dx.doi.org/10.7437/NT2236-7640/2015.03.006
Notas Técnicas, v. 5, n. 3, p. 10–18, 2015

CORDIC: Repensando Senos e Cossenos

CORDIC: Rethinking sines and cosines

Ivana Mara Gomes Andersen Cavalcanti* e Bene Regis Figueiredo
Centro Brasileiro de Pesquisas Físicas - Rua Dr. Xavier Sigaud, 150 - Urca, Rio de Janeiro - RJ - Brasil, CEP - 22290-180

Edgar Monteiro da Silva
Centro Brasileiro de Pesquisas Físicas - Rua Dr. Xavier Sigaud, 150 - Urca, Rio de Janeiro - RJ - Brasil, CEP - 22290-180 e
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca - CEFET (Campus Maracanã) - Av. Maracanã, 229 - Maracanã – Rio de Janeiro - RJ - Brasil, CEP - 20271-110

Pablo Diniz Batista
Centro Brasileiro de Pesquisas Físicas - Rua Dr. Xavier Sigaud, 150 - Urca, Rio de Janeiro - RJ - Brasil, CEP - 22290-180

Submetido em 23/10/2015

Resumo: A proposta deste trabalho é apresentar em detalhes um algoritmo iterativo via método de rotação vetorial, que exige basicamente operações de deslocamento de *bit*, de adição e subtração para a realização de funções elementares básicas, conhecido como CORDIC. Tradicionalmente, diversos autores utilizam a abordagem matemática apresentada em 1959 por J. E. Volder [1]. Entretanto, será desenvolvida aqui, passo a passo, a ideia proposta no artigo para o cálculo de funções trigonométricas através de uma tabela de arco-tangentes sem a necessidade de operações de multiplicações e divisões. O algoritmo CORDIC possui grande potencial para utilização em hardware.
Palavras-chave: CORDIC, Linguagem C, rotação vetorial, microcontrolador.

Abstract: The purpose of this paper is to present in detail an iterative algorithm via vector rotation method, which basically requires bit shift operations, addition and subtraction to perform basic elementary functions, known as CORDIC. Traditionally, many authors use the mathematical approach presented in 1959 by J.E. Volder [1]. However, here it will be developed, step by step, the idea proposed in the article for the calculation of trigonometric functions using an arc-tangent table without the need of multiplication and division operations. The CORDIC algorithm has great potential for use in hardware.

Keywords: CORDIC, C Language, rotation vector, microcontroller.

1. INTRODUÇÃO

CORDIC significa COordinate Rotation DIgital Computer (Computador Digital para Rotação de Coordenadas), conhecido também como "Algoritmo de Volder"[1]. É um algoritmo simples e eficiente para o cálculo de funções hiperbólicas e trigonométricas usando apenas operações de adição, subtração, deslocamento binário e Look-Up Table (LUT) [2]. O CORDIC se faz útil quando nenhum hardware multiplicador está disponível, como por exemplo, um chip básico de FPGA (Field Programmable Gate Array). O algoritmo CORDIC foi descrito pela primeira vez em 1959 por Jack E. Volder [1], no Departamento Aeroeletrônico de Convair, uma divisão da General Dynamics para substituir o computador analógico do sistema de navegação do bombardeiro B-58 por um computador digital. Originalmente, o CORDIC foi implementado usando o sistema binário. Em 1971, John Stephen Walther na Hewlett-Packard, generalizou o CORDIC para que este pudesse calcular funções hiperbólicas, exponenciais e logarítmicas, multiplicações e divisões (via deslocamento de bits) [3, 4]. O algoritmo foi amplamente utilizado em calculadoras de bolso, pela facilidade de implementação de diversas funções matemáticas em chips sem muitos recursos de hardware.

A popularidade do CORDIC se deve ao seu eficiente potencial de aplicação e baixo custo na geração de funções trigonométricas e logarítmicas, multiplicação de números complexos, inversão de matrizes, processamento de sinais e imagens [5]. Como exemplos de aplicações populares temos: (1) cálculo da cinemática direta e inversa de robôs manipuladores [5, 6]; (2) rotação de vetor no plano tridimensional para gráficos e animações [5] e (3) codificação da fala [7]. Embora o CORDIC não seja um algoritmo rápido para executar as operações descritas, o seu uso se deve à sua simplicidade de implementação em hardware [8–10]. A área de Processamento Digital de Sinais necessita de algoritmos versáteis e rápidos no tratamento de sinais (uso de circuitos integrados de aplicação específica – ASIC), sendo que o CORDIC preenche esses requisitos descritos e vem sendo amplamente utilizado em operações matemáticas de um DSP

*Electronic address: ivana@cbpf.br

(Digital Signal Processor). O algoritmo também pode ser utilizado em qualquer outra aplicação que requeira o processamento de sinais em tempo real, como controle e automação de dispositivos [11–13].

Uma limitação apresentada pelo algoritmo CORDIC é sua latência de cálculo; todavia isso pode ser minimizado transformando-se o algoritmo computacional em segmentos independentes e para implementar esses segmentos individuais usa-se diferentes processadores CORDIC [14].

