**Design Document**

**For**

**Optimized Sine Cosine Algorithm**

**VERSION: 1.0**

**DATE: 15-Feb-2016**

**Prepared By:**

**Jason Erlenbeck**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Author** | **Version** | **Date** |
| Initial Version | Jason Erlenbeck | 1.0 | 15-Feb-2016 |

**Optimized SinCos Algorithm**

1. **Background:**

The following diagram is an overview of the operations that are performed during digital motor angle initialization to compute their correction tables.

Generate Reconstructed Waveform Pseudo Code:

*for n = 0 to 128* // size of correction table

As shown in the pseudo code there is several calls to the sine and cosine function (128 \* 12). Furthermore, the algorithm requires both the sine and cosine of the same angle. During benchmarking activities the following algorithm was developed and shown to have significant time improvements (~= 5.5 mSec vs ~= 2.4 mSec) over the standard sine and cosine library functions.

1. **Sine and Cosine as Taylor Series:**
2. **Range Reduction:**

Although the Taylor Series for sine and cosine are exact it technically takes an infinite number of terms. The purpose of limiting the value of x into the Taylor Series is so that it doesn’t take many terms before the factorial in the denominator dwarfs the numerator. To limit the input range of x the following identity is used:

For reasons that will be discussed later B will be declared as 22.5n, where n is an integer between 0 and 15. Using this technique the largest x input into the Taylor Series will be +/- 11.25 Degrees (0.1963 Radians).

Now that the maximum input value of x has been established it is possible to determine the number of terms in the Taylor Series that are required to achieve acceptable levels of accuracy.

| **# of Terms** | **Sin(x)** | **Sin(x) Max Error** | **Cos(x)** | **Cos(x) Max Error** |
| --- | --- | --- | --- | --- |
| **1** |  | **0.001259** |  | **0.0192** |
| **2** |  | **2.4297E-6** |  | **6.18515E-5** |
| **3** |  | **2.2312E-9** |  | **7.953E-8** |

As the table above shows an acceptable level of accuracy can be achieved with only 2 terms in the sine and cosine Taylor Series once the input has been range limited to +/- 11.25 degrees.

1. **Algorithm:**

**Calculate n & B:**

As shown before, n is the largest integer that satisfies: 22.5n < x. This can be performed using integer math where:

The above equation will leave a remainder that ranges from 0 to 22.5 degrees. By adding in a 0.5 term the result will round up leaving a remainder with the desired range of +/- 11.25 Degrees. Therefore n is calculated as:

The B term is a simple multiple of n:

**Determine A:**

The A term is the effective remainder of the input angle *x* minus the B term.

**Compute Sin(A) & Cos(A):**

Once input x has been range limited into A the term limited Taylor Series of sine and cosine can be computed:

**Compute Sin(x):**

Now that Sin(A) & Cos(A) have been determined they can be combined with Sin(B) & Cos(B) to determine Sin(x). The B term is a function of n, which was earlier stated to be a multiple of 22.5 degrees of which there are 16 per revolution. Input x is has no theoretical bound but determining Sin(B) & Cos(B) can be simplified by normalizing n into a single revolution. With n having 16 steps per revolution the normalization is a simple bit mask.

With normalized n determined it can be used as an index into a switch case statement to determine Sin(x)

Switch (n\_nrml)

Case 0:

Sinx = Sin(A)Cos(22.5 \* 0) + Sin(22.5 \* 0)Cos(A)

Case 1:

Sinx = Sin(A)Cos(22.5 \* 1) + Sin(22.5 \* 1)Cos(A)

Case 2:

Sinx = Sin(A)Cos(22.5 \* 2) + Sin(22.5 \* 2)Cos(A)

Case 3:

Sinx = Sin(A)Cos(22.5 \* 3) + Sin(22.5 \* 3)Cos(A)

Case 4:

Sinx = Sin(A)Cos(22.5 \* 4) + Sin(22.5 \* 4)Cos(A)

Case 5:

Sinx = Sin(A)Cos(22.5 \* 5) + Sin(22.5 \* 5)Cos(A)

Case 6:

