1 Extended Kalman Filter - Flux and Speed Observer

1.1 The model of motor used

The motor that this flux observer targets is a squirrel-cage induction motor.

The state space representation of the motor used in the EKF, as given in [1] (with modifications to allow multiple pole pairs as used in [2]) in the form $\hat{\dot{x}} = A\hat{x} + Bu$ and $y = C\hat{x}$ is as follows:

$$\hat{x} = \begin{bmatrix} -\frac{L_m^2 R_r + L_r^2 Rs}{\sigma L_s L_r^2} & 0 & \frac{L_m R_r}{\sigma L_s L_r^2} & \frac{p L_m \omega}{2\sigma L_s L_r} & 0 \\ 0 & -\frac{L_m^2 R_r + L_r^2 Rs}{\sigma L_s L_r^2} & -\frac{p L_m \omega}{2\sigma L_s L_r} & \frac{L_m R_r}{\sigma L_s L_r^2} & 0 \\ \frac{L_m R_r}{L_r} & 0 & -\frac{Rr}{Lr} & -\frac{p}{2}\omega & 0 \\ 0 & \frac{L_m R_r}{L_r} & \frac{p}{2}\omega & -\frac{Rr}{Lr} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{ds} \\ i_{qs} \\ \psi_{dr} \\ \psi_{qr} \\ \omega \end{bmatrix} + \frac{1}{\sigma L_s} \begin{bmatrix} v_{ds} \\ v_{qs} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{ds} \\ i_{qs} \\ \psi_{dr} \\ \psi_{qr} \\ \omega \end{bmatrix}$$

Where the symbols are defined as follows:

L_s	Stator inductance	R_s	Stator resistance
L_r	Rotor inductance	R_r	Rotor resistance
L_m	Mutual inductance	ω	Rotor angular velocity
i_{ds}	Direct stator current	v_{ds}	Direct stator voltage
i_{qs}	Quadrature stator current	v_{qs}	Quadrature stator voltage
ψ_{dr}	Direct rotor flux	σ	$1 - L_m^2/(L_s L_r)$
ψ_{qr}	Quadrature rotor flux	p	Number of motor poles

1.2 Discretisation

Rather than using a discretisation of the form $A_d = e^{A\tau} = \mathscr{L}^{-1}(sI - A)^{-1}|_{\tau=t}$, which would be computationally intensive to calculate at each time-step the approximation $A_d = I + A\tau$ is used, where τ is the model sample time.

This leads to the main model being:

$$\hat{x}_{k+1} = \begin{bmatrix} 1 - \frac{L_m^2 R_r + L_r^2 Rs}{\sigma L_s L_r^2}\tau & 0 & \frac{L_m R_r}{\sigma L_s L_r^2}\tau & \frac{p L_m \omega}{2\sigma L_s L_r}\tau & 0 \\ 0 & 1 - \frac{L_m^2 R_r + L_r^2 Rs}{\sigma L_s L_r^2}\tau & -\frac{p L_m \omega}{2\sigma L_s L_r}\tau & \frac{L_m R_r}{\sigma L_s L_r^2}\tau & 0 \\ \frac{L_m R_r}{L_r}\tau & 0 & 1 - \frac{Rr}{Lr}\tau & -\frac{p}{2}\tau\omega & 0 \\ 0 & \frac{L_m R_r}{L_r} & \frac{p}{2}\tau\omega & 1 - \frac{Rr}{Lr}\tau & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \hat{x}_k + \frac{\tau}{\sigma L_s} \begin{bmatrix} v_{ds} \\ v_{qs} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

1.3 The Kalman gain

The Kalman gain is calculated iteratively - estimation of the matrix P, update of the Kalman gain, then update of the matrix P using the new Kalman gain.

The first stage is to calculate an estimate of the P matrix at step k given the P matrix at step $k-1$.

```
% Estimate the P matrix
P = F * P * F' + Q;
```

Where the Q matrix is the process noise covariance matrix and the F matrix is the partial derivative of $A_d\hat{x}_k + B_d u_k$ with respect to $\hat{x}_k$ and takes the form:

$$F_k = \begin{bmatrix} 1 - \frac{L_m^2 R_r + L_r^2 Rs}{\sigma L_s L_r^2}\tau & 0 & \frac{L_m R_r}{\sigma L_s L_r^2}\tau & \frac{pL_m\omega}{2\sigma L_s L_r}\tau & \frac{pL_m}{2\sigma L_s L_r}\phi_{qr}\tau \\ 0 & 1 - \frac{L_m^2 R_r + L_r^2 Rs}{\sigma L_s L_r^2}\tau & -\frac{pL_m\omega}{2\sigma L_s L_r}\tau & \frac{L_m R_r}{\sigma L_s L_r^2}\tau & -\frac{pL_m}{2\sigma L_s L_r}\phi_{dr}\tau \\ \frac{L_m R_r}{L_r}\tau & 0 & 1 - \frac{Rr}{Lr}\tau & -\frac{p}{2}\omega\tau & -\frac{p}{2}\phi_{qr}\tau \\ 0 & \frac{L_m R_r}{L_r} & \frac{p}{2}\omega\tau & 1 - \frac{Rr}{Lr}\tau & \frac{p}{2}\phi_{dr}\tau \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The Kalman gain is then calulated as follows:

```
% Calculate P * C' once only
% PCdash = P * C';
PCdash = P(:,[1:2]);

% Update the Kalman gain
L = PCdash/(PCdash([1:2],:) + R);
```