Na seção *II* deste trabalho será apresentada uma análise matemática do CORDIC, passo a passo, com o método de pseudo-rotações [1] de um vetor num plano bidimensional. A seção *III* consiste na implementação do algoritmo desenvolvido em linguagem C em computadores. A seção *IV* mostra a implementação das funções seno e cosseno em um microcontrolador desenvolvido no Laboratório de Eletrônica e Processamento de Sinais. Finalmente, as seções seguintes apresentam uma análise das funções trigonométricas, seno e cosseno e as conclusões desde estudo.

2. O ALGORITMO CORDIC

2.1. Pense em um número

O primeiro passo consiste em um simples exercício de adivinhação. Pense em um ângulo X contido entre 0^o e 90^o. Como determinar X? A condição inicial a partir da qual o problema será abordado, consiste na premissa de que o número X deve estar entre o intervalo A e B. Neste caso $A = 0^o$ e $B = 90^o$. A Figura 1 apresenta o fluxograma de um algoritmo capaz de determinar esse número a partir de uma única pergunta: X é menor do que Y? A partir da resposta, o algoritmo é capaz de restringir o intervalo pela metade (aqui entra o conceito de pseudo-rotação). As condições iniciais são: $S = 45^o$ e $Y = 45^o$. Portanto, se $X < 45^o$ então, o novo intervalo será modificado para $A = 0^o$ e $B = 45^o$ ($0^o < X < 45^o$); caso contrário, teremos $A = 45^o$ e $B = 90^o$ ($45^o < X < 90^o$). Nota-se que as variáveis A e B não são utilizadas explicitamente no algoritmo, porém, a pergunta é reapresentada considerando $Y = 22,5^o$ ou $Y = 67,5^o$ a depender da primeira resposta. Este processo pode ser repetido até que o intervalo seja suficientemente pequeno, de tal forma que o número seja o próprio intervalo.

A cada iteração, Y se aproxima de X, de tal modo que após N iterações o erro esteja na terceira casa decimal. Para que isso aconteça, o valor de S é adicionado ou subtraído a Y, a depender se X é menor ou maior do que Y, respectivamente. Além disso, para que Y convirja para X é necessário que S diminua a cada iteração, isto é, S é dividido pela metade depois de cada teste de condição. A Tabela 2.1 apresenta os valores dessas variáveis supondo X igual a 84 para N inteiro variando de 1 até 15.

Observe que após 15 iterações o algoritmo determina o valor de X com uma precisão na segunda casa decimal. A

[1] O nome pseudo-rotações vem da ideia de que os vetores não realizam rotações completas, mas sofrem ajustes de ângulos como descrito na subseção *A* da seção *II* desta Nota Técnica.

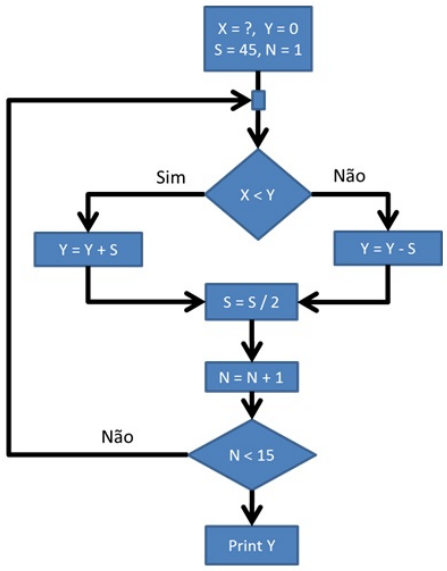

Figura 1: Fluxograma para determinar o valor de X.

N	Y	S
01	0.00	45.00
02	45.00	22.50
03	67.50	11.25
04	78.75	5.62
05	84.38	2.81
06	81.56	1.41
07	82.97	0.70
08	83.67	0.35
09	84.02	0.18
10	83.85	0.09
11	83.94	0.04
12	83.98	0.02
13	84.00	0.01
14	83.99	0.01
15	**84.00**	**0.00**

Tabela I: Dinâmica dos valores de variáveis para N de 1 até 15.

Figura 2 apresenta uma elaboração deste algoritmo em linguagem C.

A Figura 3 apresenta a tela de saída do programa considerando X igual a 84. A única diferença dessa implementação é a presença da variável E utilizada para armazenar o erro absoluto entre X e Y para cada iteração.

A partir desse ponto, seria interessante determinar o valor de N (número de iterações) necessário para que o erro entre X e Y seja sempre menor que um Erro Absoluto para todos os valores de X entre 0^o e 90^o. Neste caso, o Erro Absoluto pode ser considerado uma variável de entrada como mostra a Figura 4. Os resultados mostram que o número de iterações aumenta à medida em que o erro absoluto diminui, pois é necessário um maior número de iterações para que Y se aproxime de X.

```c
#include <stdio.h>
#include <conio.h>
#include <math.h>
//------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    int   i;
    float E;
    float S = 45.0;
    float X = 84.0;
    float Y = 0.0;

    printf("\n\n\ti\t Y\t S \t Erro \n\n");
    for ( i = 0; i < 15; i++)
    {
          E = fabs( X - Y);
          printf("\t %d\t %3.2f\t %2.2f\t %2.5f\n",i,Y,S,E);
          if ( Y < X )
               Y = Y + S;
          else
               Y = Y - S;
          S = S / 2.0;
    }
    getch();
    return 0;
}
```

Figura 2: Elaboração do algoritmo em linguagem C.

