Gettting the FAMM-DD operational

Getting the FAMM-DD operational

DCT 2002.76

Jeroen de Boeij (s461064)
December 15, 2002

Preface

The FAMM Direct Drive is a pick-and-place robot developed at Philips CFT. The main goal was to design the fastest robot ever built. Several types have been developed and the latest version uses no transmission and is therefore called Direct Drive. The FAMM is used for research and was never intended to be a commercial robot. For almost 5 years the FAMM have been obsolete at Philips, so the robot was transported to the Dynamics & Control laboratorium for further research.

Upon delivery the controller hardware was removed. The power amplifiers were available and also the interface for the sensors was intact. In order to get the FAMM operational the power amplifiers and the sensors needed to be connected to the dSpace controller board. After connecting the FAMM in- and outputs to the dSpace controller board a C-code Matlab-Simulink S-function had to be written to control the FAMM from the Matlab-Simulink environment.

The goal of my assignment was to get the FAMM operational, together with Jeroen van Helvoort. In this report is described which changes were made to the original configuration and how everything is connected to the dSpace controller board. Also the framework for the C-code is described.

Contents

1 Hardware Analysis **4**
- 1.1 Robot . . . 4
  - 1.1.1 Motors . . . 4
  - 1.1.2 Encoders . . . 5
  - 1.1.3 Sensors . . . 5
  - 1.1.4 Security . . . 7
- 1.2 Control Cabinet . . . 7
  - 1.2.1 Main Switch Panel (A2) . . . 7
  - 1.2.2 Data Acquisition Unit (A4) . . . 7
  - 1.2.3 Power Amplifiers (A6) . . . 8
  - 1.2.4 Transformer (A8) . . . 9

2 Hardware Adaptations **10**
- 2.1 Main Switch Panel . . . 10
  - 2.1.1 Emergency Stop A2-X05 . . . 10
  - 2.1.2 Phase Guard . . . 10
- 2.2 Data Acquisition Unit . . . 10
  - 2.2.1 Encoders . . . 11
  - 2.2.2 Motor Commands . . . 12
  - 2.2.3 Digital I/O . . . 12
- 2.3 Power Amplifiers . . . 14

3 FAMM Procedures **15**
- 3.1 Commutation . . . 15
  - 3.1.1 Principle of a 3 phase AC Motor . . . 15
  - 3.1.2 Zero Search Procedure . . . 16
- 3.2 Homing . . . 17
  - 3.2.1 Homing Motor 1 and 2 . . . 17
  - 3.2.2 Homing Motor 3 and 4 . . . 17
- 3.3 Safety . . . 18
  - 3.3.1 Position Violation . . . 18
  - 3.3.2 Velocity Violation . . . 18
  - 3.3.3 Current Violation . . . 18

4 Simulink Implementation **20**
- 4.1 Structure of FAMMDD.mdl . . . 20
- 4.2 dSpace Interface . . . 22
- 4.3 S-Function . . . 23
  - 4.3.1 I/O Definitions . . . 23
  - 4.3.2 S-Function Framework . . . 23
  - 4.3.3 Status Definitions . . . 25

5 Conclusion **26**

A Connector Layout **27**
A.1 Connector A4-X2 27
A.2 Connector A4-X20 27
A.3 Connector A4-X21 28
A.4 Connector A4-X24 29
A.5 Connector A4-X25 30
A.6 Connector A4-X26 31
A.7 Connector digital I/O 32

B PC Board **33**

C Simulink Layouts **34**

D C-Code **35**
D.1 IO Definitions 35
D.2 Status Definitions 36
D.3 S-Function 37
D.4 Workspace Initialization 44
D.5 Reference Generator 46

Chapter 1

Hardware Analysis

In order to connect te robot to the dSpace controller board a thorough analysis of the hardware is necessary. The FAMM consists of two parts: the robot and a control cabinet. The robot itself is connected to an control cabinet with three connectors. Two for the power supply to the motors (380V AC) and one for output of the sensors. In this chapter both the robot as the control cabinet are analyzed.

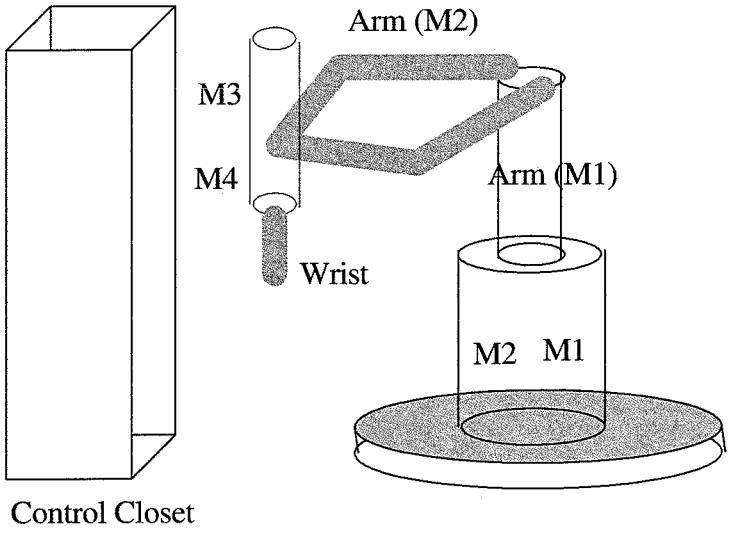

Figure 1.1: FAMM: control cabinet and robot

1.1 Robot

In this section the motors, encoders and sensors of the robot are discussed.

1.1.1 Motors

The robot has four motors, two in the main axis and two in the wrist. All four motors are 380V (three phases) AC motors. The Direct Drive version of the FAMM [4] doesn't a transmission between the motors and the axes.

The motors in the main axis (motor 1 and 2) operate at a maximum current level of 12 [A] and have eight pole pairs. When the motors move in the same direction, the arm of the robot is rotating. If one motor is moving in the opposite direction, the arm of the robot is translating.

The motors in the wrist operate at a maximum of 9 [A] and have four pole pairs. The movement of the wrist also depends on the rotation direction of motor 3 and 4. When both motors rotate in the same direction, the wrist is translating. When one motor is turning in the opposite direction, the wrist is rotating. This differs from the main axis movement, because the motors in the wrist are not mounted in the same direction, as is the case in the main axis.

All four motors are powered by three current phases (R S and T). To move the motor phase S must have a $\frac{2}{3}\pi$ phase shift with respect to phase R. The direction of movement is determined by the sign of the amplitude of the current. The third phase T is generated in the control cabinet from R and S (T=-S-R), so to rotate the motors only two phases have to be generated by the user. More about this topic is found in chapter 3.

1.1.2 Encoders

Each motor has an incremental encoder. Also, at the end of the main axis a fifth encoder is present. This fifth encoder enables the user to measure torsion in the main axis. For the control of the FAMM this encoder is not important and therefore this encoder is not connected to the dSpace controller board.

Encoders of motor 1 and 2

The encoders in the main axis for motor 1 and 2 are sine wave encoders. These encoders output a sine, cosine and an index pulse. In the control cabinet the sine and cosine are interpolated to improve the resolution by a factor four. After this interpolation the signals are converted to standard S0-S90 signals, which increases the resolution again by a factor four. Furthermore there is a transmission between de motors and the encoders. The transmission factor is 1:10 so one revolution of the motor equals ten encoder revolution. The encoders have five thousand increments per revolution. This means that one motor revolution equals $(4 * 4 * 10 * 5000 = 800.000)$ eight hundred thousand increments. This information from Philips CFT has not yet been verified, so the number of increments per revolution still needs to be checked.

Encoders of motor 3 and 4

In the wrist there are also two encoders for motor 3 and 4. These encoders are standard encoders and its signals are standard S0-S90, which implies an increase of its resolution by a factor four. The encoders are connected to the motors by an 1:4 transmission. This transmission also improves the resolution by a factor four. The resolution is $(4*4*2500 = 40.000)$ forty thousand increments per revolution. This information from Philips CFT has also not yet been checked.

1.1.3 Sensors

The robot has 14 sensors. These sensors are used to mark important positions in the working range of the robot. There are three types of sensors.

EWG Sensors

These sensors mark the end of the normal working area (in Dutch: Einde Werk Gebied) of the robot. There are five EWG sensors:

1. EWG M1
   This sensor marks the end of the normal working area of motor 1. Its signal is true when operating inside the working area. When passing the border, the signal becomes false but only close to the border. When moving across the border, the signal becomes true again. So EWG M1 outputs a pulse with the value false when the end of the working area is reached.

2. EWG M2
   This sensor works the same as EWG M1 except that it indicates the end of the normal working area of motor 2.

3. EWG ARM
   This EWG sensor indicates the working area of the arm. The arm can translate but its movement is bounded by the inner and outer position of the working area. When the arm is between its bounds the sensor outputs the value true. When the arm reaches the end of its working area (either its inner or outer bound) the output value becomes false. When the arm is outside its normal working area the value of EWG ARM stays false.

4. EWG M3
   The EWG M3 sensor marks the outer bound of the translation of the wrist. It outputs the value true when the wrist has reached or passed its outer position.

5. EWG M4
   This sensor marks the inner bound of the working area of the wrist. When outside the working area, it outputs the value true.

EOS Sensors

The EOS sensors indicate the End Of Stroke of the four motors and the arm. When the robot passes this sensor it is close to its mechanical limitations. Moving further will cause serious damage. The characteristics of these sensors are discussed below.

1. EOS M1
   The EOS sensor of motor 1 outputs false when the end of stroke is reached, otherwise true. When moving further a claxon is activated to indicate the end of stroke when moving the robot by hand.

2. EOS M2
   This sensor indicates the end of stroke of motor 2. It also outputs the value false when motor 2 reaches its end of stroke. This motor also has a claxon available, which is activated when the motor passes the end of stroke sensor.

3. EOS ARM
   The EOS ARM sensor marks the inner and outer position of the arm. When the arm is between its bound, the value is true. If the arm reaches the end of stroke the value becomes false.

4. EOS M3
   This sensor indicates the end of stroke of the wrist at its outer bound. When the wrist reaches its outer bound the value of the sensor becomes true, else it has value false.

5. EOS M4
   The EOS M4 sensor marks the inner position bound of the wrist. When the inner bound is reached the sensor outputs the value true, else the value is false.

ZERO Sensors

The ZERO sensors are used to mark the homing positions of the motors. Each motor has its own zero sensor. These sensors are used in the homing procedure which is described in chapter 3. The zero sensors are not the actual zero, but they are located very close to an index pulse of the encoder. Each encoder has one index, but because of the transmission one revolution of the motor means multiple revolutions of the encoder. The index pulse is much more accurate then the zero sensor. The zero sensors are used to make the right index pulse easier to find.

PTC

The robot has a thermocouple (PTC) for each robot. These thermocouples are connected to the power amplifiers in the control cabinet. This is an extra safety measure to prevent the motors from overheating. The sensors are not implemented on the dSpace controller board but they are connected to the power amps. When the motors warming up too much, the power amps will be switched off.

1.1.4 Security

The robot is equipped with two safety measures. The first is a short-circuit switch. This switch is opened when the power on the robot is enabled. When the switch is closed the motors are short-circuited and the generation of current by the motors, when the robot is moved by hand, is prevented. Also moving the robot is much more difficult if the switch is closed. Unwanted movement, when the robot is not enabled, will therefore be difficult. Also during an emergency procedure the power on the robot will be disabled and the high damping will cause the robot to slow down faster.

Another switch is connected as an emergency stop at the control cabinet. This switch has to be closed (short-circuited) in order to disable the emergency stop and to allow power on the robot. During the installation and testing of the hardware of the FAMM it was not possible to determine how this switch was working. Therefore this switch has been disabled in the control cabinet. More about safety procedures is found in chapter 3 later in this report.

1.2 Control Cabinet

The control cabinet is a closet which contains (from bottom to top) a transformer (section A8), power amplifiers (section A6) for the four motors of the robot, a connection board and data acquisition unit which processes the encoder output of the encoders of motor 1 and 2 (section A4) and the main switch panel (section A2).

In this section the parts of the control cabinet are described and some important features are discussed. Also the connectors connected to each part of the panel are described. All connectors have a X-number and also the A-number of the section e.g. connector A2-X1 is the main power connector on the main switch panel (section A2). Information about connections of each pin of the connector is found in chapter 2 and appendix A.

1.2.1 Main Switch Panel (A2)

The main switch panel (in Dutch: Hoofd Schakel Paneel) or HSP as it is referred to in the Philips documentation is the main power supply for all the hardware. It contains a number of switches and relays. Also all the hardware and software emergency stops are connected to this panel. To enable the power to the power amplifiers and the robot the power supply must be OK and also all the emergency stops must be disabled.

Several connectors are connected to the HSP. These are listed in table 1.1.

This is the original configuration. Later some adaptations will be made, which are described in the next chapter.

