Notes on Permanent Magnet Synchronous Machine (PMSM) Model Predictive Control (MPC) and Operation Under Open Circuit Fault

Hakan Saraç

Contents

1 Introduction

In this handnote, I will be explaining my findings on PMSM control using MPC. At first, I will introduce the formula sets and PMSM state space model and how to implement MPC. Then, for a 3 phase PMSM I will be explaining my implementation and test results. After that, the implementation of MPC for a two-three-phase group machine will be discussed. Lastly, the two-three-phase group machine will be investigated under open circuit faults.

In the conclusions part, I have mentioned the problems I need to overcome and in the appendix the matlab function codes are provided.

2 MPC for PMSM

In the implementation of MPC, there are parameters we need to have in order to be able to make an prediction. For a motor drive application, mainly, we need to predict the torque output of the machine.

To achieve the torque prediction, in an induction machine, the rotor flux parameters needs to be estimated. In a PMSM, since the rotor flux density is a constant known parameters, the estimation is not necessary. A control block diagram is provided in [1].

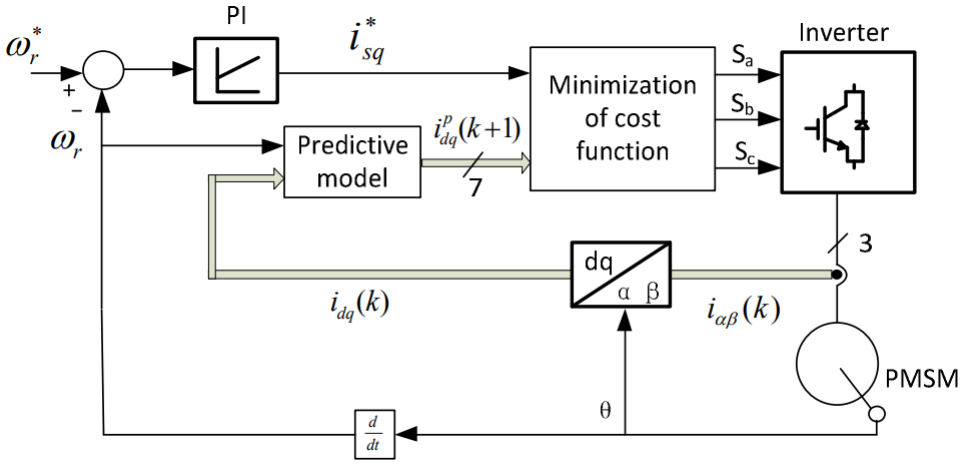

Figure 1: MPC PMSM control block diagram [1]

$$\begin{bmatrix} i_d(k+1) \\ i_q(k+1) \\ \omega(k+1) \end{bmatrix} = \begin{bmatrix} 1-\frac{R}{L_d}T_s & \frac{L_q\,\omega(k)}{L_d}T_s & 0 \\ 0 & 1-\frac{R}{L_q}T_s & -\frac{(L_d i_d(k)+\lambda_{PM})}{L_q}T_s \\ 0 & \frac{1.5p^2\lambda_{PM}}{Jeq}T_s & 1-\frac{B}{Jeq}T_s \end{bmatrix} \begin{bmatrix} i_d(k) \\ i_q(k) \\ \omega(k) \end{bmatrix} + \begin{bmatrix} \frac{1}{L_d}T_s & 0 & 0 \\ 0 & \frac{1}{L_q}T_s & 0 \\ 0 & 0 & -\frac{p}{Jeq}T_s \end{bmatrix} \begin{bmatrix} u_d(k) \\ u_q(k) \\ T_L(k) \end{bmatrix}$$

Figure 2: State Space Model [1]

3 One Module Implementation

3.1 Model

The matlab model can be seen in 3. The matlab code inside the large function diagram is provided in Listing 1. Some of the properties of the simulation and machine parameters can be listed as follows:

- Sampling Time = 1e-6
- Algorithm Frequency = 40kHz

- Rs = 351mOhm
- Ls = 3.5mHenry
- Back Emf At 600RPM = 80.3 Volt
- Flux per pole = 0.18 Weber
- Vdc = 270 Volts

The system operates in a stable way, however, torque ripple at steady state is considerably high.

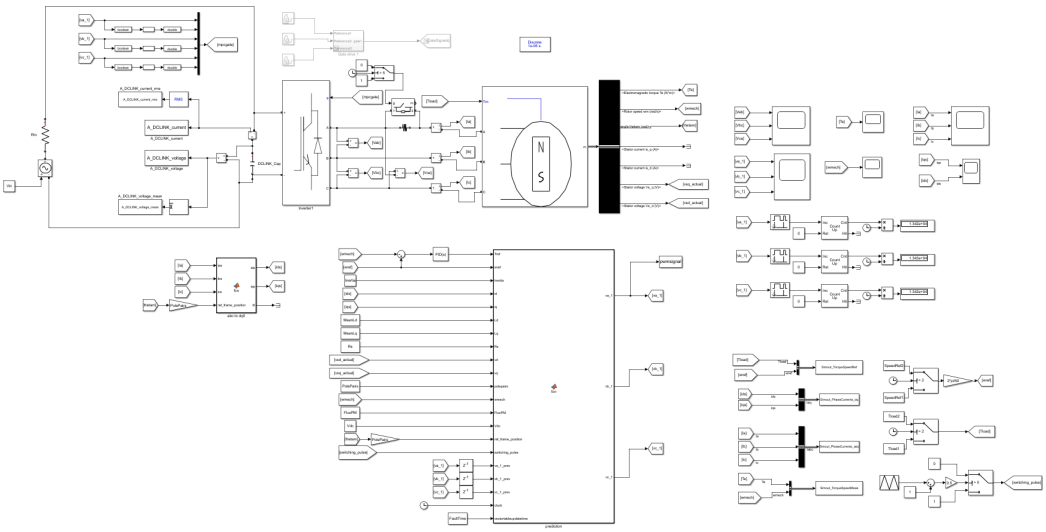

Figure 3: One Module Simulink Model

3.2 Simulation Results

The algorithm evaluates the cost function (line 47 of Listing 1) for 8 possible switching vectors for a two level inverter. The evalution is done for 40kHz rate. Due to the property of the MPC algorithm, the switching frequency varies. In one second, the switchings are around 12-14 kHz. For the beginning, the cost function only evaluates the torque error and the Id current. Some limits for the protection can be seen in line 41 of Listing 1.

The obtained results are listed in Figures 4, 5, 6.

