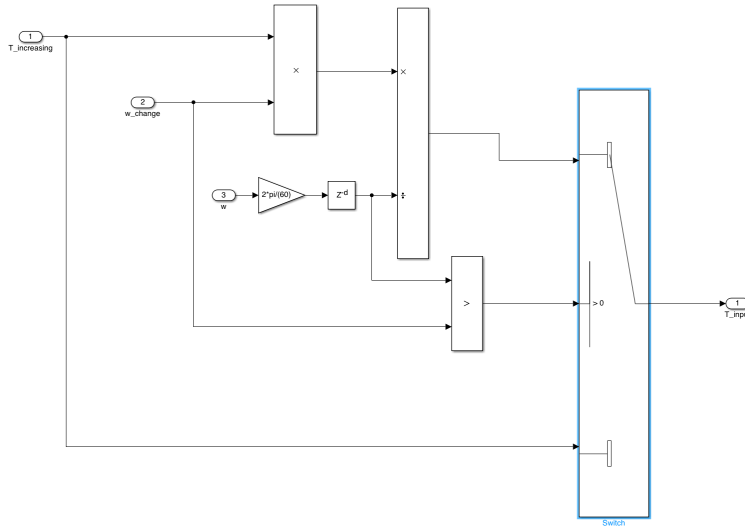


Figure 4.101: Torque curves subsystem's Simulink model

*bm\_cmd1 subsystem* is intended to evaluate bench motor command and it is constituted of a single output representing rotational speed value to be considered for evaluating bench motor command signal. Since minimum and maximum rotational speed values are known user can define an increment to be considered for moving along rotational speed axis and selecting new working point where evaluating test motor characteristics. This increment can be computed as ratio between maximum and minimum rotational speed value difference and rotational speed values' number to be considered to move from initial to final rotational speed value. It can be used to compute time by time new working point since it can be added to previous rotational speed for obtaining new rotational speed value which corresponds to new working point. In Simulink, this logic is implemented by means model shown in figure 4.102 where sum block is implemented with an input port connected to a constant block (rotational speed increment) and other input connected to discrete delay's output whose value changes each time a time delay is expired in order to compute rotational speed value as sum between previous rotational speed value and rotational speed increment. Time delay is defined by user on the basis of transient phase that characterizes interesting parameters' trend (time needed to reach steady-state value). Sum Simulink block's output represents bench motor command that must be sent to Control reference model after being converted in voltage signal. Moreover, discrete delay's initial output is defined by means an external port which is connected to product block's output port that allows to compute increment opposite value such 0 rad/s is the first rotational speed value considered to select a working point.

Figure 4.102: *bm\_cmd1* subsystem's Simulink model

Instead, *tm\_cmd1 subsystem* is intended to evaluate test motor command and this time it is constituted of a more complex model. Indeed, this time on right part Switch Simulink block is implemented that allows to select torque value to be considered for selecting working point. This because two different working areas in  $\omega - T$  plane can be distinguished: constant torque working area in which motor works at constant torque value and constant power working area where motor works at constant power. Switch from one area to other one happens at nominal rotational speed and at that point test motor torque command must be computed in such a way that its power remains constant. In Simulink model, condition to switch from one area to other one is set by a GreaterThan Simulink block that allows to compare rotational speed value read by rounds per minute measurement system and  $w\_change$  that represents rotational speed value at which motor working condition must change and once switch happens test motor torque command is computed dividing the result of product between considered torque value and  $w\_change$  for rotational speed sensed by rounds per minute measurement system that must be converted in rad/s.


 Figure 4.103: *tm\_cmd1* subsystem's Simulink model

To parametrize Simulink blocks implemented inside *Torque\_curves subsystem* a corresponding MATLAB script has to be defined and run before to start Simulink simulation in order to load variables in workspace. Among all variables to be defined there are *w\_eff\_bp\_vec\_bench* and *w\_eff\_bp\_vec\_test* representing rotational speed values' vector to evaluate bench and test motor power loss matrix, *w\_sampling* representing number of rotational speed values to be considered during test, *w\_increasing* representing rotational speed increment, *T\_increasing* representing torque value to be considered for performing test, *w\_n\_test* representing test motor nominal rotational speed, *delay\_w* that defines for how long test motor must be tested in each working point, *sample* representing sample time to be considered for ToFile Simulink blocks used to store interesting variables.

```

Torque_curves
348 %w_increasing
349 w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6 628.32];
350 w_max_bench=max(w_eff_bp_vec_bench);
351 w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471.2389 523.5988 575.9587 628.3185];
352 w_max_test=max(w_eff_bp_vec_test);
353 w_max=min(w_max_bench,w_max_test);
354
355 w_sampling=20;
356 w_increasing=w_max/w_sampling;
357
358 %T_increasing
359 T_increasing=20;
360
361 %w_change
362 w_n_bench= 4000 /60*2*pi;
363 w_n_test = 5000 /60*2*pi;
364 w_change=min(w_n_bench,w_n_test); %rotational speed at which the area change happens
365
366 %bm_cmd1
367 delay_w=25;
368
369 sample=0.1; %sample time for interesting parameters to be stored
370 T_sim=(delay_w+(delay_w*w_sampling))
    
```

Figure 4.104: Torque curves' MATLAB script

- *3D maps tests*

In this case, test motor's behavior has to be evaluated in entire coverable area of  $\omega - T$  plane so, a very huge number of working points has to be considered. For this reason, both torque and rotational speed values time by time have to be changed in order to select working points belonged to entire coverable  $\omega - T$  plane area and test motor characteristics can be evaluated. Usually, to do this a constant torque value is considered and time by time rotational speed value is increased until its maximum value is reached. At that point, torque value has to be increased and the same procedure for rotational speed value has to be repeated again in order to select other working points and test motor characteristics can be evaluated. Test must be stopped once maximum torque value is reached to avoid mechanical and electrical problems for electric motor under test. To perform this kind of test user must uncomment *3D\_maps subsystem* implemented on the bottom part of HMI subsystem and comment out other three subsystem. This subsystem contains model shown in figure 4.105 and it is constituted of two subsystems: *bm\_cmd0* and *tm\_cmd0*.

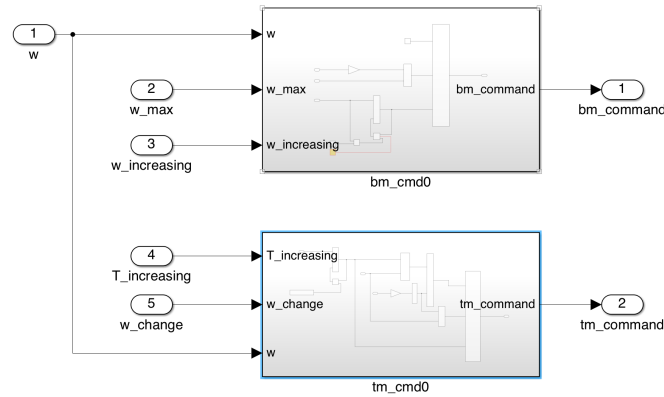
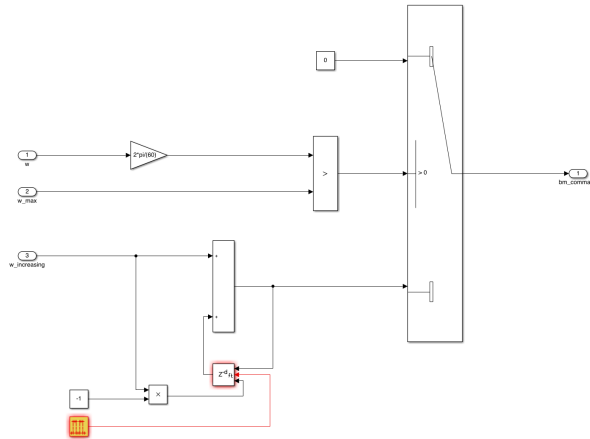


Figure 4.105: *3D\_maps* subsystem's Simulink model

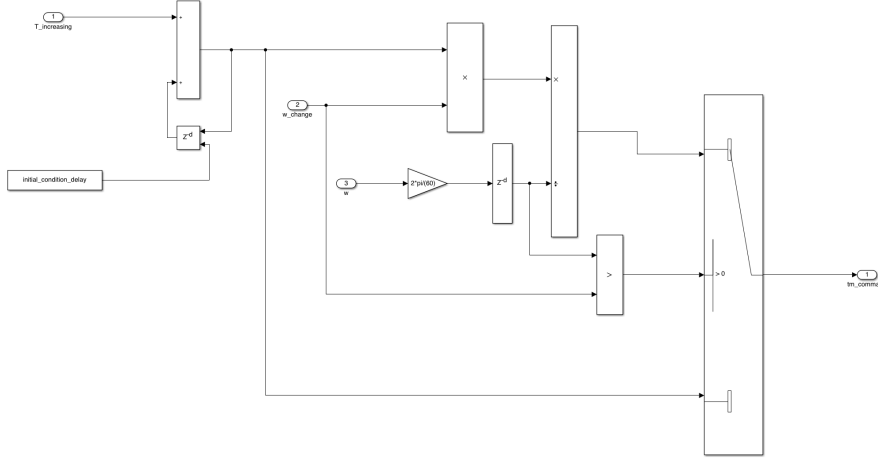
*bm\_cmd0 subsystem* is intended to evaluate bench motor command and it is constituted of a Switch Simulink block which allows to select rotational speed value to be considered for evaluating bench motor torque command. This because rotational speed value has to increase until to reach its maximum value and once it is crossed rotational speed value has to set equal to initial rotational speed value (0 rad/s). Switch condition is set by means GreaterThan Simulink block that allows to compare rotational speed value sensed by rounds per minute measurement system and *w\_max* that represents maximum rotational speed value that can be considered for testing electric motor under test.



Since minimum and maximum rotational speed values are known user can define an increment to be considered for moving along rotational speed axis and selecting new working point where evaluating test motor characteristics. This increment can be computed as ratio between maximum and minimum rotational speed value difference and rotational speed values' number to be considered to move from initial to final rotational speed value. It can be used to compute time by time new working point since it can be added to previous rotational speed for obtaining new rotational speed value which corresponds to new working point. In Simulink, this logic is implemented by means model shown in figure 4.106 where sum block is implemented with an input port connected to a constant block (rotational speed increment) and other input connected to discrete delay's output whose value changes each time a time delay is expired in order to compute rotational speed value as sum between previous rotational speed value and rotational speed increment. Time delay is defined by user on the basis of transient phase that characterizes interesting parameters' trend (time needed to reach steady-state value). Sum Simulink block's output represents bench motor command that must be sent to Control reference model after being converted in voltage signal. Moreover, discrete delay's initial output is defined by means an external port which is connected to product block's output port that allows to compute increment opposite value such 0 rad/s is the first rotational speed value considered to select a working point while delay reset is implemented by means a pulse generator that each time maximum rotational speed value is reached allows to reset delay block. This is done taken into care how long each rotational speed value lasts to stay in specific working point and evaluating test motor's characteristics.


 Figure 4.106: *bm\_cmd0* subsystem's Simulink model

Instead, *tm\_cmd0 subsystem* is intended to evaluate test motor command and this time it is constituted of a more complex model. Indeed, this time on right part Switch Simulink block is implemented that allows to select torque value to be considered for selecting working point. This because two different working areas in  $\omega - T$  plane can be distinguished: constant torque working area in which motor works at constant torque value and constant power working area where motor works at constant power. Switch from one area to other one happens at nominal rotational speed and at that point test motor torque command must be computed in such a way that its power remains constant. In Simulink model, condition to switch from one area to other one is set by a GreaterThan Simulink block that allows to compare rotational speed value read by rounds per minute measurement system and  $w\_change$  that represents rotational speed value at which motor working condition must change and once switch happens test motor torque command is computed dividing the result of product between considered torque value and  $w\_change$  for rotational speed sensed by rounds per minute measurement system that must be converted in rad/s. When motor stays in constant torque working area, torque value to be considered for computing test motor torque command is computed by a Sum Simulink block that allows to sum torque increment, computed as ration between maximum and minimum torque value difference and torque values' number to be considered to test motor, and discrete delay's output whose value changes each time a time delay is expired in order to compute torque value as sum between previous torque value and torque increment. Time delay is defined by user on the basis of transient phase that characterizes interesting parameters' trend (time needed to reach steady-state value). Sum Simulink block's output represents bench motor command that is connected to Switch Simulink block. Moreover, discrete delay's initial output is defined by means an external port that is connected to a Constant Simulink block whose value is defined by user through corresponding MATLAB script (*initial\_condition\_delay*) such that user is able to decide whether 0Nm value has to be considered to select working points or not.


 Figure 4.107: *tm\_cmd0* subsystem's Simulink model

To parametrize all Simulink blocks implemented inside *3D\_maps subsystem* a corresponding MATLAB script must be defined and run before to start Simulink simulation such that all variables are load into workspace and Simulink blocks can use them. Among all needed variables there are:  $w\_eff\_bp\_vec\_bench$  and  $w\_eff\_bp\_vec\_test$  that represent vectors containing rotational speed values considered to evaluate bench and test motor power and efficiency matrices,  $w\_sampling$  defining how many rotational speed values have to be considered between minimum and maximum rotational speed values during test,  $w\_increasing$  that represents rotational speed increment,  $T\_sampling$  that defines how many torque values must be considered during test between minimum and maximum torque values,  $T\_eff\_bp\_vec\_bench$  and  $T\_eff\_bp\_vec\_test$  that represent vector containing torque values considered to evaluate bench and test motor power and efficiency matrices,  $w\_n\_bench$  that represent bench motor nominal rotational speed,  $w\_n\_test$  that represents test motor nominal rotational speed,  $w\_change$  that represents rotational speed value at which motor switches from constant torque working area to constant power working area,  $delay\_w$  and  $delay\_T$  that represents time duration to be considered for each rotational speed and torque value,  $pulse\_generator\_period$  and  $pulse\_generator\_phase\_delay$  that represent variables to parametrize pulse generator block used to reset discrete delay of *bm\_cm0* subsystem,  $initial\_condition\_delay$  that represents initial output value of discrete delay implemented in *tm\_cmd0* subsystem,  $sample$  representing sample time to be considered for ToFile Simulink block used to store interesting variables,  $T\_sim$  that defines simulation time.

```

3D_maps
261 %w_increasing & w_max
262 w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6 628.32];
263 w_max_bench=max(w_eff_bp_vec_bench);
264 w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471.2389 523.5988 575.9587 628.3185];
265 w_max_test=max(w_eff_bp_vec_test);
266 w_max=min(w_max_bench,w_max_test);
267
268 w_sampling=20;
269 w_increasing=w_max/w_sampling;
270
271 %T_increasing
272 T_sampling=10;
273 T_start=0;
274 T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 20.9 26.9];
275 T_m_max_bench=max(T_eff_bp_vec_bench);
276 T_eff_bp_vec_test=[15 24 43 62 83];
277 T_m_max_test=max(T_eff_bp_vec_test);
278 T_end=min(T_m_max_bench,T_m_max_test);
279 T_increasing=(T_end-T_start)/(T_sampling);
280
281 %w_change
282 w_n_bench= 4000 /60*2*pi;
283 w_n_test = 5000 /60*2*pi;
284 w_change=min(w_n_bench,w_n_test); %rotational speed at which the area change happens
285
286 %bm_cmd0
287 delay_w=25;
288 pulse_generator_period=(delay_w+(delay_w*w_sampling));
289 pulse_generator_phase_delay=(delay_w+(delay_w*w_sampling));
290 %tm_cmd0
291 delay_T=(delay_w+(delay_w*w_sampling));
292 initial_condition_delay=-T_increasing; %=0 in cases T=0 must not be considered during simulation...
293 ... otherwise put it equal to -T_increasing
294
295 sample=0.1; %sample time for interesting parameters to be stored
296 T_sim=(delay_w+(delay_w*w_sampling))*T_sampling

```

Figure 4.108: 3D maps' MATLAB script

#### 4.2.5 User subsystem

It contains model shown in figure 4.109 including all user actions to be performed for starting to use system. Since user could need to send signals both to Plant reference model and to Human-Machine-Interface subsystem, two output ports are implemented in order to model interfaces by means User is able to communicate to Plant reference model (*user\_plant output port*) and to HMI subsystem (*user\_HMI output port*). Then, two input ports are implemented to model interfaces by means User subsystem can receive signals form Plant (*plant\_user input port*) and HMI (*HMI\_user input port*). In this case, since User does not receive signals from Plant reference model and does not need to send signals to it, plant\_user input port and user plant output port are connected each other by means a discrete delay that allows to avoid algebraic loop error during simulation while HMI\_user input port is connected to a terminator Simulink block because in this case User subsystem does not receive signals from HMI subsystem. Output port 1 (*user\_HMI*) is connected to Mux Simulink block's output port such that two signals can be sent to HMI subsystem: bench and test motor's voltage power supply values.

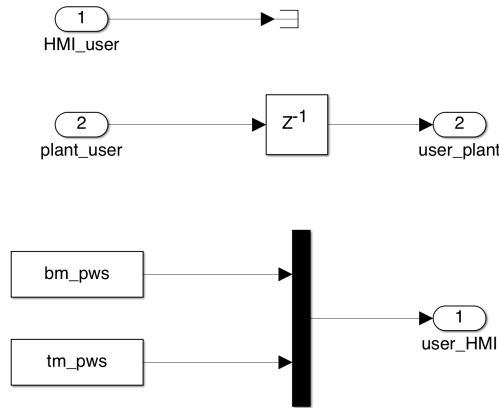


Figure 4.109: User subsystem Simulink model

To parametrize User subsystem a corresponding MATLAB script has to be defined and run before to start simulation such that variables are loaded into MATLAB workspace and Simulink blocks can use them. Here, only two variables are defined: *bm\_pws* and *tm\_pws* which are defined to set voltage values to be provided respectively to bench and test motor.

```

User Data
Define the parameters related to the User.

259 %collect user action
260
261 %%bm_tm_pws
262 bm_pws=96; %V
263 tm_pws=66; %V
264
265 return
266

```

Figure 4.110: User subsystem's MATLAB script

### 4.3 Dashboard and control panel

Observing MTM template (figure 4.64), two different panes highlighted in blue on left part are implemented. The one on upper part represents what is called *dashboard* which represents physical commands by means user is able to pilot system while the one on bottom part represents *control panel* through which user is able to check system's behavior while it works. In dashboard panel different physical components are implemented: *on/off button* by means user can start/stop system to work, indeed it is connected to constant 0 put inside HMI so that when it is pushed constant changes its value from 0 to 1 such that control law implemented inside Control Logic

reference model is perturbed and specific action is taken to start or stop system (on the basis of logic described in section 4.2.3.1); *emergency switch* by means user can stop system while it is working when something goes wrong, indeed it like on/off button case is connected to constant Simulink block implemented inside HMI subsystem whose initial value is 0 such that when it is toggled from Off to On state constant value changes from 0 to 1 causing a perturbation in control law implemented inside Control Logic reference model that is designed in such a way that when that perturbation happen controller is able to stop system working; *green LED* and *red LED* by means user is able to understand system's state (working or stopped); *floating scope* by means user is able to check sent or received signals by components implemented in dashboard. Instead, in panel control two floating scopes are implemented by means user can check signals coming out from plant and from HMI.