1.2.2 Data Acquisition Unit (A4)

In the Data Acquisition Unit (or VME unit as it is referenced in the Philips documentation) all inputs and outputs are processed. At Philips the four DMC (Digital Motion Control) units for the control of the four motors where also in this unit. All relevant inputs and outputs were connected to these DMC units. Since these DMC units where removed before transportation to the D&C lab, only a bunch of connectors, formerly connected to the DMC units, were present. All the I/O

connector	#pins	function
A2-X01	6	380 [V] main power
A2-X02	6	380 [V] power output to the transformer
A2-X03	6	power on/off
A2-X04	4	Emergency Stop
A2-X05	4	Emergency Stop
A2-X06	4	Emergency Stop
A2-X07	4	Emergency Stop
A2-X08	3	220 [V] power supply (output)
A2-X09	3	220 [V] power supply (output)

Table 1.1: List of connectors on the HSP unit

connections have to be connected to the dSpace controller board, so a lot of changes, described in the next chapter, were made to this unit.

In this unit there are two DC sources present for 15V DC and 24V DC. All digital signals related with the PA unit operate at +15 [V], all other signals at +24 [V]. Also the +24 [V] DC supply is used for the electronics at two printed circuit boards (PC boards). On these two PC boards (sine wave converters) the signals from the sine wave encoders of motor 1 and 2 are interpolated and converted to standard S0-S90 encoder signals. Also there is some additional logic to combine the five EWG and five EOS sensors to one EWG and 2 EOS signals. This function is not used in the lab. Also there is a pushbutton available to reset the power amps. More detailed information about the connections to the sine wave converters is found in chapter 2 and appendix A.

All the connectors at the Data Acquisition Unit (A4) are listed in table 1.2.

connector	#pins	function
A4-X01	3	220 [V] input
A4-X02	7	relais to switch power on/off
A4-X03	-	not in use
A4-X04	15	encoder motor 1
A4-X05	9	control signals to PA 1
A4-X08	15	encoder motor 2
A4-X09	9	control signals to PA 2
A4-X12	15	encoder motor 3
A4-X13	9	control signals to PA 3
A4-X16	15	encoder motor 4
A4-X17	9	control signals to PA 4
A4-X18	15	encoder 5 (for measuring torsion in main axis)
A4-X20	25	PA error signals
A4-X21	25	EWG, EOS and ZERO sensors plus sensor power supply
A4-X22	-	input to DMC units; connection removed
A4-X23	-	output from DMC units; connection removed
A4-X24	15	PA enable signals
A4-X25	64	connector sine wave converter 1
A4-X26	64	connector sine wave converter 2

Table 1.2: List of connectors on the VME unit

1.2.3 Power Amplifiers (A6)

The Power Amp section convert the control signals for each motor (phase R and S) to real current signals to the motors (phase R, S and T). Phase T is computed automatically by the amplifier.

The control signals can range from -10 [V] to +10 [V]. The amplifiers are AC amps so the input signals must be sine waves. Constant inputs will cause damage to the amps, [2].

Besides signals R and S there are other signals to control the amps. These are digital signals operating at +15[V]. Every amp has an enable signal and it sends a PA OK signal if the amp self check is completed successfully when it is turned on. Also there is a reset signal. One signal is used to reset all amps.

The outputs of the amps are three phases R, S and T. The power amps for motor 1 and 2 can output a maximum of 12 [A], the amps for motor 3 and 4 have a maximum output of 9 [A].

In table 1.3 all connectors on the PA unit are listed.

connector	#pins	function
A6-X01	3	220 [V] power supply
A6-X02	6	380 [V] power supply
A6-X03	10	Three phase current to motor 1
A6-X04	10	Three phase current to motor 2
A6-X05	10	Three phase current to motor 3
A6-X06	10	Three phase current to motor 4
A6-X07	10	380 [V] power
A6-X08	9	control signals from DMC for motor 1
A6-X09	9	control signals from DMC for motor 2
A6-X10	9	control signals from DMC for motor 3
A6-X11	9	control signals from DMC for motor 4
A6-X12	15	Thermocouple PTC
A6-X13	15	PA enable and reset signals
A6-X14	25	status output from PA's; not in use

Table 1.3: List of connectors on the PA unit

1.2.4 Transformer (A8)

The last unit to be discussed is the transformer (A8). There is no Philips documentation available about this unit, so there is little to discuss. The transformer is powered by a cable connected to A2-X02 on the HSP unit. This is the power input of the transformer. The output is connected to the PA unit (A6).

Chapter 2

Hardware Adaptations

To be able to control the FAMM in the Matlab-Simulink environment the hardware must be connected to the dSpace controller board. Several changes have been made to the connector layout described in chapter 1. In this chapter for each part of the control cabinet will be described which changes were made.

2.1 Main Switch Panel

Two changes are made to the HSP. One of the emergency stops is disabled and also one switch, which seemed to malfunction, has been bypassed.

2.1.1 Emergency Stop A2-X05

The main switch panel supplies power to the other units. In order to enable the power on the robot, several conditions (switches) must be checked. All the emergency stops have to be true. An emergency stop is true when the wire is short-circuited. When an emergency stop is pushed, the connection will be broken and the power is switched off. Originally only two emergency stops were connected. The other two connections for emergency stops were fitted with decoys. One of the connected emergency stops was connected inside the robot. It was not possible to find out how this emergency stop was working. In order to get the HSP working, it was necessary to short-circuit the connection inside the HSP and disable this emergency stop. Therefore emergency stop connection A2-X05 is disabled.

2.1.2 Phase Guard

One of the switches inside the HSP is a phase guard. This switch is connected to an input, the three phases and an output. If all three phases are present and conform certain specifications the input and output are connected (true). If not, input and output are not connected and it will not be possible to enable power to the robot. The phase guard inside the HSP never switched to true, even when it was for sure that all three phases were present and correct. The power supply in the lab is conform all specifications and therefore the phase guard has been disabled. The output wire is removed from pin 12 and connected to pin 11. The result is that the input and output are always connected.

2.2 Data Acquisition Unit

In the data acquisition unit the most changes to the original configuration were made. A large number of connections have been redirected to the dSpace controller board. In this section all

changes are discussed. The signals from and to the dSpace controller can be divided into three parts.

1. **Encoder signals**
   dSpace has a standard connection for S0-S90 encoder signals. The signals from encoder 3 and 4 can be redirected directly to dSpace. The signals from encoder 1 and 2 on the contrary are converted on the sine wave converters to S0-S90 signals (encoder 5 is also converted on the sine wave converters, but is not implemented). The S0-S90 signals from encoder 1 and 2 have to be redirected from the sine wave converters to the dSpace controller board.

   When the robot is moving at maximum speed, the signal has a frequency of 1.5 MHz. The maximum sample rate of the dSpace encoder input is 8.3 MHz. Therefore all position information is sampled but a small error will occur when moving at high speeds.

2. **Motor commands**
   The signals send to the power amps are currents varying between -10 [V] and +10 [V]. For each motor 2 control signals are necessary, so a total of eight outputs must be implemented. dSpace has 8 outputs that can operate in this range so these outputs are used to generate the input signals for the power amps.

3. **Digital signals**
   The FAMM has a number of digital input and output signals. dSpace has a 50 pins I/O (input/output) connector for these signals. In total 32 digital signals can be connected to this I/O in blocks of eight signals. For each block can be defined whether it is an input or output block. The EOS, EWG and ZERO sensors and also the alarm and PA OK signals are digital inputs (total: 20). The PA enable and power off signals are digital output signals (total: 5).

2.2.1 Encoders

Encoder 3 and 4 were connected to A4-X12 and A4-X16 to connect them to the DMC units. These connections are now obsolete. The encoder signals from encoder 3 and 4 are standard S0-S90 signals, the only problem was that the signals were not connected on the right pin to connect them to dSpace (enc 3 and enc 4). For both encoders the cable is reconfigured to change the connections to the right pins.

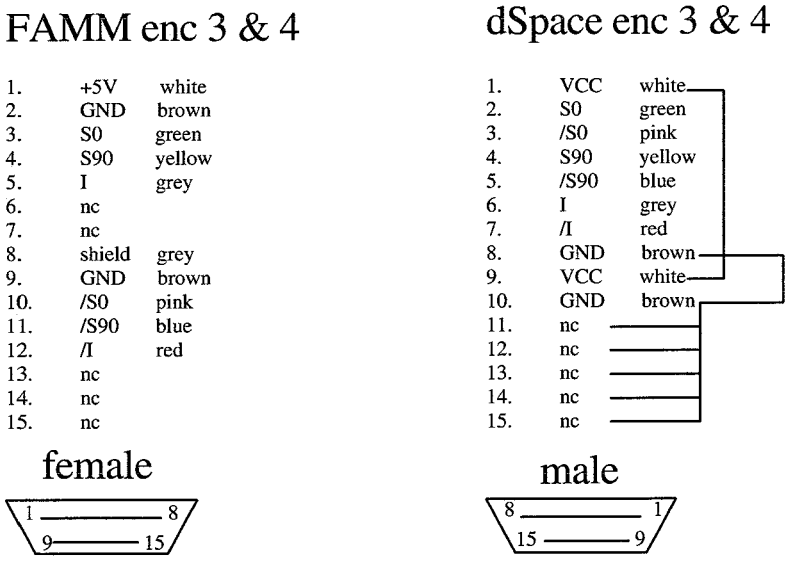

Figure 2.1: Layout of cable from FAMM to dSpace for encoder 3 and 4

The S0-S90 signals from encoder 1 and 2 are generated on the sine wave converters. These signals are connected to the back of the obsolete connectors A4-X12 (for encoder 1) and A4-X16 (for encoder 2). These connectors are connected with a standard cable to the dSpace controller board (enc 1 and enc 2). The symbol "/" in table 2.1 means *not*.

The connection scheme of connections A4-X12 and A4-X16 is the same as the dSpace connections of encoder 3 and 4 (figure 2.1).

Sine wave	back A4-X12	enc 1	Sine wave	back A4-X16	enc 2
-	A4-X12 p1	VCC (+5V)	-	A4-X16 p1	VCC (+5V)
A4-X26 26a	A4-X12 p2	S0	A4-X26 29a	A4-X16 p2	S0
A4-X26 26c	A4-X12 p3	/S0	A4-X26 29c	A4-X16 p3	/S0
A4-X26 27a	A4-X12 p4	S90	A4-X26 30a	A4-X16 p4	S90
A4-X26 27c	A4-X12 p5	/S90	A4-X26 30c	A4-X16 p5	/S90
A4-X26 28a	A4-X12 p6	I	A4-X26 31a	A4-X16 p6	I
A4-X26 28c	A4-X12 p7	/I	A4-X26 31c	A4-X16 p7	/I
-	A4-X12 p8	GND	-	A4-X16 p8	GND
-	A4-X12 p9	VCC (+5V)	-	A4-X16 p9	VCC (+5V)
-	A4-X12 p10–p15	GND	-	A4-X16 p10–p15	GND

Table 2.1: Signals of encoder 1 and 2 redirected to A4-X12 and A4-X16

2.2.2 Motor Commands

The analogous signals to the power amps were sent by the DMC units from connectors A4-X5, A4-X9, A4-X13 and A4-X17. These connectors are now obsolete and as discussed later in this chapter, used for the digital I/O.

2.2.3 Digital I/O

All the digital signals are connected to the dSpace digital I/O. Originally all the digital sensors (EOS, EWG, ZERO, alarm and Power off) operated on 24V and all the PA signals (PA enable and PA OK) on 15V. The dSpace controller board can operate only +5 [V] digital signals. The operating voltage of all the digital signals is changed to +15 [V]. It is checked whether all sensors would operate at this voltage but above +12 [V] all sensors worked correctly. The digital output signals (PA Enable and Power off) are switched with relays by the digital I/O signals (5V). The PA enable signals are on (true) when the wires are short-circuited and off (false) when not connected. The Power off signal is on (true, power switched off) when the wire is disconnected. When short-circuited the Power off signal is off (false) and the power to the robot can be switched on. The relays are located on an extra PC board, which contains additional electronics used in the data acquisition unit (A4). All the input signals are converted from +15 [V] to +5 [V] by converters located on two additional PC boards. For the layout of the PC boards see appendix B.

The connectors A4-X5, A4-X9, A4-X13 and A4-X17 (all 9 pins) are obsolete and are now used for the digital I/O. The digital I/O uses blocks of eight signals as described before. Each connector represents one block of eight signals and a ground pin. All signals that must be converted to +5 [V] are first converted and then connected to one of the connectors at the back of panel A4. To connect A4-X5, A4-X9, A4-X13 and A4-X17 to the digital I/O a special cable is made. The layout of this cable can be derived from table 2.2, nc means not connected.

The exact layout of the cable connecting A4-X5, A4-X9, A4-X13 and A4-X17 to the digital I/O at the dSpace controller board is listed in appendix A.

