

به نام خدا



طراحی سیستم‌های دیجیتال

پروژه پایانی - CORDIC

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

استاد:

جناب آقای دکتر بهاروند

---

نام، نام خانوادگی و شماره دانشجویی اعضای گروه:

علیرضا ایلامی - ۹۷۱۰۱۲۸۶

محمدمتین فتوحی - ۹۷۱۰۶۱۴۳

سید مهدی فقیه - ۹۷۱۰۶۱۹۸

علی قاسمی - ۹۷۱۰۶۲۰۵

امیرمهدی نامجو - ۹۷۱۰۷۲۱۲

محمدرضا یوسف پور - ۹۷۱۰۶۳۲۴

## فهرست مطالب

۱	مقدمه	۲
۱.۱	معرفی اجمالی و تاریخچه	۲
۲.۱	هدف الگوریتم	۲
۳.۱	پایه ریاضی	۳
۱.۳.۱	حالت Rotation	۳
۲.۳.۱	حالت Vectoring	۶
۴.۱	شکل دقیق الگوریتم	۸
۵.۱	کاربردها	۱۰
۱.۵.۱	سخت افزار	۱۰
۲.۵.۱	نرم افزار	۱۰
۱۱	مراجع	

## ۱ مقدمه

### ۱.۱ معرفی اجمالی و تاریخچه

هدف اصلی ما در این پروژه، طراحی یک واحد CORDIC است. CORDIC مخفف واژه COordinate Rotation DIgital Computer به معنی کامپیوتر دیجیتال چرخش مختصاتی است. این الگوریتم اولین بار در سال ۱۹۵۶ توسط آقای جک ولدر (Jack E. Volder)، که یک مهندس اویونیک (مهندس الکترونیک مرتبط به صنعت هوانوردی) بود، برای سیستم مسیریابی بمب افکن B-۵۸ ابداع شد. هر چند بعضی از ایده‌های استفاده شده در این روش، حتی در قرن هفدهم ذکر شده بودند. ویژگی اصلی این الگوریتم هم این است که امکان پیاده‌سازی آن با عناصر ساده جمع‌کننده و شیفتر دهنده وجود دارد و نیازی به استفاده از واحدهای پیشرفته‌تر نظیر ضرب‌کننده وجود ندارد. همچنین باید توجه کرد که در زمان ساخت و توسعه این الگوریتم، کامپیوتر واژه‌ای برای اشاره به دستگاه محاسبه‌گر بوده است و کامپیوتر با آن مفهوم سیستم متشکل از پردازنده و حافظه و مفاهیم مطرح شده توسط تورینگ، هر چند وجود داشت، اما هنوز کاربرد آن گسترده نشده بود. از این رو نباید لغت کامپیوتر در نام این الگوریتم را با تعریف امروزی آن اشتباه گرفت. [۱] [۲]

### ۲.۱ هدف الگوریتم

الگوریتم CORDIC در شکل ساده و رایج خود، الگوریتمی برای محاسبه مقادیر توابع مختلف علی‌الخصوص توابع مثلثاتی و هذلولوی است. از الگوریتم CORDIC در مواردی برای محاسبه توابع لگاریتمی، جذر گرفتن و موارد نظیر این هم استفاده می‌شود اما شکل اولیه آن مبتنی بر همان توابع مثلثاتی است. در اصل الگوریتم اصلی که در این جا قصد پیاده‌سازی آن را داریم، دو حالت عملکردی مختلف دارد. حالت Rotation و حالت Vectoring. در حالت Rotation، یک بردار همراستا با محور  $x$  و همچنین یک زاویه به الگوریتم داده شده و این الگوریتم، در اثر چرخش‌های متوالی در مراحل پشت سرهم، بردار اولیه را می‌چرخاند تا در نهایت برآیند تمامی این چرخش‌ها، با دقت مشخصی برابر با زاویه داده شده به برنامه بشود. در حالت Vectoring، یک بردار دلخواه داده می‌شود و الگوریتم سعی می‌کند در اثر چرخش‌های متوالی و منظم، مولفه  $y$  بردار را از بین ببرد و آن را بر محور  $x$  منطبق کند. با این کار، از طریق مولفه  $x$  نهایی می‌توان اندازه بردار اولیه را بدست آورد و همچنین با بررسی روند چرخش‌های انجام شده، می‌توان به زاویه بردار اولیه هم پی برد. [۲] [۳] [۴]