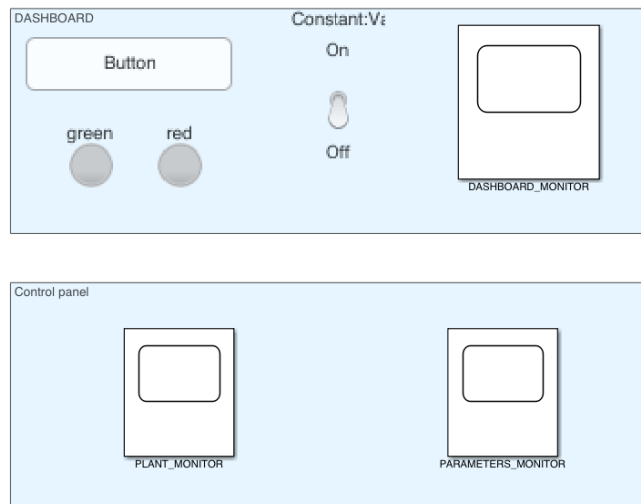


Figure 4.111: Dashboard and Control panel in MTM template

# Chapter 5

## System design: model-based testing

Once model is built some tests must be performed to validate it. For doing this model-based approach establishes three different tests by means designers can understand whether their considered technical details are able to satisfy user's requirements. These three tests are listed and described below.

- *MIL testing*

MIL stands for *Model-in-the-loop* and it represents test by which designed controller is tested in development ambient, like Simulink, adopting plant model. So, in this case both designed controller and plant models are run in the same development machine to test designed controller's behavior.

- *SIL testing*

SIL stands for *Software-in-the-loop* and it represents test by which designed controller's corresponding C code is tested in development ambient, like Simulink, using always plant's model. So, also in this case both controller's corresponding C code and plant's model are run in the same development machine to check whether obtained controller C code works properly: obtained results must be the same to those obtained in MIL testing case.

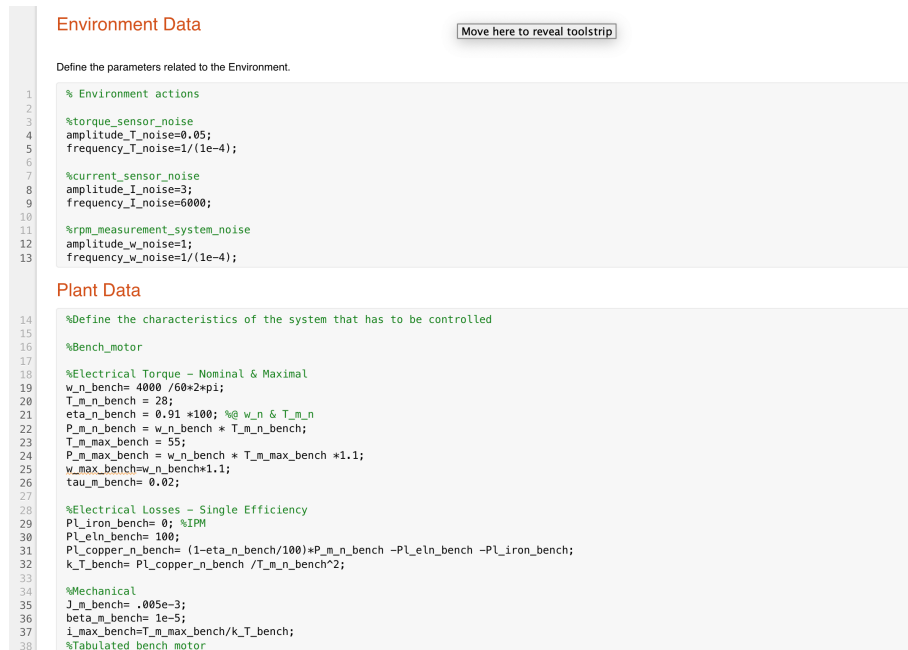
- *PIL testing*

PIL stands for *Processor-in-the-loop* and it represents test where obtained controller's C code is run in a target hardware (rapid prototyping hardware) while plant's model is run in development machine. Here, controller C code's behavior is tested inside hardware before to test system's performance by using real plant. In this thesis work, this test is not performed because dSpace (considered rapid prototyping hardware) implies a very expensive license to perform it that is not available.

## 5.1 MIL testing

To perform this test, MATLAB script containing needed variables to parametrize all Simulink blocks used to implement MTM, described in chapter 4, has to be run. This is needed to load all variables into workspace such that Simulink blocks can use them during simulation for performing computations and gets results. Once this is done, Simulink model can be run pressing run pushbutton implemented in “SIMULATION panel” and when simulation starts user has to press on/off dashboard button to start system working. On/off dashboard button must be keep pressed for at least 0.01 s until green LED is switched on. At that point, user can wait for simulation end in order to obtain interesting variable values stored in .mat file positioned in selected path.

MATLAB script to be defined and run, shown in figure 5.1, is constituted of different sections as described in previous chapter: the first section contains all needed variables to parametrize Simulink blocks implemented inside all reference models and subsystems and it must be always run to load all defined variables in MATLAB workspace while the other sections are related to the four subsystems implemented on bottom part of HMI subsystem so on the basis of what test user wants to perform (single working point, nominal power working point, torque curves, 3D maps) corresponding section has to be run in order to load into MATLAB workspace needed parameters.



```

1  % Environment actions
2
3  %torque_sensor_noise
4  amplitude_T_noise=0.05;
5  frequency_T_noise=1/(1e-4);
6
7  %current_sensor_noise
8  amplitude_I_noise=3;
9  frequency_I_noise=6000;
10
11 %rpm_measurement_system_noise
12 amplitude_w_noise=1;
13 frequency_w_noise=1/(1e-4);
14
15 %Define the characteristics of the system that has to be controlled
16 %Bench_motor
17
18 %Electrical Torque - Nominal & Maximal
19 w_n_bench= 4000 /60*2*pi;
20 T_m_n_bench = 28;
21 eta_n_bench = 0.91 *100; %@ w_n & T_m_n
22 P_m_n_bench = w_n_bench * T_m_n_bench;
23 T_m_max_bench = 55;
24 P_m_max_bench = w_n_bench * T_m_max_bench *1.1;
25 w_max_bench=w_n_bench*1.1;
26 tau_m_bench= 0.02;
27
28 %Electrical Losses - Single Efficiency
29 PL_iron_bench= 0; %IPM
30 PL_eln_bench= 100;
31 PL_copper_n_bench= (1-eta_n_bench/100)*P_m_n_bench -PL_eln_bench -PL_iron_bench;
32 K_T_bench= PL_copper_n_bench /T_m_n_bench^2;
33
34 %Mechanical
35 J_m_bench= .005e-3;
36 beta_m_bench= 1e-5;
37 i_max_bench=T_m_max_bench/k_T_bench;
38 %Tabulated_bench_motor

```



## System design: model-based testing

```
39 %Electrical_torque_section
40 w_t_vec_bench=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593 366.5191 418.8798 471.2389 523.5988 575.9587 628.3185]; %rad/s
41 T_t_vec_bench=[52.5 52.5 52.5 52.5 52.5 52.5 52.5 52.5 47 42]; %Move here to reveal toolbar
42
43 %Electrical_losses_section
44 %Tabulated_loss_data
45 w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6 628.32];
46 T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 20.9 26.9];
47 T_m_min_bench=min(T_eff_bp_vec_bench);
48 T_m_max_bench=max(T_eff_bp_vec_bench);
49 w_max_bench=max(w_eff_bp_vec_bench);
50
51 b=2000;
52 losses_table_bench=[52.4 85.1 151.9 126.8 304.9 468.1 712.5;
53 115.1 133.4 223.7 223.6 350.5 701.4 801.2;
54 155.1 104.8 311.2 311.4 693.4 1005.9 1299.9;
55 175.3 240.7 434.4 351.7 556.3 1056.4 1407.6;
56 200.1 244.1 572.9 395.7 813.2 1204.9 1013.4;
57 224.1 342.5 545.2 458.9 732.2 864.7 b;
58 240.1 466.8 490.3 b b b;
59 260.3 432.3 b b b b];
60
61 %Tabulated_efficiency_data
62 d=85;
63 efficiency_table_bench=[82.8 85.8 88 91.9 84.5 82.6 84.5;
64 79.6 88.4 90.5 92.2 90.2 86.2 89.2;
65 80.2 93.1 90.9 93.3 87.3 86.7 91.2;
66 80.3 87.4 89.3 93.5 90.9 87.8 91.6;
67 80.2 88.8 88.1 93.5 88.7 88.1 91.9;
68 78.9 86.1 89.2 93.2 90.3 92.4 d;
69 79.5 87.1 91.8 d d d;
70 78.7 85.2 d d d d];
71
72 %Test motor
73 %Electrical Torque - Nominal & Maximal
74 w_n_test = 5000 /60*2*pi;
75 P_m_n_test=18500;
76 T_m_n_test = P_m_n_test/w_n_test;
77 V_n_test=66; %V
78 I_n_test=115; %A
79 cos_phi_test=0.86;
80 eta_n_test =(P_m_n_test/(sqrt(3)*V_n_test*I_n_test*cos_phi_test))*100;
81 w_max_test=7500/60*2*pi;
82 T_m_max_test = 83;
83 P_m_max_test = w_max_test * T_m_max_test;
84 tau_m_test = 0.02;
85 %Electrical Losses - Single Efficiency
86 PL_iron_test = 0;
87 PL_eln_test = 100;
88 PL_copper_n_test=(1-eta_n_test/100)*P_m_n_test -PL_eln_test -PL_iron_test;
89 k_T_test= PL_copper_n_test /T_m_n_test^2; %Move here to reveal toolbar
90 I_max_test=T_m_max_test/k_T_test;
91
92 %Tabulated_test_motor
93 %Electrical_torque_section
94 w_t_vec_test=[52.3599 104.7198 157.0796 209.4395 261.7994 314.1593 366.5191 418.8798 471.2389 523.5988 575.9587 628.3185]; %rad/s
95 T_t_vec_test=[83 82.5 82 81.5 80.5 80.5 77.5 68 56 46.5 34.5]; %Nm
96
97 %Electrical_losses_section
98 %Tabulated_loss_data
99 w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471.2389 523.5988 575.9587 628.3185];
100 T_eff_bp_vec_test=[15 24 43 62 83];
101 T_m_min_test=min(T_eff_bp_vec_test);
102 T_m_max_test=max(T_eff_bp_vec_test);
103 w_max_bench=max(w_eff_bp_vec_bench);
104
105 losses_table_test=[143.8356 290.4494 954.5455 1.9857e+03 2.8393e+03
106 157.8947 311.1782 972.2222 2.8326e+03 3.5795e+03
107 236.6421 381.5789 1.8410e+03 2.4633e+03 3.6286e+03
108 204.5455 434.2105 1.1290e+03 2.4167e+03 4.0905e+03
109 219.1558 510.5263 1.3784e+03 2.7333e+03 4.2007e+03
110 335.5263 689.3617 1.5326e+03 2.7009e+03 3.8022e+03
111 454.7872 846.7742 1.6781e+03 2.9659e+03 3.5405e+03
112 564.5161 912.1622 1.9111e+03 3.2880e+03 3.1379e+03
113 632.4324 978.2609 1.8750e+03 2.9138e+03 2.8143e+03];
114
115 %Tabulated_efficiency_data
116 efficiency_table_test=[91.25 89 82.5 76.25 70;
117 95 94 90 86.25 82.5;
118 95 95 92.5 90 87.5;
119 96.25 95 93 90 87;
120 96.25 95 92.5 90 86.25;
121 95 94 92 89 86.66;
122 94 93 91.25 88 86.66;
123 93 92.5 90 86.25 87;
124 92.5 92 90 87 87.5];
125
126 %Sensors
127 %torque_sensor
128 max_T=500; % max torque value that can be measured by sensor
129 Vp_T=10; % max voltage value at the output of the sensor
130 k_torque=Vp_T/max_T;
131
132 %current_sensor
133 up_I_limit=20;
134 low_I_limit=4;
135 k_current=16/300;
136 current_offset=4;
137 R_load=250; %Move here to reveal toolbar
138
139 %rotational_speed_conversion
140 Vp_w=10;
141 rpm_max=6000;
142 k_speed=Vp_w/rpm_max;
143
144 %bn_pwm_signal
145 Vp_bn_pwm=5;
146 bn_pwm=96;
147 k_bn_pwm=Vp_bn_pwm/bn_pwm;
148
149 %tm_pwm_signal
150 Vp_tm_pwm=5;
151 tm_pwm=66;
152 k_tm_pwm=Vp_tm_pwm/tm_pwm;
153
154 % ADC
155 % LP filter
156 Ts1=1e-4;
157 wc1=2303*2*pi;
158 s=tf('s');
159 z=tf(1, Ts1);
160 H_z1=wc1^2/(s^2+(s+wc1)^2);
161 H_z1=2*pk(c2d(H_z1, Ts1, 'zoh'));
162 [NUM_LPF, DEN_LPF]=tfdata(H_z1, 'v');
163
164 %rotational_speed_sensor
165 Vp_w=10;
166 N=16;
167 quant_w=Vp_w/(2^N-1);
168
169 %bn_trqcnd_signal
170 Vp_bn_trqcnd=5;
171 N=16;
172 quant_bn_trqcnd=Vp_bn_trqcnd/(2^N-1);
173
174 %tm_trqcnd_signal
175 Vp_tm_trqcnd=5;
176 N=16;
177 quant_tm_trqcnd=Vp_tm_trqcnd/(2^N-1);
178
179 %bn_cmd_signal
180 Vp_bn_command_signal=5;
```

## System design: model-based testing

```
179 N=16;
180 quant_bn_cmd_signal=Vp_bn_command_signal/(2*N-1);
181
182 %tm_cmd_signal
183 Vp_tm_cmd_signal=5;
184 N=16;
185 quant_tm_cmd_signal=Vp_tm_cmd_signal/(2*N-1);
186
187 % DAC
188 % LP filter
189 Ts2=1e-4;
190 wc2=500e3+2*pi;
191 s=tf('s');
192 z=tf('z',Ts2);
193 H_s2=wc2^2/(s+wc2)^2;
194 H_z2=zpk(c2d(H_s2,Ts2,'zoh'));
195 [NUM_LPF2,DEN_LPF2]=tfdata(H_z2,'v');
```

**Control Logic Data**

The Control Logic must contain the control algorithm that is intended to be deployed on the target hardware.

```
196 sampling_time=1e-2;
197 s=tf('s');
198 z=tf('z',sampling_time);
199 H_s=1/(3s+1);
200 H_z=zpk(c2d(H_s,sampling_time,'zoh'));
201 [NUM,DEN]=tfdata(H_z,'v');
202
203 %w_signal
204 Vp_w=10;
205 rpm_max=6000;
206 k_speed=Vp_w/rpm_max;
207
208 %bm_cmd_signal
209 Vp_bn_cmd_signal=5;
210 bn_cmd_signal_max=6000+2*pi/60; %rad/s
211 k_bn_cmd=Vp_bn_cmd_signal/bn_cmd_signal_max;
212
213 %tm_cmd_signal
214 Vp_tm_cmd=5; %V
215 tm_cmd_max=100; %Nm
216 k_tm_cmd=Vp_tm_cmd/tm_cmd_max;
217
```

**HMI Data**

Define the parameters related to the HMI.

```
218 % HMI actions
219
220 %signal_to_parameters
221 %current_signal
222 k_current=16/300;
223 current_offset=4;
224 R_load=250;
225
226 %w_signal
227 Vp_w=10;
228 rpm_max=6000;
229 k_speed=Vp_w/rpm_max;
230
231 %T_signal
232 max_T=500; % max torque value that can be measured by sensor
233 Vp_T=10; % max voltage value at the output of the sensor
234 k_torque=Vp_T/max_T;
235
236 %cmd_to_signals
237 Vp_bn_cmd=5;
238 bn_cmd_max=6000+2*pi/60; %rad/s
239 k_bn_cmd=Vp_bn_cmd/bn_cmd_max;
240
241 Vp_tm_cmd=5;
242 tm_cmd_max=100;
243 k_tm_cmd=Vp_tm_cmd/tm_cmd_max;
244
245 %bm_pws_signal
246 Vp_bn_pws=5;
247 bn_pws=96;
248 k_bn_pws=Vp_bn_pws/bn_pws;
249
250 %tm_pws_signal
251 Vp_tm_pws=5;
252 tm_pws=66;
253 k_tm_pws=Vp_tm_pws/tm_pws;
254
```

**User Data**

Define the parameters related to the User.

```
255 %collect user action
256
257 %Abm_tm_pws
258 bn_pws=96; %V
259 tm_pws=66; %V
260 return
```

**Constant torque\_growing\_rotational\_speed**

```
261 %w_increasing & w_max
262 w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6 628.32];
263 w_max_bench=max(w_eff_bp_vec_bench);
264 w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471.2389 523.5988 575.9587 628.3185];
265 w_max_test=max(w_eff_bp_vec_test);
266 w_max=min(w_max_bench,w_max_test);
267
268 w_sampling=20;
269 w_increasing=w_max/w_sampling;
270
271 %T_increasing
272 T_sampling=10;
273 T_start=0;
274 T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 26.9 26.9];
275 T_a_max_bench=max(T_eff_bp_vec_bench);
276 T_eff_bp_vec_test=[15 24 43 62 83];
277 T_a_max_test=max(T_eff_bp_vec_test);
278 T_end=min(T_a_max_bench,T_a_max_test);
279 T_increasing=(T_end-T_start)/(T_sampling);
280
281 %w_change
282 w_n_bench= 4000 /60+2*pi;
283 w_n_test= 5000 /60+2*pi;
284 w_change=min(w_n_bench,w_n_test); %rotational speed at which the area change happens
285
286 %bn_cmd0
287 delay_w=25;
288 pulse_generator_period=(delay_w+(delay_w*w_w_sampling));
289 pulse_generator_phase_delay=(delay_w*(delay_w_w_sampling));
290 %tm_cmd0
291 delay_T=(delay_w+(delay_w*w_w_sampling));
292 initial_condition_delay=-T_increasing; %=-0 in cases T=0 must not be considered during simulation...
293 ... otherwise put it equal to -T_increasing
294
```

```

295 sample=0.1; %sample time for interesting parameters to be stored
296 T_sim=(delay_w+(delay_ww_sampling))*T_sampling
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380

```

**Constant\_rotational\_speed\_growing\_torque**

```

%w_increasing & w_max
w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6 628.32];
w_max_bench=max(w_eff_bp_vec_bench);
w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471.2389 523.5988 575.9587 628.3185];
w_max_test=max(w_eff_bp_vec_test);
w_max=min(w_max_bench,w_max_test);

w_sampling=20;
w_start=0;

w_end=w_max;
w_increasing=(w_end-w_start)/w_sampling;
initial_condition_delay=0; %w in cases w=0 must not be considered during simulation...
... otherwise put it equal to ~w_increasing

%T_increasing
T_sampling=10;
T_start=0;
T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 20.9 26.9];
T_m_max_bench=max(T_eff_bp_vec_bench);
T_eff_bp_vec_test=[15 24 43 62 83];
T_m_max_test=max(T_eff_bp_vec_test);
T_end=min(T_m_max_bench,T_m_max_test);
T_increasing=(T_end-T_start)/(T_sampling);

%w_change
w_n_bench = 4000 / 60*2*pi;
w_n_test = 5000 / 60*2*pi;
w_change=min(w_n_bench,w_n_test); %rotational speed at which the area change happens

%bn_cm1 & tm_cm1 blocks
delay_T=25;
delay_w=T_sampling*delay_T;
pulse_generator_period=T_sampling*delay_T;
pulse_generator_phase_delay=T_sampling*delay_T;

sample=0.1; %sample time for interesting parameters to be stored
T_sim=T_sampling*delay_T+w_sampling;

```

**Torque\_curves**

```

%w_increasing
w_eff_bp_vec_bench=[104.72 209.44 314.16 366.52 418.88 471.24 523.6 628.32];
w_max_bench=max(w_eff_bp_vec_bench);
w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471.2389 523.5988 575.9587 628.3185];
w_max_test=max(w_eff_bp_vec_test);
w_max=min(w_max_bench,w_max_test);

w_sampling=20;
w_increasing=w_max/w_sampling;

%T_increasing
T_increasing=20;

%w_change
w_n_bench = 4000 / 60*2*pi;
w_n_test = 5000 / 60*2*pi;
w_change=min(w_n_bench,w_n_test); %rotational speed at which the area change happens

%bn_cm1
delay_w=25;

sample=0.1; %sample time for interesting parameters to be stored
T_sim=(delay_w+(delay_ww_sampling))

```

**Nominal\_power\_working\_point**

```

w_n_test = 5000 / 60*2*pi;
w_sampling=10;
w_increasing=w_n_test/w_sampling;

P_m_n_test=10500;
T_m_n_test = P_m_n_test/w_n_test;
T_sampling=10;
T_increasing=T_m_n_test/T_sampling;

%bn_cm2 & tm_cm2 blocks
delay_T=25;
delay_w=(T_sampling*delay_T)/w_sampling;

sample=0.1;
T_sim=delay_T+(T_sampling*delay_T)

```

**Single\_working\_point**

```

% bn_cm3 & tm_cm3 blocks
w_increasing=3500/60*2*pi; %rad/s
T_increasing=20; %Nm
T_sim=100; %s

sample=0.1; %sample time of variables to be stored

```

Figure 5.1: MTM's MATLAB script

Each time a new electric motor has to be tested below listed variables must be defined by user in the first section of previous reported MATLAB script.