Signal	Source	PC board	Back A4	digital I/O nr.	digital I/O pin
EOS M1	A4-X21 p1	Converted	A4-X5 p1	I/O 00	p1
EOS M2	A4-X21 p2	Converted	A4-X5 p2	I/O 01	p18
EOS ARM	A4-X21 p3	Converted	A4-X5 p3	I/O 02	p2
EOS M3	A4-X21 p4	Converted	A4-X5 p4	I/O 03	p19
EOS M4	A4-X21 p5	Converted	A4-X5 p5	I/O 04	p3
EWG M1	A4-X21 p6	Converted	A4-X5 p6	I/O 05	p20
EWG M2	A4-X21 p7	Converted	A4-X5 p7	I/O 06	p4
EWG ARM	A4-X21 p8	Converted	A4-X5 p8	I/O 07	p21
GND	-	-	A4-X5 p9	I/O GND	I/O GND
EWG M3	A4-X21 p9	Converted	A4-X9 p1	I/O 08	p5
EWG M4	A4-X21 p10	Converted	A4-X9 p2	I/O 09	p22
ZERO M1	A4-X21 p11	Converted	A4-X9 p3	I/O 10	p6
ZERO M2	A4-X21 p12	Converted	A4-X9 p4	I/O 11	p23
ZERO M3	A4-X21 p14	Converted	A4-X9 p5	I/O 12	p7
ZERO M4	A4-X21 p15	Converted	A4-X9 p6	I/O 13	p24
Alarm 1	A4-X26 p24a	Converted	A4-X9 p7	I/O 14	p8
Alarm 2	A4-X26 p24c	Converted	A4-X9 p8	I/O 15	p25
GND	-	-	A4-X9 p9	I/O GND	I/O GND
nc	-	-	A4-X13 p1–p4	nc	nc
PA OK 1	A4-X20 p16	Converted	A4-X13 p5	I/O 20	p11
PA OK 2	A4-X20 p19	Converted	A4-X13 p6	I/O 21	p28
PA OK 3	A4-X20 p22	Converted	A4-X13 p7	I/O 22	p12
PA OK 4	A4-X20 p25	Converted	A4-X13 p8	I/O 23	p29
GND	-	-	A4-X13 p9	I/O GND	I/O GND
PA Enable 1	PC Board	Relay PA1	A4-X17 p1	I/O 24	p13
PA Enable 2	PC Board	Relay PA2	A4-X17 p2	I/O 25	p30
PA Enable 3	PC Board	Relay PA3	A4-X17 p3	I/O 26	p14
PA Enable 4	PC Board	Relay PA4	A4-X17 p4	I/O 27	p31
Power off	PC Board	Relay Power off	A4-X17 p5	I/O 28	p15
nc	-	-	A4-X17 p6–p8	nc	nc
GND	-	-	A4-X17 p9	I/O GND	I/O GND

Table 2.2: Connections at back of A4 for digital I/O

2.3 Power Amplifiers

The control signals to the power amplifiers are now directly sent from dSpace to the connectors A6-X8, A6-X9, A6-X10 and A6-X11 at the power amplifier section. These connectors at unit A6 are 9 pins male connectors. To connect the DA (digital analog converter) channels (2 for each amplifier) from dSpace to unit A6, special connectors are made.

No changes were made to the hardware inside the power amp unit.

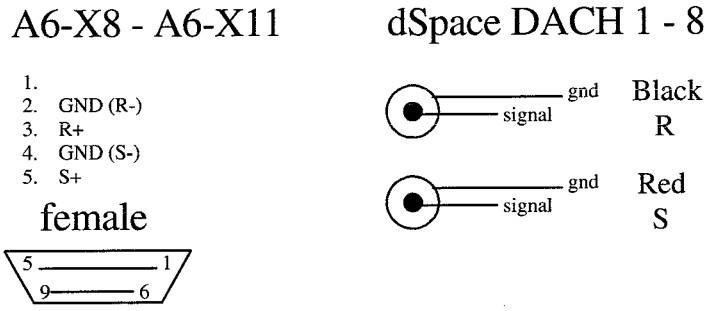

Figure 2.2: Layout of PA connector from dSpace to A6

Chapter 3

FAMM Procedures

A number of procedures are important to control the FAMM. These procedures need to be implemented in the Simulink S-Function. In this chapter is discussed how the procedures work. With the information provided in this chapter it must be possible to write the algorithms for the S-Function. A flowchart of the software, regarding the implementation of the procedures is found in chapter 4.

3.1 Commutation

The objective of this assignment was to get the FAMM operational. Considering the amount of time available for this assignment it was not possible to make a thorough study of commutation and the principles of the three phase motor. Therefore only its basic principle will be discussed.

3.1.1 Principle of a 3 phase AC Motor

The motors of the FAMM are three phase AC motors. Each phase (R, S and T) are sinusoidal currents. To move the motor phase S must have an phase shift of $\frac{2}{3}\pi$ and phase T must have an phase shift of $\frac{4}{3}\pi$ with respect to phase R. The sign of the amplitude of the currents determines the direction of movement.

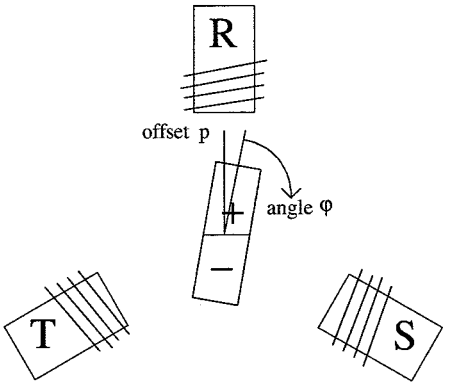

Figure 3.1: Basic layout of a 3-phase AC motor

These currents to the motors depend on the position $(\varphi + p)$ of the rotor with respect to the coils. To move the rotor of figure 3.1 clockwise the following currents must be supplied.

I_{Phase}	supplied current to coil R S or T	[A]
$\hat{I}$	amplitude of current	[A]
φ	start position of rotor	[rad]
p	offset of start position to the zero position of the rotor	[rad]

Table 3.1: Symbols and variables

$$\begin{aligned} I_{Phase,R} &= \hat{I}\sin(\varphi + p) \\ I_{Phase,S} &= \hat{I}\sin(\varphi + p + \frac{2}{3}\pi) \\ I_{Phase,T} &= \hat{I}\sin(\varphi + p + \frac{4}{3}\pi) \end{aligned} \tag{3.1}$$

The offset of the start position to the zero position of the rotor p is not know in advance. Only if the offset p is estimated well, the thrust force on the rotor will be maximal. Suppose that the currents to the coil have a user defined offset $\hat{p}$. If $(p - \hat{p}) = 0$, the thrust force on the rotor is maximal, but if $(p - \hat{p}) = \pm\frac{\pi}{2}$ the thrust force will be zero.

3.1.2 Zero Search Procedure

To estimate the initial offset p a zero search procedure is used. This procedure tries to find the $\hat{p}$ for which the thrust force is zero (equilibrium point). At this equilibrium point the rotor will not move at all, regardless of the amplitude of the current supplied. If this point ($\hat{p}$) is found, this estimate only has to be shifted by $\pm\frac{\pi}{2}$ to obtain the real p. Philips provided the assembly code of the zero search procedure. This procedure was similar to the zero search procedure for the H-drive [1].

The zero search procedure is based on small vibration pulses to move the rotor as little as possible. The total current pulse endures 10Δ and comprises of 6 sine parts of period Δ, 2Δ, Δ, Δ, 2Δ, Δ respectively and a pause of 2Δ. This current signal will keep the net displacement approximately zero. In order to keep the movement small, Δ must be kept small (typical: $\Delta = 1 \ldots 10$ ms).

During a pulse the estimated offset $\hat{p}$ is kept constant. The amount of movement and the direction of the movement provides information about how close $\hat{p}$ is to the equilibrium point. With the information about the movement the estimate $\hat{p}$ is adapted. The angle of the motors is measured by an incremental encoder. Therefore the movement must be above a certain level (= detection level) to ensure that they are reliable enough to adjust $\hat{I}$ and $\hat{p}$. What the detection level should be, is not yet determined but it will be approximately 10–100 increments.

Before the zero search can start a move test has to be performed in order to test whether there is sufficient movement to start a zero search procedure. If the movement for three pulses exceeds the detection level for each successive pulse than the move test is successful. If the movement stays below detection level the amplitude $\hat{I}$ is increased and the offset $\hat{p}$ is shifted by $\frac{\pi}{2}$ until the test succeeds. If the test does not succeed after a large number of iterations (e.g. 50) or when the maximum current is reached, the test is unsuccessful.

The following algorithm is used to find the equilibrium point.

- If the movement stays below the detection level, the amplitude $\hat{I}$ is increased, without changing $\hat{p}$. If the maximum amplitude is reached, the equilibrium point is found and the zero search procedure will stop.

- If the movement exceeds the detection level, $\hat{p}$ is shifted to try to reduce the movement during the next vibration pulse.
- if the sign of the current total movement is equal to the sign of the previous total movement, the equilibrium point, the equilibrium point is not passed during the last vibration pulse. The estimate $\hat{p}$ is shifted an angle $d\hat{p}$ in the direction opposite to the sign of the total movement during the last pulse. The step size $d\hat{p}$ is not reduced.
- If the sign of the current total movement differs from the sign of the previous total movement, the equilibrium point is passed during the last vibration pulse. Step size $d\hat{p}$ is reduced and the estimate $\hat{p}$ is shifted an angle $d\hat{p}$ with the opposite sign as the sign of the total movement.

When the equilibrium point is found a phase of $\frac{\pi}{2}$ is added to the estimate $\hat{p}$ to obtain the value of p. This value of $\hat{p}$ is used to calculate the currents to the coil according to equation 3.1.

In this section the principle of a 3-phase AC motor is discussed with just three coils. The motors in the FAMM have eight or four pole pairs. Therefore equation 3.1 must be adapted with n as the number of pole pairs, according to the theory for a linear 3-phase AC motor [1]. It is not proven that the theory can be interpolated to a rotating motor, so this needs to be checked. The zero search algorithm remains the same.

$$\begin{aligned} I_{Phase,R} &= \hat{I}\sin(\frac{\pi\varphi}{n} + p) \\ I_{Phase,S} &= \hat{I}\sin(\frac{\pi\varphi}{n} + p + \frac{2}{3}\pi) \\ I_{Phase,T} &= \hat{I}\sin(\frac{\pi\varphi}{n} + p + \frac{4}{3}\pi) \end{aligned} \tag{3.2}$$

3.2 Homing

Before the robot can operate properly it is necessary to find out what the absolute position of the robot is. Since the motors are equipped with incremental encoders it is not possible to obtain the absolute position from the encoders. Therefore the FAMM has a special procedure for finding the absolute zero position (home).

3.2.1 Homing Motor 1 and 2

First, the homing procedure of motor 1 is described. Motor 2 has to run along as slave. All trajectories are constant velocity profiles (jog mode). First motor 1 must move towards EWG 1. If this point is reached motor 1 has to stop and then move in the other direction to find sensor ZERO 1. Now motor 1 must stop again. Since the motor does not stop instantaneously it will have a small overshoot and thus pass ZERO 1. Now the jogging speed is reduced and motor 1 has to move back to ZERO 1. If ZERO 1 is reached motor 1 has to move further until its encoder index is found. This is the actual zero position of motor 1 and encoder 1 is reset to zero. Before resetting the encoder the position must be stored to apply commutation. The homing procedure of motor 2 is exactly the same as for motor 1. Only all sensors involved have index 2. Motor 1 has to run along as slave.

3.2.2 Homing Motor 3 and 4

In order to home motor 3 and 4 the wrist first must move upwards until EWG 4 is reached. At this position ZERO 3 and 4 are located. First motor 3 has to find its zero. The wrist has to rotate to find this zero. Therefore motor 4 has to run along as slave of motor 3 (in exactly the opposite direction). If ZERO 3 is found motor 3 must stop and move back to ZERO 3 with reduced jogging speed. When ZERO 3 is reached again, motor 3 has to move further until the index of encoder 3 is found. This is the zero position of motor 3. The position must be stored and the encoder can

be reset to zero. The homing procedure of motor 4 is the same as for motor 3. Motor 3 is slave of motor 4 and must run in the opposite direction of motor 4. The wrist is rotated until ZERO 4 is found. The wrist stops rotating and moves back with reduced jogging speed. When ZERO 4 is found for the second time, motor 4 must move further until the index of encoder 4 is found. Now motor 4 reached its zero. The position is stored and the encoder is reset to zero.

3.3 Safety

Several safety procedures must be implemented on the FAMM to prevent mechanical or electrical damage. Because the FAMM can operate on high speeds there are no mechanical dampers to absorb the energy when the end of stroke is reached (These dampers would be enormous to absorb al the mechanical energy [3]). This requires a special safety procedure when the FAMM is outside its normal working area (Position violation). Also the velocity at which the FAMM can move is bounded. If the velocity of the FAMM is higher than allowed, the FAMM must be slowed down (Velocity violation). The current to the motors is limited. Also the power amps inputs are not allowed to reach levels higher dan $\pm$10V. Therefore a saturation must be implemented (Current Violation).

3.3.1 Position Violation

The FAMM is constructed without shock absorbers at its end of stroke. Therefore the EWG and EOS sensors are located at a special position [3]. If the FAMM reached one of the EWG sensors, the software must generate a quick stop trajectory that can be followed by the motors (PD controller). This trajectory must be chosen in such a manner that the motors must brake at maximum power. Then the motors are able to slow down the FAMM sufficiently. When one of the EOS sensors is activated, the power to the robot is switched off (emergency stop). Due to the brake trajectory the motors have almost stopped moving and when the power is shut down (EOS sensor reached) normal friction is sufficient to completely slow down the robot without damage. When a position violation occurs at a lower speed, the EOS sensor may not be reached. Nevertheless an emergency stop (switch off power) must be generated by the software if the robot is slowed down.