Sinx = Sin(A)Cos(22.5 \* 6) + Sin(22.5 \* 6)Cos(A)

Case 7:

Sinx = Sin(A)Cos(22.5 \* 7) + Sin(22.5 \* 7)Cos(A)

Case 8:

Sinx = Sin(A)Cos(22.5 \* 8) + Sin(22.5 \* 8)Cos(A)

Case 9:

Sinx = Sin(A)Cos(22.5 \* 9) + Sin(22.5 \* 9)Cos(A)

Case 10:

Sinx = Sin(A)Cos(22.5 \* 10) + Sin(22.5 \* 10)Cos(A)

Case 11:

Sinx = Sin(A)Cos(22.5 \* 11) + Sin(22.5 \* 11)Cos(A)

Case 12:

Sinx = Sin(A)Cos(22.5 \* 12) + Sin(22.5 \* 12)Cos(A)

Case 13:

Sinx = Sin(A)Cos(22.5 \* 13) + Sin(22.5 \* 13)Cos(A)

Case 14:

Sinx = Sin(A)Cos(22.5 \* 14) + Sin(22.5 \* 14)Cos(A)

Case 15:

Sinx = Sin(A)Cos(22.5 \* 15) + Sin(22.5 \* 15)Cos(A)

The Sin(B) & Cos(B) terms above can be simplified due to their symmetrical nature into 3 predefined constants:

**Cos(x) for Free (almost):**

The previous calculation computed Sin(x) as a function of Sin(A), Cos(A), Sin(B) & Cos(B). As it turns out cosine has a similar trig identity that uses the exact same terms.

The switch case can be modified to simultaneously be used to return the sine and cosine of the same angle at the same time:

Switch (n\_nrml)

Case 0:

Sinx = Sin(A)Cos(22.5 \* 0) + Sin(22.5 \* 0)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 0) + Sin(A) Sin(22.5 \* 0)

Case 1:

Sinx = Sin(A)Cos(22.5 \* 1) + Sin(22.5 \* 1)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 1) + Sin(A) Sin(22.5 \* 1)

Case 2:

Sinx = Sin(A)Cos(22.5 \* 2) + Sin(22.5 \* 2)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 2) + Sin(A) Sin(22.5 \* 2)

Case 3:

Sinx = Sin(A)Cos(22.5 \* 3) + Sin(22.5 \* 3)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 3) + Sin(A) Sin(22.5 \* 3)

Case 4:

Sinx = Sin(A)Cos(22.5 \* 4) + Sin(22.5 \* 4)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 4) + Sin(A) Sin(22.5 \* 4)

Case 5:

Sinx = Sin(A)Cos(22.5 \* 5) + Sin(22.5 \* 5)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 5) + Sin(A) Sin(22.5 \* 5)

Case 6:

Sinx = Sin(A)Cos(22.5 \* 6) + Sin(22.5 \* 6)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 6) + Sin(A) Sin(22.5 \* 6)

Case 7:

Sinx = Sin(A)Cos(22.5 \* 7) + Sin(22.5 \* 7)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 7) + Sin(A) Sin(22.5 \* 7)

Case 8:

Sinx = Sin(A)Cos(22.5 \* 8) + Sin(22.5 \* 8)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 8) + Sin(A) Sin(22.5 \* 8)

Case 9:

Sinx = Sin(A)Cos(22.5 \* 9) + Sin(22.5 \* 9)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 9) + Sin(A) Sin(22.5 \* 9)

Case 10:

Sinx = Sin(A)Cos(22.5 \* 10) + Sin(22.5 \* 10)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 10) + Sin(A) Sin(22.5 \*10)

Case 11:

Sinx = Sin(A)Cos(22.5 \* 11) + Sin(22.5 \* 11)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 11) + Sin(A) Sin(22.5 \* 11)

Case 12:

Sinx = Sin(A)Cos(22.5 \* 12) + Sin(22.5 \* 12)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 12) + Sin(A) Sin(22.5 \* 12)

Case 13:

Sinx = Sin(A)Cos(22.5 \* 13) + Sin(22.5 \* 13)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 13) + Sin(A) Sin(22.5 \* 13)