- $w\_n\_test$
- $P\_m\_n\_test$
- $V\_n\_test$
- $I\_n\_test$

- *cos\_fi\_test*
- *w\_max\_test*
- *T\_m\_max\_test*
- *tau\_m\_test*
- *Pl\_iron\_test*
- *Pl\_eln\_test*
- *w\_t\_vec\_test*
- *T\_t\_vec\_test*
- *w\_eff\_bp\_vec\_test*
- *T\_eff\_bp\_vec\_test*
- *losses\_table\_test*
- *efficiency\_table\_test*
- *tm\_pws*

While in sections related to variables to be load into MATLAB workspace on the basis of test to be performed besides to define some of previous variables below listed must be defined.

- *w\_sampling*
- *T\_sampling*
- *T\_increasing*
- *delay\_w*
- *delay\_T*
- *initial\_condition\_delay*
- *sample*

Once all needed variables are load into MATLAB workspace, Modular-Technical-Model Simulink file can be open and run in order to start simulation. At that point on/off dashboard button has to be pressed to start system and obtain corresponding results. Indeed, at the end in selected path four .mat files are created containing interesting parameters that can be used to evaluate test motor mechanical and electrical characteristics. Results for each kind of test that can be performed are shown below.

### 5.1.1 Single working point MIL testing

To perform this kind of test, procedure described in subsection 4.2.4 related to Single working point test must be adopted to set properly MTM Simulink model and to load needed variables into MATLAB workspace. At the end of simulation, four .mat files in selected path, as shown in figure 5.2, are created: *T.mat* file containing torque values sensed by torque sensor on test motor's shaft, *w.mat* file containing rotational speed values measured by rounds per minute measurement system attached to test motor's shaft, *bm\_I\_sensor.mat* and *tm\_I\_sensor.mat* files containing respectively bench and test motor phase current values sensed by current sensors.

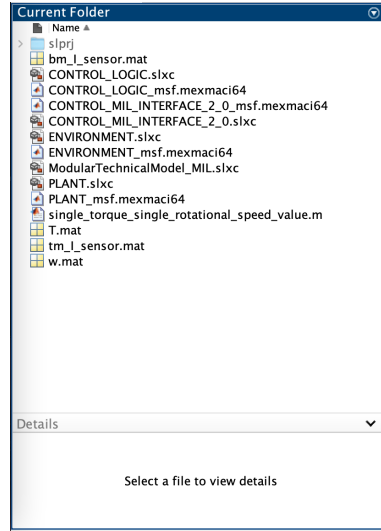


Figure 5.2: Single working point MIL testing path

Once these files have been obtained, *single\_working\_point.m* file, whose MATLAB code is shown in figure 5.3, can be used to obtain desired test motor mechanical and electrical characteristics related to that considered working point. Before to run this file, some specific variables have to be defined:  $V_{bm}$  and  $V_{tm}$  representing voltage values used to provide respectively bench and test motor Simulink blocks,  $n$  representing stored samples' number for each interesting parameter,  $t$  representing considered simulation time,  $sample$  representing sample time considered for Simulink ToFile blocks implemented in HMI subsystem to store interesting values,  $\theta_{amb}$  and  $\theta_{tm\_max}$  representing respectively ambient temperature and maximum reachable test motor temperature (measured in K),  $c_{tm}$  representing specific heat capacity that is evaluated on the basis of used materials to build test motor (measured in J/(K kg)),  $m_{tm}$  representing test motor weight (measured in kg).

```

1  clear all
2  close all
3  clc
4
5  %%input_data
6  load('bm_I_sensor.mat')
7  load('tm_I_sensor.mat')
8  load('T.mat')
9  load('w.mat')
10
11  T_vec=(T(2,:));
12  w_rpm_vec=(w(2,:));
13  bm_I_sensor_vec=(bm_I_sensor(2,:));
14  tm_I_sensor_vec=(tm_I_sensor(2,:));
15
16  %%set parameters
17  V_bm=96; %V
18  V_tm=66; %V
19
20  n=2001; %number of samples stored for each interesting parameters
21  t=200; %s considered to perform simulation
22  sample=0.1; %sample time used to store interesting variables
23
24  theta_amb=293.15; %K ambient temperature
25  theta_tm_max=453.15; %K max temperature that can be reached by motor
26  ... (isolation class H)
27
28  c_tm=400; %specific heat capacity (J/(K kg)) considering iron and copper
29  m_tm=25; %test motor weight (kg)
30
31  %%parameters_computation:
32
33  %%absorbed power
34  P_a_bm_vec=V_bm*bm_I_sensor_vec;
35  P_a_tm_vec=V_tm*tm_I_sensor_vec;
36
37  %%mechanical power
38  w_rads_vec=w_rpm_vec*2*pi/60;
39
40  for i=1:1:n
41      P_shaft_vec(i,1)=T_vec(i,1)*w_rads_vec(i,1);
42  end
43
44  %%temperature
45
46  for i=1:1:n
47      R_th_tm_vec(i,1)=theta_tm_max/P_d_tm_vec(i,1);
48  end
49
50  C_th_tm=c_tm*m_tm; %thermal capacity
51  tau_th_tm_vec=C_th_tm/R_th_tm_vec;
52  tau_th_tm_vec=-tau_th_tm_vec;
53
54  for i=1:1:n
55      theta_tm_vec(i,1)=theta_amb+((P_d_tm_vec(i,1)*R_th_tm_vec(i,1)));
56  end
57
58  t_vec=0:sample:t;
59  tiledlayout(4,2)
60  nexttile
61  plot(t_vec,w_rpm_vec,'r'), grid on
62  title('Test motor speed rotational speed')
63  xlabel('t [s]')
64  ylabel('\omega [rpm]')
65  legend('\omega')
66  nexttile
67  plot(t_vec,T_vec,'r'), grid on
68  title('Test motor Torque')
69  xlabel('t [s]')
70  ylabel('T [Nm]')
71  legend('T')
72  nexttile
73  plot(t_vec,tm_I_sensor_vec), grid on
74  title('Test motor current')
75  xlabel('t [s]')
76  ylabel('I [A]')
77  legend('I')
78  nexttile
79  plot(t_vec,eta_tm_perc_vec,'r'), grid on
80  title('Test motor efficiency')
81  xlabel('t [s]')
82  ylabel('\eta [%]')
83  legend('\eta')
84
85  P_shaft_vec(i,1)=T_vec(i,1)*w_rads_vec(i,1);
86  end
87
88  %%dissipated power
89  for i=1:1:n
90      P_d_bm_vec(i,1)=P_a_bm_vec(i,1)-P_shaft_vec(i,1);
91      P_d_tm_vec(i,1)=(P_a_tm_vec(i,1)-P_shaft_vec(i,1));
92  end
93
94  %%rendimento
95  for i=1:1:n
96      eta_tm_vec(i,1)=P_shaft_vec(i,1)/inv(P_a_tm_vec(i,1));
97  end
98
99  %%total current demand
100  for i=1:1:n
101      I_tot_vec(i,1)=bm_I_sensor_vec(i,1)+tm_I_sensor_vec(i,1);
102  end
103
104  %%total absorbed power
105  for i=1:1:n
106      P_a_tot_vec(i,1)=P_a_bm_vec(i,1)+P_a_tm_vec(i,1);
107  end
108
109  %%total dissipated power
110  for i=1:1:n
111      P_d_tot_vec(i,1)=P_d_bm_vec(i,1)+P_d_tm_vec(i,1);
112  end
113
114  %%eta_percentage
115  eta_tm_perc_vec=100*eta_tm_vec;
116
117  nexttile
118  plot(t_vec,T_vec,'r'), grid on
119  title('Test motor Torque')
120  xlabel('t [s]')
121  ylabel('T [Nm]')
122  legend('T')
123  nexttile
124  plot(t_vec,tm_I_sensor_vec), grid on
125  title('Test motor current')
126  xlabel('t [s]')
127  ylabel('I [A]')
128  legend('I')
129  nexttile
130  plot(t_vec,eta_tm_perc_vec,'r'), grid on
131  title('Test motor efficiency')
132  xlabel('t [s]')
133  ylabel('\eta [%]')
134  legend('\eta')
135  nexttile
136  plot(t_vec,P_a_tm_vec,'r'), grid on
137  title('Test motor absorbed power')
138  xlabel('t [s]')
139  ylabel('P_{a} [kW]')
140  legend('P_{a}')
141  nexttile
142  plot(t_vec,P_d_tm_vec,'r'), grid on
143  title('Test motor power shaft')
144  xlabel('t [s]')
145  ylabel('P_{d} [kW]')
146  legend('P_{d}')
147  nexttile
148  plot(t_vec,theta_tm_vec,'r'), grid on
149  title('Test motor temperature')
150  xlabel('t [s]')
151  ylabel('\theta [K]')
152  legend('\theta')
153  nexttile
154  plot(t_vec,P_a_tot_vec,'b'), grid on, hold on
155  plot(t_vec,P_d_tot_vec,'r')
156  title('Power recirculating effect')
157  xlabel('t [s]')
158  ylabel('P_{a} P_{d} [W]')
159  legend('P_{a}', 'P_{d}')

```

Figure 5.3: *Single\_working\_point.m*'s MATLAB script

Once this is done, MATLAB *single\_working\_point.m* script can be run to obtain results below shown (figure 5.4) where test motor is tested considering the working point that corresponds to intersection between 20 Nm and 366.52 rad/s in  $\omega$ - $T$  plane. Chosen torque value is used to pilot test motor while chosen rotational speed value is used to pilot bench motor.

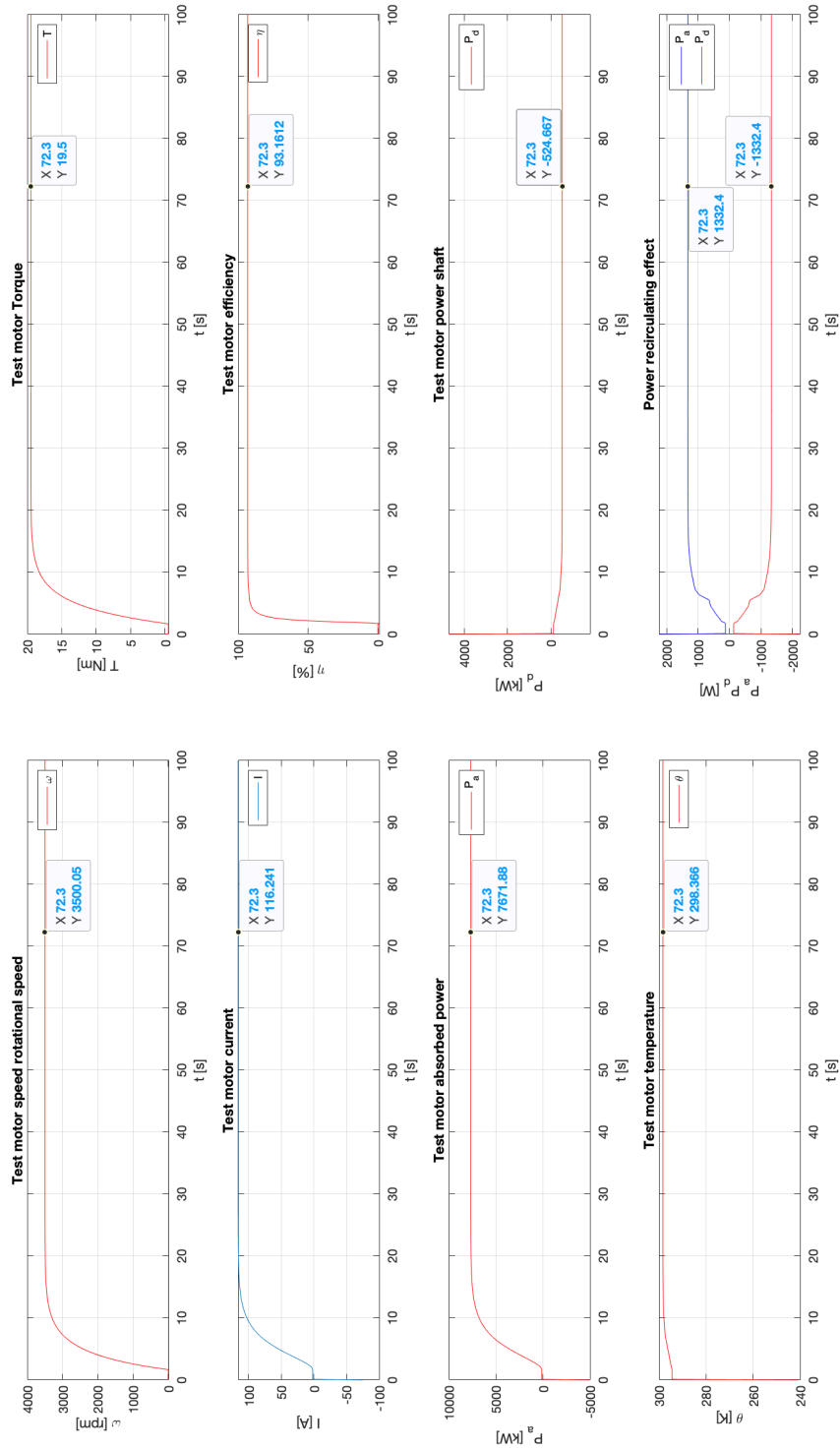


Figure 5.4: Single working point results in system design phase MIL testing

### 5.1.2 Nominal power working point MIL testing

To perform this test, procedure described in subsection 4.2.4 related to Nominal power working point test must be adopted to set properly MTM Simulink model and to load needed variables in MATLAB workspace. At the end of simulation, the same .mat file described in previous case are generated inside selected path that can be used to obtain desired test motor mechanical and electrical characteristics related to working points covered during testing. To obtain these desired characteristics *nominal\_power\_working\_point.m file*, shown in figure 5.6, can be used where the same variables described in previous case have to be defined before to run MATLAB script.

```

1 - clear all
2 - close all
3 - clc
4
5 %input_data
6 - load('bm_I_sensor.mat')
7 - load('tm_I_sensor.mat')
8 - load('t.mat')
9 - load('w.mat')
10
11 - T_vec=(T(2,:));
12 - w_rpm_vec=w(2,:);
13 - bm_I_sensor_vec=(bm_I_sensor(2,:));
14 - tm_I_sensor_vec=(tm_I_sensor(2,:));
15
16 %set parameters
17 - V_bm=96; %V
18 - V_tm=66; %V
19
20 - n=2751; %number of samples stored for each interesting parameters
21 - t=275; %s considered to perform simulation
22 - sample=0.1; %sample time used to store interesting variables
23
24 - theta_amb=293.15; %K ambient temperature
25 - theta_tm_max=453.15; %K max temperature that can be reached by moto
26 - ... (insulation class H)
27
28 - c_tm=400; %specific heat capacity (J/(K kg)) considering iron and c
29 - m_tm=25; %test motor weight (kg)
30
31 %parameters computation:
32
33 %absorbed power
34 - P_a_bm_vec=V_bm*bm_I_sensor_vec;
35 - P_a_tm_vec=V_tm*tm_I_sensor_vec;
36
37 %mechanical power
38 - w_rads_vec=w_rpm_vec*2*pi/60;
39
40 - for i=1:1:n
41 - P_shaft_vec(i,1)=T_vec(i,1)*w_rads_vec(i,1);
42 - end
43
44
45
46 %dissipated power
47
48 - for i=1:1:n
49 - P_d_bm_vec(i,1)=P_a_bm_vec(i,1)-P_shaft_vec(i,1);
50 - P_d_tm_vec(i,1)=P_a_tm_vec(i,1)-P_shaft_vec(i,1);
51 - end
52
53 %rendimento
54
55 - for i=1:1:n
56 - eta_tm_vec(i,1)=P_shaft_vec(i,1)*inv(P_a_tm_vec(i,1));
57 - end
58
59 %total current demand
60
61 - for i=1:1:n
62 - I_tot_vec(i,1)=bm_I_sensor_vec(i,1)+tm_I_sensor_vec(i,1);
63 - end
64
65 %total absorbed power
66
67 - for i=1:1:n
68 - P_a_tot_vec(i,1)=P_a_bm_vec(i,1)+P_a_tm_vec(i,1);
69 - end
70
71 %total dissipated power
72
73 - for i=1:1:n
74 - P_d_tot_vec(i,1)=P_d_bm_vec(i,1)+P_d_tm_vec(i,1);
75 - end
76
77 %eta_percentage
78
79
80
81
82
83
84
85
86
87
88 %eta_percentage
89
90 - eta_tm_perc_vec=100*eta_tm_vec;
91
92 %temperature
93
94 - for i=1:1:n
95 - R_th_tm_vec(i,1)=theta_tm_max/(P_d_tm_vec(i,1));
96 - end
97
98
99 - C_th_tm=c_tm*m_tm; %thermal capacity
100 - tau_th_tm_vec=C_th_tm/R_th_tm_vec;
101 - tau_th_tm_vec=tau_th_tm_vec;
102
103 - for i=1:1:n
104 - theta_tm_vec(i,1)=theta_amb+((P_d_tm_vec(i,1)*R_th_tm_vec(i,1)));
105 - end
106
107 %plots
108 - t_vec=[0:sample:t];
109 - tiledlayout(3,3)
110 - nexttile
111 - plot(t_vec,w_rpm_vec,'r'), grid on
112 - title('Test motor speed rotational speed')
113 - xlabel('t [s]')
114 - ylabel('omega [rpm]')
115 - legend('omega')
116 - nexttile
117 - plot(t_vec,T_vec,'r'), grid on
118 - title('Test motor Torque')
119 - xlabel('t [s]')
120 - ylabel('T [Nm]')
121 - legend('T')
122 - nexttile
123 - plot(t_vec,tm_I_sensor_vec), grid on
124 - title('Test motor current')
125 - xlabel('t [s]')
126 - ylabel('I [A]')
127 - legend('I')
128 - nexttile
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178

```

Figure 5.5: *Nominal\_powe\_working\_point.m's* MATLAB script

Running this MATLAB file, results shown in figure 5.7 can be obtained.