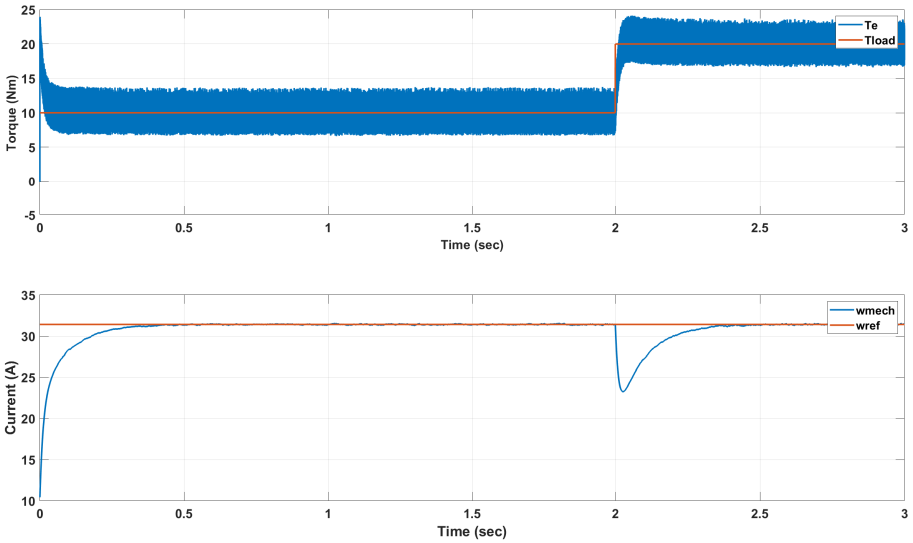

Figure 4: Torque and Rotor Speed for One Module

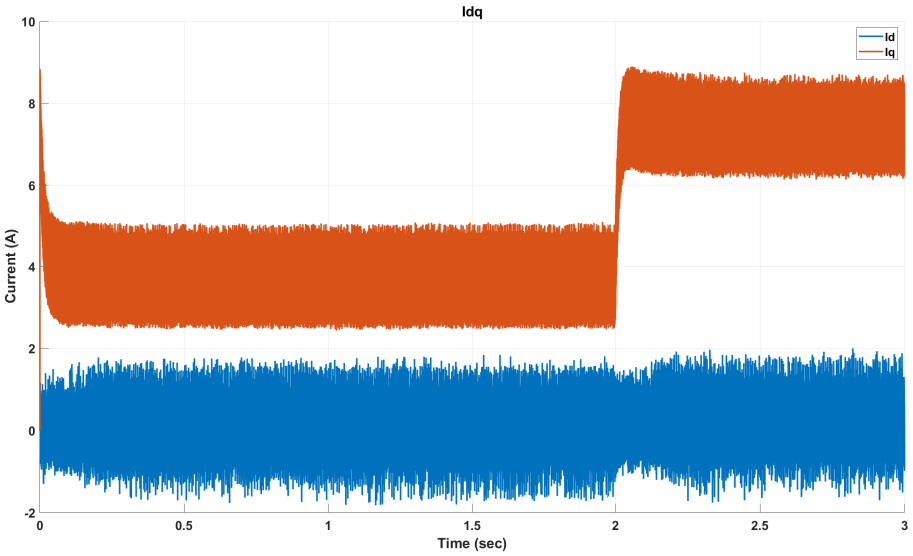

Figure 5: Id Iq for One Module

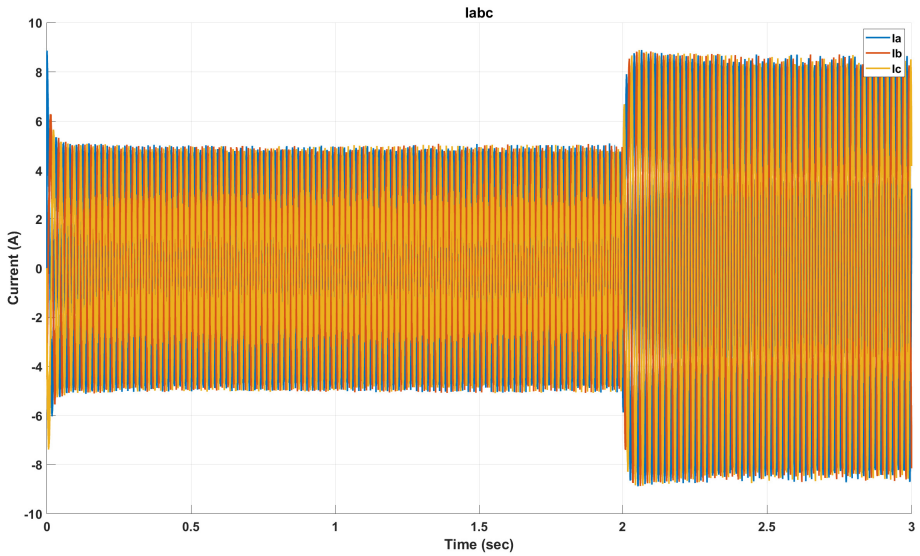

Figure 6: Phase Currents for One Module

4 Two Module Implementation

The configuration of the two module implementation is provided in Figure 7. There are two 3-phase winding groups that are contributing the torque. In construction of this model, first the cost functions are evaluated seperately for each module.

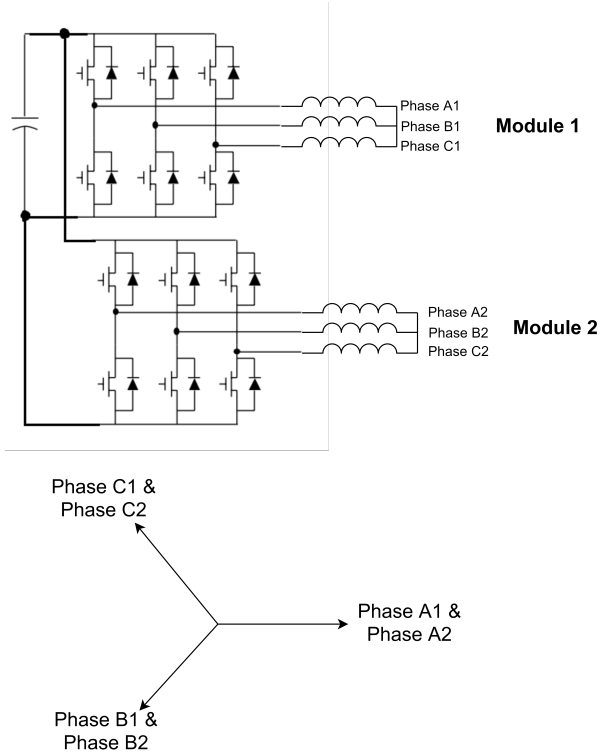

Figure 7: The drive circuity and winding configuration

The simulink model is consists of two PMSMs whose shafts are attached to each other, as can be seen in Figure 8. That way, the machines have the same angular speed and position during the simulation.

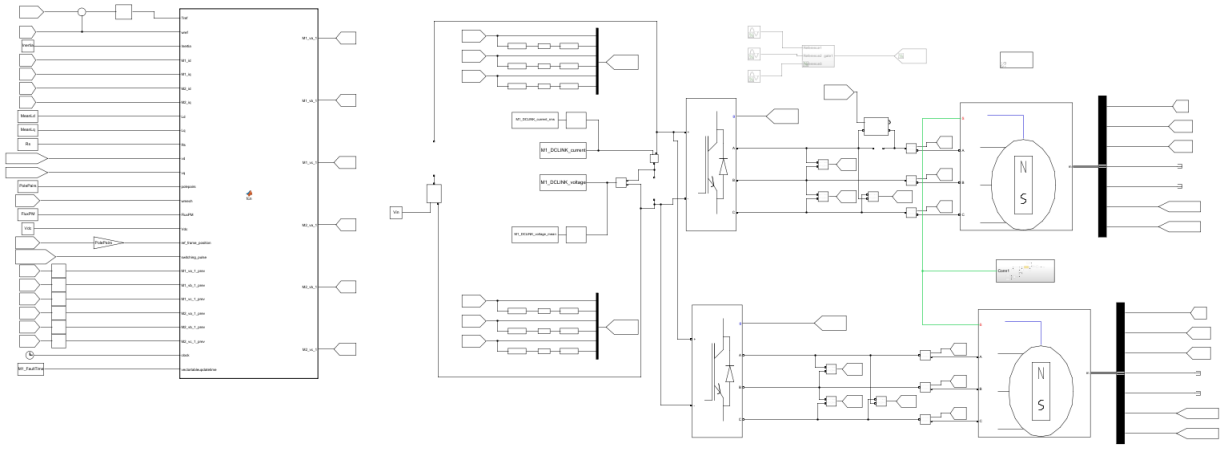

Figure 8: Two Module Simulink Model

For two module configuration, the cost function has been evaluated with different configurations. These are:

- Seperate cost functions for each module
- Combined cost function with the same switching vectors evaluated
- Combined cost function with all switching combinations evaluated

4.1 Seperate Cost Functions for Each Module

The seperate cost functions can be seen in line 60 and line 61 of Listing 2.

4.1.1 Healthy Operation

It can be seen from Figures 9, 10, 11, 12, that the steady state operation is achieved for seperate cost functions. However, the torque ripple is still considerably high for both modules.