Finally the estimate of the P matrix is updated using the new value of the Kalman gain.

```
% Calculate (I-L*C) once only
IminusLC(1,1) = 1 - L(1,1);
IminusLC(2,1) = -L(2,1);
IminusLC(3,1) = -L(3,1);
IminusLC(4,1) = -L(4,1);
IminusLC(5,1) = -L(5,1);
IminusLC(1,2) = -L(1,2);
IminusLC(2,2) = 1 - L(2,2);
IminusLC(3,2) = -L(3,2);
IminusLC(4,2) = -L(4,2);
IminusLC(5,2) = -L(5,2);

% Update the estimate of the P matrix
P = IminusLC * P;
```

1.4 Code optimisations

The first optimisation is that as much calculation as possible is done off-line in the block mask in the Simulink diagram and passed as inline parameters to the Embedded MATLAB code. This means that the Embedded Target need not perform unnecessary calculations which would yield the same result each time.

```
sigma = 1 - Lm^2/(Ls*Lr);

var1 = (1 - T*(Lm^2*Rr + Lr^2*Rs)/(sigma*Ls*Lr^2));
var2 = T*Lm*Rr/(sigma*Ls*Lr^2);
var3star = poles*T*Lm/(2*(sigma*Ls*Lr));
var4 = T*Lm*Rr/Lr;
var5 = (1-T*Rr/Lr);
var6 = poles*T/2;

b = T/(sigma*Ls);
```

This generates an A_d matrix that would look something like:

```
%A = [var1,   0,      var2,          var3,          0;
%     0,       var1,   -var3,         var2,          0;
%     var4,    0,      var5,           -var6*X(5),     0;
%     0,       var4,   var6*X(5),      var5,           0;
%     0,       0,      0,              0,              1];
```

Similarly, the F matrix looks something like:

```
%   F = [var1,    0,       var2,       var3,          var3star*X(4);
%        0,       var1,    -var3,      var2,          -var3star*X(3);
%        var4,    0,       var5,       -var6*X(5),    -var6*X(4);
%        0,       var4,    var6*X(5),  var5,          var6*X(3);
%        0,       0,       0,          0,             1];
```

However, to save on CPU cycles, these matrices are initialised in Embedded MATLAB with zeros, and non-zero elements individually set. This is because when specifying the matrix, the C-code generated by Embedded MATLAB builds up the matrix in a series of arrays then copies them into place in the persistent matrix variable. It saves on memory accesses to initialise all points to a constant of zero and then place the non-zero values in afterwards.

Additionally, the A_d matrix `A` and the F_k matrix `F` are declared *persistent* and after a first-time initialisation, only values which are dependent on time are updated on each model iteration.

Where the `A` and `F` matrices are the same, they are set at the same time, and where possible, if two elements are to be set to the same value they are set at the same time, so that if inline-parameters are not enabled in Simulink there is a likelihood that the constant will still be in a register for the subsequent assignments. In addition, where a multiplication takes place, if the result is required in multiple places, it is carried out only once, and then the second destination is assigned to the value of the first to avoid carrying out the multiplication operation twice.

When an interim result is to be used more than once, it is explicitly calculated beforehand in a temporary variable to avoid unnecessary matrix multiplication.

Knowledge about the positions of the zeros B and C matrices are used so that rather than performing an expensive matrix multiplication, an alternative reshape is performed.

Single precision floating point is chosen, as the operations for this are slightly faster than for double precision. Tests run on the workstation against a simulation performed by SimPowerSystems showed that this did not provide any noticable loss of numerical performance.

1.5 Testing the filter

A simulation of an induction motor was performed using SimPowerSystems, using a modified version of one of the demonstration models, configured to output the two rotor flux values, the two stator current values, the two stator voltage values and the angular velocity to a file - this was done because the induction motor simulation was prohibitively slow, and testing the filter's performance as an observer required testing with varying parameters.

1.6 Testing on the Real Time Target Hardware

To test the performance on the MPC555 prototype board, the EKF block was run at different sampling frequencies alongside another, low-power function connected to an output. It was observed at each frequency whether the target failed due to sample-time over-runs. After the optimisations described above, and enabling inline parameter values in Real Time Workshop, it was possible to run the EKF at a sampling frequency of 1 kHz when the MPC555 was clocked at 40 MHz.

Order of magnitude calculations suggest that this could be improved by using more highly optimised matrix multiplication algorithms - the Embedded MATLAB coder produces code which explicitly multiplies the matrices rather than using the Real Time Workshop matrix multiplication functions.

References

[1] Bimal K. Bose. *Modern Power Electronics and AC Drives*. Prentice Hall PTR, 2002.

[2] K. L. Shi, T. F. Chan, Y. K. Wong, and S. L. Ho. Speed estimation of an induction motor drive using an optimized extended kalman filter. *IEEE Transactions on Industrial Electronics*, 49(1):124–133, Feb 2002.