Here, rotational speed and torque values increase with a step trends until their nominal values are reached (523.6 rad/s and 20 Nm). Each step lasts 25 s that is equal to user considered delay value to change used rotational speed and torque values to pilot bench and test motor. For each pair of rotational speed and torque values a working point is defined and for it test motor characteristics are computed. Here, a very interesting parameter to be considered is reached temperature once nominal power working point is selected (last step). In that point, reached temperature is 319.86 K (about 47 °C) so it is far from acceptable maximum temperature value (453.15 K, about 180 °C).

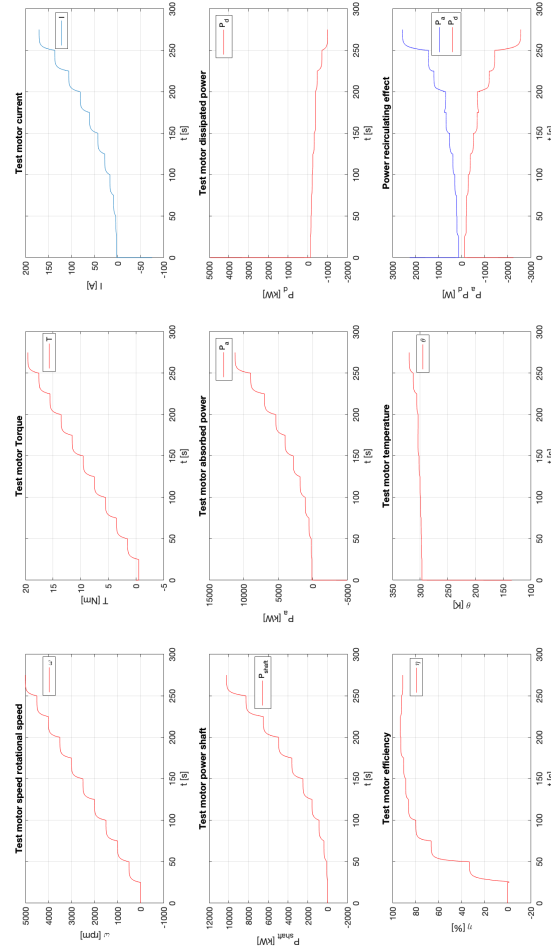


Figure 5.6: Nominal power working point results in system design phase MIL testing

### 5.1.3 Torque curves MIL testing

To perform this test, procedure described in subsection 4.2.4 related to Torque curves test must be adopted to set properly MTM Simulink model and to load needed variables in MATLAB workspace. In this case, like previous ones, at the end of simulation the same four .mat files are created in selected path that can be adopted to evaluate desired test motor torque curves. To obtain these, *Torque\_curves.m* file shown in figure 5.6 must be used where besides to define the same variables considered in previous cases, *T\_test* variable must be defined that represents torque value considered to pilot electric motor under test.

```

1 - clear all
2 - close all
3 - clc
4
5 %input_data
6 - load('bm_I_sensor.mat')
7 - load('tm_I_sensor.mat')
8 - load('T.mat')
9 - load('w.mat')
10
11 - T_vec=(T(2,:));
12 - w_rpm_vec=w(2,:);
13 - bm_I_sensor_vec=(bm_I_sensor(2,:));
14 - tm_I_sensor_vec=(tm_I_sensor(2,:));
15
16 %set parameters
17 - V_bm=96; %V
18 - V_tm=66; %V
19 - T_test=20; %torque value considered to perform test
20 - n=21; %number of samples stored for each interesting parameters
21 - t=500; %s considered to perform simulation
22 - sample=25; %sample time used to store interesting variables
23 - theta_amb=293.15; %K ambient temperature
24 - theta_tm_max=453.15; %K max temperature that can be reached by moto
25 - ....(insulation class H)
26 - c_tm=400; %specific heat capacity (J/(K kg)) considering iron and c
27 - m_tm=25; %test motor weight (kg)
28
29 %parameters_computation
30 %absorbed_power
31 - P_a_bm_vec=V_bm*bm_I_sensor_vec;
32 - P_a_tm_vec=V_tm*tm_I_sensor_vec;
33
34 %mechanical_power
35 - w_rads_vec=w_rpm_vec*2*pi/60;
36
37 for i=1:1:n
38     P_shaft_vec(i,1)=T_vec(i,1)*w_rads_vec(i,1);
39 end
40
41 %dissipated_power
42 for i=1:1:n
43     P_d_tm_vec(i,1)=P_a_tm_vec(i,1)+P_a_bm_vec(i,1);
44 end
45
46 P_d_bm_vec(i,1)=P_a_bm_vec(i,1)-P_shaft_vec(i,1);
47 P_d_tm_vec(i,1)=P_a_tm_vec(i,1)-P_shaft_vec(i,1);
48 end
49
50 %rendimento
51 for i=1:1:n
52     eta_tm_vec(i,1)=P_shaft_vec(i,1)/inv(P_a_tm_vec(i,1));
53 end
54
55 %total_current_demand
56 for i=1:1:n
57     I_tot_vec(i,1)=bm_I_sensor_vec(i,1)+tm_I_sensor_vec(i,1);
58 end
59
60 %total_absorbed_power
61 for i=1:1:n
62     P_a_tot_vec(i,1)=P_a_bm_vec(i,1)+P_a_tm_vec(i,1);
63 end
64
65 %total_dissipated_power
66 for i=1:1:n
67     P_d_tot_vec(i,1)=P_d_bm_vec(i,1)+P_d_tm_vec(i,1);
68 end
69
70 %eta_percentage
71 eta_tm_perc_vec=100*eta_tm_vec;
72
73 %temperature
74 for i=1:1:n
75     R_th_tm_vec(i,1)=theta_tm_max/P_d_tm_vec(i,1);
76 end
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101 %plots
102 t_vec=0:sample:t;
103 tiledlayout(3,1)
104 nexttile
105 [hAx,hLine1,hLine2] = plotyy(w_rpm_vec,eta_tm_perc_vec,w_rpm_vec,P_shaft_vec), grid on, hold all;
106 plot(w_rpm_vec,T_vec,'m')
107 set(hLine1,'color','red');
108 set(hLine2,'color','blue');
109 set(hAx,'ycolor','k');
110 set(hAx(1),'YTick',[0:5:100]);
111 set(hAx(2),'YTick',[0:1:20]);
112 title('Results considering T = ', num2str(T_test), ' Nm')
113 xlabel('\omega [rpm]')
114 ylabel(hAx(1),'eta [%] T [Nm]') % left y-axis
115 ylabel(hAx(2),'P_shaft [kW]') % right y-axis
116 legend('eta','T','P_shaft')
117 nexttile
118 plot(t_vec,P_a_tot_vec,'b'), grid on, hold on
119 plot(t_vec,P_d_tot_vec,'r')
120 title('Power recirculating effect')
121 xlabel('t [s]')
122 ylabel('P_a) P_d [W]')
123 legend('P_a','P_d')
124 nexttile
125 plot(t_vec,theta_tm_vec,'r'), grid on
126 title('Test motor temperature')
127 xlabel('t [s]')
128 ylabel('\theta [K]')
129 legend('\theta')
130

```

Figure 5.7: *Torque\_curves.m*'s MATLAB script

Running this MATLAB file, results shown in figure 5.8 can be obtained.

Here, in the first graph efficiency, motor shaft power and torque trends are shown while in second graph temperature trend is shown by means test motor's temperature can be taken under control (it is very important that it does not overcome maximum admissible value to avoid problems with electrical circuits implemented inside), and in third graph absorbed and dissipated power trends are shown in order to check whether power recirculating effect is verified or not.

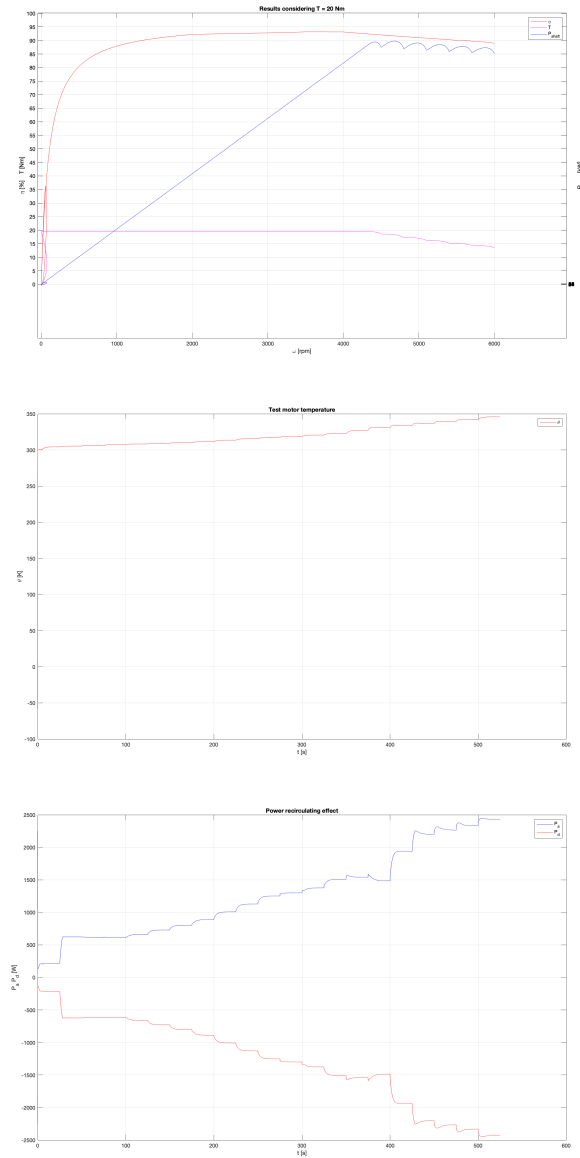


Figure 5.8: Torque curves results in system design phase MIL testing

### 5.1.4 3D maps MIL testing

To perform this test, the procedure described in subsection 4.2.4 related to 3D maps test must be adopted to set properly MTM Simulink model and to load all needed variables into MATLAB workspace. Also in this case, like all the previous cases, at the end of simulation in selected path four .mat files are generated containing interesting parameter values that can be used to obtain desired test motor 3D efficiency and dissipated power maps and electric and mechanical characteristics related to all working points considered during test. To do this, at the end of simulation maps.m file can be used to obtain desired efficiency and dissipated 3D maps. Here, besides to define all variables described in the first case, some parameters have to be defines such as:  $T\_sampling$  representing considered torque values' number for selecting all desired working points,  $w\_sampling$  representing considered rotational speed values' number to select desired working points,  $w\_eff\_bp\_vec\_test$  and  $T\_eff\_bp\_vec\_test$  that represent respectively considered rotational speed and torque.

```

1 - clear all
2 - close all
3 - clc
4
5 %input_data
6 load('bm_I_sensor.mat')
7 load('tm_I_sensor.mat')
8 load('T.mat')
9 load('w.mat')
10
11 T_vec=(T(2,:))';
12 w_rpm_vec=(w(2,:))';
13 bm_I_sensor_vec=(bm_I_sensor(2,:))';
14 tm_I_sensor_vec=(tm_I_sensor(2,:))';
15
16 T_vec(1,:)=[];
17 w_rpm_vec(1,:)=[];
18 bm_I_sensor_vec(1,:)=[];
19 tm_I_sensor_vec(1,:)=[];
20
21 %set parameters
22 V_bm=96; %V
23 V_tm=66; %V
24
25 ns=52500; %number of samples stored for each interesting parameters
26 t=525; %s considered to perform simulation
27 sample=0.1; %sample time used to store interesting variables
28
29 theta_amb=293.15; %K ambient temperature
30 theta_tm_max=453.15; %K max temperature that can be reached by moto
31 ... (isolation class H)
32
33 c_tm=400; %specific heat capacity (J/(K kg)) considering iron and c
34 m_tm=25; %test motor weight (kg)
35
36 T_sampling=10;
37 w_sampling=20;
38 w_eff_bp_vec_test=[104.7198 209.4395 314.1593 366.5191 418.879 471.];
39 T_eff_bp_vec_test=[15 24 43 62 83];
40
41 %parameters_computation:
42
43 %absorbed_power
44 P_a_bm_vec=V_bm*bm_I_sensor_vec;
45 P_a_tm_vec=V_tm*tm_I_sensor_vec;
46
47 %mechanical_power
48 w_rads_vec=w_rpm_vec*2*pi/60;
49
50 for i=1:1:n
51     P_shaft_vec(i,1)=T_vec(i,1)*w_rads_vec(i,1);
52 end
53
54 %dissipated_power
55 for i=1:1:n
56     P_d_bm_vec(i,1)=P_a_bm_vec(i,1)-P_shaft_vec(i,1);
57     P_d_tm_vec(i,1)=(P_a_tm_vec(i,1)-P_shaft_vec(i,1));
58 end
59
60 %rendimento
61 for i=1:1:n
62     eta_tm_vec(i,1)=P_shaft_vec(i,1)*inv(P_a_tm_vec(i,1));
63 end
64
65 %total_current_demand
66 for i=1:1:n
67     I_tot_vec(i,1)=bm_I_sensor_vec(i,1)+tm_I_sensor_vec(i,1);
68 end
69
70 %total_absorbed_power
71 for i=1:1:n
72     P_a_tot_vec(i,1)=P_a_bm_vec(i,1)+P_a_tm_vec(i,1);
73 end
74
75 %total_dissipated_power
76 for i=1:1:n
77     P_d_tot_vec(i,1)=P_d_bm_vec(i,1)+P_d_tm_vec(i,1);
78 end
79
80
81
82
83
84
85
86
87
88
89
90

```

```

91 %eta_percentage
92 eta_tm_perc_vec=100*eta_tm_vec;
93
94 %temperature
95
96 for i=1:1:n
97     R_th_tm_vec(i,1)=theta_tm_max/P_d_tm_vec(i,1);
98 end
99
100 C_th_tm=c_tm*m_tm; %thermal capacity
101 tau_th_tm_vec=C_th_tm*R_th_tm_vec;
102 tau_th_tm_vec=tau_th_tm_vec;
103
104 for i=1:1:n
105     theta_tm_vec(i,1)=theta_amb+(P_d_tm_vec(i,1)*R_th_tm_vec(i,1));
106 end
107
108 %plots
109 t_vec=[0:sample:t*T_sampling];
110 t_vec(1,:)=[];
111 tiledlayout(3,3)
112 [hAx,hLine1,hLine2] = plotyy(w_rpm_vec,eta_tm_perc_vec,w_rpm_vec,P_
113 plot(w_rpm_vec,T_vec,'m')
114 set(hLine1,'color','red');
115 set(hLine2,'color','blue');
116 set(hAx,'ycolor',{'k','k'});
117 set(hAx(1),'YTick',[0:5:100]);
118 set(hAx(2),'YTick',[0:1:20]);
119 T_test=20;
120 title('Results considering T = ', num2str(T_test), ' Nm')
121 xlabel('omega [rpm]')
122 ylabel(hAx(1),'eta [%] T [Nm]') % left y-axis
123 ylabel(hAx(2),'P_shaft [kW]') % right y-axis
124 legend('eta','T','P_shaft')
125 nexttile
126 plot(t_vec,w_rpm_vec,'r'), grid on
127 title('Test motor speed rotational speed')
128 xlabel('t [s]')
129
130 title('Power recirculating effect')
131 xlabel('t [s]')
132 ylabel('P_a [kW]')
133 legend('P_a','P_d')
134
135 T_matrix=reshape(T_vec,[t/sample,T_sampling]);
136 w_rpm_matrix=reshape(w_rpm_vec,[t/sample,T_sampling]);
137 w_rpm_matrix=reshape(w_rpm_vec,[t/sample,T_sampling]);
138 P_a_tm_matrix=reshape(P_a_tm_vec,[t/sample,T_sampling]);
139 P_a_tm_matrix=reshape(P_a_tm_vec,[t/sample,T_sampling]);
140 P_a_tot_matrix=reshape(P_a_tot_vec,[t/sample,T_sampling]);
141 P_d_tm_matrix=reshape(P_d_tm_vec,[t/sample,T_sampling]);
142 P_d_tot_matrix=reshape(P_d_tot_vec,[t/sample,T_sampling]);
143 P_shaft_matrix=reshape(P_shaft_vec,[t/sample,T_sampling]);
144 bn_t_sensor_matrix=reshape(bn_t_sensor_vec,[t/sample,T_sampling]);
145 tm_t_sensor_matrix=reshape(tm_t_sensor_vec,[t/sample,T_sampling]);
146 I_tot_matrix=reshape(I_tot_vec,[t/sample,T_sampling]);
147 eta_tm_perc_matrix=reshape(eta_tm_perc_vec,[t/sample,T_sampling]);
148
149 %substitute first column elements:
150 T_matrix(:,1)=[];
151 P_shaft_matrix(:,1)=[];
152 eta_tm_perc_matrix(:,1)=[];
153
154 %maps:
155 w_eff_bp_vec_bench=[84.72 209.44 314.16 366.52 418.88 471.24 523.6 62
156 w_max_bench=max(w_eff_bp_vec_bench);
157 w_max_test=max(w_eff_bp_vec_test);
158 w_max=min(w_max_bench,w_max_test);
159 w_increasing=w_max/w_sampling;
160 T_start=0;
161 T_eff_bp_vec_bench=[2.4 4.3 11.4 13.8 15.2 20.9 26.9];
162 T_max_bench=max(T_eff_bp_vec_bench);
163 T_max_test=max(T_eff_bp_vec_test);
164 T_end=min(T_max_bench,T_max_test);
165
166 %efficiency_map
167 figure;
168 speed(w_rpm_matrix,'
169 torques(T_matrix,'
170 efficiency(eta_tm_perc_matrix,'
171 x0=0;
172 x1=max(speed(:))+0.05*max(speed(:));
173
174 y0=0;
175 y1=max(torque(:))+0.01*max(torque(:));
176 ny=1/((T_end-T_start)/T_sampling);
177 x= linspace(x0,x1,nx);
178 y= linspace(y0,y1,ny);
179 [X,Y]=meshgrid(x,y);
180 Z=griddata(speed,torque,efficiency,X,Y);
181 contourf(X,Y,Z), grid on, hold all
182 val=[0:5:100];
183 [C]=contour(X,Y,Z,val,'k-');
184 clabel(C,val);
185 xlabel('Speed [rpm]','fontWeight','bold','fontSize',10);
186 ylabel('Torque [Nm]','fontWeight','bold','fontSize',10);
187 colormap(jet);
188 h=colorbar;
189 set(get(h,'label'),'string','Efficiency map (%)', 'fontWeight',
190
191 %dissipated_power_map
192 figure;
193 speed(w_rpm_matrix,'
194 torque(T_matrix,'
195 dissipated_power(P_d_tm_matrix,'
196 x0=0;
197 x1=max(speed(:))+0.05*max(speed(:));
198 nx=1/w_increasing;
199 y0=0;
200 y1=max(torque(:))+0.01*max(torque(:));
201 ny=1/((T_end-T_start)/T_sampling);
202 x= linspace(x0,x1,nx);
203 y= linspace(y0,y1,ny);
204 [X,Y]=meshgrid(x,y);
205 Z=griddata(speed,torque,dissipated_power,X,Y);
206 contourf(X,Y,Z), grid on, hold all
207 val=[0:100:1200];
208 [C]=contour(X,Y,Z,val,'k-');
209 clabel(C,val);
210 xlabel('Speed [rpm]','fontWeight','bold','fontSize',10);
211 ylabel('Torque [Nm]','fontWeight','bold','fontSize',10);
212 colormap(jet);
213 h=colorbar;
214 set(get(h,'label'),'string','Dissipated power (W)', 'fontWeight',

```

Figure 5.9: *3D\_maps.m*'s MATLAB script

Running this MATLAB script, results shown in following figures are obtained. Here, besides dissipated power and efficiency 3D maps also mechanical and electrical characteristics are shown from which all selected working points to evaluate test motor's behavior can be distinguished and for each of them corresponding parameter values can be read. Moreover, also temperature trend is shown where reached temperature values are shown.