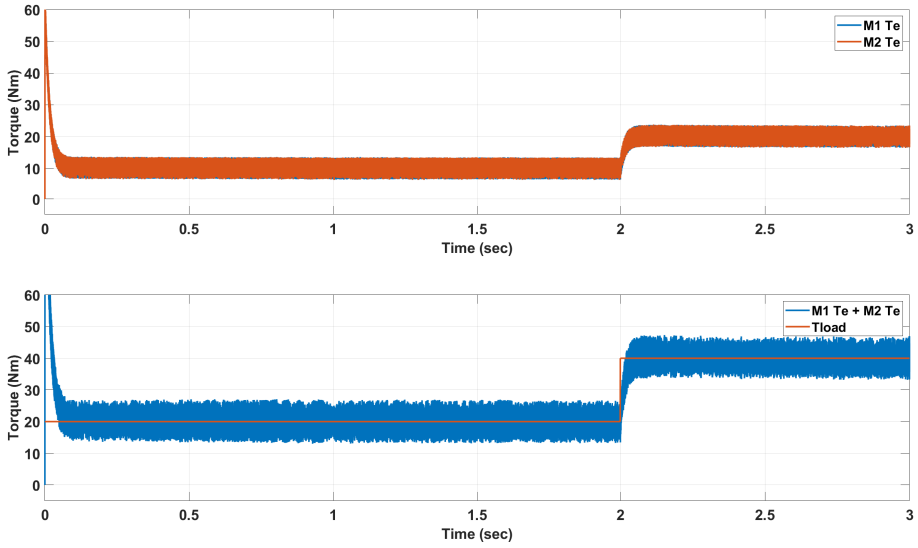

Figure 9: Torque generation of each module with seperate cost functions with no open circuit fault

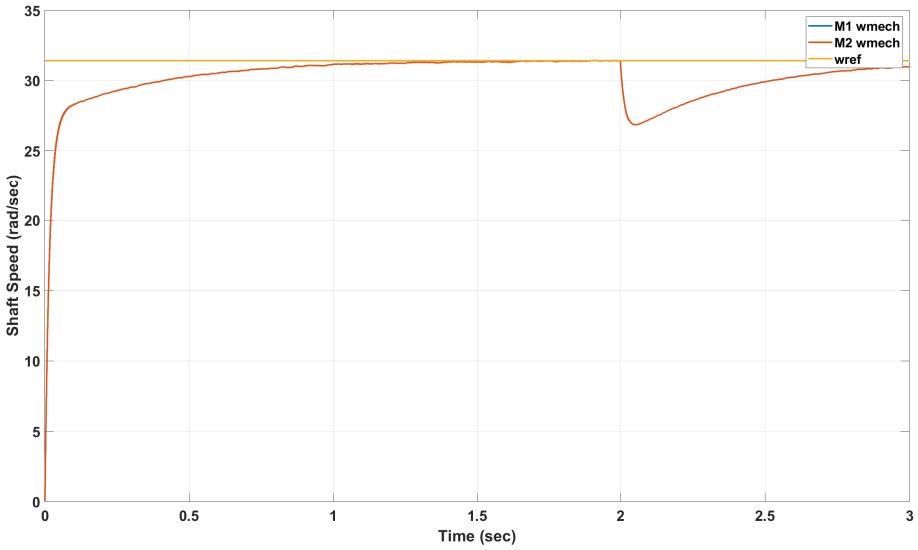

Figure 10: Shaft speed with seperate cost functions no open circuit fault

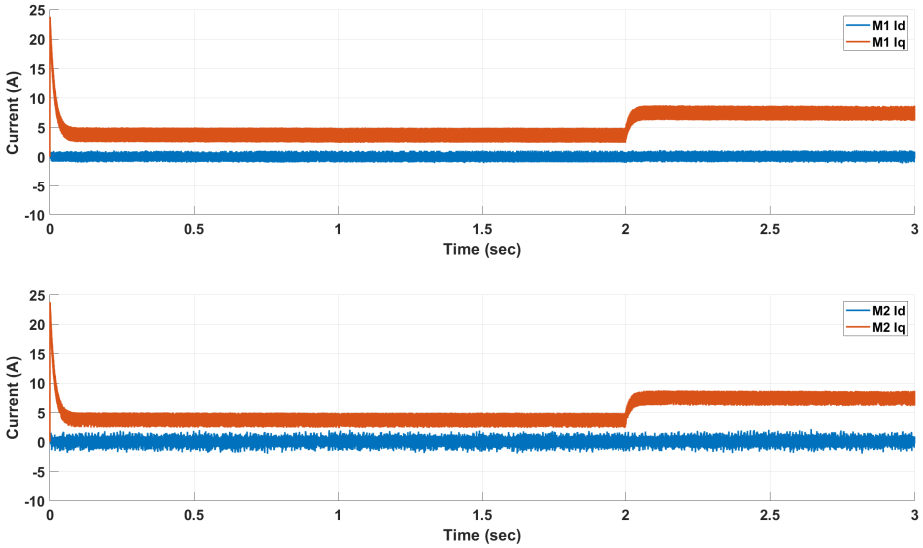

Figure 11: Id Iq for Two Modules with seperate cost functions no open circuit fault

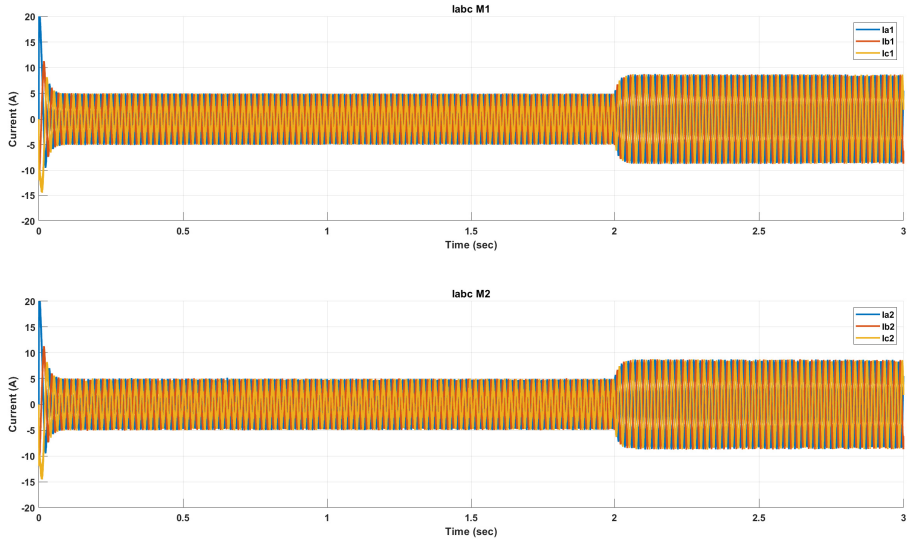

Figure 12: Phase Currents for Two Modules with seperate cost functions no open circuit fault

4.1.2 Faulty Operation

Even though this approach is suitable for steady state operations, the fault performance is not as expected. The reason is that, since each module is evaluated seperately, each module tries to optimize its own parameters. During the fault, the torque ripple of the faulty module directly added to the shaft torque which not desired (see Figure 13).

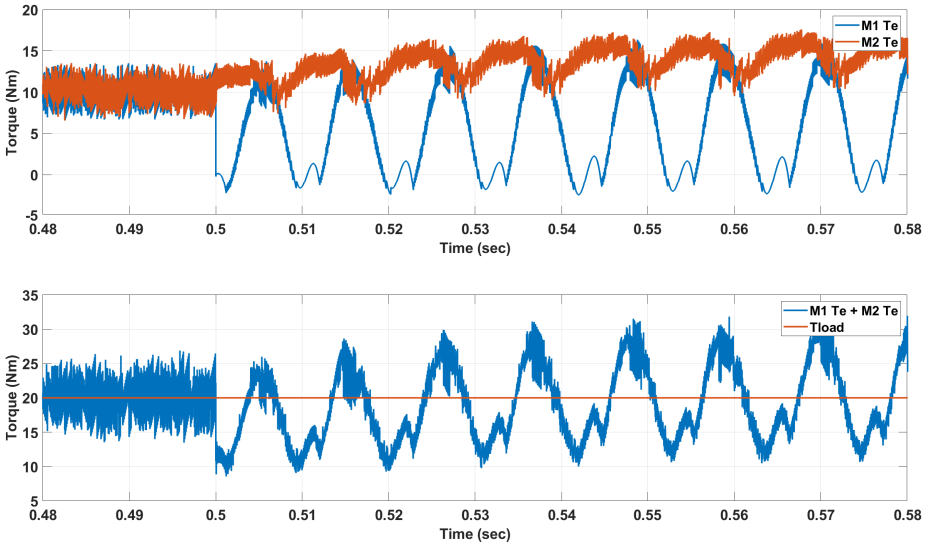

Figure 13: Torque Generation at Fault Moment of each module for seperate cost functions

4.2 Combined Cost Function

The combined cost function can be seen in line 60 of Listing 3. Note that the combined cost function is evaluated.

4.2.1 Healthy Operation

The simulation results for Healthy operation for combined cost function is provided in Figures 14, 15, 16, 17.