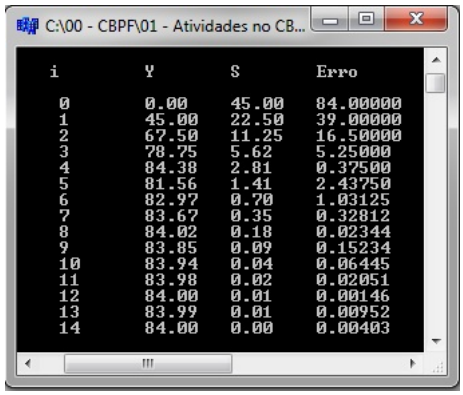

Figura 3: Tela de saída do programa.

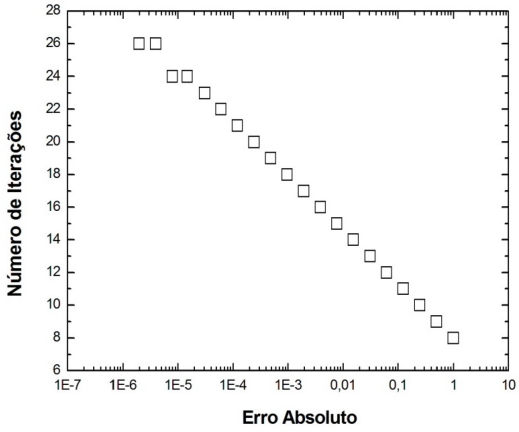

Figura 4: Gráfico da relação entre o erro absoluto das medidas e o número de iterações.

2.2. Representação em coordenadas polares

É possível utilizar a mesma ideia do fluxograma apresentado na Figura 1 para determinar o seno e o cosseno de um ângulo X. O primeiro passo consiste em representar os ângulos X e Y em coordenadas polares, como mostra a Figura 5. Nesse caso, sabe-se que a projeção dos dois vetores A e B nos eixos x e y correspondem aos valores de seno e cosseno, respectivamente, como mostra as equações

$$x' = x\cos\phi - y\sin\phi \quad (1)$$

$$y' = x\sin\phi + y\cos\phi \quad (2)$$

O grande lance do CORDIC é compreender que quando operamos em Y estamos também rotacionando o vetor B no plano de coordenadas polares. O algoritmo deve ser modificado de tal maneira que a cada iteração, as projeções de Y nos eixos x e y serão calculadas. Portanto, os valores de seno e cosseno de X serão obtidos após N iterações, considerando que Y se aproxima de X.

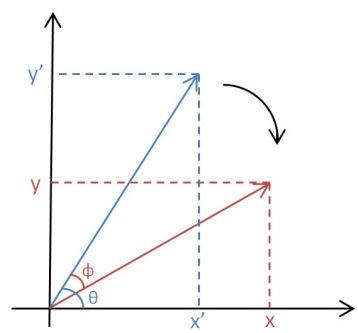

Figura 5: Rotação de vetores de um ângulo X.

As representações das projeções do vetor B nos eixos x e y, são dadas por x' e y', portanto, o seno e o cosseno do ângulo em questão. O algoritmo além de introduzir a ideia de rotação de vetores, propõe também um método eficiente para calcular a matriz de rotação representada pela equação 2, utilizando apenas operações de soma, subtração e deslocamento de bits. Essa discussão ocorrerá na próxima seção.

2.3. Implementação Matemática

Através de rotações de um vetor é possível calcular funções trigonométricas. No caso do método do Cordic, são realizadas pseudo-rotações de um vetor em ângulos arbitrários inteiros utilizando-se somente somas, subtrações e deslocamentos. O algoritmo é derivado das equações gerais de rotação de um vetor com coordenadas (x,y) por um ângulo ϕ qualquer, tal como na representação da Figura 5 [16, 17].

Realizando as análises trigonométricas para os ângulos θ e ϕ, teremos:

$$sin\theta = \frac{y'}{h}$$
$$cos\theta = \frac{x'}{h}$$

$$y' = hsin\theta \quad (3)$$
$$x' = hcos\theta \quad (4)$$

$$y = hsin(\theta - \phi) = h(sin\theta cos\phi - sin\phi cos\theta)$$
$$x = hcos(\theta - \phi) = h(cos\theta cos\phi + sin\theta sin\phi)$$

$$y = hsin\theta cos\phi - hsin\phi cos\theta \quad (5)$$
$$x = hcos\phi cos\theta + hsin\phi sin\theta \quad (6)$$

Substituindo 3 e 4 em 5 e 6, respectivamente, teremos:

$$y = y'cos\phi - x'sin\phi \quad (7)$$
$$x = x'cos\phi + y'sin\phi \quad (8)$$

Colocando y' na equação 7 em evidência,

$$y' = x'\frac{sin\phi}{cos\phi} + y\frac{1}{cos\phi} \quad (9)$$

e substituindo este resultado em 8,

$$x = x'cos\phi + \left(x'\frac{sin\phi}{cos\phi}sin\phi + \frac{1}{cos\phi}ysin\phi\right)$$
$$x = \frac{x'cos^2\phi + x'sin^2\phi + ysin\phi}{cos\phi}$$

Evidenciando x',

$$x' = xcos\phi - ysin\phi \quad (10)$$

Substituindo o resultado da equação 10 em 9,

$$y' = \frac{sin\phi}{cos\phi}(xcos\phi - ysin\phi) + \frac{1}{cos\phi}y$$
$$y' = xsin\phi - y\frac{sin^2\phi}{cos\phi} + y\frac{1}{cos\phi}$$
$$y' = xsin\phi + \frac{y}{cos\phi}(1 - sin^2\phi)$$

Portanto,

$$y' = xsin\phi + ycos\phi \quad (11)$$

Colocando o $cos\phi$ em evidência nas equações 10 e 11, podemos expressar a rotação vetorial por:

$$x' = cos\phi(x - ytg\phi)$$

e

$$y' = cos\phi(y + xtg\phi)$$

Que gira um vetor em um plano cartesiano pelo ângulo ϕ.