3.3.2 Velocity Violation

The maximum velocity at which the FAMM can safely operate is 8 [m/s]. If the FAMM reaches higher velocities, it must be slowed down. This can be done by a simple velocity brake (D controller). This brake can slow down the robot until the robot reached a safe speed. It is not necessary to generate an emergency stop when this violation occurs.

3.3.3 Current Violation

The motors have a maximum current that can be handled. Motor 1 and 2 can operate at a maximum of 12 [A], motor 3 and 4 at a maximum of 9 [A]. The power amplifiers can handle an input signal varying between -10 [V] and +10 [V]. So the currents computed by the user controller and/or safety controllers must be checked for possible overload. If the amplitude current is above I_{max} its value must be set to I_{max} (clipping). Furthermore the amplitude of the currents must be converted from [A] to [V]. For PA 1 and 2 the maximum input is 10 [V] and the maximum output is 12 [A], for PA 3 and 4 the maximum is also 10 [V] but the maximum output is 9 [A]. The scaling factors from [A] to [V] are therefore $\frac{10}{12}$ and $\frac{10}{9}$ respectively. The dSpace controller board also has a scaling factor 10, so an input of 1 [V] to the dSpace controller board leads to an output of 10 [V]. This must be corrected by scaling the input voltage by $\frac{1}{10}$.

$\frac{1}{10}$	dSpace constant
$\frac{10}{12}$	conversion from [A] to [V] for motor 1 and 2
$\frac{10}{9}$	conversion from [A] to [V] for motor 3 and 4

Table 3.2: Factors to convert controller signals to dSpace inputs

Suppose the amplitude of the current to motor 1 must be 3.4 [A]. First it must be converted to voltage ($3.4 * \frac{10}{12} = 2.8333$ [V]) then it must be corrected for the dSpace scaling factor ($2.8333 * \frac{1}{10} =$ 0.2833 [V]). This is the actual signal sent to the dSpace controller board. The dSpace controller board will then output (0.2833*10=2.8333 [V]) to PA 1.

Chapter 4

Simulink Implementation

To use the FAMM within the Matlab-Simulink environment the connections from and to the dSpace controller board must be implemented in a Simulink model (FAMMDD.mdl). Furthermore a user S-function must be written in ANSI C-code to implement all the FAMM procedures such as zero search, homing, safety and the calculation of the motor commands to the power amps. In this chapter, first the structure of the model FAMMDD.mdl and the dSpace interface will be discussed then the s-function itself. As there was little time available for this project the procedures have not yet been implemented in the s-function and only the framework of the C-code is presented. The s-function must also enable an user controller after initialization of the FAMM. The user controller is not discussed in this report.

4.1 Structure of FAMMDD.mdl

The main Simulink block is the block that later will be used by other users. It has only the most elementary inputs and outputs. All information that a normal user does not need to know for normal use is hidden.

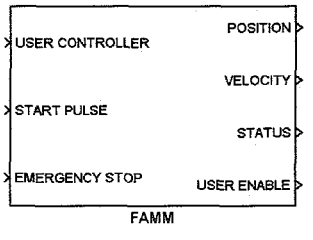

Figure 4.1: Main Simulink block

The block has three input ports:

- USER CONTROLLER (vector 4)
  Vector with currents that have to be applied to the different motors [current M1, current M2, current M3, current M4]. These currents are 1 dimensional control signals, which are converted to three phase signals by the FAMM C-code.
- START PULSE (scalar)
  This scalar switches the FAMM on (1) or off (0). Its default value at startup is 0. When set to 1 the initialization of the FAMM is started.
- EMERGENCY STOP (scalar)
  This scalar switches the emergency stop on (1) or off (0). When switched on the FAMM

stops as quickly as possible and the power is switched off when the FAMM stoppes moving. Its default value at startup is 0.

Furthermore it has four output ports:

- POSITION (vector 4)
  Vector containing the angles of the four motors [M1, M2, M3, M4].
- VELOCITY (vector 4)
  This vector contains estimates of the angular velocity of the four motors [M1, M2, M3, M4]. The estimates are generated by a first order filter. See the FAMM code (FAMMDD_Main.c, function: mdlDerivatives) for the filter implementation.
- STATUS (scalar)
  Outputs the status number of the FAMM. More about the FAMM status in the S-function section later this chapter.
- USER ENABLE (scalar)
  This scalar enables the user controller when the initialization of the FAMM is finished. The user controller therefore must be a enabled subsystem. At startup the value is 0 (disabled), after initialization the user controller is enabled (value 1).

Inside this subsystem the dSpace interface and the FAMM S-function are present. By masking the subsystem these features will not be visible to other users. All FAMM outputs are connected as inputs in the S-function and all FAMM inputs are outputs of the S-function.

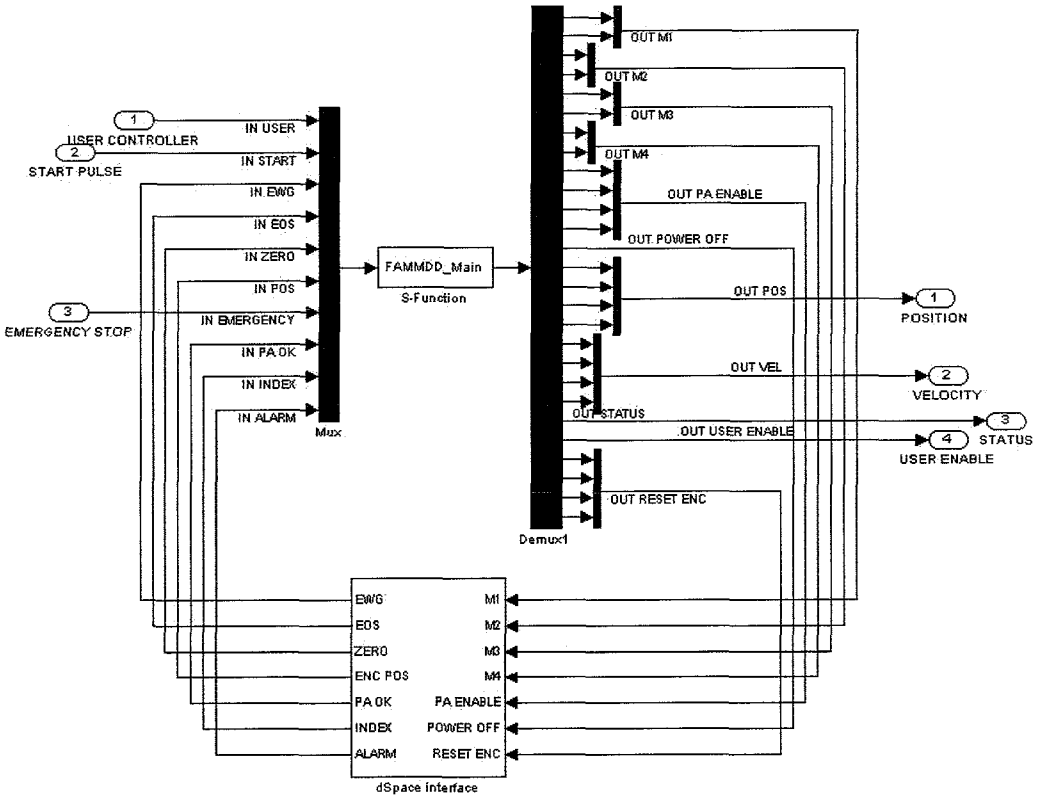

Figure 4.2: Layout of the main block

4.2 dSpace Interface

The dSpace interface contains all the connections to and from the dSpace controller board. All the output and input signals of the FAMM, which are discussed before are present in this block. The layout of the subsystem is available in appendix C.

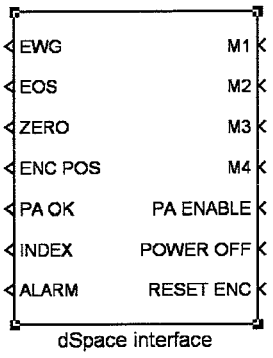

Figure 4.3: dSpace interface block

The block has seven input ports:

- M1 (vector 2)
  M1 contains the R- and S-phase of the three phase currents for motor 1 [phase R, phase S].
- M2 (vector 2)
  Contains R- and S-phase for motor 2 [phase R, phase S].
- M3 (vector 2)
  Contains R- and S-phase for motor 3 [phase R, phase S].
- M4 (vector 2)
  Contains R- and S-phase for motor 4 [phase R, phase S].
- PA ENABLE (vector 4)
  This signal enables the power amps [PA1, PA2, PA3, PA4]. If value of the element is 0 the corresponding power amp is disabled, if the value is 1 the power amp is enabled.
- POWER OFF (scalar)
  Scalar to switch the power to the robot on (0) or off (1).
- RESET ENC (vector 40)
  Resets the encoders to the specified value [enc1, enc2, enc3, enc 4].

Furthermore it also has seven output ports:

- EWG (vector 5)
  Contains the signals from the EWG sensors [EWG M1, EWG M2, EWG ARM, EWG M3, EWG M4].
- EOS (vector 5)
  Contains the signals from the EOS sensors [EOS M1, EOS M2, EOS ARM, EOS M3, EOS M4].
- ZERO (vector 4)
  Contains the signal from the ZERO sensors [ZERO M1, ZERO M2, ZERO M3, ZERO M4].

- ENC POS (vector 4)
  Contains the encoder positions [enc1, enc2, enc3, enc4]. These positions are converted to angles in the S-function.
- PA OK (vector 4)
  Contains the PA OK signals [PA1, PA2, PA3, PA4]. The value 1 means OK, 0 implies an error.
- INDEX (vector 4)
  Contains the indices of the encoders [enc1, enc2, enc3, enc4]. The value 1 indicates an index, otherwise the value is 0.
- ALARM (vector 2)
  Contains the alarm signals [alarm1, alarm2]. It is not clear what these signals represent.

4.3 S-Function

The s-function is programmed in ANSI C code. Matlab-Simulink provides a framework for user defined s-functions in C-code. It is assumed that the reader of this report is familiar with the simulink environment and the writing of s-functions, if not, see the simulink-manual for more information.

During the programming process the following rules are used:

- Global variables and constants (defines) have CAPITAL names. All local variables and constants have normal names.
- All global variables have round braces (), all local variables have rectangular braces [].

4.3.1 I/O Definitions

First a C-code library is defined to initialize all the inputs and outputs of the s-function. The s-function itself has one input vector (uPtrs) and one output vector (yPtrs). To make the programming more easy, the input- and output vectors are divided into parts that represent a group of signals (e.g EWG sensors). Also an index in each part is defined (e.g. EWG(2)=EWG ARM signal). The C-code (FAMMDD_Main.h) of this library is found in appendix D.

4.3.2 S-Function Framework

The main s-function (FAMMDD_Main.c) is structured according to the Simulink specifications. First global variables and constants are defined. Then the structure of the s-function is initialized. These functions are only called at the startup of the s-function. The last part of the s-function contains the functions called during realtime operation.

The normal initialization procedure for the FAMM is the following. All the states of the s-function are mentioned. A flowchart is included in figure 4.4

- Waiting for start
  Enable the power amps and allow power to the robot. Wait for the start pulse to start initialization
- Move test
  Test if there is enough movement for a successful zero search procedure. If this test fails, the zero search failed. This test can be performed for all four motors at once.

Function	Tasks
mdlInitializeSizes	Initialize: 1. S-function parameters 2. Number of continuous (8) and discrete (0) states 3. Length of input and output port 4. Length of workspace vectors
mdlInitializeSampleTimes	Initialize sample time
mdlInitializeConditions	Initialize: 1. Workspace global variables 2. Continuous and discrete states initial values
mdlOutputs	Calculate output signals depending of the status of the FAMM. In this function the FAMM procedures must be implemented.
mdlUpdate	Check whether the FAMM must change its status.
mdlDerivatives	Calculate the derivatives of the continuous states. State 0...3 represent velocity estimates. State 4...7 represent the integrant of the error for the I-action.

Table 4.1: Functions in the s-function, FAMMDD_Main.c

- Zero search
  Try to find the position of the magnets with respect to the coils. When all motors find its zero the homing procedure can start. Otherwise the zero search failed. This procedure can also be performed parallel for all motors, for more information see the H-drive report [1].

- Homing
  Because the homing procedure is complicated and a master slave configuration is necessary, the homing is done for one motor at a time. Maybe it is possible to home motor 3 at the same time as motor 1 and motor 4 at the same time as motor 2, but this is not tested.

- Moving
  When the home is found, the robot moves to the start position in the middle of the normal working area.

- Ready
  If the starting point is reached, the initialization is finished and control is given to the user.

- Safety
  During user operation (Ready), the software must perform safety checks to prevent overload (current violation) and mechanical damage (position and velocity violation). Also a emergency stop must be generated on the user's command.

The C-code for the FAMM is far from ready for implementation. The procedures are not added to the code. The s-function FAMMDD_Main is found in appendix D and provides a starting point for further programming. For the generation of trajectories for homing and emergency stop, a c-code reference generator is included in this report.

What also can be implemented is the possibility for the user to set the FAMM standby (set start pulse from 1 back to 0). This state will switch off the power to the robot but will remember the control variables so that the user can resume his research just by setting the start pulse back to 1 and without all the initialization procedures.