Case 14:

Sinx = Sin(A)Cos(22.5 \* 14) + Sin(22.5 \* 14)Cos(A)

Cosx = Cos(A)Cos(22.5 \* 14) + Sin(A) Sin(22.5 \* 14)

Case 15:

Sinx = Sin(A)Cos(22.5 \* 15) + Sin(22.5 \* 15)Cos(A)

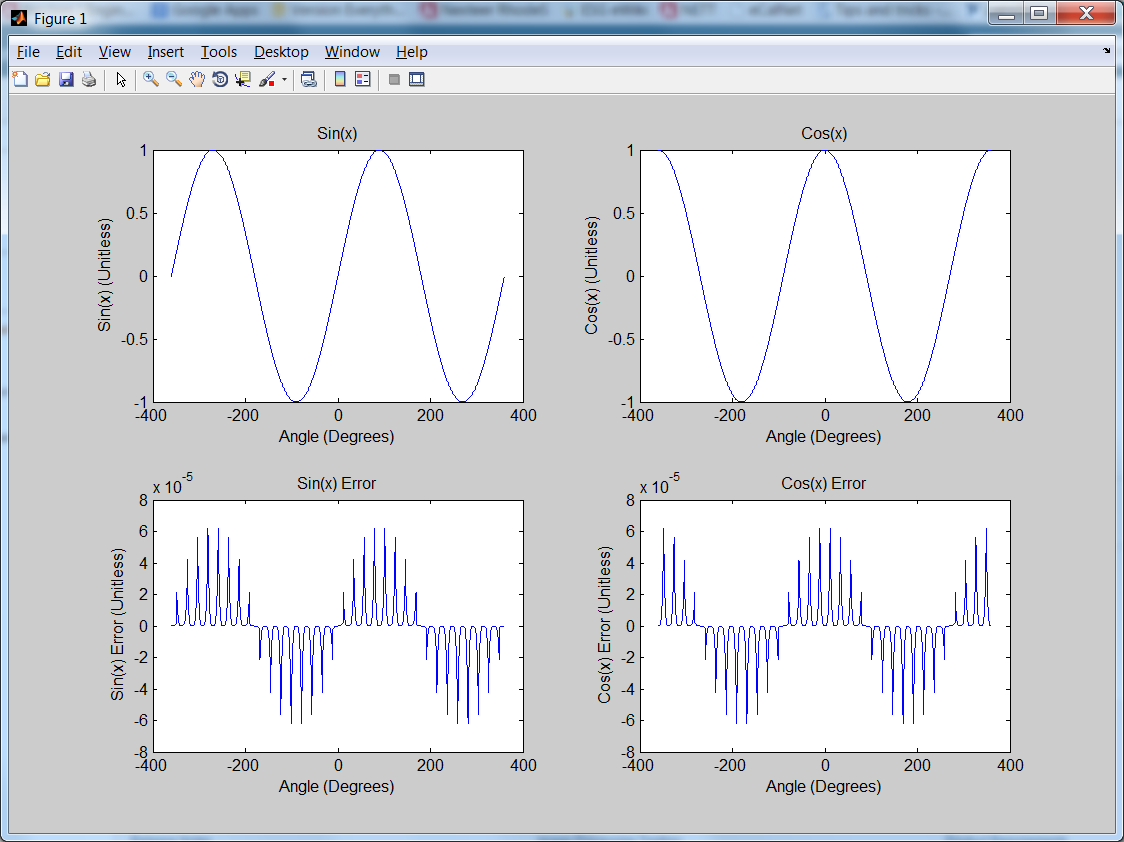
Cosx = Cos(A)Cos(22.5 \* 15) + Sin(A) Sin(22.5 \* 15)

1. **Performance:**

The following graph shows the output and error of the algorithm. The maximum errors for sine and cosine are:

SinErrorMax = 6.1799e-05

CosErrorMax = 6.1618e-05



1. **Matlab Code:**

%% Optimized SinCos Algorithm

ArraySize = uint16(100000);

PiOverEight = pi/8;