Uma grande observação do CORDIC é restringir o ângulo de rotação [16] a

$$tg\phi = \pm 2^{-i} \quad (12)$$

de tal forma que serão gerados somente valores numéricos na base binária.

Teremos que,

$$x' = cos(tg^{-1}2^{-i})(x - y2^{-i}) \quad (13)$$
$$y' = cos(tg^{-1}2^{-i})(y + x2^{-i}) \quad (14)$$

Se os ângulos de rotação são limitados à equação 12, a multiplicação por $tg\phi$ será substituída por uma operação de deslocamento (*shift*), chamada *operação bit-a-bit*. A vantagem da operação *bit-a-bit* é que ela pode ser implementada facilmente em hardware. Além disso, cada uma delas usa apenas um clock do processador, sendo altamente eficiente [18].

Fazendo-se pequenas rotações iterativas, é possível girar um vetor por um ângulo arbitrário qualquer, tomando *i* como a possibilidade de girar ou não o vetor. Assim, a rotação iterativa é definida como

$$x_{i+1} = K_i(x_i - y_i d_i 2^{-i}) \quad (15)$$
$$y_{i+1} = K_i(y_i + x_i d_i 2^{-i}) \quad (16)$$

Onde $d = \pm 1$ a depender se a rotação será pra "+" ou para −". Se d for $+1$, o vetor girará no sentido anti-horário, enquanto se d for -1, o vetor irá girar no sentido horário (tem-se aqui a mesma ideia da subseção *A* da seção *II*). Temos ainda que

$$K_i = cos(tg^{-1}2^{-i}) = \frac{1}{\sqrt{1 + 2^{-2i}}} \quad (17)$$

O tratamento do fator K_i pode ser considerado como ganho de processamento. Esse valor é de aproximadamente $0,6073$ quando o número de iterações tende ao infinito. Este fator é obtido através de identidades trigonométricas básicas:

$$cos^2\theta + sin^2\theta = 1 \quad (18)$$

e

$$tg\theta = \frac{sin\theta}{cos\theta}$$
$$\therefore sin\theta = cos\theta tg\theta$$
$$cos^2\theta + cos^2\theta tg^2\theta = 1$$
$$cos^2\theta = \frac{1}{1+tg^2\theta}$$

Fazendo $\theta = tg^{-1}2^{-i}$, teremos:

$$cos^2(tg^{-1}2^{-i}) = \frac{1}{1+tg^2(tg^{-1}2^{-i})}$$
$$= \frac{1}{1+2^{-2i}}$$
$$cos(tg^{-1}2^{-i}) = \frac{1}{\sqrt{1+2^{-2i}}}$$

O produto de todos os termos de K_i de $[0,n]$ pode ser representado por K_n,

$$K_n = \prod_{i=0}^{n} \frac{1}{\sqrt{1+2^{-2i}}} \quad (19)$$

Os valores de K podem ser calculados à priori e armazenados em uma tabela.

O algoritmo de rotação tem um ganho de A_n de aproximadamente $1,647$. O valor exato do ganho depende do número de iterações que por sua vez, depende da equação

$$A_n = \prod_{i=0}^{n} \sqrt{1+2^{-2i}} \quad (20)$$

O ângulo de rotação composto é definido exclusivamente pela sequência das direções das rotações elementares. Essa sequência pode ser representada por um vetor de decisão [19]. O conjunto de todos os possíveis vetores de decisão é um sistema de medição angular com base em arcos-tangentes binários. No algoritmo, usa-se um somador-subtrator adicional que acumula os ângulos de rotação elementares em cada iteração. Os ângulos elementares podem ser expressos em qualquer unidade angular conveniente. Esses valores angulares são fornecidos por uma *Look-Up* Table (uma entrada por cada iteração) ou são diretamente conectados, dependendo da implementação. Este ângulo acumulador adiciona uma terceira equação para compor o ângulo de rotação, z_{i+1}, que é dada por:

$$z_{i+1} = z_i - d_i tg^{-1}2^{-i} \quad (21)$$

O algoritmo CORDIC pode ser empregado em dois modos de operação: Vetorial e Rotacional (ver Figura 6).

No modo rotacional, o vetor de entrada é girado por um dado ângulo de entrada z_0. O sinal de d_i determina o sentido de rotação em cada iteração, de tal forma que o seu valor absoluto z_i é diminuído após cada iteração. Resumindo: as entradas são o vetor $\vec{r} = (x,y)$ e o ângulo de giro θ e a saída

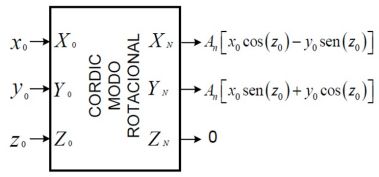

Figura 6: Somador-subtrator adicional que acumula os ângulos de rotação elementares em cada iteração.