## ۳.۱ پایه ریاضی

## ۱.۳.۱ حالت Rotation

ابتدا باید به نحوه مدل کردن چرخش یک نقطه در دستگاه مختصات بپردازیم. اگر نقطه  $v = (x, y)$  را به اندازه زاویه  $\theta$  به صورت پادساعتگرد (در جهت مثلثاتی) دوران بدهیم، نقطه  $v' = (x', y')$  به صورت زیر بدست می‌آید:

$$x' = x \cos(\theta) - y \sin(\theta)$$

$$y' = x \sin(\theta) + y \cos(\theta)$$

از طرف دیگر، یک دوران به اندازه  $\theta$  را می‌توان مجموعی از  $n$  دوران دیگر  $\theta_1, \theta_2, \dots, \theta_n$  دانست که  $\theta = \theta_1 + \theta_2 + \dots + \theta_n$

در نتیجه می‌توان برای انجام یک دوران، آن را به چشم تعدادی دوران دیگر دانست و با نماد گذاری نقطه شروع به شکل  $v_0 = (x_0, y_0)$  و نتیجه هر کدام از  $n$  دوران به صورت  $v_i = (x_i, y_i)$  داریم:

$$x_{i+1} = x_i \cos(\theta_{i+1}) - y_i \sin(\theta_{i+1})$$

$$y_{i+1} = x_i \sin(\theta_{i+1}) + y_i \cos(\theta_{i+1})$$

از طرف دیگر، با فاکتور گرفتن از  $\cos(\theta_{i+1})$  در عبارات بالا داریم:

$$x_{i+1} = \cos(\theta_{i+1})(x_i - y_i \tan(\theta_{i+1}))$$

$$y_{i+1} = \cos(\theta_{i+1})(x_i \tan(\theta_{i+1}) + y_i)$$

اگر عبارت شامل  $\cos$  را کنار بگذاریم، درون پرانتز شاهد یک جمع (تفریق) و یک ضرب هستیم. در سخت افزار ضرب به حالت کلی، هزینه نسبتاً بالایی دارد؛ اما الگوریتم CORDIC سعی می‌کند با ایده هوشمندانه ضرب در توان‌های 2 این مشکل را برطرف کند. زوایایی که قرار است برای چرخش‌ها انتخاب بشوند، باید به این صورت باشند که:

$$\tan(\theta_{i+1}) = \pm 2^{-i}$$

از آن جایی که عملیات‌های ضرب یا تقسیم بر توان‌های 2 در یک سیستم دیجیتالی به راحتی از طریق شیفت دادن قابل انجام است، بدون نیاز به استفاده از ضرب کننده، می‌توانیم عملیات‌ها را انجام بدهیم. نکته مهمی که باقی می‌ماند، کسینوس‌هایی است که در هر مرحله در حال ضرب شدن هستند. ابتدا باید توجه کرد که:

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

در نتیجه، در اصل بسته به این که قرار است این محاسبات تا چند مرحله ادامه پیدا کنند، عدد ضرب شده نهایی به صورت:

$$K = \prod_i \frac{1}{\sqrt{1 + 2^{-2i}}}$$

خواهد بود.

جدول محاسبات مربوط به  $K$  در زیر آمده است:

$i$	$2^{-i}$	$\arctan(2^{-i})$	$\cos(\arctan(2^{-i}))$	$\prod_i \cos(\arctan(2^{-i}))$
0	1.0000	0.7854	0.7071	0.7071
1	0.5000	0.4636	0.8944	0.6325
2	0.2500	0.2450	0.9701	0.6136
3	0.1250	0.1244	0.9923	0.6088
4	0.0625	0.0624	0.9981	0.6076
5	0.0312	0.0312	0.9995	0.6074
6	0.0156	0.0156	0.9999	0.6073
7	0.0078	0.0078	1.0000	0.6073
8	0.0039	0.0039	1.0000	0.6073
9	0.0020	0.0020	1.0000	0.6073
10	0.0010	0.0010	1.0000	0.6073
11	0.0005	0.0005	1.0000	0.6073
12	0.0002	0.0002	1.0000	0.6073
13	0.0001	0.0001	1.0000	0.6073
14	0.0001	0.0001	1.0000	0.6073
15	0.0000	0.0000	1.0000	0.6073