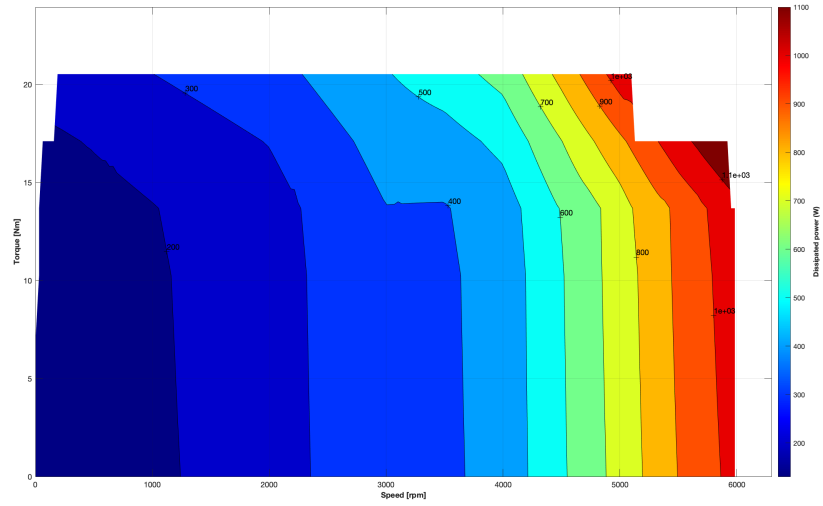


Figure 5.10: Dissipated power 3D map result in system design phase MIL testing

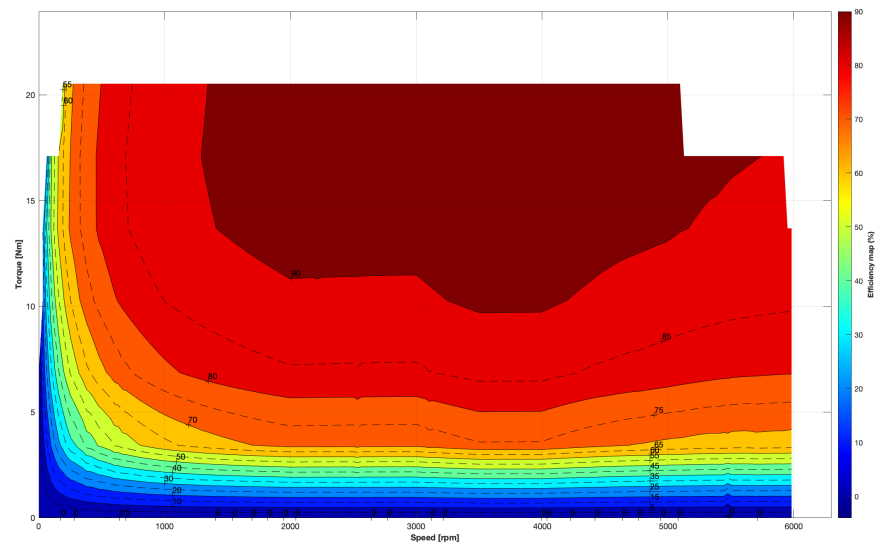


Figure 5.11: Efficiency 3D map result in system design phase MIL testing

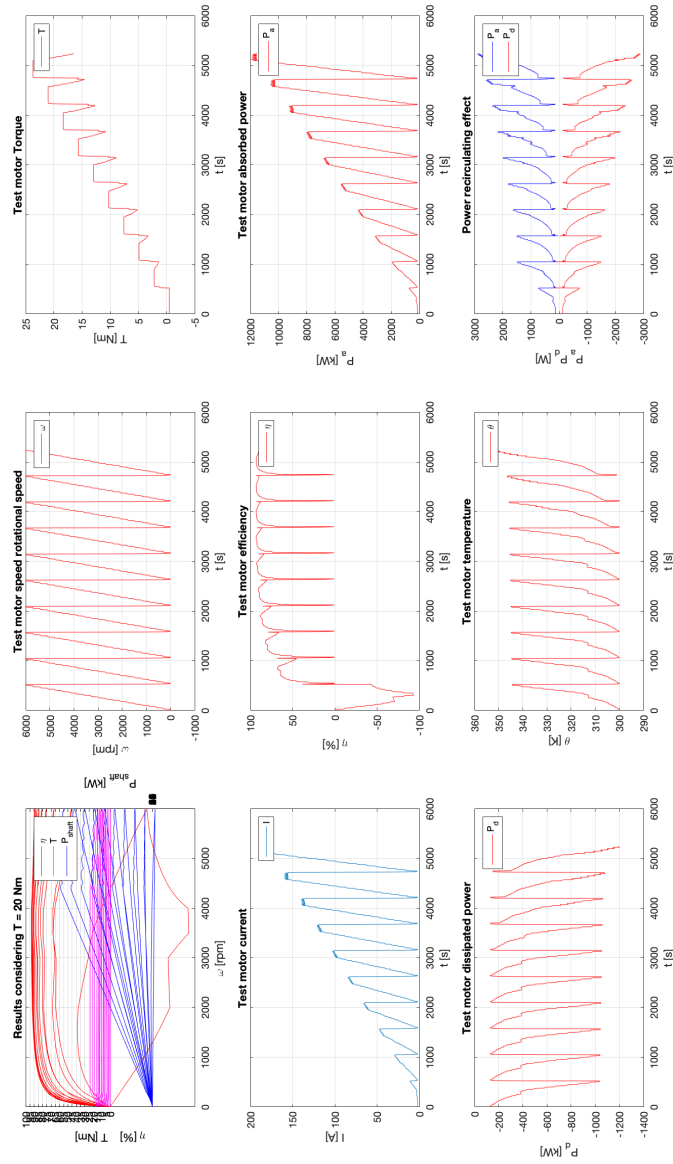


Figure 5.12: Test motor parameter trends in system design phase MIL testing

## 5.2 SIL testing

This test allows to check corresponding Control Logic C code's behavior to understand whether it allows to obtain the same results obtained in MIL testing case. Here, first of all Control Logic's C code has to be obtained and then running it in development machine such as also plant's model results have to be obtained in order to compare them to those obtained in MIL testing case. To obtain Control Logic's C code, reference model's block parameters window must be opened and from *Simulation mode* drop-down menu *Software-in-the-loop (SIL)* must be chosen, as shown in figure 5.13.

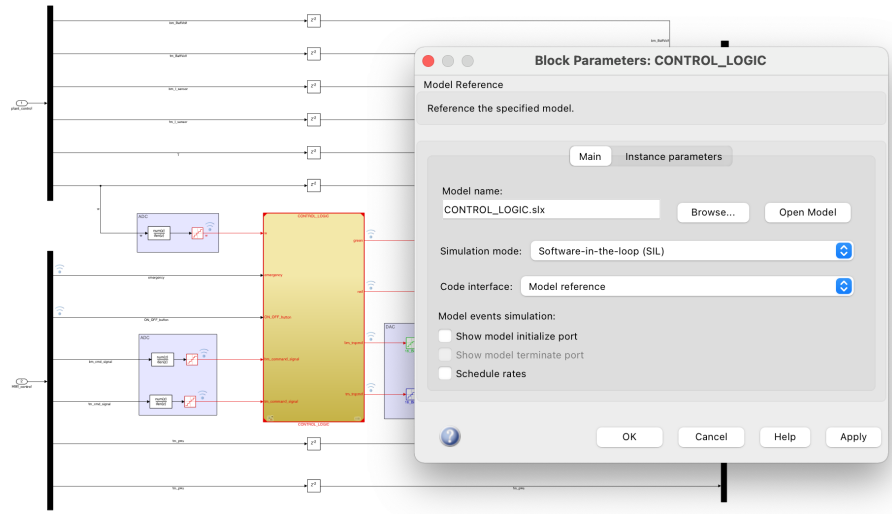


Figure 5.13: Control Logic block parameters setting in SIL testing

Once this is done, one of the four possible tests that can be chosen must be selected setting properly MTM Simulink model and running the corresponding MATLAB section to load needed parameters in MATLAB workspace. At that point Simulink model can be run and on/off dashboard button has to be pressed to start system and obtain both corresponding Control Logic's C code and simulation results stored in four .mat file inside selected path, like MIL testing case. To check C code generation status, diagnostic viewer window can be opened from Simulink model in order to see what is shown in figure 5.14 and understand whether Control Logic's C code is successfully generated or not. This could be sufficient to understand that obtained C code works properly but in order to be more precise simulation results have to be compared to those obtained in MIL testing case.



```
Diagnostics
ModularTechnicalModel_MIL

# Simulation
03.28 PM

Updating Model Reference Code Generation Targets
Elapsed: 0.06 sec

# Updating Model Reference Code Generation Targets @ 1
Elapsed: 0.06 min

## Starting serial model reference code generation build
## Starting build procedure for CONTROL_LOGIC
## Generating code and artifacts to 'Model specific' folder structure
## Generating code into build folder: /Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC
## Invoking Target Language Compiler on CONTROL_LOGIC.rtu
## Using System Target File: Applications/MATLAB_R2020b.app/tw/c/ert/ert.tlc
## Loading TLC Function Libraries
## Initial pass through model to cache user defined code
## Caching model source code
## Writing header file CONTROL_LOGIC_types.h
## Writing source file CONTROL_LOGIC.o

...

## Writing header file CONTROL_LOGIC_private.h
## Writing header file CONTROL_LOGIC.h
## Writing header file types.h
## Writing header file model_reference_types.h
## TLC code generation complete.
## Using toolchain Clang v3.1 | gmake (64-bit Mac)
## Creating /Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC/coderassumptions/lib/CONTROL_LOGIC_ca.m ...
## Building "/CONTROL_LOGIC.ca": Applications/MATLAB_R2020b.app/bin/maci64/gmake -f "/CONTROL_LOGIC.ca" all
xcrun clang -o isysroot/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKS/MacOSX11.sdk arch x86_64 -fno-common -exceptions -mmacosx-version-min=10.14 -D__CLASSIC_INTERFACE__ -DALLOWCONTINPCN=0 -DITERMCHN1=1 -DMAT_FILE=0 -DMULTI_INSTANCE_CODE=0 -DINTERGER_CODE=0 -DINTQ=0 -DIO1SIG=0 -DIO1SIG_HANDLER=0 -DIO1SIG_HANDLER_ENABLED=1 -DIO1STREAM_RX_BUFFER_BYTE_SIZE=5000 -DIO1STREAM_TX_BUFFER_BYTE_SIZE=5000 -DMEM_UNIT_BYTE=1 -DMeandm_Twinst_T=1/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC -I/Applications/MATLAB_R2020b.app/extern/include -I/Applications/MATLAB_R2020b.app/simlink/include -I/Applications/MATLAB_R2020b.app/tw/c/sc -I/Applications/MATLAB_R2020b.app/tw/c/sc/ext_mode/common -I/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/sharedutils -o "/CONTROL_LOGIC.o"
## Creating static library "/CONTROL_LOGIC_twlib.a" ...
xcrun ar rvs "/CONTROL_LOGIC_twlib.a" CONTROL_LOGIC.o
ar: creating archive "/CONTROL_LOGIC_twlib.a"
a - CONTROL_LOGIC.o
## Created: "/CONTROL_LOGIC_twlib.a"
## Successfully generated all binary outputs.
## Successful completion of build procedure for CONTROL_LOGIC

# Build Summary @ 1
Elapsed: 0.06 sec

Code generation targets built:

Model Action Rebuild Reason
CONTROL_LOGIC Code generated and compiled CONTROL_LOGIC.o does not exist.
1 of 1 models built (0 models already up to date)
Build duration: 0h 3m 7.181s

## Preparing to start SIS simulation ...
Building with 'Xcode with Clang'.
MEX completed successfully.
## Using toolchain Clang v3.1 | gmake (64-bit Mac)
## Creating /Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC/coderassumptions/lib/CONTROL_LOGIC_ca.m ...
## Building "/CONTROL_LOGIC.ca": Applications/MATLAB_R2020b.app/bin/maci64/gmake -f "/CONTROL_LOGIC.ca" all
xcrun clang -o isysroot/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKS/MacOSX11.sdk arch x86_64 -fno-common -exceptions -mmacosx-version-min=10.14 -D__CLASSIC_INTERFACE__ -DALLOWCONTINPCN=0 -DITERMCHN1=1 -DMAT_FILE=0 -DMULTI_INSTANCE_CODE=0 -DINTERGER_CODE=0 -DINTQ=0 -DIO1SIG=0 -DIO1SIG_HANDLER=0 -DIO1SIG_HANDLER_ENABLED=1 -DIO1STREAM_RX_BUFFER_BYTE_SIZE=5000 -DIO1STREAM_TX_BUFFER_BYTE_SIZE=5000 -DMEM_UNIT_BYTE=1 -DMeandm_Twinst_T=1/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC -I/Applications/MATLAB_R2020b.app/extern/include -I/Applications/MATLAB_R2020b.app/simlink/include -I/Applications/MATLAB_R2020b.app/tw/c/sc -I/Applications/MATLAB_R2020b.app/tw/c/sc/ext_mode/common -I/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/sharedutils -o "/CONTROL_LOGIC.o"
## Creating static library "/CONTROL_LOGIC_twlib.a" ...
xcrun ar rvs "/CONTROL_LOGIC_twlib.a" CONTROL_LOGIC.o
ar: creating archive "/CONTROL_LOGIC_twlib.a"
a - CONTROL_LOGIC.o
## Created: "/CONTROL_LOGIC_twlib.a"
## Successfully generated all binary outputs.

## Using toolchain Clang v3.1 | gmake (64-bit Mac)
## Creating /Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC/ca/CONTROL_LOGIC_ca.m ...
## Building "/CONTROL_LOGIC.ca": Applications/MATLAB_R2020b.app/bin/maci64/gmake -f "/CONTROL_LOGIC.ca" all
xcrun clang -o isysroot/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKS/MacOSX11.sdk arch x86_64 -fno-common -exceptions -mmacosx-version-min=10.14 -D__CLASSIC_INTERFACE__ -DALLOWCONTINPCN=0 -DITERMCHN1=1 -DMAT_FILE=0 -DMULTI_INSTANCE_CODE=0 -DINTERGER_CODE=0 -DINTQ=0 -DIO1SIG=0 -DIO1SIG_HANDLER=0 -DIO1SIG_HANDLER_ENABLED=1 -DIO1STREAM_RX_BUFFER_BYTE_SIZE=5000 -DIO1STREAM_TX_BUFFER_BYTE_SIZE=5000 -DMEM_UNIT_BYTE=1 -DMeandm_Twinst_T=1/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC -I/Applications/MATLAB_R2020b.app/extern/include -I/Applications/MATLAB_R2020b.app/simlink/include -I/Applications/MATLAB_R2020b.app/tw/c/sc -I/Applications/MATLAB_R2020b.app/tw/c/sc/ext_mode/common -I/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/sharedutils -o "/CONTROL_LOGIC.o"
## Creating static library "/CONTROL_LOGIC_twlib.a" ...
xcrun ar rvs "/CONTROL_LOGIC_twlib.a" CONTROL_LOGIC.o
ar: creating archive "/CONTROL_LOGIC_twlib.a"
a - CONTROL_LOGIC.o
## Created: "/CONTROL_LOGIC_twlib.a"
## Successfully generated all binary outputs.

## Using toolchain Clang v3.1 | gmake (64-bit Mac)
## Creating /Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC/ca/CONTROL_LOGIC_ca.m ...
## Building "/CONTROL_LOGIC.ca": Applications/MATLAB_R2020b.app/bin/maci64/gmake -f "/CONTROL_LOGIC.ca" all
xcrun clang -o isysroot/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKS/MacOSX11.sdk arch x86_64 -fno-common -exceptions -mmacosx-version-min=10.14 -D__CLASSIC_INTERFACE__ -DALLOWCONTINPCN=0 -DITERMCHN1=1 -DMAT_FILE=0 -DMULTI_INSTANCE_CODE=0 -DINTERGER_CODE=0 -DINTQ=0 -DIO1SIG=0 -DIO1SIG_HANDLER=0 -DIO1SIG_HANDLER_ENABLED=1 -DIO1STREAM_RX_BUFFER_BYTE_SIZE=5000 -DIO1STREAM_TX_BUFFER_BYTE_SIZE=5000 -DMEM_UNIT_BYTE=1 -DMeandm_Twinst_T=1/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC -I/Applications/MATLAB_R2020b.app/extern/include -I/Applications/MATLAB_R2020b.app/simlink/include -I/Applications/MATLAB_R2020b.app/tw/c/sc -I/Applications/MATLAB_R2020b.app/tw/c/sc/ext_mode/common -I/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/sharedutils -o "/CONTROL_LOGIC.o"
## Creating static library "/CONTROL_LOGIC_twlib.a" ...
xcrun ar rvs "/CONTROL_LOGIC_twlib.a" CONTROL_LOGIC.o
ar: creating archive "/CONTROL_LOGIC_twlib.a"
a - CONTROL_LOGIC.o
## Created: "/CONTROL_LOGIC_twlib.a"
## Successfully generated all binary outputs.

## Using toolchain Clang v3.1 | gmake (64-bit Mac)
## Creating /Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC/ca/CONTROL_LOGIC_ca.m ...
## Building "/CONTROL_LOGIC.ca": Applications/MATLAB_R2020b.app/bin/maci64/gmake -f "/CONTROL_LOGIC.ca" all
xcrun clang -o isysroot/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKS/MacOSX11.sdk arch x86_64 -fno-common -exceptions -mmacosx-version-min=10.14 -D__CLASSIC_INTERFACE__ -DALLOWCONTINPCN=0 -DITERMCHN1=1 -DMAT_FILE=0 -DMULTI_INSTANCE_CODE=0 -DINTERGER_CODE=0 -DINTQ=0 -DIO1SIG=0 -DIO1SIG_HANDLER=0 -DIO1SIG_HANDLER_ENABLED=1 -DIO1STREAM_RX_BUFFER_BYTE_SIZE=5000 -DIO1STREAM_TX_BUFFER_BYTE_SIZE=5000 -DMEM_UNIT_BYTE=1 -DMeandm_Twinst_T=1/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC -I/Applications/MATLAB_R2020b.app/extern/include -I/Applications/MATLAB_R2020b.app/simlink/include -I/Applications/MATLAB_R2020b.app/tw/c/sc -I/Applications/MATLAB_R2020b.app/tw/c/sc/ext_mode/common -I/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/sharedutils -o "/CONTROL_LOGIC.o"
## Creating static library "/CONTROL_LOGIC_twlib.a" ...
xcrun ar rvs "/CONTROL_LOGIC_twlib.a" CONTROL_LOGIC.o
ar: creating archive "/CONTROL_LOGIC_twlib.a"
a - CONTROL_LOGIC.o
## Created: "/CONTROL_LOGIC_twlib.a"
## Successfully generated all binary outputs.

## Using toolchain Clang v3.1 | gmake (64-bit Mac)
## Creating /Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC/ca/CONTROL_LOGIC_ca.m ...
## Building "/CONTROL_LOGIC.ca": Applications/MATLAB_R2020b.app/bin/maci64/gmake -f "/CONTROL_LOGIC.ca" all
xcrun clang -o isysroot/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKS/MacOSX11.sdk arch x86_64 -fno-common -exceptions -mmacosx-version-min=10.14 -D__CLASSIC_INTERFACE__ -DALLOWCONTINPCN=0 -DITERMCHN1=1 -DMAT_FILE=0 -DMULTI_INSTANCE_CODE=0 -DINTERGER_CODE=0 -DINTQ=0 -DIO1SIG=0 -DIO1SIG_HANDLER=0 -DIO1SIG_HANDLER_ENABLED=1 -DIO1STREAM_RX_BUFFER_BYTE_SIZE=5000 -DIO1STREAM_TX_BUFFER_BYTE_SIZE=5000 -DMEM_UNIT_BYTE=1 -DMeandm_Twinst_T=1/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/CONTROL_LOGIC -I/Applications/MATLAB_R2020b.app/extern/include -I/Applications/MATLAB_R2020b.app/simlink/include -I/Applications/MATLAB_R2020b.app/tw/c/sc -I/Applications/MATLAB_R2020b.app/tw/c/sc/ext_mode/common -I/Users/gaetanocomblo/Downloads/Test_bench_project/sprj/ert/sharedutils -o "/CONTROL_LOGIC.o"
## Creating static library "/CONTROL_LOGIC_twlib.a" ...
xcrun ar rvs "/CONTROL_LOGIC_twlib.a" CONTROL_LOGIC.o
ar: creating archive "/CONTROL_LOGIC_twlib.a"
a - CONTROL_LOGIC.o
## Created: "/CONTROL_LOGIC_twlib.a"
## Successfully generated all binary outputs.
```