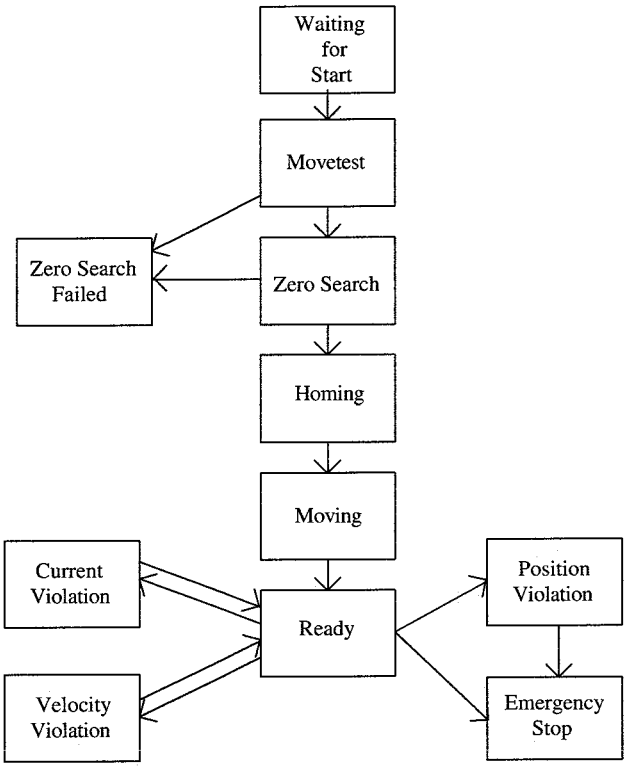

Figure 4.4: Flowchart of the S-function

4.3.3 Status Definitions

The s-function works with different states of the FAMM. Depending of the status of the FAMM a part of the code is executed (mdlOutputs). In mdlUpdate for each status is checked whether the FAMM can move on to the next status. If there is an emergency situation the status is set to emergency stop. The C-code (FAMMDD_IOPorts.h) of this library is also found in appendix D.

Chapter 5

Conclusion

The hardware of the FAMM is analyzed and the changes made to the hardware are documented. This will make it easier to make additional changes to the FAMM for further research. Also a start is made with the s-function. The code is far from ready, but the code represented in this report will be a good start for further programming.

In order to control the FAMM correctly it is necessary to calculate the motor currents (phase R, S and T). The equation 3.2 presented in chapter 3 for the generation of the current therefore must be checked. This equation is derived from the theory for a linear motor motion system [1] and it is not proven that this is valid for the motors used in the FAMM. When the equations are correct the motor commands can be calculated correctly and only then other procedures can be implemented in the C-code.

Appendix A

Connector Layout

Only the connectors that are used in chapter 2 are listed in this appendix.

A.1 Connector A4-X2

Pin	Signal
p2	VCC (+15V)
p3	Power off

Table A.1: Layout of connector A4-X2

A.2 Connector A4-X20

The pins 1–12 contain A,B,C of PA1, A,B,C of PA2, A,B,C of PA3 and A,B,C of PA4. These signals are error signals from the power amps, but are not used. Pin 13 is not connected. Pin 15, 18, 21 and 24 contain return PA1–PA4. These signals are not used either.

Pin	Signal
p1–p13	nc
p14	+15V
p15	nc
p16	PA OK 1
p17	+15V
p18	nc
p19	PA OK 2
p20	+15V
p21	nc
p22	PA OK 3
p23	+15V
p24	nc
p25	PA OK 4

Table A.2: Layout of connector A4-X20

A.3 Connector A4-X21

Pin	Signal
p1	EOS M1
p2	EOS M2
p3	EOS ARM
p4	EOS M3
p5	EOS M4
p6	EWG M1
p7	EWG M2
p8	EWG ARM
p9	EWG M3
p10	EWG M4
p11	ZERO M1
p12	ZERO M2
p13	nc
p14	ZERO M3
p15	ZERO M4
p16–p23	nc
p24	GND
p25	VCC (+15V)

Table A.3: Layout of connector A4-X21

A.4 Connector A4-X24

Pin	Signal
p1	+15V
p2	+15V
p3	+15V
p4	+15V
p5	Reset+ (A4-X25 p14a)
p6	nc
p7	nc
p8	GND
p9	enable PA1
p10	enable PA2
p11	enable PA3
p12	enable PA4
p13	nc
p14	nc
p15	reset PA (A4-X25 p14c)

Table A.4: Layout of connector A4-X24

A.5 Connector A4-X25

Only the pins that are connected are mentioned in the table below.

pin	source	signal	pin	source	signal
1a	A4-X18 p3	A	1c	A4-X18 p10	not A
2a	A4-X18 p4	B	2c	A4-X18 p11	not B
6a	A4-X21 p1	EOS M1	6c	A4-X21 p2	EOS M2
7a	A4-X21 p3	EOS ARM	7c	A4-X21 p4	EOS M3
8a	A4-X21 p5	EOS M4			
9a	-	EOS 1	9c	-	EOS 2
12a	-	Power off	12c	-	Power off ground
13a	A4-X2 p2	relais	13c	A4-X2 p3	relais
14a	A4-X24 p5	enable PA	14c	A4-X24 p15	reset
17a	-	+5V DC	17c	-	+5V DC
18a	-	0V DC	18c	-	0V DC
19a	-	+15V DC	19c	-	+15V DC
20a	-	0V DC	20c	-	0V DC
21a	-	-15V DC	21c	-	-15V DC
22a	-	+24V DC	22c	-	+24V DC
23a	-	0V DC	23b	-	0V DC
26a	-	S0 enc 1	26c	-	not S0 enc 1
27a	-	S90 enc 1	27c	-	not S90 enc 1

Table A.5: Pin layout X4-X25

A.6 Connector A4-X26

Only the pins that are connected are mentioned in the table below.

pin	source	signal	pin	source	signal
1a	A4-X4 p10	not A	1c	A4-X4 p3	A
2a	A4-X4 p4	B	2c	A4-X4 p11	not B
3a	A4-X4 p5	I	3c	A4-X4 p12	not I
4a	-	ground	4c	-	ground
5a	A4-X8 p10	not A	5c	A4-X8 p3	A
6a	A4-X8 p4	B	6c	A4-X8 p11	not B
7a	A4-X8 p5	I	7c	A4-X8 p12	not I
8a	-	ground	8c	-	ground
17a	-	+5V DC	17c	-	+5V DC
18a	-	0V DC	18c	-	0V DC
19a	-	+15V DC	19c	-	+15V DC
20a	-	0V DC	20c	-	0V DC
21a	-	-15V DC	21c	-	-15V DC
22a	-	+24V DC	22c	-	+24V DC
23a	-	0V DC	23b	-	0V DC
24a	-	alarm 1	24c	-	alarm 2
26a	-	S0 enc 1	26c	-	not S0 enc 1
27a	-	S90 enc 1	27c	-	not S90 enc 1
28a	-	I enc 1	28c	-	not I enc 1
29a	-	S0 enc 2	29c	-	not S0 enc 2
30a	-	S90 enc 2	30c	-	not S90 enc 2
31a	-	I enc 2	31c	-	not I enc 2

Table A.6: Pin layout X4-X26

A.7 Connector digital I/O

I/O Number	Signal	Block I/O	I/O pin	Source
00	EOS M1	Block 1 input	p1	A4-X5 p1
01	EOS M2	Block 1 input	p18	A4-X5 p2
02	EOS ARM	Block 1 input	p2	A4-X5 p3
03	EOS M3	Block 1 input	p19	A4-X5 p4
04	EOS M4	Block 1 input	p3	A4-X5 p5
05	EWG M1	Block 1 input	p20	A4-X5 p6
06	EWG M2	Block 1 input	p4	A4-X5 p7
07	EWG ARM	Block 1 input	p21	A4-X5 p8
08	EWG M3	Block 2 input	p5	A4-X9 p1
09	EWG M4	Block 2 input	p22	A4-X9 p2
10	ZERO M1	Block 2 input	p6	A4-X9 p3
11	ZERO M2	Block 2 input	p23	A4-X9 p4
12	ZERO M3	Block 2 input	p7	A4-X9 p5
13	ZERO M4	Block 2 input	p24	A4-X9 p6
14	ALARM 1	Block 2 input	p8	A4-X9 p7
15	ALARM 2	Block 2 input	p25	A4-X9 p8
16	nc	Block 3 input	p9	-
17	nc	Block 3 input	p26	-
18	nc	Block 3 input	p10	-
19	nc	Block 3 input	p27	-
20	PA OK 1	Block 3 input	p11	A4-X13 p5
21	PA OK 2	Block 3 input	p28	A4-X13 p6
22	PA OK 3	Block 3 input	p12	A4-X13 p7
23	PA OK 4	Block 3 input	p29	A4-X13 p8
24	PA ENABLE 1	Block 4 output	p13	A4-X17 p1
25	PA ENABLE 2	Block 4 output	p30	A4-X17 p2
26	PA ENABLE 3	Block 4 output	p14	A4-X17 p3
27	PA ENABLE 4	Block 4 output	p31	A4-X17 p4
28	POWER OFF	Block 4 output	p15	A4-x17 p5
29	nc	Block 4 output	p32	-
30	nc	Block 4 output	p16	-
31	nc	Block 4 output	p33	-
GND	GND	-	p34–p45	GND p9
/INT0	nc	-	p46	-
/INT1	nc	-	p47	-
/INT2	nc	-	p48	-
/INT3	nc	-	p49	-
VCC (+5V)	nc	-	p50	-