EightOverPi = 8/pi;

sin225 = sin(22.5/180\*pi);

sin450 = sin(45.0/180\*pi);

sin675 = sin(67.5/180\*pi);

x = linspace(-2\*pi, 2\*pi,ArraySize)';

SinX = zeros(ArraySize,1);

CosX = zeros(ArraySize,1);

n = int16(floor(x \* EightOverPi + 0.5));

A = (x - double(n) \* PiOverEight);

SinA = A - (A.^3)/6;%+(A.^5)/120;

CosA = 1 - (A.^2)/2;%+(A.^4)/24;

n\_nrml = bitand(n,15);

for i = 1:ArraySize

switch n\_nrml(i)

case 0

SinX(i) = SinA(i);

CosX(i) = CosA(i);

case 1

SinX(i) = sin675 \* SinA(i) + sin225 \* CosA(i);

CosX(i) = -sin225 \* SinA(i) + sin675 \* CosA(i);

case 2

SinX(i) = sin450 \* SinA(i) + sin450 \* CosA(i);

CosX(i) = -sin450 \* SinA(i) + sin450 \* CosA(i);

case 3

SinX(i) = sin225 \* SinA(i) + sin675 \* CosA(i);

CosX(i) = -sin675 \* SinA(i) + sin225 \* CosA(i);

case 4

SinX(i) = CosA(i);

CosX(i) = -SinA(i);

case 5

SinX(i) = -sin225 \* SinA(i) + sin675 \* CosA(i);

CosX(i) = -sin675 \* SinA(i) + -sin225 \* CosA(i);

case 6

SinX(i) = -sin450 \* SinA(i) + sin450 \* CosA(i);

CosX(i) = -sin450 \* SinA(i) + -sin450 \* CosA(i);

case 7

SinX(i) = -sin675 \* SinA(i) + sin225 \* CosA(i);

CosX(i) = -sin225 \* SinA(i) + -sin675 \* CosA(i);

case 8

SinX(i) = -SinA(i);

CosX(i) = -CosA(i);

case 9

SinX(i) = -sin675 \* SinA(i) + -sin225 \* CosA(i);

CosX(i) = sin225 \* SinA(i) + -sin675 \* CosA(i);

case 10

SinX(i) = -sin450 \* SinA(i) + -sin450 \* CosA(i);

CosX(i) = sin450 \* SinA(i) + -sin450 \* CosA(i);

case 11

SinX(i) = -sin225 \* SinA(i) + -sin675 \* CosA(i);

CosX(i) = sin675 \* SinA(i) + -sin225 \* CosA(i);

case 12

SinX(i) = -CosA(i);

CosX(i) = SinA(i);

case 13

SinX(i) = sin225 \* SinA(i) + -sin675 \* CosA(i);

CosX(i) = sin675 \* SinA(i) + sin225 \* CosA(i);

case 14

SinX(i) = sin450 \* SinA(i) + -sin450 \* CosA(i);

CosX(i) = sin450 \* SinA(i) + sin450 \* CosA(i);

case 15

SinX(i) = sin675 \* SinA(i) + -sin225 \* CosA(i);

CosX(i) = sin225 \* SinA(i) + sin675 \* CosA(i);

end

end

SinError = sin(x) - SinX;

CosError = cos(x) - CosX;

SinErrorMax = max(SinError)

CosErrorMax = max(CosError)

subplot(2,2,1) % first subplot

plot(x\*180/pi,SinX)

title('Sin(x)')

xlabel('Angle (Degrees)')

ylabel('Sin(x) (Unitless)')

subplot(2,2,2) % second subplot

plot(x\*180/pi,CosX)

title('Cos(x)')

xlabel('Angle (Degrees)')

ylabel('Cos(x) (Unitless)')

subplot(2,2,3) % third subplot

plot(x\*180/pi,SinError)

title('Sin(x) Error')

xlabel('Angle (Degrees)')

ylabel('Sin(x) Error (Unitless)')

subplot(2,2,4) % fourth subplot

plot(x\*180/pi,CosError)

title('Cos(x) Error')

xlabel('Angle (Degrees)')

ylabel('Cos(x) Error (Unitless)')