é o vetor girado com novas coordenadas $\vec{r'} = (x',y')$. Desta forma, o vetor de entrada gira um ângulo especifico, que é introduzido como parâmetro.

O cálculo da função seno e cosseno utiliza o modo rotacional. A Figura 7 mostra isso em forma de bloco [2].

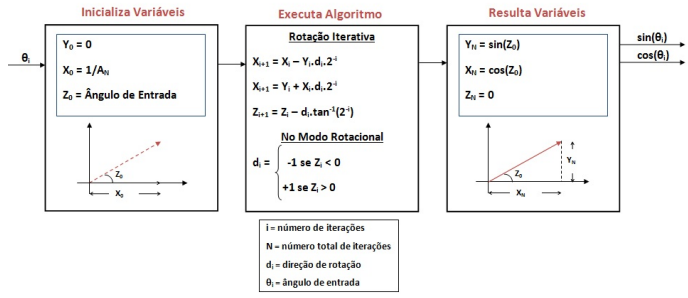

Figura 7: Função seno e cosseno.

Usando-se as equações do algoritmo, obtém-se rotações para ângulos menores que 90^o. Para atingir rotações maiores, deve-se realizar primeiro uma rotação de $\pm 90^o$ (garantindo que o vetor fique apenas no 1^o ou 4^o quadrante). Portanto, o algoritmo tem duas etapas: iniciação com rotação por $\pm 90^o$ e pseudo-rotações iterativas que buscam levar a zero a variável y_i. O vetor aumenta sua amplitude a cada iteração pelo valor de A_n.

Resumindo as equações principais, teremos:

Para $d = -1$, $z_i < 0$

$$x_{i+1} = x_i + y_i 2^{-i}$$
$$y_{i+1} = y_i - x_i 2^{-i}$$
$$z_{i+1} = z_i + tg^{-1}2^{-i}$$

Para $d = 1$, $z_i > 0$

$$x_{i+1} = x_i - y_i 2^{-i}$$
$$y_{i+1} = y_i + x_i 2^{-i}$$
$$z_{i+1} = z_i - tg^{-1}2^{-i}$$

2.4. Implementação em Software - Linguagem C

Como primeiro passo de estudo do algoritmo CORDIC, foi desenvolvido um programa em Linguagem C para computadores usuais implementando o algoritmo CORDIC. Este

[2] ©2007 Microchip Technology Inc. DS01061A.

Tabela de arcotangentes

i	$\tan(2^{-i})$	$\theta = \arctan(2^{-i})$	θ em graus	θ em binário
0	1,000000000	0,785398163	45	0,1100100
1	0,500000000	0,463647609	26,56505118	0,0111011
2	0,250000000	0,244978663	14,03624347	0,0011111
3	0,125000000	0,124354995	7,125016349	0,0001111
4	0,062500000	0,062418810	3,576334375	0,0000111
5	0,031250000	0,031239833	1,789910608	0,0000011
6	0,015625000	0,015623729	0,89517371	0,0000001

Tabela II: Exemplo de tabela pré-calculada com os valores das Arcotangentes utilizadas no algoritmo CORDIC, aqui utilizando 8 bits de casas decimais

procedimento mostra-se vantajoso, pois facilita o entendimento do algoritmo. A Linguagem C foi utilizada por ser uma linguagem de programação compilada de propósito geral, estruturada, iterativa e procedimental. A vantagem de utilizar Linguagem C é que é simples, possui um conjunto de bibliotecas de rotinas padronizadas, uso de linguagem de pré-processamento, ponteiros, acesso de baixo-nível através de inclusões de código Assembly, estruturas de variáveis que permitem que dados relacionados sejam combinados e manipulados como um todo, além de existirem poucas arquiteturas onde não há a possibilidade da utilização desta ampla linguagem [20].

Um fluxograma foi elaborado com o algoritmo implementado em linguagem C (Figura 8) utilizando-se uma LUT com valores de acordo com a Tabela 2.4.

No programa desenvolvido neste projeto, foi criada uma LUT para valores de arcotangentes e $K's$ para diversos ângulos inteiros gerados aleatoriamente via comandos básicos. Para facilitar o método de rotação de ângulos, caso um ângulo estivesse no 3º, 4º ou 5º quadrante, os mesmos eram reduzidos ao 1º. O fluxograma da Figura 8 representa bem o processo. A Figura 9 mostra a rotina desenvolvida para computadores.

A popularidade da linguagem C encorajou a criação de tecnologias para ferramentas como bibliotecas, compiladores e sistemas operacionais; ainda oferece um ambiente de desenvolvimento de aplicações em microcontroladores, utilizando-se toda sua potencialidade, de forma que estes tornaram-se cada vez mais adaptáveis à utilização em projetos como será explanado na seção seguinte.