Table A.7: Layout connector digital I/O

Appendix B

PC Board

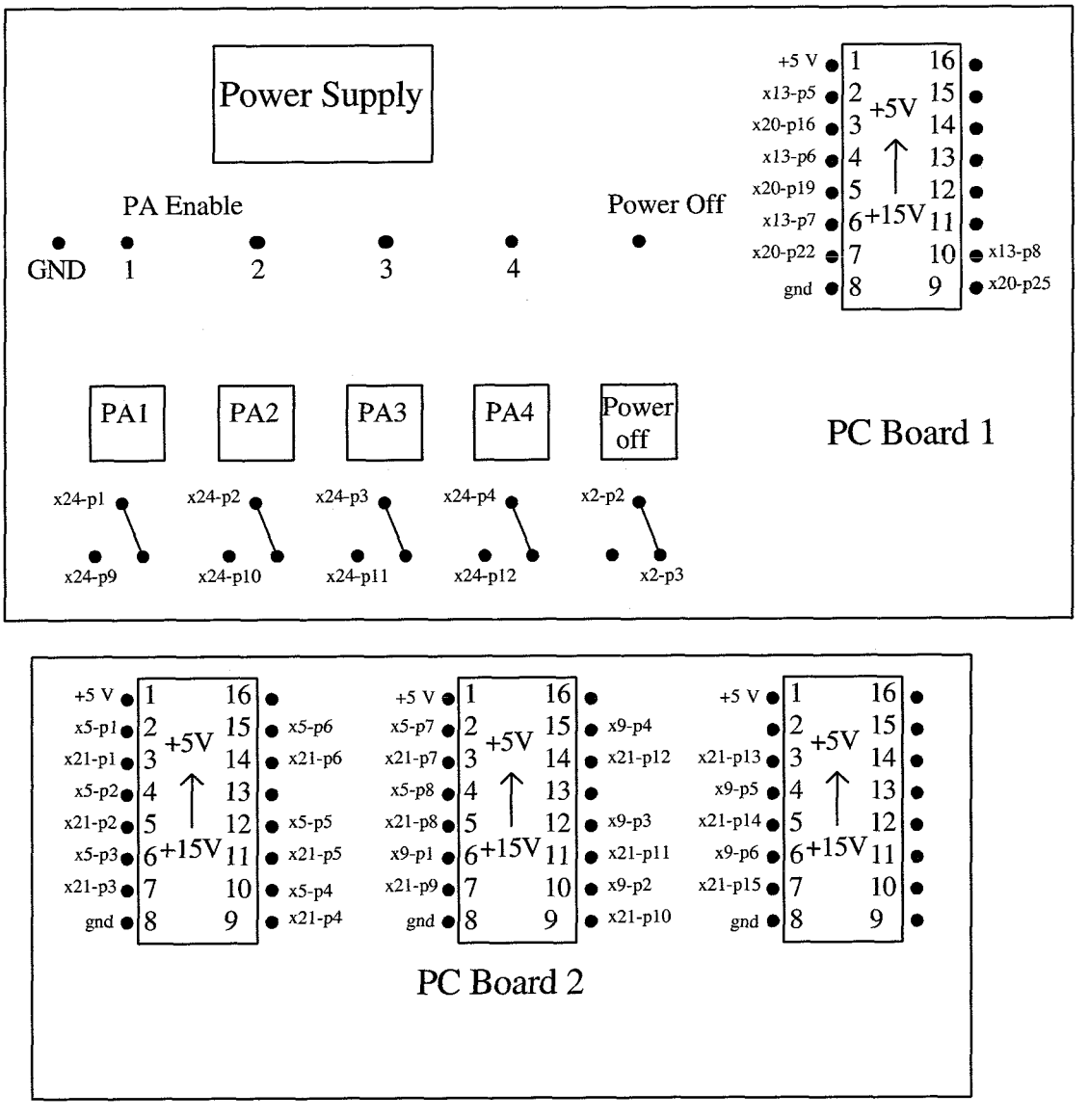

Figure B.1: Layout of the PC board

Appendix C

Simulink Layouts

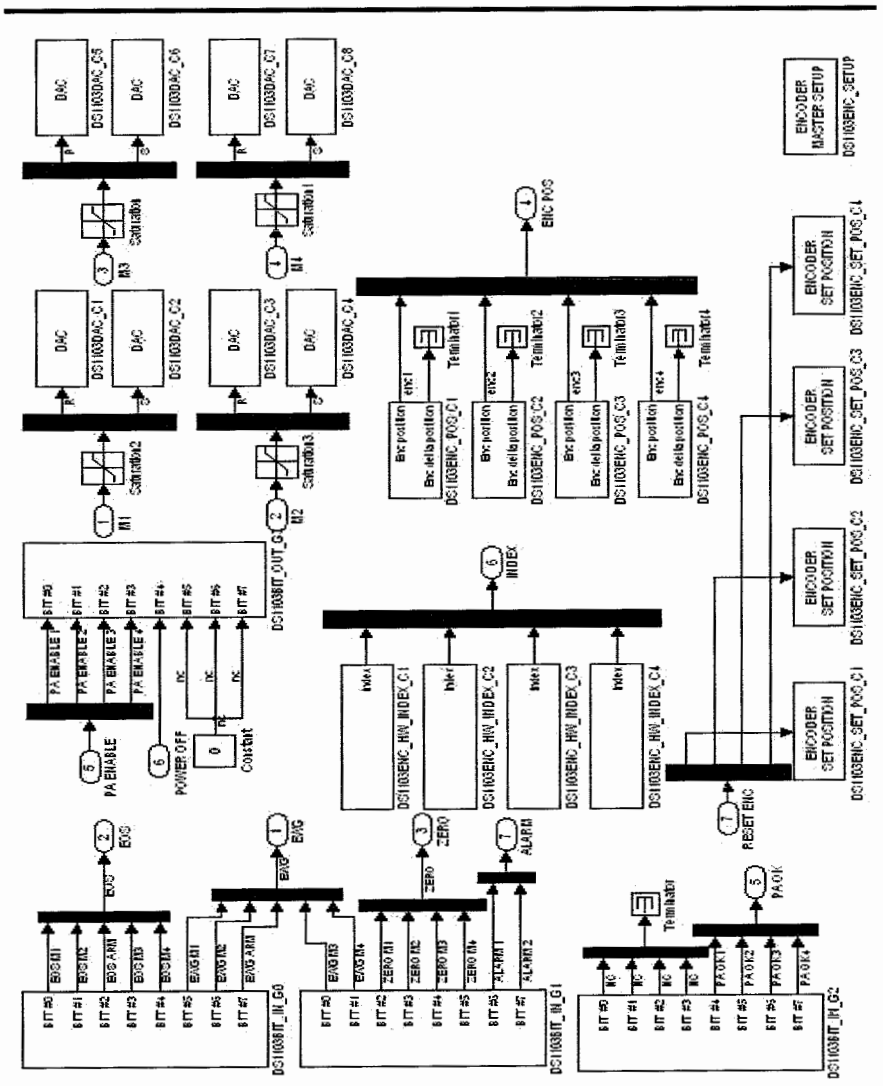

Figure C.1: Layout of the dSpace interface block.

Appendix D

C-Code

D.1 IO Definitions

```
//=========================================================================
// FAMM_IOPorts.h
//=========================================================================

#define U(element)     (*uPtrs[element])
#define Y(element)     (yPtrs[element])

// noa  = 4
// 0    =   Motor 1
// 1    =   Motor 2
// 2    =   Motor 3
// 3    =   Motor 4

// working area sensors (EWG and EOS):
// 0    =   Motor 1
// 1    =   Motor 2
// 2    =   Motor Arm
// 3    =   Motor 3
// 4    =   Motor 4

// zero's, PA and encoder indices
// 0    =   Motor 1
// 1    =   Motor 2
// 2    =   Motor 3
// 3    =   Motor 4


//=========================================================================
// inputs
//=========================================================================

#define IN_USER(element)          U(0+element)         //  U(0)....U(3)
#define IN_START                  U(4)                 //  U(4)
#define IN_EWG(element)           U(5+element)         //  U(5)....U(9)
#define IN_EOS(element)           U(10+element)        //  U(10)...U(14)
```

```
#define IN_ZERO(element)          U(15+element)       //  U(15)...U(18)
#define IN_POS(element)           U(19+element)       //  U(19)...U(22)
#define IN_EMERGENCY              U(23)               //  U(23)
#define IN_PA_OK(element)         U(24+element)       //  U(24)...U(27)
#define IN_INDEX(element)         U(28+element)       //  U(28)...U(31)
#define IN_ALARM(element)         U(32+element)       //  U(32)...U(33)


//=====================================================================
// outputs
//=====================================================================

#define OUT_M1(element)           Y(0+element)        //  Y(0)....Y(1)
#define OUT_M2(element)           Y(2+element)        //  Y(2)....Y(3)
#define OUT_M3(element)           Y(4+element)        //  Y(4)....Y(5)
#define OUT_M4(element)           Y(6+element)        //  Y(6)....Y(7)
#define OUT_PA_ENABLE(element)    Y(8+element)        //  Y(8)....Y(11)
#define OUT_POWER_OFF             Y(12)               //  Y(12)
#define OUT_POS(element)          Y(13+element)       //  Y(13)...Y(16)
#define OUT_VEL(element)          Y(17+element)       //  Y(17)...Y(20)
#define OUT_STATUS                Y(21)               //  Y(21)
#define OUT_USER_ENABLE           Y(22)               //  Y(22)
#define OUT_RESET_ENC(element)    Y(23+element)       //  Y(23)...Y(26)
```

D.2 Status Definitions

```
//=====================================================================
// FAMMDD_Main.h
//=====================================================================

// initialisation
#define WAITING_FOR_START   0
#define MOVETEST            1
#define ZERO_SEARCH         2
#define ZERO_SEARCH_FAILED  3

//homing
#define HOMING_M1           4
#define HOMING_M2           5
#define HOMING_M3           6
#define HOMING_M4           7
#define MOVING              8

//during operation
#define READY               9
#define POS_VIOLATION       10
#define VEL_VIOLATION       11
#define I_VIOLATION         12
#define EMERGENCY_STOP      13
```

D.3 S-Function

```
/*
     FAMMDD_Main.c (Matlab 6.0 version)

     (c) Jeroen de Boeij (2002)

     last update: September 20, 2002

     This program is based on the VRS software of Rene' van de Molengraft
     and on the aligning software of Antoine Verweij
     and on the Hdrive software of Loy Rovers and Stef Hendriks

     The allignmentpulses are based on a sinus
*/

//=============================================================================
// setup
//=============================================================================

#define S_FUNCTION_NAME FAMMDD_Main
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include "math.h"

#define PI                  3.14159265358979323846264338327795
#define NOM                 4       //  number of motors that are controlled
#define NINPUTS             34      //  number of inputs (see ioports.h)
#define NOUTPUTS            27      //  number of outputs (see ioports.h)
#define NSTATES             8
//      number of states (2 for each motor, for velocity estimate and I-action)
#define NRWRK               226
//      2*1 + 19*nom for FAMMDD_Main.c +37*nom for FAMMDD_Jog.c
#define NIWRK               58
//      2+19 for FAMMDD_Main.c and 37 for FAMMDD_Jog.c


//=============================================================================
// variables for basic control
//=============================================================================

#define STATUS                      prwrk[piwrk[0]]
//      global status of the FAMM
#define FAMM_POS(element)           prwrk[piwrk[1]+element]
//      position of the FAMM motors
#define FAMM_VEL(element)           prwrk[piwrk[2]+element]
//      velocity of the FAMM motors
#define US(element)                 prwrk[piwrk[3]+element]
//      control output
#define START_PULSE                 prwrk[piwrk[4]]
#define SCALE(element)              prwrk[piwrk[5]+element]
#define FACTM0                      2*PI*1.25e-6
#define FACTM1                      2*PI*1.25e-6
#define FACTM2                      2*PI*2.5e-5
```

```
#define FACTM3                      2*PI*2.5e-5
#define IMAX_MAIN_HIGH              12
//      maximum current for motors m1 and m2 during emergency [A]
#define IMAX_WRIST_HIGH             9
//      maximum current for motors m3 and m4 during emergency [A]
#define IMAX_MAIN_LOW               9
//      maximum current for motors m1 and m2 during normal use [A]
#define IMAX_WRIST_LOW              6.75
//      maximum current for motors m3 and m4 during normal use [A]
#define IMAX_ZERO                   1
//      maximum current for zero search (commutation)
#define DELAY                       1
//      time delay between zero search, homing, moving and ready [sec]

//=============================================================================
// zero search variables
//=============================================================================

#define PHI(element)                prwrk[piwrk[6]+element]
//      commutation angle te be determined
#define MOVETEST_NOTTESTED          0
#define MOVETEST_SUCCEEDED          1
#define MOVETEST_FAILED             2
#define MOVETEST_STATUS(element)    prwrk[piwrk[7]+element]
//      result of movetest for each motor. 1=succes; 2=failure;
#define IREF(element)               prwrk[piwrk[8]+element]
//      amplitude of excitation pulses
#define DISC_STEP(element)          prwrk[piwrk[9]+element]
//      stepcount for vibrationpulses
#define RESULT(element)             prwrk[piwrk[10]+element]
//      result of vibration pulse
#define TEMP0(element)              prwrk[piwrk[11]+element]
//      variable to temporarily store result of pulses
#define TEMP1(element)              prwrk[piwrk[12]+element]
#define TEMP2(element)              prwrk[piwrk[13]+element]
#define TEMP3(element)              prwrk[piwrk[14]+element]
#define TEMP4(element)              prwrk[piwrk[15]+element]
#define TEST_COUNT(element)         prwrk[piwrk[16]+element]
//      number of vibration pulses
#define AMPLITUDE_COUNT(element)    prwrk[piwrk[17]+element]
//      Counter of adjustments
#define ZERO_SEARCH_READY(element)  prwrk[piwrk[18]+element]
#define PREVIOUS_RESULT(element)    prwrk[piwrk[19]+element]
#define DPHI(element)               prwrk[piwrk[20]+element]
//      adjustment angle of commutation angle PHI
#define ZERO_SEARCH_SUCCEEDED       1

//=============================================================================
// homing variables
//=============================================================================



//=============================================================================
```

```
// control variables
//=============================================================================


//=============================================================================
// safety layer variables
//=============================================================================


//=============================================================================
// headers and includes
//=============================================================================

#include "FAMMDD_IOPorts.h"
#include "FAMMDD_Main.h"
#include "FAMMDD_Work.c"
#include "FAMMDD_Jog.c"



//=============================================================================
// mdlInitializeSizes
//=============================================================================

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S,0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {return;}

    ssSetNumContStates(S,NSTATES);
    ssSetNumDiscStates(S,0);

    if (!ssSetNumInputPorts(S,1)) {return;}
    ssSetInputPortWidth(S,0,NINPUTS);
    ssSetInputPortDirectFeedThrough(S,0,0);

    if (!ssSetNumOutputPorts(S,1)) {return;}
    ssSetOutputPortWidth(S,0,NOUTPUTS);

    ssSetNumSampleTimes(S,1);
    ssSetNumRWork(S,NRWRK);
    ssSetNumIWork(S,NIWRK);
    ssSetNumPWork(S,0);
    ssSetNumModes(S,0);
    ssSetNumNonsampledZCs(S,0);
}


//=============================================================================
// mdlInitializeSampleTimes
//=============================================================================

static void mdlInitializeSampleTimes(SimStruct *S)
{
```

```
    ssSetSampleTime(S,0,CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S,0,0.0);
}


//=============================================================================
// mdlInitializeConditions
//=============================================================================

#define MDL_INITIALIZE_CONDITIONS

static void mdlInitializeConditions(SimStruct *S)
{
    int_T   *piwrk  =   ssGetIWork(S);
    real_T  *prwrk  =   ssGetRWork(S);
    real_T  *yPtrs  =   ssGetOutputPortRealSignal(S,0);
    real_T  *x      =   ssGetContStates(S);
    int     i;

    rwrk_init_all(S);
    STATUS=WAITING_FOR_START;
    // SCALE is not used at the moment
    SCALE(0)=FACTM0;
    SCALE(1)=FACTM1;
    SCALE(2)=FACTM2;
    SCALE(3)=FACTM3;

    for(i=0; i<NOM; i++) {
        US(i)=0.0;              // zeroise control output
        x[i]=0.0;               // zeroise continuous states
        x[4+i]=0.0;
        OUT_PA_ENABLE(i)=0;     // disable PA
        PHI(i)=0;               // zeroise commutation angle
        MOVETEST_STATUS(i)=0;   // set movetest result to zero
        TEMP0(i)=0;             // zeroise temporary variables
        TEMP1(i)=0;
        TEMP2(i)=0;
        TEMP3(i)=0;
        TEMP4(i)=0;
        IREF(i)=0.25;           // startvalue of IREF
        ZERO_SEARCH_READY(i)=0; // zeroise search result
        DPHI(i)=PI/2;
    }
    OUT_POWER_OFF=1;
}


//=============================================================================
// mdlOutputs
//=============================================================================

static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T              *x      =   ssGetContStates(S);
```

```
real_T              *yPtrs  =   ssGetOutputPortRealSignal(S,0);
InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);
int_T               *piwrk  =   ssGetIWork(S);
real_T              *prwrk  =   ssGetRWork(S);
int_T               i,k,istat;
double              time;

//get time
time=ssGetT(S);

//get current position
FAMM_POS(0)=FACTM0*IN_POS(0);
FAMM_POS(1)=FACTM1*IN_POS(1);
FAMM_POS(2)=FACTM2*IN_POS(2);
FAMM_POS(3)=FACTM3*IN_POS(3);

istat=0;
OUT_STATUS=STATUS;

istat=STATUS;

switch(istat) {

    case WAITING_FOR_START:

        for (i=0; i<NOM; i++) {
            US(i)=0.0;
            OUT_POS(i)=0.0;
            OUT_VEL(i)=0.0;

            //  zeroise output to motors (PA)
            for(k=0; k<2; k++) {
                OUT_M1(k)=0.0;
                OUT_M2(k)=0.0;
                OUT_M3(k)=0.0;
                OUT_M4(k)=0.0;
            }
            OUT_PA_ENABLE(i)=1;
            //  enable PA
            OUT_RESET_ENC(i)=0;
        }
        OUT_POWER_OFF=0;
        //  allow power switched on
        OUT_USER_ENABLE=0;
        //  disable user controller
        STATUS=WAITING_FOR_START;
        OUT_STATUS=STATUS;
    break;

    case MOVETEST:
        // perform movetest and send signal to motors
        STATUS=MOVETEST;
        OUT_STATUS=STATUS;
        for (i=0; i<NOM; i++) {
```

```
                OUT_POS(i)=FAMM_POS(i);
            }
        break;

        case ZERO_SEARCH:
            // perform zerosearch and send signal to motors
            STATUS=ZERO_SEARCH;
            OUT_STATUS=STATUS;
        break;
    }
}

//=============================================================================
// mdlUpdate
//=============================================================================

#define MDL_UPDATE

static void mdlUpdate(SimStruct *S, int_T tid)
{
    real_T              *x      =   ssGetContStates(S);
    InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);
    int_T               *piwrk  =   ssGetIWork(S);
    real_T              *prwrk  =   ssGetRWork(S);
    int_T               istat,i,count;

    istat=STATUS;

    switch (istat) {

        case WAITING_FOR_START:
            if (IN_START==1) {
                STATUS=MOVETEST;
            }
        break;

        case MOVETEST:
            count=0;
            // if three MOVETESTs have been succesful, update status
            for (i=0; i<NOM; i++) {
                if (TEST_COUNT(i)==3) {
                    MOVETEST_STATUS(i)=MOVETEST_SUCCEEDED;
                    count++;
                }
            }

            // if all motors heeft succeeded testing update status
            if (count==NOM) {
                STATUS=ZERO_SEARCH;
            }

            // if one of the motors did not succeed -> search failed
            for (i=0; i<NOM; i++) {
                if (MOVETEST_STATUS(i)==MOVETEST_FAILED) {
```

```
                STATUS=ZERO_SEARCH_FAILED;
            }
        }

        // if one of the EWG sensors is activated -> emergency stop
        for (i=0; i<NOM; i++) {
            if (IN_EWG(i)==0 || IN_EOS(0)==0 || IN_EOS(1)==0) {
                STATUS=EMERGENCY_STOP;
            }
        }
    break;

    case ZERO_SEARCH:

        // if one of the motors did not find zero
        for (i=0; i<NOM; i++) {
            if (AMPLITUDE_COUNT(i)>50) {
                STATUS=ZERO_SEARCH_FAILED;
            }
        }

        // check which motors did find it's zero
        count=0;
        for (i=0; i<NOM; i++) {
            if (ZERO_SEARCH_READY(i)==1) {
                count++;
            }
        }

        // if all motors did find zero
        if (count==NOM) {
            for (i=0; i<NOM; i++) {
                PHI(i)=PHI(i)+PI/2;
            }
            STATUS=HOMING_M1;
        }

        // if one of the EWG sensors is activated -> emergency stop
        for (i=0; i<NOM; i++) {
            if (IN_EWG(i)==0 || IN_EOS(0)==0 || IN_EOS(1)==0) {
                STATUS=EMERGENCY_STOP;
            }
        }
    break;

    case HOMING_M1:
        STATUS=HOMING_M1;
    break;
  }
}


//=============================================================================
// mdlDerivatives
```

```
//=============================================================================

#define MDL_DERIVATIVES

static void mdlDerivatives(SimStruct *S)
{
    real_T              *yPtrs  =   ssGetOutputPortRealSignal(S,0);
    InputRealPtrsType   uPtrs   =   ssGetInputPortRealSignalPtrs(S,0);
    real_T              *prwrk  =   ssGetRWork(S);
    int_T               *piwrk  =   ssGetIWork(S);
    real_T              *x      =   ssGetContStates(S);
    real_T              *dx     =   ssGetdX(S);
    int_T               i;

    //velocity estimates
    for (i=0; i<NOM; i++) {
        FAMM_VEL(i)=100*(FAMM_POS(i)-x[i]);
        dx[i]=FAMM_VEL(i);
        dx[i+4]=0.0;
        OUT_VEL(i)=FAMM_VEL(i);
    }
}


//=============================================================================
// mdlTerminate()
//=============================================================================

static void mdlTerminate(SimStruct *S)
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

D.4 Workspace Initialization