[illegible]

126

Here, SIL testing is done considering only 3D maps test, which represents the most complex test, that can be done with designed model. So, whether the corresponding Control Logic's C code works properly in this case for sure it will work properly in all the other cases. Performing this test, again, the four .mat files seen in all the other cases are obtained. These files contain interesting parameter values that can be used to obtain desired test motor efficiency and dissipated power 3D maps. To do this, the same maps.m file described in subsection 5.1.4 can be used to obtain the corresponding results to be compared to those obtained in 3D maps MIL testing case (results reported in subsection 5.1.4).

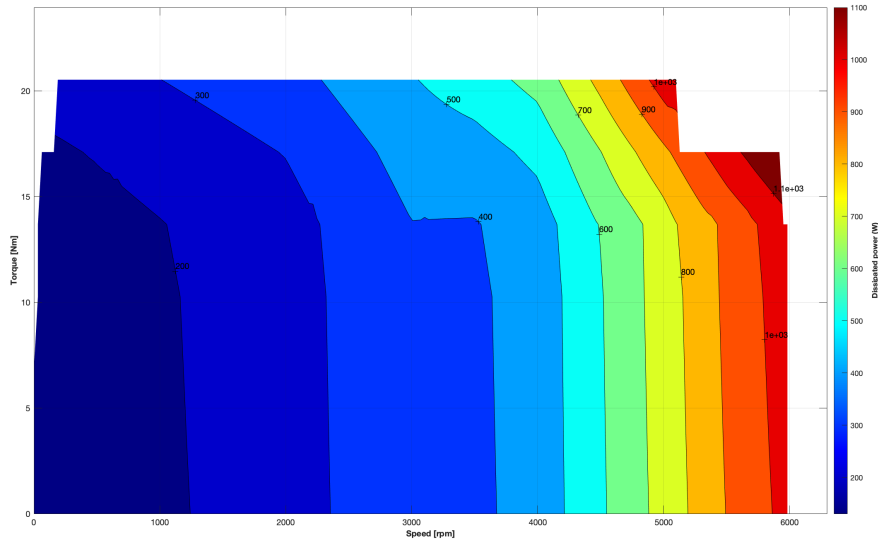


Figure 5.15: Dissipated power 3D map in SIL testing

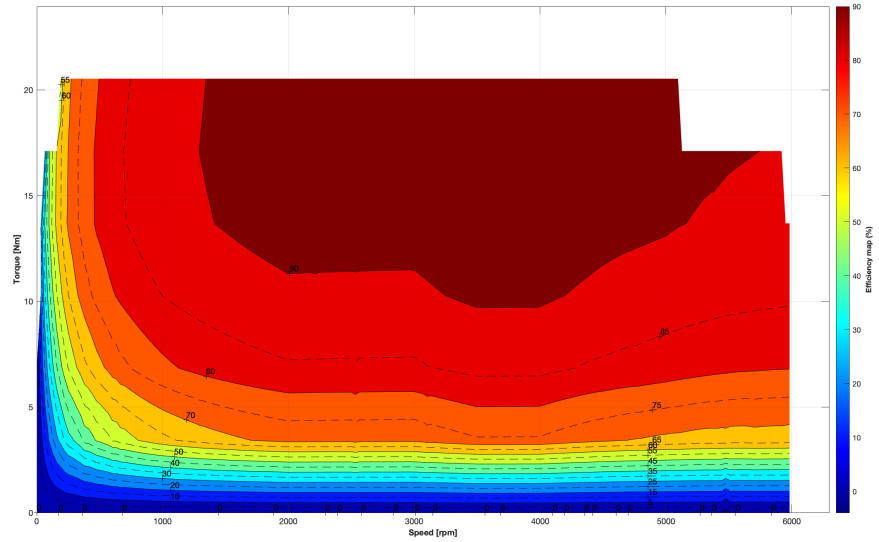


Figure 5.16: Efficiency 3D map in SIL testing

Comparing these results to those obtained in 3D maps MIL testing case reported in subsection 5.1.4 it is possible to observe that results are perfectly equal to each other. This means that obtained Control Logic's C code works properly and it is validated. For one more confirmation, test harness can be performed. To do this Control Logic reference model has to be translated into subsystem such that *treat as atomic unit option* from block parameters window can be checked. Once this is done, signals coming out and entering to subsystem must be logged in order to see their trends over the time. To store these trends Logging Signal option in model setting must be checked, and a name and save format must be chosen. In this way, at the end of simulation in MATLAB workspace a new variable is created (*out variable*) containing logged signal values. This variable can be stored in .mat file digiting `save('out_SIL', 'out')` command in MATLAB command window. *Out\_SIL* is the name of new .mat file while *out* is MATLAB workspace variable to be stored in that file. After this, Control Logic test harness can be created by right click on Control Logic subsystem and selecting *test harness option* from drop-down. Then, *create for Control Logic option* must be chosen in order to open a new window from which a name and SIL option from *Verification Mode* drop-down menu can be selected as shown in figure 5.19.

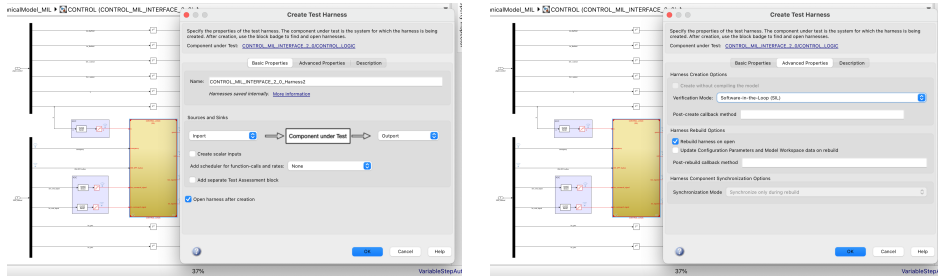


Figure 5.17: Control Logic reference model setting for test harness generation

Once Control Logic test harness is created (figure 5.18), out variable before stored must be loaded in workspace in order to obtain before logged signals. This can be done through `SIL_signals=out.get('SIL_signals')` command where `SIL_signals` before equal signal represents name used for creating new variable in MATLAB workspace while the one in brackets represents variable to be extracted from out variable. Then, Control Logic input signals have to be obtained from new defined variable by using command before described. Among these signals there is *w* signal, *emergency* signal, *ON\_OFF\_button* signal, *bm\_cmd\_signal* and *tm\_cmd\_signal*. Then, Control Logic test harness signals must be set through model setting input section while output signals must be logged in order to store their values and trends. Also in this case Signal Logging option from Control Logic test harness model setting must be checked and a name and save format must be defined.

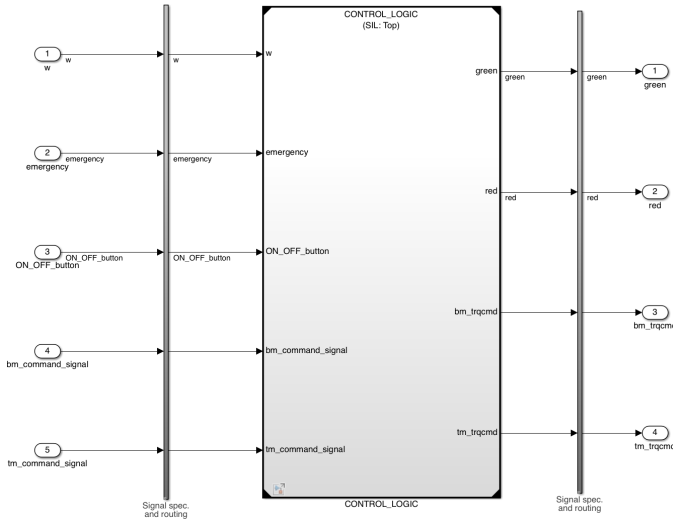


Figure 5.18: Control Logic's test harness model

Once Control Logic test harness is set, it can be run to obtain a new out variable into MATLAB workspace that must be stored by means the

same command before described. At that point, Control Logic subsystem output signals obtained in test harness case must be compared to those obtained in MIL testing case to check whether obtained Control Logic's C code works properly or not: if output signals are perfectly overlapped each other, for sure, obtained C code works properly and designer can go ahead with design procedure, otherwise it must stop and go back for doing again system design. To perform comparison between MIL Control Logic output signals and test harness Control Logic output signals, four different .m file can be used: *green\_comparison.mlx* file to compare green LED output signal, *red\_comparison.mlx* file to compare red LED output signal, *bm\_trqcmd\_comparison.mlx* file to compare bench motor command output signal, *tm\_trqcmd\_comparison.mlx* file to compare test motor command output signal.

In figure 5.21 *green\_comparison.mlx*'s MATLAB script is shown that allows to obtain results shown in figure 5.22. Here, the two signals result perfectly overlapped each other so in this output signal case for sure obtained Control Logic C code works properly.

```

1  clear all
2  close all
3  clc
4
5  load('out_SIL.mat')
6  SIL_signals=out.get('SIL_signals');
7  green_SIL=SIL_signals.get('green');
8  clear out
9  clear SIL_signals
10
11 load('out_harness.mat')
12 harness_signals=out.get('harness_signals');
13 green_harness=harness_signals.get('green');
14 clear out
15 clear harness_signals
16
17 figure,
18 plot(green_SIL.Values,'r'), hold on
19 plot(green_harness.Values,'b')
20
21 grid on

```

Figure 5.19: *green\_comparison.mlx*'s MATLAB script

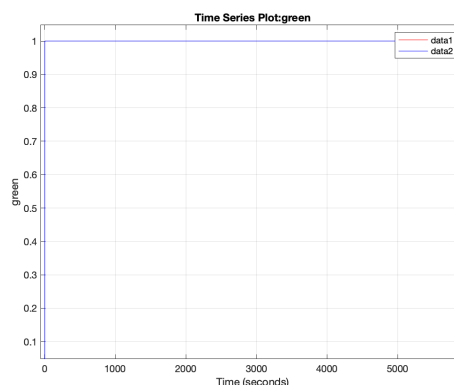


Figure 5.20: MIL and SIL Control Logic green signal comparison

In figure 5.23 *red\_comparison.mlx*'s MATLAB script is shown that allows to obtain results shown in figure 5.24. Also in this case, the two signals result perfectly overlapped each other so also in this output signal case obtained Control Logic C code works properly.

```

1  clear all
2  close all
3  clc
4
5  load('out_SIL.mat')
6  SIL_signals=out.get('SIL_signals');
7  red_SIL=SIL_signals.get('red');
8  clear out
9  clear SIL_signals
10
11 load('out_harness.mat')
12 harness_signals=out.get('harness_signals');
13 red_harness=harness_signals.get('red');
14 clear out
15 clear harness_signals
16
17 figure,
18 plot(red_SIL.Values,'r'), hold on
19 plot(red_harness.Values,'b')
20
21 grid on

```

Figure 5.21: *red\_comparison.mlx* MATLAB script

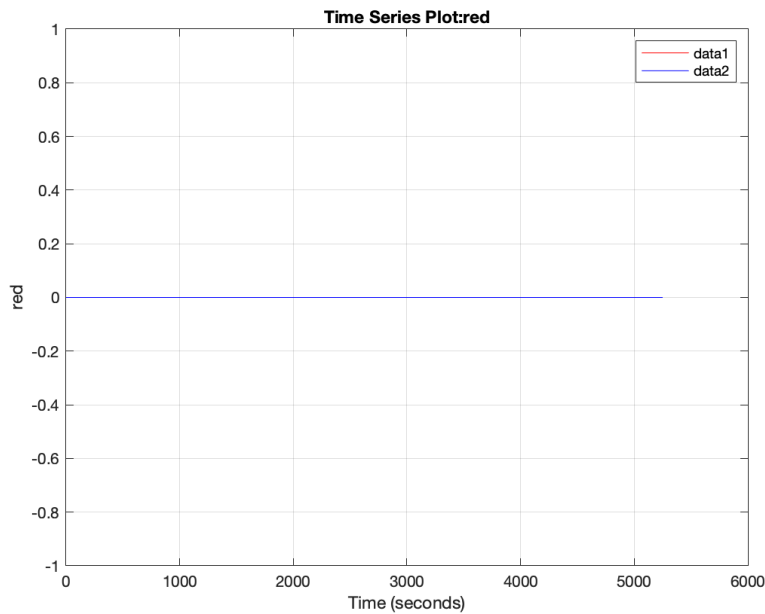


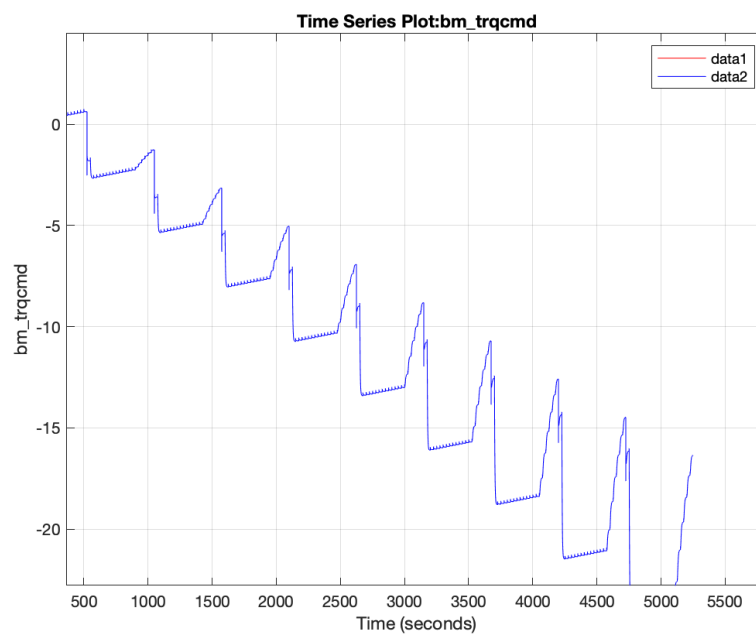
Figure 5.22: MIL and SIL Control Logic red signal comparison

In figure 5.25 *bm\_trqcmd\_comparison.mlx*'s MATLAB script is shown that allows to obtain results shown in figure 5.26 where the two signals are perfectly overlapped each other so also in this output signal case the proper Control Logic C code operation is verified.

```

1  clear all
2  close all
3  clc
4
5  load('out_SIL.mat')
6  SIL_signals=out.get('SIL_signals');
7  bm_trqcmd_SIL=SIL_signals.get('bm_trqcmd');
8  clear out
9  clear SIL_signals
10
11 load('out_harness.mat')
12 harness_signals=out.get('harness_signals');
13 bm_trqcmd_harness=harness_signals.get('bm_trqcmd');
14 clear out
15 clear harness_signals
16
17 figure,
18 plot(bm_trqcmd_SIL.Values,'r'), hold on
19 plot(bm_trqcmd_harness.Values,'b')
20
21 grid on

```

Figure 5.23: *bm\_trqcmd\_comparison.mlx*'s MATLAB scriptFigure 5.24: MIL and SIL Control Logic *bm\_trqcmd* signal comparison

In figure 5.27 *bm\_trqcmd\_comparison*'s MATLAB is shown that allows to obtain results shown in figure 5.28. Here, the two signals result perfectly overlapped each other which means that also in this case obtained Control Logic C code works in a proper way.



```

1 clear all
2 close all
3 clc
4
5 load('out_SIL.mat')
6 SIL_signals=out.get('SIL_signals');
7 tm_trqcmd_SIL=SIL_signals.get('tm_trqcmd');
8 clear out
9 clear SIL_signals
10
11 load('out_harness.mat')
12 harness_signals=out.get('harness_signals');
13 tm_trqcmd_harness=harness_signals.get('tm_trqcmd');
14 clear out
15 clear harness_signals
16
17 figure,
18 plot(tm_trqcmd_SIL.Values,'r'), hold on
19 plot(tm_trqcmd_harness.Values,'b')
20
21 grid on

```

Figure 5.25: *tm\_trqcmd\_comparison.mlx*'s MATLAB script

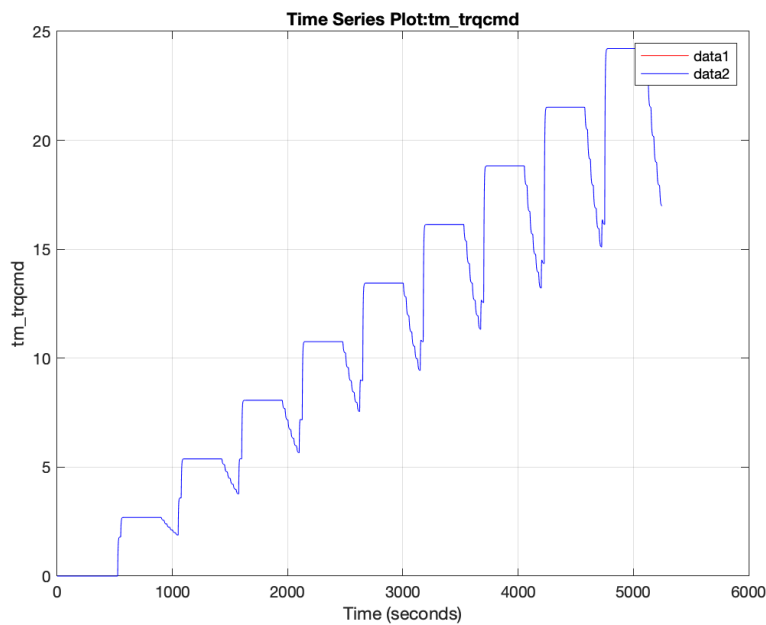


Figure 5.26: MIL and SIL Control Logic *tm\_trqcmd* signal comparison



# Chapter 6

## RCP code production

Once System design phase is completed physical system's model is obtained and validated. At that point, *Rapid Control Prototyping code production* V-shaped model's phase has to be performed in order to develop, optimize and test Control Logic's C code in real-time by using what is called *Rapid-Control-Prototyping-Platform (RCPP)* without manual programming. This is very useful because in System design phase physical system's model has been built in no real-time conditions and before to decide which kind of VMU has to be chosen to implement final physical system, system's behavior has to be verified in presence of real interfaces. So, this time what has to be changed are interfaces implemented in Control Interface reference model that allow communication between control law and other parts of system in order to perform both MIL testing and SIL testing in presence of real interfaces to verify system model and corresponding Control Logic C code's behavior. To do this, before modelled interfaces in Control interface reference model have to be substituted with new models that allow to simulate behavior of interfaces implemented in physical hardware. For this reason, *dSpace library*, described in *210128 dSPACE\_interfaces \_documentation file*, is adopted that allows to substitute discrete filters and quantization blocks, described in subsection 4.2.3, with ADC and DAC blocks as shown in figure 6.1.