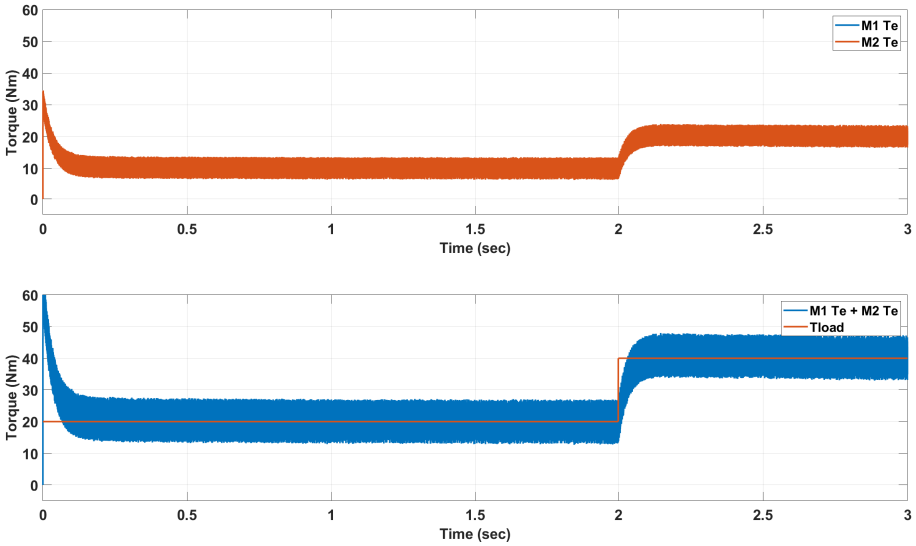

Figure 14: Torque generation of each module with combined cost function with no open circuit fault

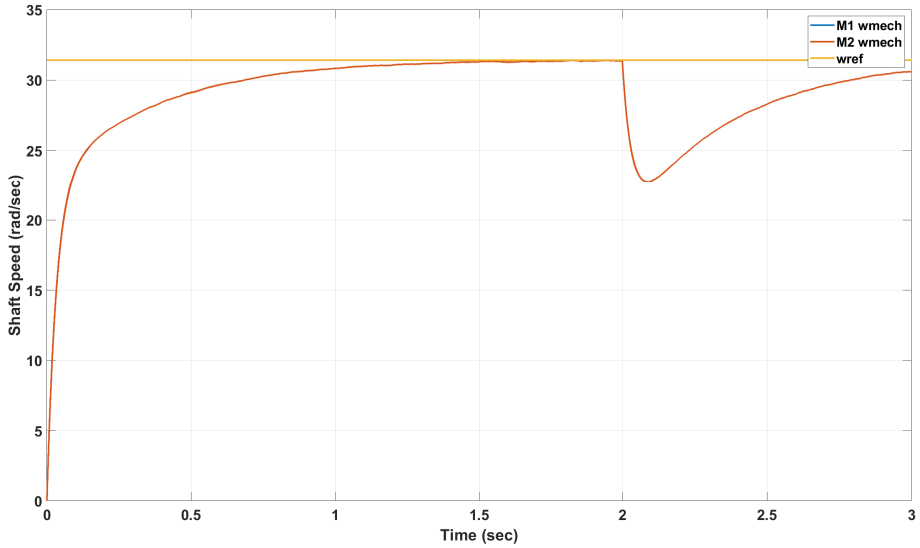

Figure 15: Shaft speed with combined cost function no open circuit fault

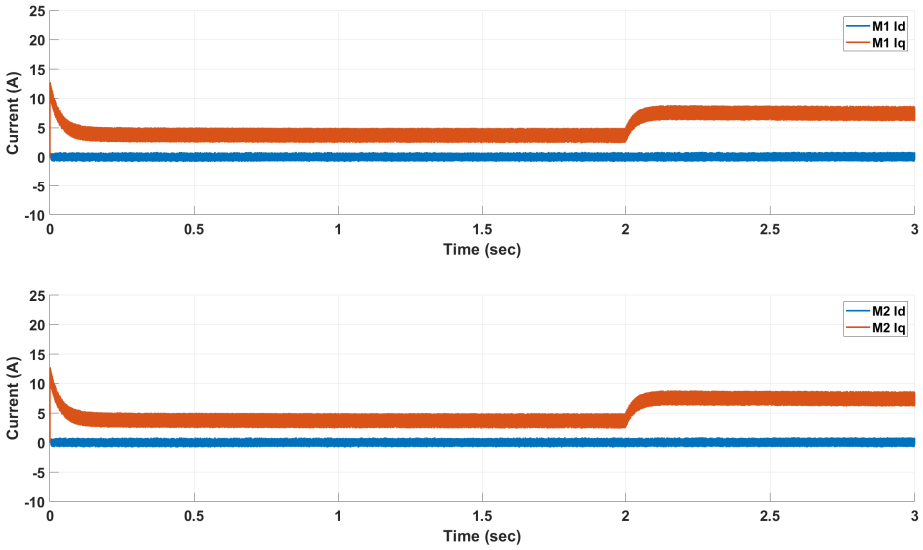

Figure 16: Id Iq for Two Modules with combined cost function no open circuit fault

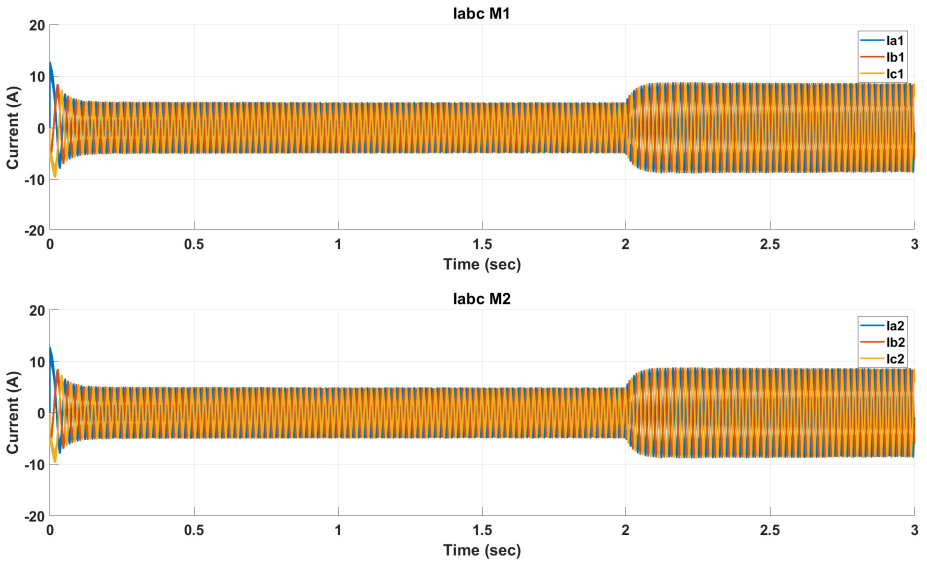

Figure 17: Phase Currents for Two Modules with combined cost function no open circuit fault

4.2.2 Faulty Operation

In the open circuit operation, the combined cost function method resulted in a more desirable operation. As can be seen in Figure 18. However, in this configuration, the same switching vectors are evaluated for both modules. To be more clear, only 8 switching vectors (same vector for both modules) are evaluted for the combined cost function. The switching vector chosen in this configuration may not be optimum, since all the vector combinations are not evaluted. This is further evaluted in the next section.

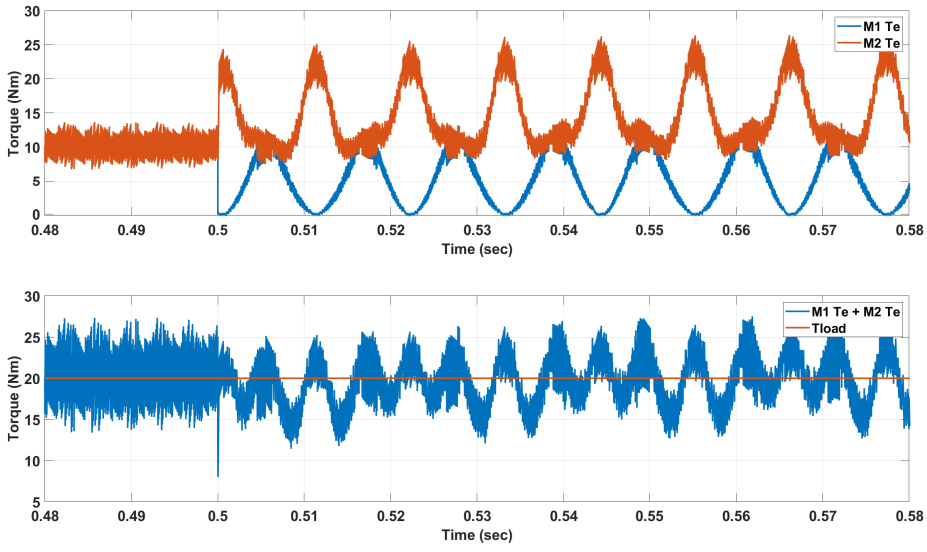

Figure 18: Torque Generation at Fault Moment of each module for combined cost function

4.3 Combined Cost Function V2, evaluation of every switching vector combinations for two modules

In this section, the switching vector evaluation for every possible switching combinations (8*8=64) for two modules is provided. This approach can provide the optimum switching vector since all the combinations

are evaluated with the cost function. The drawback is that the computational load increases exponentially.