```
//=============================================================================
// FAMMDD_Work.c
//=============================================================================

// init globals in work-space for re-entrancy

#define JOG_IDX 21


//=============================================================================
// rwrk_init_var
//=============================================================================
```

```
int rwrk_init_var(int *pivar, int *pidx, int nrw, SimStruct *S)
{
     int_T     *piwrk = ssGetIWork(S);

     piwrk[pivar[0]] = pidx[0];
     pivar[0]++;
     pidx[0] = pidx[0]+nrw;

     return 1;
}


//=============================================================================
// rwrk_init_all
//=============================================================================

int rwrk_init_all(SimStruct *S)
{
     int  ivar,idx;

     ivar = 0;
     idx = 0;

    //==============================
    //  Variables in FAMMDD_Main.c
    //==============================


     // basic control (6, 4*4 + 2*1)
     rwrk_init_var(&ivar,&idx,1,S);     // STATUS
     rwrk_init_var(&ivar,&idx,4,S);     // FAMM_POS
     rwrk_init_var(&ivar,&idx,4,S);     // FAMM_VEL
     rwrk_init_var(&ivar,&idx,4,S);     // US
     rwrk_init_var(&ivar,&idx,1,S);     // START_PULSE
     rwrk_init_var(&ivar,&idx,4,S);     // SCALE

     // zero search (15, 15*4)
     rwrk_init_var(&ivar,&idx,4,S);     // PHI
     rwrk_init_var(&ivar,&idx,4,S);     // MOVETEST_STATUS
     rwrk_init_var(&ivar,&idx,4,S);     // IREF
     rwrk_init_var(&ivar,&idx,4,S);     // DISC_STEP
     rwrk_init_var(&ivar,&idx,4,S);     // RESULT
     rwrk_init_var(&ivar,&idx,4,S);     // TEMP0
     rwrk_init_var(&ivar,&idx,4,S);     // TEMP1
     rwrk_init_var(&ivar,&idx,4,S);     // TEMP2
     rwrk_init_var(&ivar,&idx,4,S);     // TEMP3
     rwrk_init_var(&ivar,&idx,4,S);     // TEMP4
     rwrk_init_var(&ivar,&idx,4,S);     // TEST_COUNT
     rwrk_init_var(&ivar,&idx,4,S);     // AMPLITUDE_COUNT
     rwrk_init_var(&ivar,&idx,4,S);     // ZERO_SEARCH_READY
     rwrk_init_var(&ivar,&idx,4,S);     // PREVIOUS_RESULT
     rwrk_init_var(&ivar,&idx,4,S);     // DPHI
```

```
    //==================================
    //  Variables in FAMMDD_Jog.c
    //==================================

     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);
     rwrk_init_var(&ivar,&idx,4,S);


     return 1;
}
```

D.5 Reference Generator

```
// jogging and point-to-point motion based on third-degree setpoint profile
// Rene' van de Molengraft, April, 14th, 2000
// saves settings for four different axes
// time t is the absolute time
```

```
// ---------------------------------------------------------------------------
// Defines and includes
// ---------------------------------------------------------------------------



// shortcuts for jogging variables in workspace
// #define JOG_IDX                  See FAMMDD_Work.c for starting index
#define t0(element)                 prwrk[piwrk[JOG_IDX+0]+element]
#define t1(element)                 prwrk[piwrk[JOG_IDX+1]+element]
#define t2(element)                 prwrk[piwrk[JOG_IDX+2]+element]
#define t3(element)                 prwrk[piwrk[JOG_IDX+3]+element]
#define t4(element)                 prwrk[piwrk[JOG_IDX+4]+element]
#define t5(element)                 prwrk[piwrk[JOG_IDX+5]+element]
#define t6(element)                 prwrk[piwrk[JOG_IDX+6]+element]
#define t7(element)                 prwrk[piwrk[JOG_IDX+7]+element]
#define s0(element)                 prwrk[piwrk[JOG_IDX+8]+element]
#define s1(element)                 prwrk[piwrk[JOG_IDX+9]+element]
#define s2(element)                 prwrk[piwrk[JOG_IDX+10]+element]
#define s3(element)                 prwrk[piwrk[JOG_IDX+11]+element]
#define s4(element)                 prwrk[piwrk[JOG_IDX+12]+element]
#define s5(element)                 prwrk[piwrk[JOG_IDX+13]+element]
#define s6(element)                 prwrk[piwrk[JOG_IDX+14]+element]
#define s7(element)                 prwrk[piwrk[JOG_IDX+15]+element]
#define v0(element)                 prwrk[piwrk[JOG_IDX+16]+element]
#define v1(element)                 prwrk[piwrk[JOG_IDX+17]+element]
#define v2(element)                 prwrk[piwrk[JOG_IDX+18]+element]
#define v3(element)                 prwrk[piwrk[JOG_IDX+19]+element]
#define v4(element)                 prwrk[piwrk[JOG_IDX+20]+element]
#define v5(element)                 prwrk[piwrk[JOG_IDX+21]+element]
#define v6(element)                 prwrk[piwrk[JOG_IDX+22]+element]
#define v7(element)                 prwrk[piwrk[JOG_IDX+23]+element]
#define a0(element)                 prwrk[piwrk[JOG_IDX+24]+element]
#define a1(element)                 prwrk[piwrk[JOG_IDX+25]+element]
#define a2(element)                 prwrk[piwrk[JOG_IDX+26]+element]
#define a3(element)                 prwrk[piwrk[JOG_IDX+27]+element]
#define a4(element)                 prwrk[piwrk[JOG_IDX+28]+element]
#define a5(element)                 prwrk[piwrk[JOG_IDX+29]+element]
#define a6(element)                 prwrk[piwrk[JOG_IDX+30]+element]
#define a7(element)                 prwrk[piwrk[JOG_IDX+31]+element]
#define delta(element)              prwrk[piwrk[JOG_IDX+32]+element]
#define gamma(element)              prwrk[piwrk[JOG_IDX+33]+element]
#define jerk(element)               prwrk[piwrk[JOG_IDX+34]+element]
#define idir(element)               prwrk[piwrk[JOG_IDX+35]+element]
#define xstrt(element)              prwrk[piwrk[JOG_IDX+36]+element]
```

```
// ------------------------------------------------------------------------
// jog_ini()
// ------------------------------------------------------------------------



void jog_ini(real_T xstart, real_T tstart, real_T vdes, real_T tdes,
  real_T maxjerk, int_T i, SimStruct *S)
{
     /*
     input arguments
          xstart    : start position
          tstart    : start time
          vdes      : desired jogging speed
          tdes      : time to reach vdes
          maxjerk   : jerk in acceleration phase
          i         : axis id
     */

     real_T    det;
     int_T     *piwrk=ssGetIWork(S);
     real_T    *prwrk=ssGetRWork(S);

     // vdes en jerk positive, idir contains the direction of the movement

     if (vdes>=0.0) {
       idir(i)=1.0;
     } else {
       idir(i)=-1.0;
       vdes=-vdes;
     }

     if (maxjerk>=0.0) {
       jerk(i)=maxjerk;
     } else {
       jerk(i)=-maxjerk;
     }

//   compute jerk period delta
     det=tdes*tdes*jerk(i)*jerk(i)-4.0*jerk(i)*vdes;
     if (det<0) {
       ssSetErrorStatus(S,
         "JOG_INI: vdes cannot be reached (increase jerk and/or tdes).");
     } else {
       delta(i)=(tdes*jerk(i)-sqrt(det))/(2.0*jerk(i));