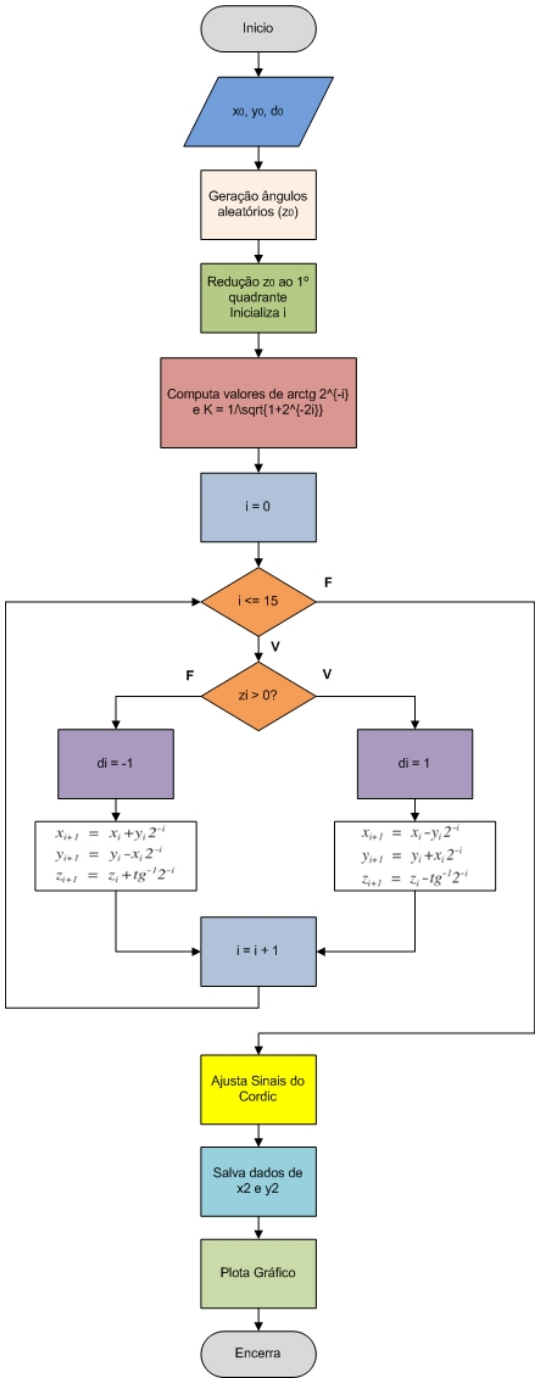

Figura 8: O fluxograma do algoritmo CORDIC mostra passo-a-passo o processo iterativo do método de rotação. Através de uma LUT de arco-tangentes para ângulos gerados no programa em C, é possível através de adições e subtrações, calcular valores de seno e cosseno para todos os ângulos da tabela, com melhor aproximação a depender do número de passos iterativos.

2.5. Implementação em Hardware - Microcontrolador

Como segunda etapa deste projeto, será utilizado o sistema de controle e aquisição de dados [24] (ver Figura 10) desenvolvido recentemente no Laboratório de Eletrônica e Processamento de Sinal do CBPF [3] para testar a implementação do algoritmo. É importante destacar que este circuito eletrônico contém o microcontrolador $PIC18F45K50$, uma porta USB para a comunicação com periféricos externos e um conversor analógico digital a partir do $MCP4822$. Além disso, o programa será interpretado por um processador de 8 bits a uma velocidade de 16 MHz.

O código desenvolvido em linguagem C para a função seno e cosseno a partir do CORDIC está apresentado na Figura 11.

[3] O projeto e o desenvolvimento deste hardware faz parte de uma dissertação de mestrado no CBPF e será apresentado em detalhes em trabalhos futuros.

```
void Cordic (int x1, int y1, int ang_aux, int &ang_out,
             int &s_cord, int &c_cord)
{
    int i;
    int x2, y2;

        for ( i = 0; i < IMAX + 1; i ++ )
        {
            if ( ang_aux < 0 )
            {
                x2 = x1 + (y1 >> i);
                y2 = y1 - (x1 >> i);
                ang_out = ang_aux + Arctg[i];
            }
            else
            {
                x2 = x1 - (y1 >> i);
                y2 = y1 + (x1 >> i);
                ang_out = ang_aux - Arctg[i];
            }
                x1 = x2;
                y1 = y2;
                ang_aux = ang_out;
        }

    c_cord = x2;
    s_cord = y2;
}
```

Figura 9: Rotina desenvolvida em Linguagem C para computadores.

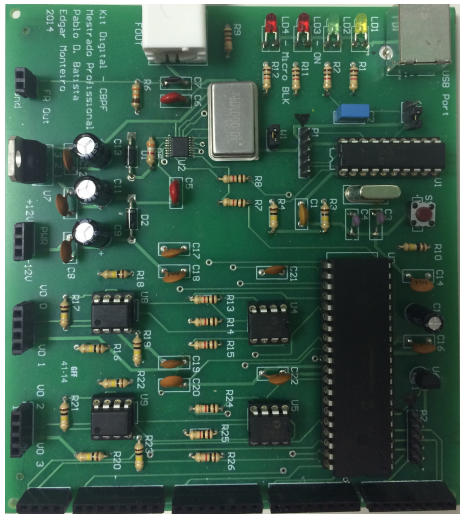

Figura 10: Foto do hardware desenvolvido para controle e aquisição de dados.