The total cost function at line 59 of listings 4. There two for loops for evaluating to evaluate every switching combinations.

4.3.1 Healthy Operation

The simulation results for Healthy operation for combined cost function is provided in Figures 19, 20, 21, 22. In the cost function, the following parameters evaluated (as can be seen in line 59 of listing 4).

- Torque error
- Module 1 Id current
- Module 2 Id current
- Torque and current limits.

In the healthy operation simulations, it can be seen that the combined torque generated seem to be not different from other simulations (Figure 19). However, individual current and torque values of the modules seem to have a high ripples. In my opinion there needs to be an extra constraint for the selection of switching vectors. "Lack of constraints cause freedom of choice of switching vectors, which is not optimum", see Section 4.3.2. At first, I tried to make the modules to share the same torque value. However, under the faulty condition, the healthy modules started to produce the same torque as the faulty module, their torque ripples are added up, which is not desired. Therefore I concluded that there needs to be an extra constraint on the cost function evalution such as minimizing the conduction loss for both modules.

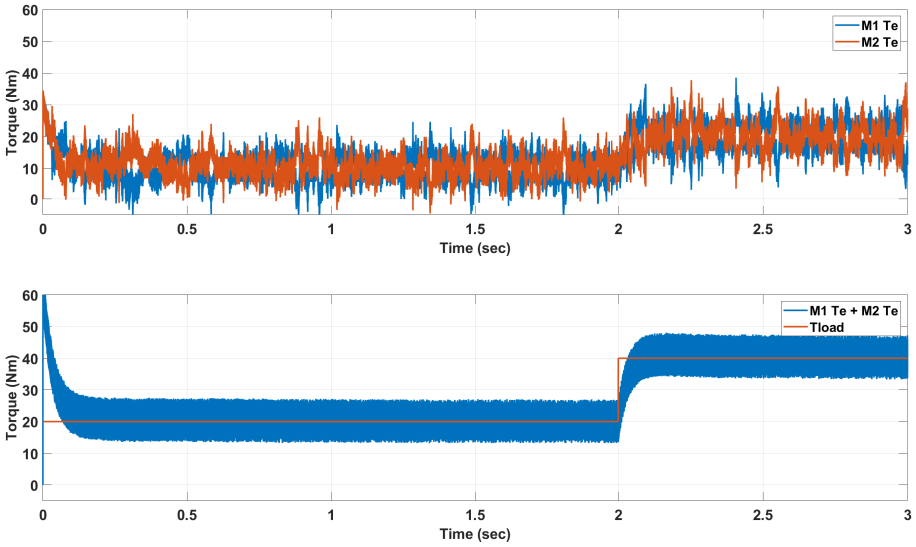

Figure 19: Torque generation of each module with combined cost function V2 with no open circuit fault

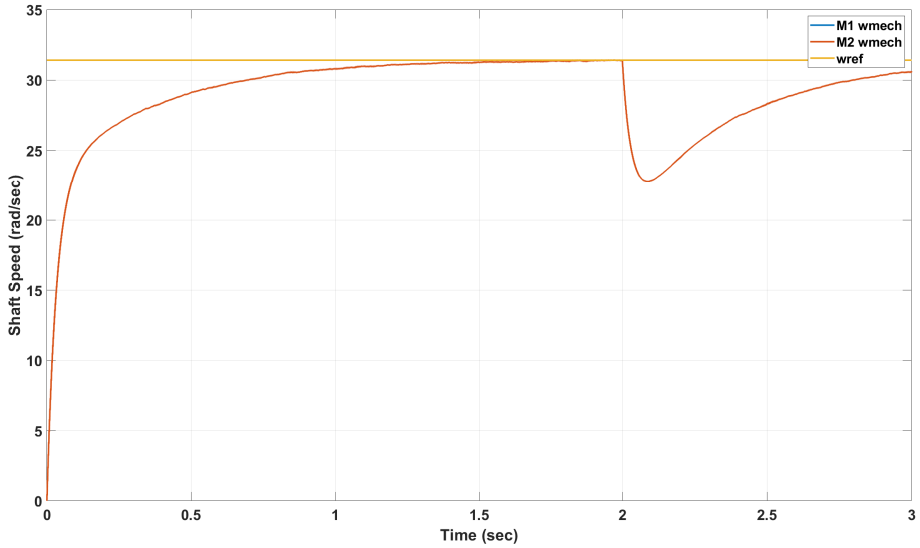

Figure 20: Shaft speed with combined cost function V2 no open circuit fault

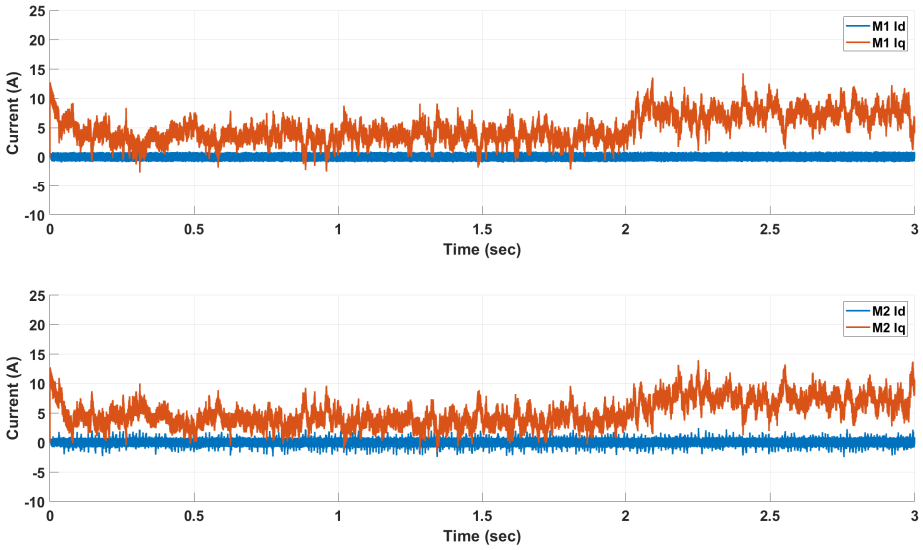

Figure 21: Id Iq for Two Modules with combined cost function V2 no open circuit fault

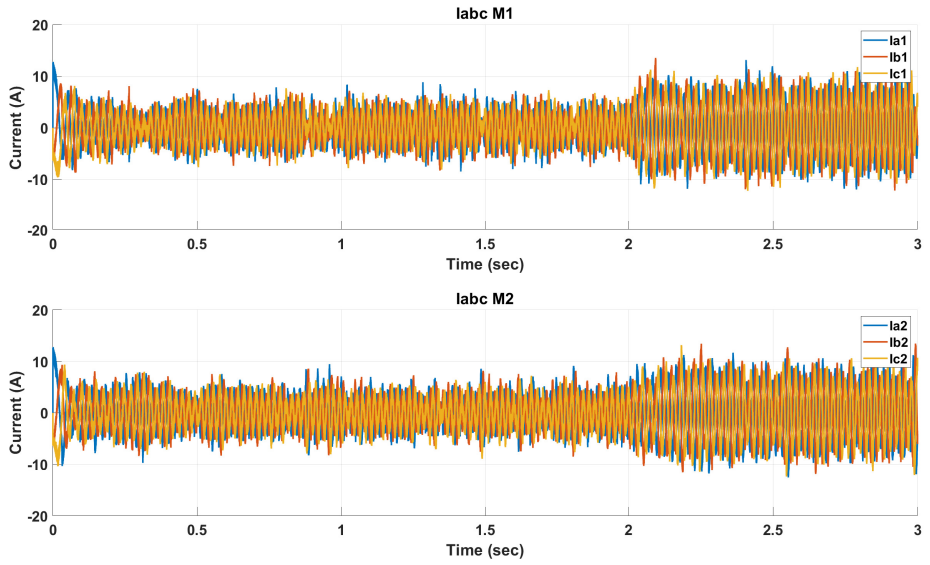

Figure 22: Phase Currents for Two Modules with combined cost function V2 no open circuit fault

4.3.2 Faulty Operation

The problem of freedom of switching vector choice that is mentioned in 4.3.1 causes more drastic problem under fault condition. As can be seen in Figure 23, even though the combined torque of the system is not absurd, the individual torque generated by each module gets absurd values.