همان طور که مشخص است، با دقت چهار رقم اعشار سری مذکور به 0.6073 همگراست. با کمک نرم افزارهای ریاضیاتی نظیر Mathematica نیز می توان بررسی کرد که:

$$\lim_{n \rightarrow \infty} \prod_{i=0}^n \frac{1}{\sqrt{1 + 2^{-2i}}} \approx 0.607253 \approx 0.6073$$

همان طور که در بالا دیدیم:

$$\tan(\theta_{i+1}) = \pm 2^{-i}$$

یعنی دو مقدار مثبت و منفی وجود دارد و زاویه هم می تواند به دو شکل مختلف مثبت و منفی باشد. برای این که بدانیم در هر مرحله، باید مقدار مثبت زاویه را اعمال کنیم یا مقدار منفی آن را باید به این توجه کنیم که ما در این جا با زوایایی با مقادیر مشخص سر و کار داریم و در ابتدا هم یک زاویه به عنوان زاویه هدف در نظر داریم که هر بار با اجرای یکی از مراحل، می توانیم زاویه هدف را کاهش داده و به مقدار جدید آپدیت کنیم. حال اگر مقدار زاویه هدف باقی مانده مثبت باشد، باید با کم کردن یک زاویه مثبت، آن را به صفر نزدیک کنیم و اگر منفی باشد، با کم کردن یک زاویه منفی، آن را به صفر نزدیک نماییم. در نتیجه فرمول کلی الگوریتم به صورت زیر در می آید:

$$x_{i+1} = x_i - d_i \cdot y_i \cdot 2^{-i}$$

$$\begin{aligned} y_{i+1} &= y_i + d_i x_i 2^{-i} \\ \theta_{i+1} &= \theta_i - d_i \arctan(2^{-i}) \end{aligned}$$

که در آن اگر  $z_i$  نامنفی باشد، مقدار  $d_i$  برابر ۱ و در غیر این صورت برابر  $-1$  است. چند نکته در این جا باقی می ماند. اولین مورد این است که بازه برد  $\arctan$  در  $\frac{\pi}{2} > \theta > -\frac{\pi}{2}$  است. برای این که عملیات چرخش برای زوایای بیش تر از  $\frac{\pi}{2}$  و کمتر از  $-\frac{\pi}{2}$  هم به خوبی جواب بدهد، باید دوران ناشی از  $\frac{\pi}{2}$  ای که این زاویه ها اضافه یا کم دارند را در همان ابتدا اعمال کنیم. برای این کار، اگر  $\theta > \frac{\pi}{2}$  باشد، در همان ابتدا تبدیل زیر را اعمال می کنیم:

$$x' = -y, y' = x, \theta' = \theta - \frac{\pi}{2}$$

و اگر  $\theta < -\frac{\pi}{2}$  باشد، در همان ابتدا تبدیل:

$$x' = y, y' = -x, \theta' = \theta + \frac{\pi}{2}$$

را اعمال می کنیم و الگوریتم را با  $x', y', \theta'$  ادامه می دهیم. نکته دیگر مربوط به این است که دو شکل دیگر از الگوریتم CORDIC به نام های Hyperbolic و Linear یعنی هذلولوی و خطی وجود دارند و شکلی که در بالا آن را توصیف کردیم، شکل Circular یا دایروی آن بود. منطق کلی این روش ها مشابه بالا است، با این تفاوت که در شکل هذلولوی، از توابع هذلولوی استفاده شده و در شکل خطی، عملاً به جای  $\arctan(2^{-i})$  خود  $2^{-i}$  قرار می گیرد. به جز این، این دو روش تنها در منفی یا مثبت بودن یکسری از عبارات با هم تفاوت دارند که به صورت خلاصه و به شکل متحدالشکل، می توان آن را به صورت زیر نمایش داد.