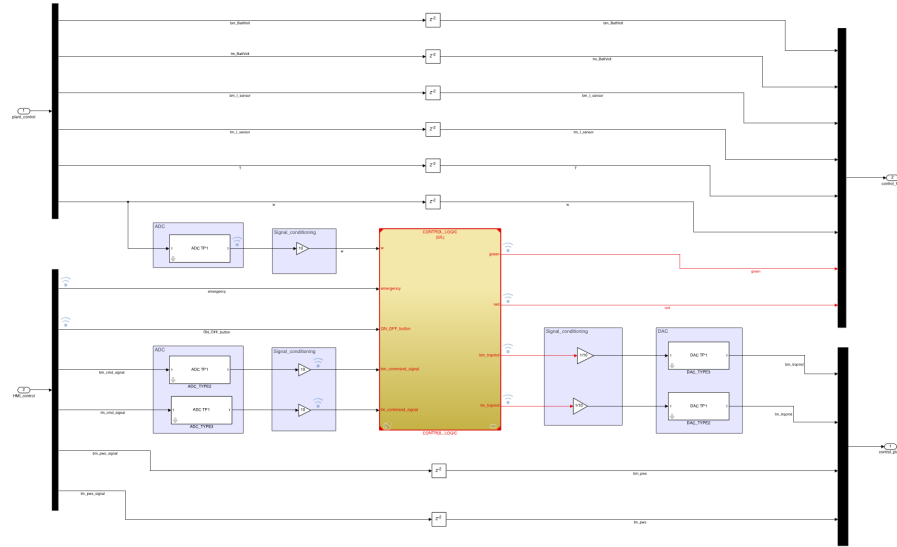


Figure 6.1: Control interface reference model in RCP phase

Opening ADC block, model reported in figure 6.2 can be observed. Here, three different blocks are implemented to convert analog input signals into digital signals: the first block is an analog filter that is intended to filter input voltage signals, the second one is a quantizer block whose quantizer interval is set on the basis of desired resolution while the third one is a gain that divides by ten signal values.

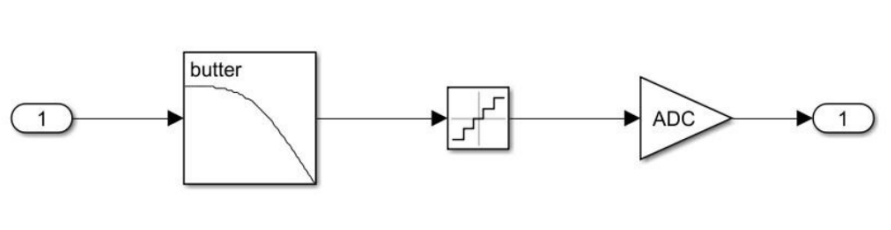


Figure 6.2: dSpace ADC block model scheme

Instead opening DAC block, model reported in figure 6.4 can be observed where the same blocks described in ADC block case are implemented. Indeed, a quantizer block is implemented to set quantization interval on the basis of desired resolution, a gain that multiplies by ten entering signals and an analog filter that allows to reconstruct original signal.

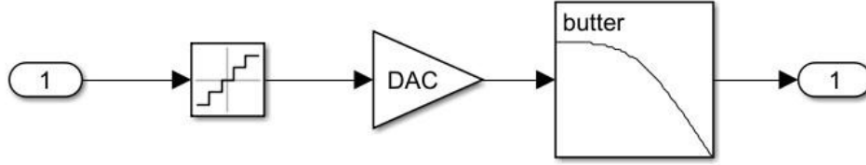


Figure 6.3: dSpace DAC block model scheme

Since dSpace ADC blocks divide by ten entering analog signals and dSpace DAC blocks multiply by ten entering digital signal, two signal conditioning sections must be implemented to adapt signals entering to Control Logic reference model and signals coming out from that reference model. Indeed, as highlighted in figure 6.4, a signal conditioning section is implemented at the input of Control Logic reference model while the other one is implemented at the output of Control Logic reference model.

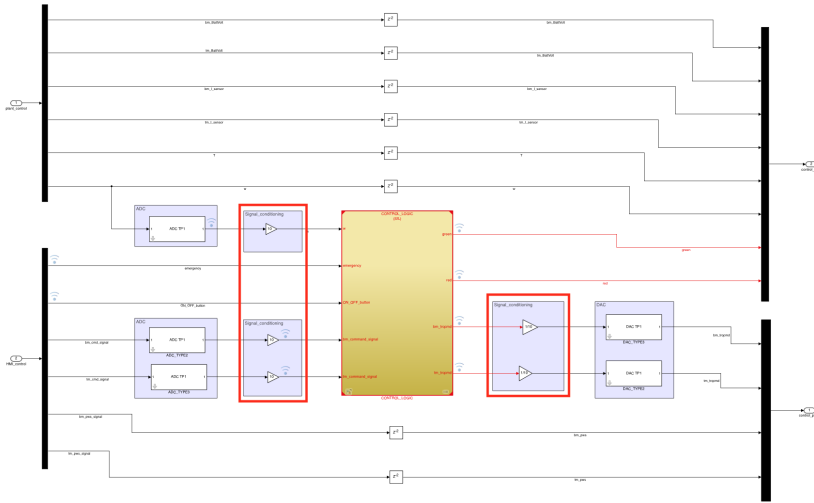


Figure 6.4: Signal conditioning sections in Control Interface reference model

This, obviously, implies a change in Control Data section of MATLAB script used to parametrize all implemented Simulink blocks in MTM. This time, indeed, no data must be defined in that section because to implement dSpace interfaces, *slblocks.m* and *dSPACE\_data.mlx* files must be run in order to add dSpace library among all other libraries in Library Browser section and to load needed parameters for parametrizing those blocks in MATLAB workspace. In figure 6.5 and 6. *slblocks.m* and *dSPACE\_data.mlx* files MATLAB script are shown.

```

1
2
3 function blkStruct = slblocks
4     Browser.Library = 'dSPACE_interfaces';
5     Browser.Name = 'dSPACE_INTERFACES';
6     blkStruct.Browser = Browser;
7
8
9

```

Figure 6.5: slblocks.m MATLAB script

HARDWARE INTERFACES DATA

Move here to reveal toolstrip

ADC TYPE1 DATA

LOW-PASS FILTER

```

1 order=1;
2 cut_f_AD=23e3*2*pi; %rad/s

```

QUANTIZER

$V\_ADC$   $\rightarrow$  peak voltage. Input Voltage Range  $\pm 10$  V

$N\_ADC$   $\rightarrow$  number of bits/ resolution

$quant\_AD$   $\rightarrow$  quantization interval

```

3 V_ADC=10;
4 N_ADC=16;
5 quant_AD=V_ADC/(2*N_ADC-1);

```

GAIN

```

6 ADC=0.1;

```

DAC TYPE1 DATA

move here to reveal toolstrip

QUANTIZER

$V\_DAC$   $\rightarrow$  peak voltage. Output Voltage Range  $\pm 10$  V

$N\_DAC$   $\rightarrow$  number of bits/ resolution

$quant\_DA$   $\rightarrow$  quantization interval

```

7 V_DAC=10;
8 N_DAC=16;
9 quant_DA=V_DAC/(2*N_DAC-1);

```

GAIN

```

10 DAC=10;

```

CONTINUOUS TRANSFER FUNCTION

```

11 cut_f_DA=500e3*2*pi;

```

Figure 6.6: slblocks.m's MATLAB script

All parameters to be load into MATLAB workspace to parametrize dSpace ADC and DAC blocks are listed and described below.

- *order*  
It defines low-pass filter's order. In this case, since a first order low-pass filter must be implemented inside ADC blocks it is set equal to 1.
- *cut\_f\_AD*  
It defines low-pass filter's cut-off frequency implemented inside ADC block.
- *V\_ADC*  
It defines peck voltage with which ADC can deal. In this case, input voltage range is set between -10 and 10 V
- *N\_ADC*  
It defines ADCs' resolution since it represents number of bits to be considered for computing quantization interval.
- *quant\_AD*  
It defines quantization interval computed by means relation defined in MATLAB script
- *ADC*  
It represents gain's value implemented inside ADC block
- *V\_DAC*  
It defines peck voltage with which DAC can deal. In this case, input voltage range is set between -10 and 10 V.
- *N\_DAC*  
It defines DACs' resolution since it represents number of bits to be considered for computing quantization interval.
- *quant\_DA*  
it defines quantization interval computed by means the same relation used in quant\_AD case defined in MATLAB script.
- *DAC*  
It defines gain's value implemented in DAC block.
- *cut\_f\_DA*  
It defines low-pass filter's cut-off frequency implemented inside DAC block.

Once real interfaces have been modelled, tests must be done to understand whether system's behavior changes or not with respect to that found during System Design phase. So, MIL and SIL testing must be done to understand whether implemented model and Control Logic's C code work properly in presence of modelled real interfaces.

## 6.1 Rapid-Control-Prototyping MIL testing

This test allows to verify system model's behavior in presence of modelled real interface. To understand whether implemented model during System design phase behaves properly or not, obtained results must be compared to those obtained during System design MIL testing: if they result equal to each other, for sure, implemented system's model behaves properly and designer can go away with design based on V-shaped model. So, also in this case to perform MIL testing the four different way to test motor must be considered in order to obtain results that have to be compared to those obtained during System Design V-shaped model phase.

### 6.1.1 RCP Single working point MIL testing

To perform this kind of test, like System design single working point MIL testing case, procedure described in subsection 4.2.4 related to Single working point test must be adopted to set properly MTM Simulink model and to load needed variables into MATLAB workspace. At the end of simulation, four .mat files in selected path are created that contain interesting parameter values that can be used to evaluate desired test motor's electrical and mechanical characteristics related to considered working point. To obtain these test motor's characteristics the same single `_working_point.m` file used in System Design MIL testing case has to be used where some variable values must be defined before to run MATLAB script. Parameters to be defined are the same to those described in subsection 5.1.1 and once MATLAB script is run results shown in figure 6.7 are obtained. Obviously, since these results must be compared to those obtained in System Design single working point MIL testing case, the same working point of  $\omega - T$  plane is considered: 20 Nm value is considered to pilot electric motor under test and 366.52 rad/s value is considered to pilot bench motor.



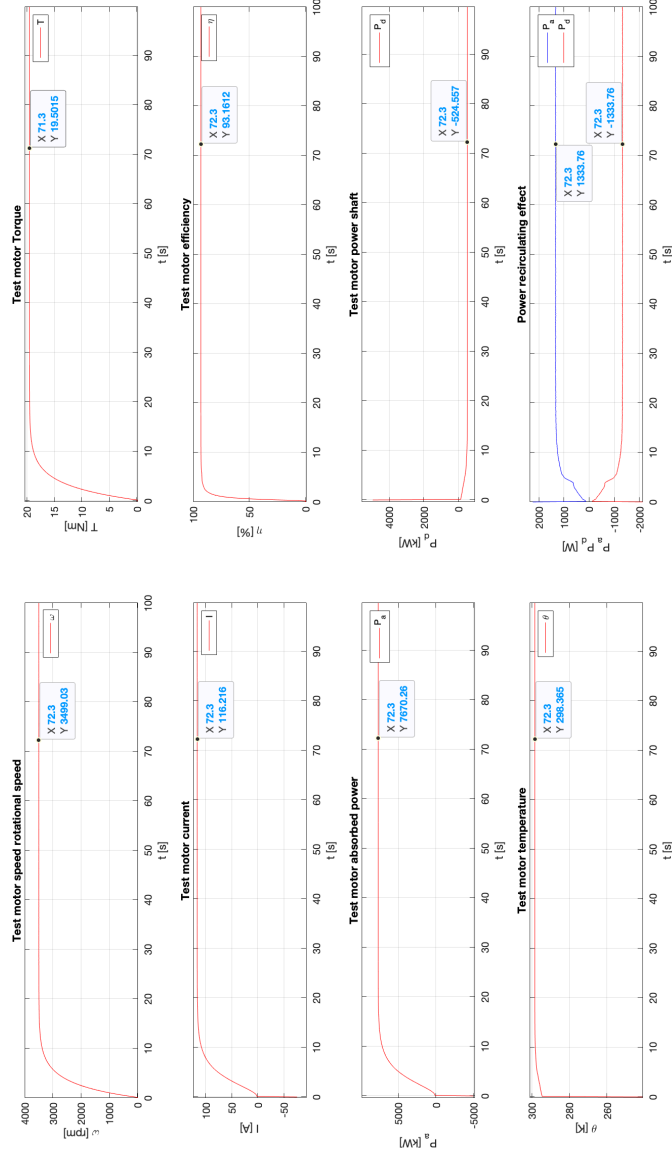


Figure 6.7: Single working point results in RCP MIL testing

Results shown in above figure are perfectly the same to those shown in figure 5.4 so, system's model works properly in presence of modelled real interfaces.

### 6.1.2 RCP Nominal power working point MIL testing

To perform this test, like System design nominal power working point MIL testing case, procedure described in subsection 4.2.4 related to Nominal power working point test must be adopted to set properly MTM Simulink model and to load needed variables in MATLAB workspace. At the end of simulation, like all other cases, in selected path four .mat files are created that contain interesting parameter values that can be used to obtain desired test motor mechanical and electrical characteristics related to working points considered during testing. To obtain these characteristics the same `nominal_power_working_point.m` file described in subsection 5.1.2 has to be used where some variables must be defined before to run the MATLAB script. Of course, parameters to be defined are the same to those described in subsection 5.1.2 and once MATLAB script is run, results shown in figure 6.8 are obtained.

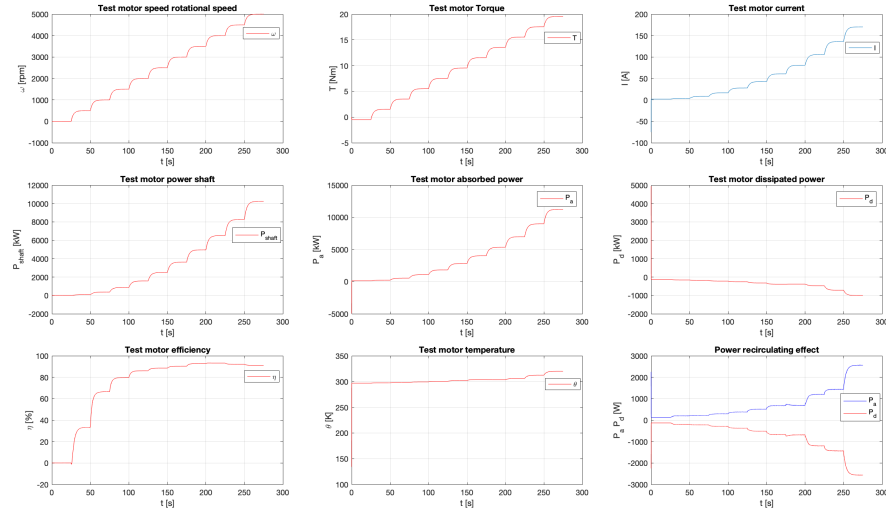
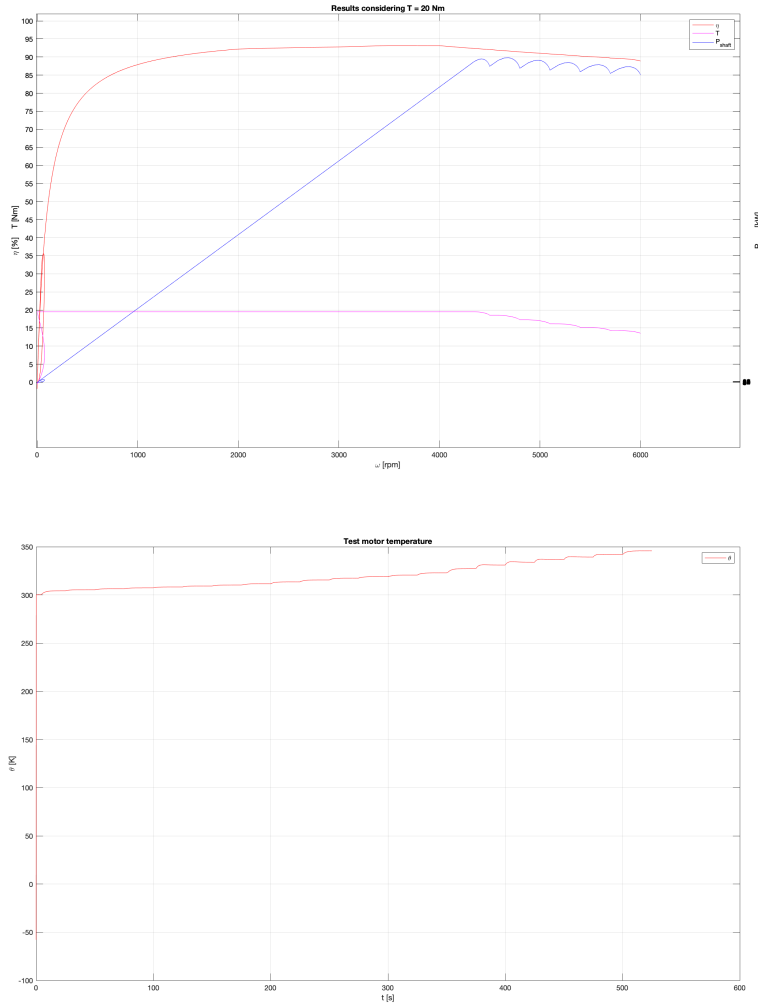


Figure 6.8: Nominal power working point results in RCP MIL testing

Comparing these results to those obtained in System Design Nominal Power Working Point MIL testing case (figure 5.6) it is possible to observe that they are equal to each other so for sure also in this case system's model works properly in presence of modelled real interfaces.