3. RESULTADOS

A Figura 12 apresenta os dados obtidos com o osciloscópio digital *TDS*2024*C* utilizado para medir o tempo necessário para que a função CORDIC seja executada no *PIC*18*F*45*K*20. O osciloscópio monitora a tensão em um dos pinos da porta usado para indicar que a função foi executada por completo. Entretanto, para reduzir o erro relacionado à medida optamos por determinar o tempo necessário para que a função fosse executada 10 vezes.

A partir dos resultados apresentados podemos concluir que a função CORDIC necessita de 0.81 ms para retornar o valor do seno e do cosseno de um ângulo qualquer. Esse tempo ainda poderia ser reduzido caso a função fosse escrita em Linguagem Assembly Além disso, essa análise é

```
void  Sin_Cos_Cordic( int theta)
{
      int   ATAN[] = { 4500,2656,1403,712,357,178,89,44,22,11};
      int   i, alfa = 0, t, tx, ty, x = 6073,y = 0, x_i, y_i;

      for ( i = 0; i < 10; i ++ )
      {
            tx = x;  ty = y;
            if ( alfa > theta )
            {
                  alfa -=  ATAN[i];
                  if ( y < 0 ){
                        ty  = (ty & 0b01111111) >> i;
                        ty  =  ty & 0b11111111;
                        x_i =  x + ty;
                  }
                  else
                        x_i   =  (x + (y >> i) );

                  if ( x < 0 ){
                        tx = (tx & 0b01111111) >> i;
                        tx =  tx & 0b11111111;
                        y_i = y - tx;
                  }
                  else
                        y_i =  (y - (x >> i) );
            }
            else
            {
                  alfa +=  ATAN[i];
                  if ( y < 0 ) {
                        ty  = (ty & 0b01111111) >> i;
                        ty  = ty & 0b11111111;
                        x_i = x - ty;
                  }
                  else
                      x_i   =  (x - (y >> i) );

                  if ( x < 0 ){
                        tx  = (tx & 0b01111111) >> i;
                        tx  = tx & 0b11111111;
                        y_i = y - tx;
                  }
                  else
                     y_i   =  (y + (x >> i) );
            }
            x = x_i;                y = y_i;
      }
      _a_sin = y;
      _b_cos = x;
}
```

Figura 11: Código do CORDIC para microcontrolador de 8 bits.

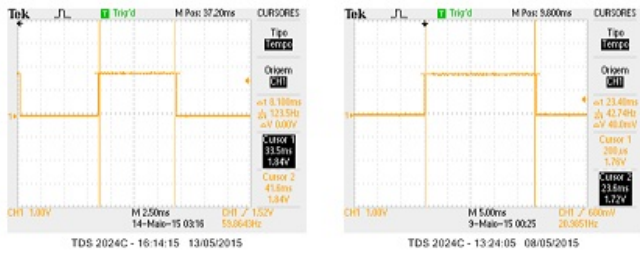

Figura 12: Medida de tempo para o calculo do seno e do cosseno no *PIC*18*F*45*K*20 executando o programa a 16 MHz. A função é executada 10 vezes.

repetida utilizando agora as funções disponibilizadas pela MICROCHIP através da biblioteca *math* resultando em um tempo de execução de 2, 34 ms.

Para finalizar os testes da implementação do CORDIC em hardware a Figura 13 apresenta a tensão em função do tempo nas saídas do conversor analógico digital presente no hardware. Para isso, as tensões são moduladas usando as funções

seno e cosseno desenvolvidas a partir do CORDIC. A partir dos resultados observamos que os sinais apresentam o comportamento esperado. Entretanto, como os dois canais de tensão são acessados sequencialmente pelo microcontrolador não é possível observar uma defasagem de 90º entre os sinais.

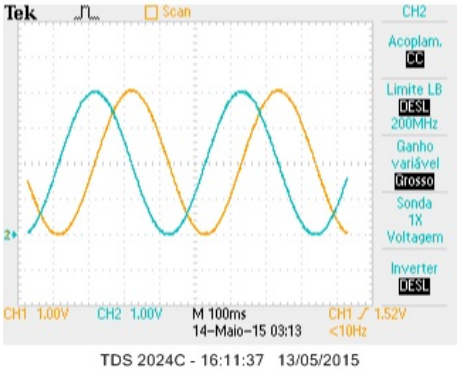

Figura 13: Medida da tensão em função do tempo nas saídas do DAC presente no hardware. Note que a tensão é modulada de acordo com as funções seno e cosseno desenvolvida a partir do CORDIC.

4. CONCLUSÕES

Este trabalho apresentou a implementação de duas principais funções trigonométricas, seno e cosseno para ângulos inteiros, através de um algoritmo de rotação vetorial conhecido como CORDIC. O algoritmo foi implementado usando apenas operações básicas de programação, tais como: adição/subtração, deslocamento de *bits* e Look-Up Table. Como o CORDIC apresenta a característica de ser mais rápido que outros algoritmos na ausência de um hardware de multiplicação/divisão, foi desenvolvida uma implementação deste algoritmo para a realização do cálculo de funções senos e cossenos em um microcontrolador.