//     compute acceleration period gamma

       gamma(i)=tdes-2.0*delta(i);

//     compute switching times

       t0(i)=tstart;
```

```
        t1(i)=t0(i)+delta(i);
        t2(i)=t1(i)+gamma(i);
        t3(i)=t2(i)+delta(i);

//      t4, t5, t6 and t7 equal infinity at startup

        t4(i)=100000.0;
        t5(i)=100000.0;
        t6(i)=100000.0;
        t7(i)=100000.0;

//      compute reference values at switching times

        xstrt(i)=xstart;
        s0(i)=0.0;

        a1(i)=jerk(i)*(t1(i)-t0(i));
        v1(i)=0.5*jerk(i)*(t1(i)-t0(i))*(t1(i)-t0(i));
        s1(i)=s0(i)+jerk(i)*(t1(i)-t0(i))*(t1(i)-t0(i))*(t1(i)-t0(i))/6.0;

        a2(i)=a1(i);
        v2(i)=v1(i)+a1(i)*(t2(i)-t1(i));
        s2(i)=s1(i)+v1(i)*(t2(i)-t1(i))+0.5*a1(i)*(t2(i)-t1(i))*(t2(i)-t1(i));

        a3(i)=a2(i)-jerk(i)*(t3(i)-t2(i));
        v3(i)=v2(i)+a2(i)*(t3(i)-t2(i))-0.5*jerk(i)*(t3(i)-t2(i))*(t3(i)-t2(i));
        s3(i)=s2(i)+v2(i)*(t3(i)-t2(i))+0.5*a2(i)*(t3(i)-t2(i))*(t3(i)-t2(i))
             -jerk(i)*(t3(i)-t2(i))*(t3(i)-t2(i))*(t3(i)-t2(i))/6.0;
    }
}




// ------------------------------------------------------------------------
// jog_get()
// ------------------------------------------------------------------------




void jog_get(real_T *qref, real_T *vref, real_T *aref, real_T t, int_T i,
  SimStruct *S)
{

//   i : axis id

    int_T     *piwrk = ssGetIWork(S);
    real_T    *prwrk = ssGetRWork(S);

    //################## moet dit niet overal index i zijn ipv [0]????
    //-> nee want de aanroepende funcie zorgt ervoor dat i van de originele
    //   vector op positie 0 komt te staan!
    if (t<=t0(i)) {
```

```
      aref[0]=0;
      vref[0]=0;
      qref[0]=s0(i);
    } else if (t<=t1(i)) {
      aref[0]=jerk(i)*(t-t0(i));
      vref[0]=0.5*jerk(i)*(t-t0(i))*(t-t0(i));
      qref[0]=s0(i)+jerk(i)*(t-t0(i))*(t-t0(i))*(t-t0(i))/6.0;
    } else if (t<=t2(i)) {
      aref[0]=a1(i);
      vref[0]=v1(i)+a1(i)*(t-t1(i));
      qref[0]=s1(i)+v1(i)*(t-t1(i))+0.5*a1(i)*(t-t1(i))*(t-t1(i));
    } else if (t<=t3(i)) {
      aref[0]=a2(i)-jerk(i)*(t-t2(i));
      vref[0]=v2(i)+a2(i)*(t-t2(i))-0.5*jerk(i)*(t-t2(i))*(t-t2(i));
      qref[0]=s2(i)+v2(i)*(t-t2(i))+0.5*a2(i)*(t-t2(i))*(t-t2(i))
             -jerk(i)*(t-t2(i))*(t-t2(i))*(t-t2(i))/6.0;
    } else if (t<=t4(i)) {
      aref[0]=0;
      vref[0]=v3(i);
      qref[0]=s3(i)+v3(i)*(t-t3(i));
    } else if (t<=t5(i)) {
      aref[0]=-jerk(i)*(t-t4(i));
      vref[0]=v4(i)-0.5*jerk(i)*(t-t4(i))*(t-t4(i));
      qref[0]=s4(i)+v4(i)*(t-t4(i))-jerk(i)*(t-t4(i))*(t-t4(i))*(t-t4(i))/6.0;
    } else if (t<=t6(i)) {
      aref[0]=a5(i);
      vref[0]=v5(i)+a5(i)*(t-t5(i));
      qref[0]=s5(i)+v5(i)*(t-t5(i))+0.5*a5(i)*(t-t5(i))*(t-t5(i));
    } else if (t<=t7(i)) {
      aref[0]=a6(i)+jerk(i)*(t-t6(i));
      vref[0]=v6(i)+a6(i)*(t-t6(i))+0.5*jerk(i)*(t-t6(i))*(t-t6(i));
      qref[0]=s6(i)+v6(i)*(t-t6(i))+0.5*a6(i)*(t-t6(i))*(t-t6(i))
             +jerk(i)*(t-t6(i))*(t-t6(i))*(t-t6(i))/6.0;
    } else {
      aref[0]=0.0;
      vref[0]=0.0;
      qref[0]=s7(i);
    }

    if (idir(i)==-1.0) {
      aref[0]=-aref[0];
      vref[0]=-vref[0];
      qref[0]=-qref[0];
    }

    qref[0]=qref[0]+xstrt(i);
}



// ------------------------------------------------------------------------
// jog_stop()
// ------------------------------------------------------------------------
```

```
void jog_stop(real_T t, int_T i, SimStruct *S)
{
//   i : axis id

     int_T     *piwrk = ssGetIWork(S);
     real_T    *prwrk = ssGetRWork(S);

     t4(i)=t;
     t5(i)=t4(i)+delta(i);
     t6(i)=t5(i)+gamma(i);
     t7(i)=t6(i)+delta(i);

     a4(i)=0.0;
     v4(i)=v3(i);
     s4(i)=s3(i)+v3(i)*(t4(i)-t3(i));

     a5(i)=-jerk(i)*(t5(i)-t4(i));
     v5(i)=v4(i)-0.5*jerk(i)*(t5(i)-t4(i))*(t5(i)-t4(i));
     s5(i)=s4(i)+v4(i)*(t5(i)-t4(i))-jerk(i)*(t5(i)-t4(i))*(t5(i)-t4(i))*(t5(i)
          -t4(i))/6.0;

     a6(i)=a5(i);
     v6(i)=v5(i)+a5(i)*(t6(i)-t5(i));
     s6(i)=s5(i)+v5(i)*(t6(i)-t5(i))+0.5*a5(i)*(t6(i)-t5(i))*(t6(i)-t5(i));

     a7(i)=a6(i)+jerk(i)*(t7(i)-t6(i));
     v7(i)=v6(i)+a6(i)*(t7(i)-t6(i))+0.5*jerk(i)*(t7(i)-t6(i))*(t7(i)-t6(i));
     s7(i)=s6(i)+v6(i)*(t7(i)-t6(i))+0.5*a6(i)*(t7(i)-t6(i))*(t7(i)-t6(i))
          +jerk(i)*(t7(i)-t6(i))*(t7(i)-t6(i))*(t7(i)-t6(i))/6.0;
}



// ------------------------------------------------------------------------
// jog_status()
// ------------------------------------------------------------------------



int_T jog_status(real_T t, int_T i,real_T tset, SimStruct *S)
{
//   i : axis id

     int_T *piwrk=ssGetIWork(S);
     real_T *prwrk=ssGetRWork(S);

     if (t>t7(i)+tset) {
       return 1;
```

```
    } else {
      return 0;
    }
}



// ------------------------------------------------------------------------
// p2p_ini()
// ------------------------------------------------------------------------



void p2p_ini(real_T xstart, real_T tstart, real_T xend, real_T vdes,
  real_T tdes, real_T maxjerk, int_T i, SimStruct *S)
{
    /*
    input arguments
        xstart    : start position
        tstart    : start time
        xend      : end position
        vdes      : desired jogging speed
        tdes      : time to reach vdes
        maxjerk   : jerk in acceleration phase
        i         : axis id
    */

    real_T    det,disp;

    int_T     *piwrk = ssGetIWork(S);
    real_T    *prwrk = ssGetRWork(S);

    disp=xend-xstart;
    if (disp<0.0) {
      disp=-disp;
    }

    if (vdes>=0.0) {
      idir(i)=1.0;
    } else {
      idir(i)=-1.0;
      vdes=-vdes;
    }

    if (maxjerk>=0.0) {
      jerk(i)=maxjerk;
    } else {
      jerk(i)=-maxjerk;
    }

//   compute jerk period delta
```

```
      det=tdes*tdes*jerk(i)*jerk(i)-4.0*jerk(i)*vdes;
      if (det<0) {
        ssSetErrorStatus(S,
          "P2P_INI: vdes cannot be reached (increase jerk and/or tdes).");
      } else {
        delta(i)=(tdes*jerk(i)-sqrt(det))/(2.0*jerk(i));

//      compute acceleration period gamma

        gamma(i)=tdes-2.0*delta(i);

//      compute switching times

        t0(i)=tstart;
        t1(i)=t0(i)+delta(i);
        t2(i)=t1(i)+gamma(i);
        t3(i)=t2(i)+delta(i);

//      compute t4

        t4(i)=t3(i)+(disp-2.0*jerk(i)*delta(i)*delta(i)*delta(i)
             -3.0*jerk(i)*delta(i)*delta(i)*gamma(i)
             -jerk(i)*delta(i)*gamma(i)*gamma(i))/vdes;
        if (t4(i)<t3(i)) {
          ssSetErrorStatus(S,
            "P2P_INI: vdes too high for displacement (decrease vdes).");
        } else {
          t5(i)=t4(i)+delta(i);
          t6(i)=t5(i)+gamma(i);
          t7(i)=t6(i)+delta(i);

//        compute reference values at switching times

          xstrt(i)=xstart;
          s0(i)=0.0;

          a1(i)=jerk(i)*(t1(i)-t0(i));
          v1(i)=0.5*jerk(i)*(t1(i)-t0(i))*(t1(i)-t0(i));
          s1(i)=s0(i)+jerk(i)*(t1(i)-t0(i))*(t1(i)-t0(i))*(t1(i)-t0(i))/6.0;

          a2(i)=a1(i);
          v2(i)=v1(i)+a1(i)*(t2(i)-t1(i));
          s2(i)=s1(i)+v1(i)*(t2(i)-t1(i))+0.5*a1(i)*(t2(i)-t1(i))*(t2(i)-t1(i));

          a3(i)=a2(i)-jerk(i)*(t3(i)-t2(i));
          v3(i)=v2(i)+a2(i)*(t3(i)-t2(i))-0.5*jerk(i)*(t3(i)-t2(i))*(t3(i)-t2(i));
          s3(i)=s2(i)+v2(i)*(t3(i)-t2(i))+0.5*a2(i)*(t3(i)-t2(i))*(t3(i)-t2(i))
               -jerk(i)*(t3(i)-t2(i))*(t3(i)-t2(i))*(t3(i)-t2(i))/6.0;

          a4(i)=0.0;
          v4(i)=v3(i);
          s4(i)=s3(i)+v3(i)*(t4(i)-t3(i));

          a5(i)=-jerk(i)*(t5(i)-t4(i));
```

```
        v5(i)=v4(i)-0.5*jerk(i)*(t5(i)-t4(i))*(t5(i)-t4(i));
        s5(i)=s4(i)+v4(i)*(t5(i)-t4(i))
             -jerk(i)*(t5(i)-t4(i))*(t5(i)-t4(i))*(t5(i)-t4(i))/6.0;

        a6(i)=a5(i);
        v6(i)=v5(i)+a5(i)*(t6(i)-t5(i));
        s6(i)=s5(i)+v5(i)*(t6(i)-t5(i))+0.5*a5(i)*(t6(i)-t5(i))*(t6(i)-t5(i));

        a7(i)=a6(i)+jerk(i)*(t7(i)-t6(i));
        v7(i)=v6(i)+a6(i)*(t7(i)-t6(i))+0.5*jerk(i)*(t7(i)-t6(i))*(t7(i)-t6(i));
        s7(i)=s6(i)+v6(i)*(t7(i)-t6(i))+0.5*a6(i)*(t7(i)-t6(i))*(t7(i)-t6(i))
             +jerk(i)*(t7(i)-t6(i))*(t7(i)-t6(i))*(t7(i)-t6(i))/6.0;
      }
    }
}



// ------------------------------------------------------------------------
// p2p_get()
// ------------------------------------------------------------------------



void p2p_get(real_T *qref, real_T *vref, real_T *aref, real_T t, int_T i,
  SimStruct *S)
{
    jog_get(qref,vref,aref,t,i,S);
}



// ------------------------------------------------------------------------
// p2p_status()
// ------------------------------------------------------------------------



int_T p2p_status(real_T t, int_T i,real_T tset, SimStruct *S)
{
    return jog_status(t,i,tset,S);
}
```

Bibliography

[1] Controlling the H-drive, A.F. Rovers, DCT Report 2002.12,
Eindhoven University of Technology, February 2002

[2] FAMM hardware documentation, edited September 2002 by J. Vogels, Philips CFT

[3] FAMM: Fast and Accurate Manipulator Modules, Soemers, H.M.J., CTR 545.91.0122, 1990, Philips CFT

[4] Performance improvement of the FAMMDD: The computed torque feedforward, Pasqualini, A., CTB 595-97-5269, 1997, Philips CFT