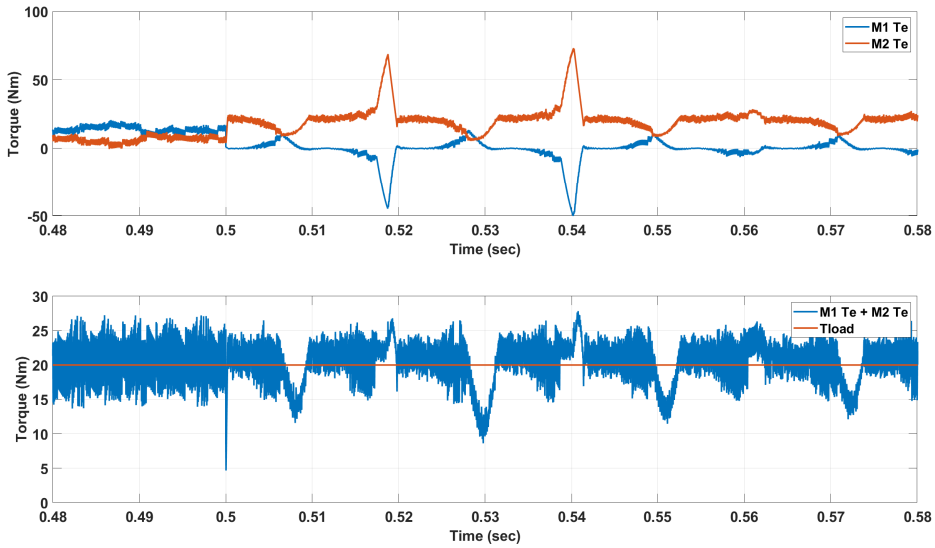

Figure 23: Torque Generation at Fault Moment of each module for combined cost function V2

5 Conclusion

As I have stated in 4.3.1 and 4.3.2, there needs to be extra constaint on the choice of an optimum vector choice. After this point, I am planning to add more constraints on the cost function that is mentioned in 4.3.1, such as minimizing the conduction losses. The problems that are needs to be solved can be listed as follows.

- In every case, the torque ripples are very high.
- An extra constaint will be used in the cost function (as mentioned in 4.3.1 and 4.3.2)
- Literature will be revised for faulty operations.
- Clarke and Park transformation will be modified for the faulty conditions.
- Rather than controlling the torque reference out of the PI controller, the iq value may need to be controlled.

Some of these problems may require longer time.

6 Appendix

Listing 1: Matlab Function for One Module Implementation

```
function [va_1, vb_1,vc_1 ] = fcn(Tref,wref,Inertia,id, iq, Ld, Lq...
    , Rs,vd, vq,polepairs,wmech,FluxPM,Vdc...
    ,ref_frame_position,switching_pulse,va_1_prev, vb_1_prev,vc_1_prev...
    ,clock,vectortableupdatetime)
VoltageVector = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];
Ts = 1/40000;
cost = zeros(1,8);
vectortobeapplied = zeros(1,3);
for i = 1:8

    %get the vectors
    if(clock>vectortableupdatetime)
        va = 1;
        vb = VoltageVector(i,2);
        vc = VoltageVector(i,3);
    else
        va = VoltageVector(i,1);
        vb = VoltageVector(i,2);
        vc = VoltageVector(i,3);
    end


    %calculate  vd vq v0
    vds = Vdc*2/3*(va*sin(ref_frame_position)...
        +vb*sin(ref_frame_position-2*pi/3)...
        +vc*sin(ref_frame_position+2*pi/3));
    vqs =  Vdc*2/3*(va*cos(ref_frame_position)...
        +vb*cos(ref_frame_position-2*pi/3)...
        +vc*cos(ref_frame_position+2*pi/3));
    v0 = Vdc*2/3*1/2*(va+vb+vc);
    %predict the future output
    id_1 = id + Ts*(vds/Ld-Rs/Ld*id...
        +Lq/Ld*polepairs*wmech*iq);
    iq_1 = iq + Ts*(vqs*Lq-Rs/Lq*iq...
        -Ld/Lq*polepairs*wmech*id-FluxPM*polepairs*wmech/Lq);
    Te_1 = 1.5*polepairs*(FluxPM*iq_1...
        +0*(Ld-Lq)*id_1*iq_1);
    % reluctance torque component is multiplied by 0,
    % since it caused unreasonable torque values.

```

```
    if(abs(iq)>30)(abs(Te_1)>50)
        protection = inf;
    else
        protection = 0;
    end

    cost(i) = 100000000*(Tref-Te_1)^2+(id_1)*1500+protection; %

end

[mincostval, index] = min(cost);
% apply the one with minimum cost
vectortobeapplied(1) = VoltageVector(index,1);
vectortobeapplied(2) = VoltageVector(index,2);
vectortobeapplied(3) = VoltageVector(index,3);


if(switching_pulse==1)
    va_1 = vectortobeapplied(1);
    vb_1 = vectortobeapplied(2);
    vc_1 = vectortobeapplied(3);
else
    va_1 = va_1_prev;
    vb_1 = vb_1_prev;
    vc_1 = vc_1_prev;
end
```