5. AGRADECIMENTOS

A realização do trabalho que conduziu a esta Nota Técnica apenas foi possível devido à colaboração de um conjunto de pessoas que deram suas contribuições em diferentes etapas do projeto. A ideia de estudar o tema aqui apresentado surgiu da realização de um projeto na disciplina de Linguagem de Programação oferecida no CBPF para o Mestrado Profissional em Instrumentação Científica com Ênfase em Física, ministrada pelo prof. Dr. Pablo Diniz Batista.

[1] Volder J.E. The CORDIC trigonometric computing technique. IRE Transactions on Electronic Computers 8 (3): 330–334, retrieved 2009-06-02 **(1959)**.

[2] Volder J.E. The Birth of CORDIC. Journal of VLSI Signal Processing 25, 101–105 **(2000)**.

[3] Walther J. Stephen. The Story of Unified CORDIC. Journal of VLSI Signal Processing 25, 107–112 **(2000)**.

[4] Pramod K., Valls J., Juang, Sridharan K. and Maharatna K. 50 Years of CORDIC: Algorithms, Architectures and Applications. IEEE Trasactions on Circuit and Systems - I: Regular Papers, vol. 56, nº9, September **(2009)**.

[5] Pramod K. and Park S. CORDIC Designs for Fixed Angle of Rotation. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, nº2, February **(2013)**.

[6] Choudhary P., Karmakar A. CORDIC based implementation of Fast Fourier Transform, Computer and Communication Technology (ICCCT), 2011 2nd International Conference on , vol., no., pp.550,555, 15-17 Sept. **(2011)**.

[7] Valls J., Sansaloni T., Pérez-Pascual A., Torres V., and Almenar V. The Use of CORDIC in Software Defined Radios: A Tutorial. IEEE Communications Magazine. September **(2006)**.

[8] Meggitt J. Pseudo Division and Pseudo Multiplication Processes. IBM Journal, pp. 210–226, April **(1962)**.

[9] Das B., Banerjee S. Unified CORDIC-based chip to realise DFT/DHT/DCT/DST, Computers and Digital Techniques, IEE Proceedings - vol.149, no.4, pp.121,127, Jul **(2002)**.

[10] Hsiao S., Hu Y. and Juang T. A memory-efficient and high-speed sine/cosine generator based on parallel CORDIC rotations. IEEE Signal Process. Lett., vol. 11, no. 2, **(2004)**.

[11] Nakagawa T., Nosaka H. Direct Digital Synthesizer with Interpolation Circuits. IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 32, NO. 5, MAY **(1997)**.

[12] Vankka J. Methods of Mapping from Phase to sino Amplitude in Direct Digital Synthesis. 1996 IEEE International Frequency Control Symposium, **(1996)**.

[13] Vankka J. A Direct Digital Synthesizer with a Tunable Error Feedback Structure. IEEE Transactions on Communications, vol. 45, nº. 4, April **(1997)**.

[14] Wu C., Wu A. and Lin C. A high-performance/low-latency vector rotational CORDIC architecture based on extended elementary angle set and trellis-based searching schemes," IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 50, no. 9, pp. 589–601, Sep. **(2003)**.

[15] Bhuria S. and Muralidhar P. FPGA Implementation of Sine and Cosine Value Generators using Cordic Algorithm for Satellite Attitude Determination and Calculators. IEEE, **(2010)**.

[16] Benavides J. Efficient Fixed-Point Trigonometry Using CORDIC Functions For PIC16F. Microchip Technology Inc. **(2007)**.

[17] Hwang K. Computer Arithmetic Principles: Architecture and Design. New York: Wiley, **(1979)**.

[18] Ravichandran S. and Asari V, Implementation of unidirectional CORDIC algorithm using precomputed rotation bits. in 45th Midwest Symp. on Circuits Syst., 2002. MWSCAS 2002, vol. 3, pp. 453–456, Aug. **(2002)**.

[19] Muller J. Elementary Functions: Algorithms and Implementation. Boston, MA: Birkhauser Boston, **(2006)**.

[20] Schildt H. Borland C++: Todo o poder do Borland C++. Completo e Total. Versão 5. Makorn Books **(1998)**.

[21] Lee J. and Lang T. Constant-factor redundant CORDIC for angle calculation and rotation. IEEE Trans. Computers. vol. 41, no. 8, pp.1016–1025, Aug. **(1992)**.

[22] Hsiao S. and Delosme J. Householder CORDIC algorithms". IEEE Trans. Computers, vol. 44, no. 8, pp. 990–1001, Aug. **(1995)**.

[23] Vachhani L., Sridharan K. and Meher P. Efficient CORDIC

algorithms and architectures for low area and high throughput implementation. IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 56, no. 1, pp.61–65, Jan. **(2009)**.

[24] Monteiro E. Dissertação de Mestrado: Sistema eletrônico para a produção e caracterização de sensores de pH baseado em dispositivos semicondutores do tipo EGFET. CBPF. Rio de Janeiro **(2015)**.

[25] Giraldeli F. Programação em C. Instituto Federal do Espírito Santo. V20.0.

[26] Disponível em: Cplusplus (http:www.cplusplus.com).

[27] Disponível em: Digital Circuits/CORDIC (*http://en.wikibooks.org/wiki/Digital_Circuits/CORDIC*). Acessado em: 13 de Setembro de 2014.

[28] Disponível em: Ponteiros (*http://pt.wikibooks.org/wiki/Programar_em_C/Ponteiros*). Acessado em 08 de Dezembro de 2014.