$$x_{i+1} = x_i - \eta \cdot d_i \cdot y_i \cdot 2^{-i}$$

$$\begin{aligned} y_{i+1} &= y_i + d_i x_i 2^{-i} \\ \theta_{i+1} &= \theta_i - d_i \cdot \xi_i \end{aligned}$$

که  $\eta$  و  $\xi$  براساس جدول زیر هستند:

Mode	$\eta$	$\xi$
Linear	0	$2^{-i}$
Circular	1	$\tan^{-1}(2^{-i})$
Hyperbolic	-1	$\tanh^{-1}(2^{-i})$

در نهایت هم بعد از اعمال همه مراحل، باید  $x$  و  $y$  نهایی در ضریب  $K$  که بالا نوشته شد، ضرب بشوند. این ضریب برای حالت هذلولوی، برابر  $K = 1.2075$  و برای حالت خطی  $K = 1$  است.

در نهایت لازم به ذکر است که هر چند الگوریتم چرخش، می‌تواند هر بردار اولیه‌ای را بچرخاند، اما در شکل استاندارد الگوریتم، بردار اولیه به صورت یک و موازی محور  $x$  داده می‌شود. در دو جدول زیر، این که ورودی‌ها و خروجی‌های هر کدام از این حالات در وضعیت استاندارد چه هستند، نمایش داده شده است:

جدول ورودی‌های استاندارد:

Mode	$x_{\text{input}}$	$y_{\text{input}}$	$\theta_{\text{input}}$
Linear	x	0	$\theta$
Circular	1	0	$\theta$
Hyperbolic	1	0	$\theta$

جدول خروجی‌های استاندارد:

Mode	$x_{\text{output}}$	$y_{\text{output}}$	$\theta_{\text{output}}$
Linear	$x$	$x \times \theta$	0
Circular	$\cos(\theta)$	$\sin(\theta)$	0
Hyperbolic	$\cosh(\theta)$	$\sinh(\theta)$	0

### ۲.۳.۱ حالت Vectoring

پایه و اساس حالت Vectoring از نظر ریاضیاتی مشابه حالت Rotation است. با این تفاوت که در حالت Rotation ما قصد داشتیم یک بردار را به اندازه زاویه مشخص شده بچرخانیم، اما این جا قصد داریم یک بردار را طوری بچرخانیم که روی محور  $x$  منطبق شده و در نتیجه این چرخش‌ها اندازه و زاویه اولیه‌اش با محور  $x$  را پیدا کنیم. از نظر ریاضیاتی، همه مواردی که در بخش قبل گفته شده، در این جا هم برقرار است. فقط در این جا باید توجه کرد که در ابتدای کار، ما زاویه‌ای در اختیار نداریم و در عوض باید با استفاده از  $x$  و  $y$  داده شده، این زاویه را پیدا کنیم. مشابه بخش قبل، اساس کلی این الگوریتم هم براساس زوایا و بردارهای ناحیه اول و چهارم مختصات است. در نتیجه، اضافه کردن زوایا، بر این اساس انجام می‌شود که آیا  $y$  بردار فعلی، مثبت است یا منفی، اگر منفی باشد، با افزایش زاویه و اگر مثبت باشد، با کاهش زاویه رو به رو هستیم. در اصل تفاوت اصلی این الگوریتم با الگوریتم بخش قبل، در مقدار  $d_i$  است که به شکل زیر مشخص می‌شود:

$$d_i = \begin{cases} 1 & \text{if } y_i < 0 \\ -1 & \text{if } y_i \geq 0 \end{cases}$$

معادلات اصلی الگوریتم، همان معادلات قبلی است:

$$x_{i+1} = x_i - \eta \cdot d_i \cdot y_i \cdot 2^{-i}$$

$$y_{i+1} = y_i + d_i x_i 2^{-i}$$

$$\theta_{i+1} = \theta_i - d_i \cdot \xi_i$$

در حالتی که ورودی و خروجی در ناحیه اول یا چهارم نباشند، تبدیل‌هایی مانند تبدیل‌های بخش قبل صورت می‌گیرد، با این تفاوت که در این جا، معیار ما مثبت یا منفی بودن  $x$  و  $y$  است. اگر  $x$  منفی و  $y$  مثبت باشد یعنی در ناحیه دوم هستیم و تبدیل زیر اعمال می‌شود:

$$x' = y, y' = -x, \theta' = \frac{\pi}{2}$$

و اگر  $x$  منفی و  $y$  هم منفی باشد، یعنی در ناحیه سوم هستیم و تبدیل زیر اعمال می‌شود:

$$x' = -y, y' = x, \theta' = \frac{-\pi}{2}$$

توجه داشته باشید که در این حالت، در ابتدا  $\theta = 0$  است، چون چیزی در مورد زاویه نمی‌دانیم و هدف پیدا کردن زاویه است. بعد از انجام تبدیلات، الگوریتم را با استفاده از  $x', y', \theta'$  پی می‌گیریم. جدول ورودی و خروجی‌های استاندارد این حالت به صورت زیر است:

جدول ورودی‌های استاندارد:

Mode	$x_{\text{input}}$	$y_{\text{input}}$	$\theta_{\text{input}}$
Linear	$x$	$y$	0
Circular	$x$	$y$	0
Hyperbolic	$x$	$y$	0

جدول خروجی‌های استاندارد:

Mode	$x_{\text{output}}$	$y_{\text{output}}$	$\theta_{\text{output}}$
Linear	$x$	0	$\frac{y}{x}$
Circular	$\sqrt{x^2 + y^2}$	0	$\tan^{-1} \frac{y}{x}$
Hyperbolic	$\sqrt{x^2 - y^2}$	0	$\tanh^{-1} \frac{y}{x}$

منابع استفاده شده برای بخش مربوط به پایه ریاضی: [۳] [۴] [۵]



## ۴.۱ شکل دقیق الگوریتم

در الگوریتم‌های زیر،  $\theta$  را با  $z$  نمایش داده ایم. الگوریتم‌ها براساس حالت Circular نوشته شده اند ولی تنها تفاوت دو حالت دیگر، همان طور که در بالا هم گفته شده، در علامت مثبت و منفی یکی از عبارات و همچنین استفاده از تابعی متفاوت با  $\arctan$  است. ضمناً بهینه سازی‌های پیاده سازی سخت افزاری (نظیر این که عملاً از یک  $K$  ثابت استفاده خواهیم کرد) در زیر اعمال نشده و شکل کلی الگوریتم نوشته شده است. [۴]

**Algorithm 1:** CORDIC Rotation

---

```

input :  $x_{in}, y_{in}, z_{in}$ :angle,  $n$ :number-of-iterations
output:  $x_{out}, y_{out}$ 
if  $!(-\frac{\pi}{2} < z_{in} < \frac{\pi}{2})$  then
  if  $z_{in} > \frac{\pi}{2}$  then
     $x = -y_{in}$ 
     $y = x_{in}$ 
     $z = z_{in} - \frac{\pi}{2}$ 
  else
     $x = y_{in}$ 
     $y = -x_{in}$ 
     $z = z_{in} + \frac{\pi}{2}$ 
  end
else
   $x = x_{in}$ 
   $y = y_{in}$ 
   $z = z_{in}$ 
end
 $K = 1$ 
for  $i \leftarrow 0$  to  $n$  do
   $d = \text{sgn}(z)$ 
   $x = x - d \cdot y \cdot 2^{-i}$ 
   $y = y + d \cdot x \cdot 2^{-i}$ 
   $z = z + d \cdot \arctan(2^{-i})$ 
   $K = K \cdot \frac{1}{\sqrt{1+2^{-2i}}}$ 
end
 $x_{out} = \frac{x}{K}$ 
 $y_{out} = \frac{y}{K}$ 