Listing 2: Matlab Function for Two Module Implementation With Seperate Cost Functions

```
function [M1_va_1, M1_vb_1, M1_vc_1, M2_va_1, M2_vb_1, M2_vc_1 ] = fcn(Tref,
    wref,Inertia,M1_id, M1_iq,M2_id, M2_iq, Ld, Lq, Rs,vd, vq,polepairs,wmech,
    FluxPM,Vdc,ref_frame_position,switching_pulse,M1_va_1_prev, M1_vb_1_prev,
    M1_vc_1_prev,M2_va_1_prev, M2_vb_1_prev,M2_vc_1_prev,clock,
    vectortableupdatetime)
M1_VoltageVector = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];
M2_VoltageVector = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];

Ts = 1/40000;

M1_cost = zeros(1,8);
M2_cost = zeros(1,8);
M1_vectortobeapplied = zeros(1,3);
M2_vectortobeapplied = zeros(1,3);

for i = 1:8

    %get the vectors
    if(clock>vectortableupdatetime)
        M1_va = 1;
        M1_vb = M1_VoltageVector(i,2);
        M1_vc = M1_VoltageVector(i,3);
    else
        M1_va = M1_VoltageVector(i,1);
        M1_vb = M1_VoltageVector(i,2);
        M1_vc = M1_VoltageVector(i,3);
    end
    M2_va = M2_VoltageVector(i,1);
```

```
25      M2_vb = M2_VoltageVector(i,2);
26      M2_vc = M2_VoltageVector(i,3);
27
28
29      %calculate  vd vq v0 for module 1
30      M1_vds = Vdc*2/3*(M1_va*sin(ref_frame_position)+M1_vb*sin(
            ref_frame_position-2*pi/3)+M1_vc*sin(ref_frame_position+2*pi/3));
31      M1_vqs =  Vdc*2/3*(M1_va*cos(ref_frame_position)+M1_vb*cos(
            ref_frame_position-2*pi/3)+M1_vc*cos(ref_frame_position+2*pi/3));
32      M1_v0 = Vdc*2/3*1/2*(M1_va+M1_vb+M1_vc);
33      %predict the future output for module 1
34      M1_id_1 = M1_id + Ts*(M1_vds/Ld-Rs/Ld*M1_id+Lq/Ld*polepairs*wmech*M1_iq);
35      M1_iq_1 = M1_iq + Ts*(M1_vqs*Lq-Rs/Lq*M1_iq-Ld/Lq*polepairs*wmech*M1_id-
            FluxPM*polepairs*wmech/Lq);
36      M1_Te_1 = 1.5*polepairs*(FluxPM*M1_iq_1+0*(Ld-Lq)*M1_id_1*M1_iq_1); %
            reluctance torque component is multiplied by 0, since it caused
            unreasonable torque values.
37
38      %calculate  vd vq v0 for module 2
39      M2_vds = Vdc*2/3*(M2_va*sin(ref_frame_position)+M2_vb*sin(
            ref_frame_position-2*pi/3)+M2_vc*sin(ref_frame_position+2*pi/3));
40      M2_vqs =  Vdc*2/3*(M2_va*cos(ref_frame_position)+M2_vb*cos(
            ref_frame_position-2*pi/3)+M2_vc*cos(ref_frame_position+2*pi/3));
41      M2_v0 = Vdc*2/3*1/2*(M2_va+M2_vb+M2_vc);
42      %predict the future output for module 2
43      M2_id_1 = M2_id + Ts*(M2_vds/Ld-Rs/Ld*M2_id+Lq/Ld*polepairs*wmech*M2_iq);
44      M2_iq_1 = M2_iq + Ts*(M2_vqs*Lq-Rs/Lq*M2_iq-Ld/Lq*polepairs*wmech*M2_id-
            FluxPM*polepairs*wmech/Lq);
45      M2_Te_1 = 1.5*polepairs*(FluxPM*M2_iq_1+0*(Ld-Lq)*M2_id_1*M2_iq_1); %
            reluctance torque component is multiplied by 0, since it caused
            unreasonable torque values.
46
47      wmech_1 = wmech + Ts*(M1_Te_1-Tref)*Inertia;
48
49      %calculate the cost
50      if(abs(M1_iq)>30)(abs(M1_Te_1)>80)
51          M1_protection = inf;
52      else
53          M1_protection = 0;
54      end
55      if(abs(M2_iq)>30)(abs(M2_Te_1)>80)
56          M2_protection = inf;
57      else
58          M2_protection = 0;
59      end
60      M1_cost(i) = 100000000*(Tref-M1_Te_1)^2+(M1_id_1)^2*1500*3+M1_protection;
            %
61      M2_cost(i) = 100000000*(Tref-M2_Te_1)^2+(M2_id_1)^2*1500*3+M2_protection;
            %
62
63  end
64
65  [mincostval, M1_index] = min(M1_cost);
66  [mincostval, M2_index] = min(M2_cost);
67  % apply the one with minimum cost
68  M1_vectortobeapplied(1) = M1_VoltageVector(M1_index,1);
```

```
M1_vectortobeapplied(2) = M1_VoltageVector(M1_index,2);
M1_vectortobeapplied(3) = M1_VoltageVector(M1_index,3);
M2_vectortobeapplied(1) = M2_VoltageVector(M2_index,1);
M2_vectortobeapplied(2) = M2_VoltageVector(M2_index,2);
M2_vectortobeapplied(3) = M2_VoltageVector(M2_index,3);


if(switching_pulse==1)
    M1_va_1 = M1_vectortobeapplied(1);
    M1_vb_1 = M1_vectortobeapplied(2);
    M1_vc_1 = M1_vectortobeapplied(3);
    M2_va_1 = M2_vectortobeapplied(1);
    M2_vb_1 = M2_vectortobeapplied(2);
    M2_vc_1 = M2_vectortobeapplied(3);
else
    M1_va_1 = M1_va_1_prev;
    M1_vb_1 = M1_vb_1_prev;
    M1_vc_1 = M1_vc_1_prev;
    M2_va_1 = M2_va_1_prev;
    M2_vb_1 = M2_vb_1_prev;
    M2_vc_1 = M2_vc_1_prev;
end
```

Listing 3: Matlab Function for Two Module Implementation Combined Cost Function

```
function [M1_va_1, M1_vb_1, M1_vc_1, M2_va_1, M2_vb_1, M2_vc_1,M1_Te_1,M2_Te_1
    ] = fcn(Tref,wref,Inertia,M1_id, M1_iq,M2_id, M2_iq, Ld, Lq, Rs,vd, vq,
    polepairs,wmech,FluxPM,Vdc,ref_frame_position,switching_pulse,M1_va_1_prev,
    M1_vb_1_prev,M1_vc_1_prev,M2_va_1_prev, M2_vb_1_prev,M2_vc_1_prev,clock,
    vectortableupdatetime)
M1_VoltageVector = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];
M2_VoltageVector = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];

Ts = 1/40000;
M1_cost = zeros(1,8);
M2_cost = zeros(1,8);
Total_Cost = zeros(1,8);
M1_vectortobeapplied = zeros(1,3);
M2_vectortobeapplied = zeros(1,3);

for i = 1:8
    %get the vectors
    if(clock>vectortableupdatetime)
        M1_va = 1;
        M1_vb = M1_VoltageVector(i,2);
        M1_vc = M1_VoltageVector(i,3);
    else
        M1_va = M1_VoltageVector(i,1);
        M1_vb = M1_VoltageVector(i,2);
        M1_vc = M1_VoltageVector(i,3);
    end
    M2_va = M2_VoltageVector(i,1);
    M2_vb = M2_VoltageVector(i,2);
    M2_vc = M2_VoltageVector(i,3);


    %calculate  vd vq v0 for module 1
```

```
29      M1_vds = Vdc*2/3*(M1_va*sin(ref_frame_position)+M1_vb*sin(
            ref_frame_position-2*pi/3)+M1_vc*sin(ref_frame_position+2*pi/3));
30      M1_vqs =  Vdc*2/3*(M1_va*cos(ref_frame_position)+M1_vb*cos(
            ref_frame_position-2*pi/3)+M1_vc*cos(ref_frame_position+2*pi/3));
31      M1_v0 = Vdc*2/3*1/2*(M1_va+M1_vb+M1_vc);
32      %predict the future output for module 1
33      M1_id_1 = M1_id + Ts*(M1_vds/Ld-Rs/Ld*M1_id+Lq/Ld*polepairs*wmech*M1_iq);
34      M1_iq_1 = M1_iq + Ts*(M1_vqs*Lq-Rs/Lq*M1_iq-Ld/Lq*polepairs*wmech*M1_id-
            FluxPM*polepairs*wmech/Lq);
35      M1_Te_1 = 1.5*polepairs*(FluxPM*M1_iq_1+0*(Ld-Lq)*M1_id_1*M1_iq_1); %
            reluctance torque component is multiplied by 0, since it caused
            unreasonable torque values.
36
37      %calculate  vd vq v0 for module 2
38      M2_vds = Vdc*2/3*(M2_va*sin(ref_frame_position)+M2_vb*sin(
            ref_frame_position-2*pi/3)+M2_vc*sin(ref_frame_position+2*pi/3));
39      M2_vqs =  Vdc*2/3*(M2_va*cos(ref_frame_position)+M2_vb*cos(
            ref_frame_position-2*pi/3)+M2_vc*cos(ref_frame_position+2*pi/3));
40      M2_v0 = Vdc*2/3*1/2*(M2_va+M2_vb+M2_vc);
41      %predict the future output for module 2
42      M2_id_1 = M2_id + Ts*(M2_vds/Ld-Rs/Ld*M2_id+Lq/Ld*polepairs*wmech*M2_iq);
43      M2_iq_1 = M2_iq + Ts*(M2_vqs*Lq-Rs/Lq*M2_iq-Ld/Lq*polepairs*wmech*M2_id-
            FluxPM*polepairs*wmech/Lq);
44      M2_Te_1 = 1.5*polepairs*(FluxPM*M2_iq_1+0*(Ld-Lq)*M2_id_1*M2_iq_1); %
            reluctance torque component is multiplied by 0, since it caused
            unreasonable torque values.
45
46      wmech_1 = wmech + Ts*(M1_Te_1-Tref)*Inertia;
47
48      %calculate the cost
49      if(abs(M1_iq)>30)(abs(M1_Te_1)>80)
50          M1_protection = inf;
51      else
52          M1_protection = 0;
53      end
54      if(abs(M2_iq)>30)(abs(M2_Te_1)>80)
55          M2_protection = inf;
56      else
57          M2_protection = 0;
58      end
59
60          Total_Cost(i) = 100000000*(Tref-M2_Te_1-M1_Te_1)^2+((M1_id_1)^2)
                *1500*15+((M2_id_1)^2)*1500*15; %
61
62  end
63
64
65
66  [mincostval, M1_index] = min(Total_Cost);
67  [mincostval, M2_index] = min(Total_Cost);
68
69  % apply the one with minimum cost
70  M1_vectortobeapplied(1) = M1_VoltageVector(M1_index,1);
71  M1_vectortobeapplied(2) = M1_VoltageVector(M1_index,2);
72  M1_vectortobeapplied(3) = M1_VoltageVector(M1_index,3);
73  M2_vectortobeapplied(1) = M2_VoltageVector(M2_index,1);
```

```
M2_vectortobeapplied(2) = M2_VoltageVector(M2_index,2);
M2_vectortobeapplied(3) = M2_VoltageVector(M2_index,3);


if(switching_pulse==1)
    M1_va_1 = M1_vectortobeapplied(1);
    M1_vb_1 = M1_vectortobeapplied(2);
    M1_vc_1 = M1_vectortobeapplied(3);
    M2_va_1 = M2_vectortobeapplied(1);
    M2_vb_1 = M2_vectortobeapplied(2);
    M2_vc_1 = M2_vectortobeapplied(3);
else
    M1_va_1 = M1_va_1_prev;
    M1_vb_1 = M1_vb_1_prev;
    M1_vc_1 = M1_vc_1_prev;
    M2_va_1 = M2_va_1_prev;
    M2_vb_1 = M2_vb_1_prev;
    M2_vc_1 = M2_vc_1_prev;
end
```