### 6.1.3 RCP Torque curves MIL testing

To perform this test, like System design torque curves case, procedure described in subsection 4.2.4 related to Torque curves test must be adopted to set properly MTM Simulink model and to load needed variables in MATLAB workspace. In this case, like previous ones, at the end of simulation the same four .mat files are created in selected path that can be used to evaluate desired test motor torque curves. To obtain these, the same Torque\_curves.m file can be adopted where parameters described in subsection 5.1.3 must be defined before to run MATLAB script and once this is done results shown in figure 6.9 are obtained



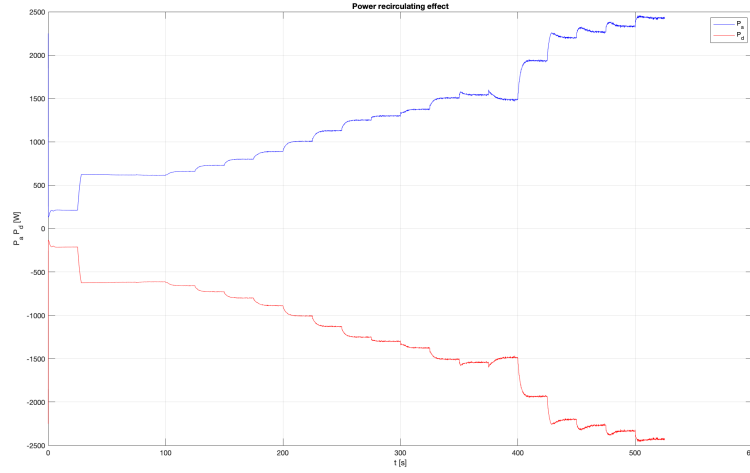


Figure 6.9: Torque curves results in RCP MIL testing

Comparing these results to those obtained in System Design Torque Curves MIL testing case it is possible to conclude that also in this case designed system's model works properly in presence of modelled real interface.

#### 6.1.4 RCP 3D maps MIL testing

To perform this test, like System design 3D maps MIL testing case, procedure described in subsection 4.2.4 related to 3D maps test must be adopted to set properly MTM Simulink model and to load all needed variables into MATLAB workspace. Also in this case, like all other cases, at the end of simulation in selected path four .mat files are generated that contain interesting parameter values that can be used to obtain desired test motor 3D efficiency and dissipated power maps and electric and mechanical characteristics related to all working points considered during test. To do this, at the end of simulation the same maps.m file described in subsection 5.1.4 can be used where some parameters must be defined before to run MATLAB script. Running that script results shown in following figures are obtained.

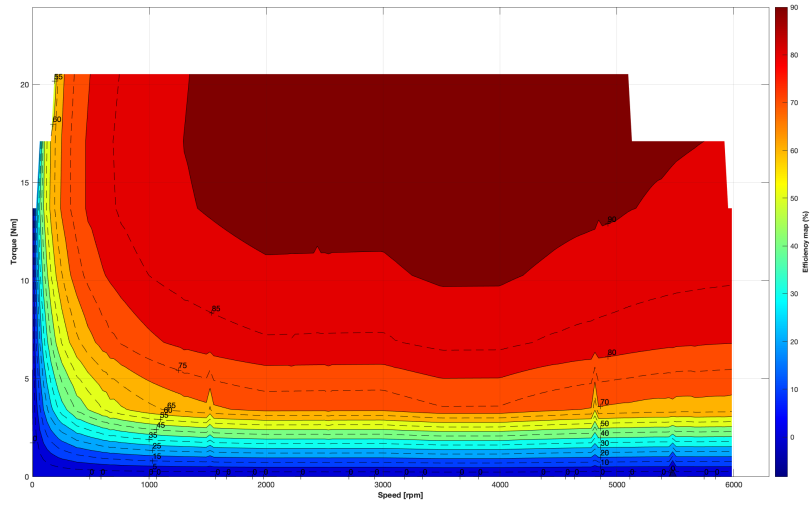


Figure 6.10: Efficiency map result in RCP MIL testing

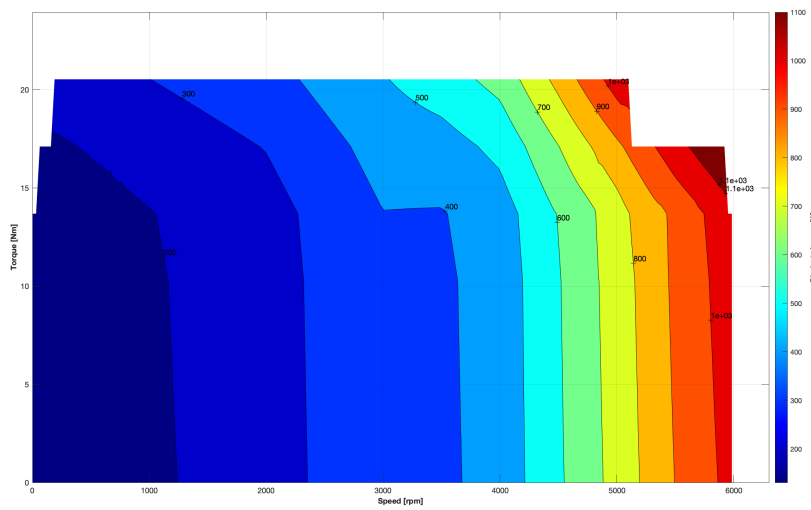


Figure 6.11: Dissipated power 3D map result in RCP MIL testing

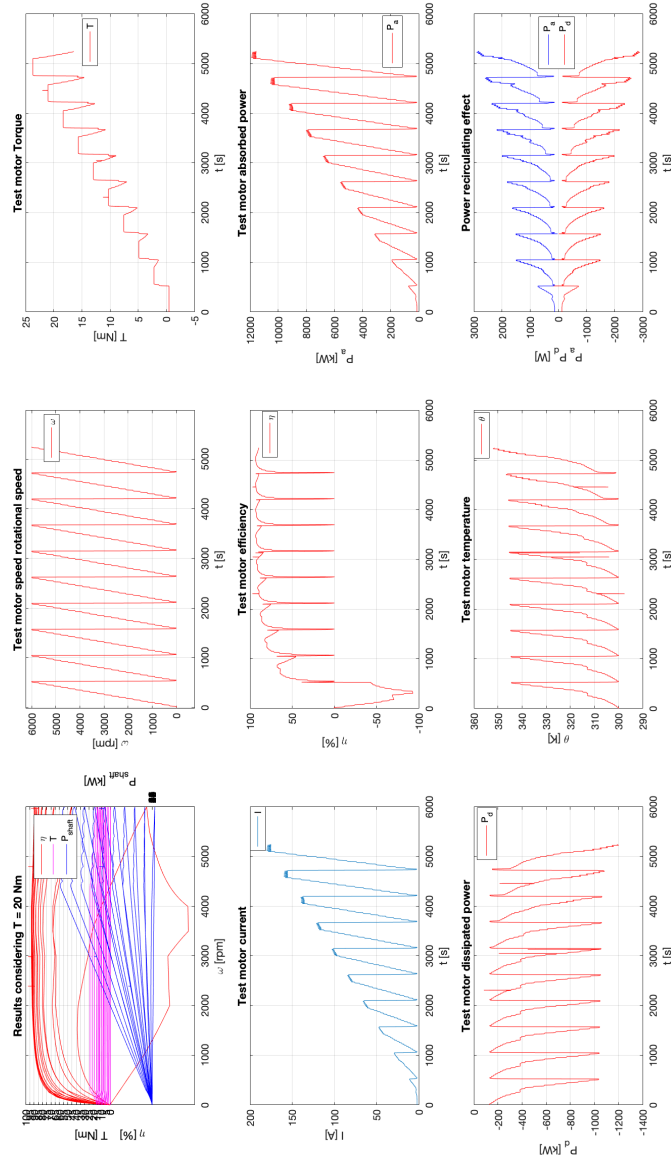


Figure 6.12: Test motor characteristics results in RCP MIL testing

Comparing these results to those obtained in System Design 3D maps MIL testing case it is possible to understand that designed system model works properly in presence of modelled real interfaces since results are perfectly equal to each other.

## 6.2 Rapid-Control-Prototyping SIL testing

After being designed, the system model is tested in presence of modelled real interface and verified that its behavior does not change with respect to the case where real interface are not modelled (model considered in System Design V-shaped model phase), corresponding Control Logic C code has to be obtained and to its behavior in presence of modelled real interfaces must be tested. Here, as explained in chapter 5 both plant's model and Control Logic's code are run in development machine. Also in this case to obtain Control Logic's C code, Software-in-the-loop must be selected from simulation mode drop-down menu that can be found in block parameters window of corresponding reference model, as shown in figure 6.13. Once this is done, all possible tests can be considered to perform simulation and obtain corresponding results to be compared to those obtained in MIL testing case and understand whether obtained C code works properly or not: if results are equal to each other, for sure, obtained Control Logic's C code works properly, otherwise there is something that does not work as desired and designer has to go back to find errors to be solved. Actually, if C code generation procedure ends in successfully manner the C code will work properly. Status of C code generation procedure can be observed by means diagnostic viewer.

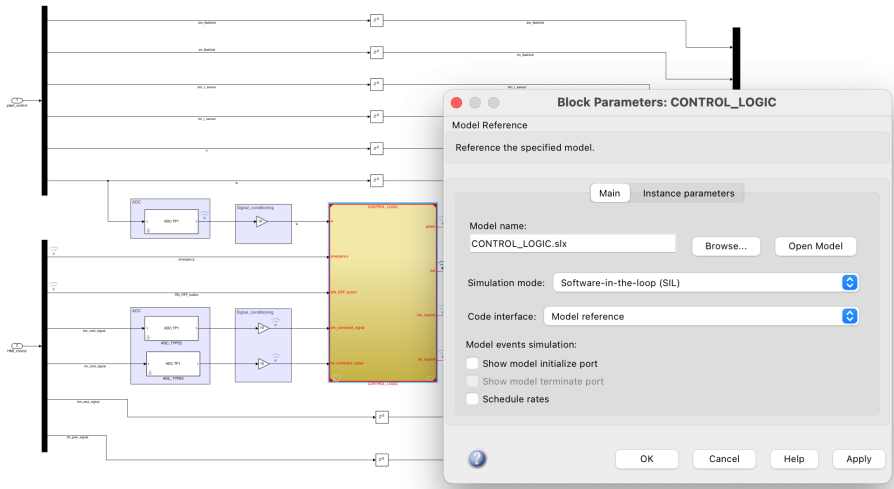


Figure 6.13: Control Logic reference model setting in RCP SIL simulation

To avoid being repetitive only a single kind of test is reported: 3D maps test which represents the hardest test due to the very huge quantity of working points that is considered. To perform this test, besides to do what before described the same procedure explained in subsection 4.2.4 related to 3D maps test has must be followed in order to set properly MTM Simulink model and to load needed variables in MATLAB workspace. Once simulation is performed in selected path always the same four files are created that

contain interesting parameter values that can be used to obtain results shown in following figures that can be compared to those obtain in RCP 3D maps MIL testing case for understanding whether obtained Control Logic's C code works properly or not such that designer can decide whether it is possible to go away with design or not. Performing this test results shown in figure 6.14 are obtained.

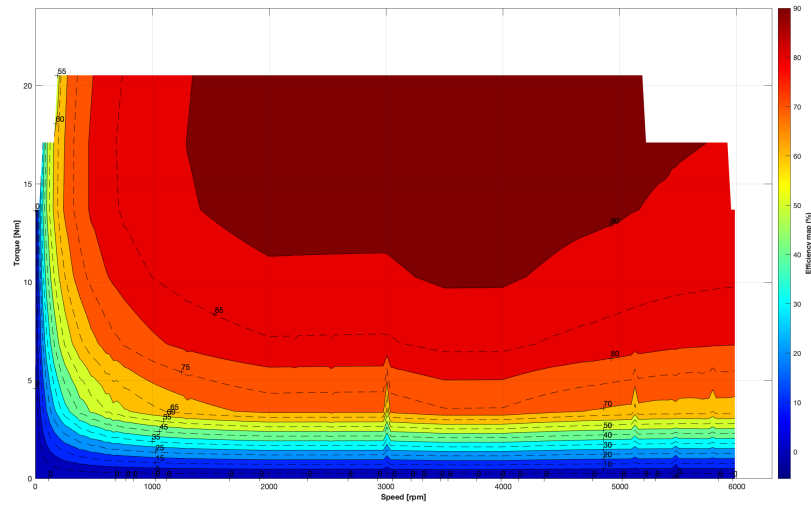


Figure 6.14: Efficiency 3D map result in RCP SIL testing

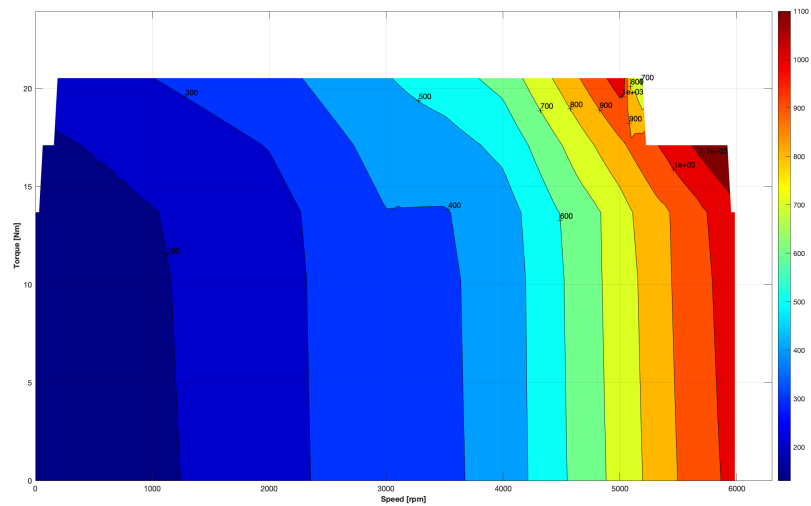


Figure 6.15: Dissipated power 3D map result in RCP SIL testing



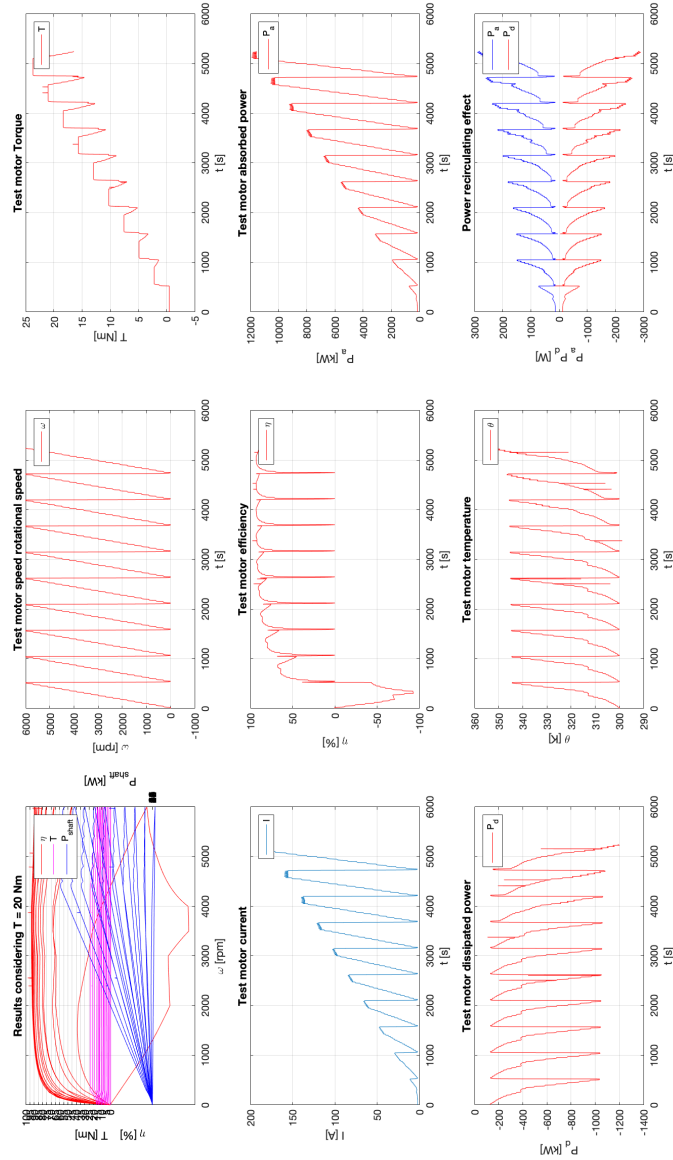


Figure 6.16: Test motor characteristics results in RCP SIL testing

Comparing these results to those obtained in RCP 3D maps MIL testing it is possible to observe that nothing changes so for sure also in this case obtained Control Logic's code works properly and designer can proceed with design.



# Bibliography

- [1] Fratino Thesis. *ICE TEST BENCH REDEVELOPMENT FOR HYBRID AND ELECTRIC POWERTRAIN*. Italy, 2019.
- [2] Coletta Thesis. *Model-based Design of an Automotive Control Code with a modified V-cycle and Modular Model approach*. Italy, 2019.
- [3] The MathWorks, Inc. *Signal Generator*  
<https://www.mathworks.com/help/simulink/slref/signalgenerator.html>
- [4] The MathWorks, Inc. *Mapped Motor*  
<https://www.mathworks.com/help/autoblks/ref/mappedmotor.html>
- [5] The MathWorks, Inc. *Rotational Inertia*  
<https://www.mathworks.com/help/vdynblks/ref/rotationalinertia.html>
- [6] The MathWorks, Inc. *Delay*  
<https://www.mathworks.com/help/simulink/slref/delay.html>
- [7] The MathWorks, Inc. *Discrete Filter*  
<https://www.mathworks.com/help/simulink/slref/discretefilter.html>
- [8] The MathWorks, Inc. *Quantizer*  
<https://www.mathworks.com/help/simulink/slref/quantizer.html>
- [9] The MathWorks, Inc. *State charts for modeling control logic*  
<https://www.mathworks.com/discovery/state-chart.html>
- [10] The MathWorks, Inc. *Switch*  
<https://www.mathworks.com/help/simulink/slref/switch.html>
- [11] The MathWorks, Inc. *Create Subsystems*  
<https://www.mathworks.com/help/simulink/ug/creating-subsystems.html>
- [12] The MathWorks, Inc. *Model References*  
<https://www.mathworks.com/help/simulink/model-reference.html>
- [13] The MathWorks, Inc. *Dashboard*  
<https://www.mathworks.com/help/simulink/dashboard.html>

- [14] The MathWorks, Inc. *Generate C Code from Simulink Model*  
<https://www.mathworks.com/help/dsp/ug/generate-c-code-from-simulink-model.html>
- [15] The MathWorks, Inc. *dSpace interfaces DAUIN - Vehicle Management Unit 210128 dSPACE\_interfaces\_documentation*. Italy, 2021.