```

---

---

**Algorithm 2:** CORDIC Vectoring

---

**input :**  $x_{in}$ ,  $y_{in}$ ,  $z_{in}$ :angle,  $n$ :number-of-iterations**output:**  $x_{out}$ ,  $z_{out}$ **if**  $x_{in} < 0$  **then**    **if**  $y_{in} \geq 0$  **then**         $x = y_{in}$          $y = -x_{in}$          $z = +\frac{\pi}{2}$     **else**         $x = -y_{in}$          $y = x_{in}$          $z = -\frac{\pi}{2}$     **end****else**     $x = x_{in}$      $y = y_{in}$      $z = 0$ **end** $K = 1$ **for**  $i \leftarrow 0$  **to**  $n$  **do**     $d = \text{sgn}(y)$      $x = x - d \cdot y \cdot 2^{-i}$      $y = y + d \cdot x \cdot 2^{-i}$      $z = z + d \cdot \arctan(2^{-i})$      $K = K \cdot \frac{1}{\sqrt{1+2^{-2i}}}$ **end** $x_{out} = \frac{x}{K}$  $z_{out} = z$ 

---

## ۵.۱ کاربردها

### ۱.۵.۱ سخت افزار

یکی از اولین کاربردهای اصلی و واقعی الگوریتم CORDIC، استفاده از آن در سیستم مسیریابی و ناوبری ماه‌نورد ناسا در پروژه Apollo در حدود سال‌های ۱۹۷۱ و ۱۹۷۲ میلادی بوده است که از آن برای سیستم‌های محاسبه زاویه افقی نسبت به اشیا و همچنین محاسبه فاصله تا سفینه اصلی استفاده شده است. [۶]

به عنوان کاربرد عمومی، CORDIC در سیستم‌هایی که نیاز به محاسبه توابع مثلثاتی داشته باشند، ولی به دلایلی نظیر هزینه، امکان استفاده از ضرب کننده در آن‌ها وجود نداشته باشد، کاربرد به سزایی دارد. در صورتی که بتوان از ضرب کننده استفاده کرد، محاسبه این توابع به کمک بسط تیلور و مک لورن آن‌ها در ترکیب با Lookup Table معمولاً سریع تر از CORDIC است ولی اگر امکان استفاده از ماژول ضرب کننده مجزا نباشد، CORDIC به مراتب نسبت به پیاده سازی نرم افزاری ضرب و سپس استفاده از ضرب برتری دارد و با سرعت بیش تری امکان انجام محاسبات را فراهم می آورد. همچنین در مواقعی که سعی بر کمینه کردن تعداد گیت‌های استفاده شده باشد و از این رو بخواهیم از ضرب کننده استفاده نکنیم (مثلاً در یک سیستم پیاده شده روی FPGA استفاده از CORDIC می تواند مورد توجه قرار بگیرد).

### ۲.۵.۱ نرم افزار

در دورانی که CPU‌ها واحد Floating-Point مجزا نداشتند و تمامی رجیسترهای آنان، به صورت Integer بودند، شرکت‌های تولید کننده پردازنده در قالب کتابخانه‌های نرم افزاری مربوط به پیاده سازی IEEE Floating Point System که امکان انجام محاسبات Floating-Point را از طریق رجیسترهای Integer مهیا می کرد، الگوریتم CORDIC را هم به صورت نرم افزاری برای محاسبه زوایای مثلثاتی پیاده سازی می کردند، اما بعدها که واحدهای مجزای Floating-Point و محاسبات آن به صورت سخت افزاری به پردازنده‌ها استفاده شد، استفاده از CORDIC به این شیوه منسوخ شد و تنها در سیستم‌هایی که از نظر زمانی و Real-Time بودن محدودیت‌های ویژه‌ای دارند، از آن استفاده می شود. [۷]

## مراجع

- [1] Volder, Jack. The birth of cordic. *VLSI Signal Processing*, 25:101–105, 06 2000.
- [2] Volder, Jack E. The cordic computing technique. in *IRE-AIEE-ACM '59 (Western)*, 1959.
- [3] Boppana, Lakshmi and Dhar, Anindya. Cordic architectures: a survey. *VLSI Design*, 2010, 03 2010.
- [4] Andraka, Ray. A survey of cordic algorithms for fpga based computers. *ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA*, 12 2001.
- [5] Valls, Javier, Kuhlmann, Martin, and Parhi, Keshab. Evaluation of cordic algorithms for fpga design. *VLSI Signal Processing*, 32:207–222, 11 2002.
- [6] Heffron, W. and Piana, F. The navigation system of the lunar roving vehicle. 01 1971.
- [7] Wikipedia. Cordic, 2020.