Listing 4: Matlab Function for Two Module Implementation Combined Cost Function V2

```
function [M1_va_1, M1_vb_1, M1_vc_1, M2_va_1, M2_vb_1, M2_vc_1,M1_Te_1,M2_Te_1
    ] = fcn(Tref,wref,Inertia,M1_id, M1_iq,M2_id, M2_iq, Ld, Lq, Rs,vd, vq,
    polepairs,wmech,FluxPM,Vdc,ref_frame_position,switching_pulse,M1_va_1_prev,
    M1_vb_1_prev,M1_vc_1_prev,M2_va_1_prev, M2_vb_1_prev,M2_vc_1_prev,clock,
    vectortableupdatetime)
M1_VoltageVector = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];
M2_VoltageVector = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 1 1 1];
Ts = 1/40000;


M1_cost = zeros(1,8);
M2_cost = zeros(1,8);
Total_Cost = zeros(1,8*8);
VectorArray = zeros(64,6);
M1_vectortobeapplied = zeros(1,3);
M2_vectortobeapplied = zeros(1,3);
k = 1;

for i = 1:8
    %get the vectors
    if(clock>vectortableupdatetime)
        M1_va = 1;
        M1_vb = M1_VoltageVector(i,2);
        M1_vc = M1_VoltageVector(i,3);
    else
        M1_va = M1_VoltageVector(i,1);
        M1_vb = M1_VoltageVector(i,2);
        M1_vc = M1_VoltageVector(i,3);
    end
    %calculate  vd vq v0 for module 1
    M1_vds = Vdc*2/3*(M1_va*sin(ref_frame_position)+M1_vb*sin(
        ref_frame_position-2*pi/3)+M1_vc*sin(ref_frame_position+2*pi/3));
    M1_vqs =  Vdc*2/3*(M1_va*cos(ref_frame_position)+M1_vb*cos(
        ref_frame_position-2*pi/3)+M1_vc*cos(ref_frame_position+2*pi/3));
    M1_v0 = Vdc*2/3*1/2*(M1_va+M1_vb+M1_vc);
```

```
30      %predict the future output for module 1
31      M1_id_1 = M1_id + Ts*(M1_vds/Ld-Rs/Ld*M1_id+Lq/Ld*polepairs*wmech*M1_iq);
32      M1_iq_1 = M1_iq + Ts*(M1_vqs*Lq-Rs/Lq*M1_iq-Ld/Lq*polepairs*wmech*M1_id-
            FluxPM*polepairs*wmech/Lq);
33      M1_Te_1 = 1.5*polepairs*(FluxPM*M1_iq_1+0*(Ld-Lq)*M1_id_1*M1_iq_1); %
            reluctance torque component is multiplied by 0, since it caused
            unreasonable torque values.
34
35
36      for j=1:8
37          M2_va = M2_VoltageVector(j,1);
38          M2_vb = M2_VoltageVector(j,2);
39          M2_vc = M2_VoltageVector(j,3);
40          %calculate  vd vq v0 for module 2
41          M2_vds = Vdc*2/3*(M2_va*sin(ref_frame_position)+M2_vb*sin(
                ref_frame_position-2*pi/3)+M2_vc*sin(ref_frame_position+2*pi/3));
42          M2_vqs =  Vdc*2/3*(M2_va*cos(ref_frame_position)+M2_vb*cos(
                ref_frame_position-2*pi/3)+M2_vc*cos(ref_frame_position+2*pi/3));
43          M2_v0 = Vdc*2/3*1/2*(M2_va+M2_vb+M2_vc);
44          %predict the future output for module 2
45          M2_id_1 = M2_id + Ts*(M2_vds/Ld-Rs/Ld*M2_id+Lq/Ld*polepairs*wmech*
                M2_iq);
46          M2_iq_1 = M2_iq + Ts*(M2_vqs*Lq-Rs/Lq*M2_iq-Ld/Lq*polepairs*wmech*
                M2_id-FluxPM*polepairs*wmech/Lq);
47          M2_Te_1 = 1.5*polepairs*(FluxPM*M2_iq_1+0*(Ld-Lq)*M2_id_1*M2_iq_1); %
                reluctance torque component is multiplied by 0, since it caused
                unreasonable torque values.
48          if(abs(M1_iq)>30)(abs(M1_Te_1)>80)
49              M1_protection = inf;
50          else
51              M1_protection = 0;
52          end
53          if(abs(M2_iq)>30)(abs(M2_Te_1)>80)
54              M2_protection = inf;
55          else
56              M2_protection = 0;
57          end
58
59          Total_Cost(k) = 100000000*(Tref-M2_Te_1-M1_Te_1)^2+((M1_id_1)^2)
                *1500*10+((M2_id_1)^2)*1500*10+M1_protection+M2_protection; %
60
61          VectorArray(k,1) = M1_VoltageVector(i,1);
62          VectorArray(k,2) = M1_VoltageVector(i,2);
63          VectorArray(k,3) = M1_VoltageVector(i,3);
64          VectorArray(k,4) = M2_VoltageVector(j,1);
65          VectorArray(k,5) = M2_VoltageVector(j,2);
66          VectorArray(k,6) = M2_VoltageVector(j,3);
67          k = k+1;
68      end
69  end
70
71
72
73  [mincostval, M1_index] = min(Total_Cost);
74  [mincostval, M2_index] = min(Total_Cost);
75  [mincostval, M1M2_index] = min(Total_Cost);
```

```
76
77 % apply the one with minimum cost
78 M1_vectortobeapplied(1) = VectorArray(M1M2_index,1);
79 M1_vectortobeapplied(2) = VectorArray(M1M2_index,2);
80 M1_vectortobeapplied(3) = VectorArray(M1M2_index,3);
81 M2_vectortobeapplied(1) = VectorArray(M1M2_index,4);
82 M2_vectortobeapplied(2) = VectorArray(M1M2_index,5);
83 M2_vectortobeapplied(3) = VectorArray(M1M2_index,6);
84
85
86 if(switching_pulse==1)
87     M1_va_1 = M1_vectortobeapplied(1);
88     M1_vb_1 = M1_vectortobeapplied(2);
89     M1_vc_1 = M1_vectortobeapplied(3);
90     M2_va_1 = M2_vectortobeapplied(1);
91     M2_vb_1 = M2_vectortobeapplied(2);
92     M2_vc_1 = M2_vectortobeapplied(3);
93 else
94     M1_va_1 = M1_va_1_prev;
95     M1_vb_1 = M1_vb_1_prev;
96     M1_vc_1 = M1_vc_1_prev;
97     M2_va_1 = M2_va_1_prev;
98     M2_vb_1 = M2_vb_1_prev;
99     M2_vc_1 = M2_vc_1_prev;
100 end
```

References

[1] Abdelsalam A. Ahmed. *Experimental Implementation of Model Predictive Control for Permanent Magnet Synchronous Motor*. waset, 2015.