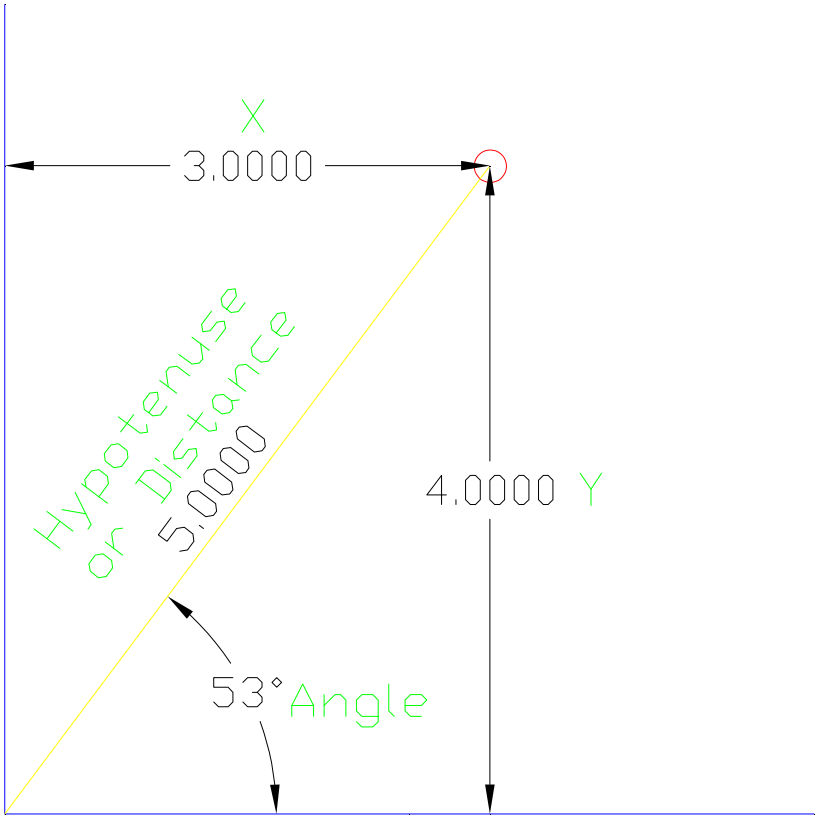
CORDIC For Dummies

CORDIC is a method of calculating a math function using much simpler math operations in a loop called a Binary Search. Most commonly CORDIC is used to calculate ATAN2 (Angle), and Hypotenuse (Distance) of a point. CORDIC can also be used to calculate other math functions like SIN and COS.

Let's say you have the coordinates (X, Y) of a point and you want to calculate the angle of this point from 0,0. You would use ATAN2 to do that calculation.

Converting from coordinates (X,Y) to (Angle, Distance) is also called rectangular to polar conversion.

In the picture below the coordinates of the red point are 3,4. By using a CORDIC algorithm we can compute the hypotenuse (5) and the angle (53).



Before getting into CORDIC let's see how a binary search works.

Guessing game (high / low)

I'm thinking of a number between 0 and 100. You make a guess and I'll tell you if you are too high or too low.
If you want to find the number in the least number of guesses what should you're first guess be ?

You should guess 50. Why ? Because that will tell you if the number is between 0 and 50 or 50 and 100. Effectively splitting the number of possible values in half.

Now if I tell you that your guess was too high, then you would know the number must be between 0 and 49. So what would your next guess be ?

Right 25. That will tell you if the number is between (0 and 25) or (25 and 50). You would keep dividing the value to by for each guess. This is the most efficient way to find the value, and is called a "Binary Search".

Here is a demonstration:

Unknown value is 80

50 = Too low so +25
75 = Too low so +12
87 = Too high so -6
81 = Too high so - 3
78 = Too low so +2
80 = YOU GOT IT!

Guessing game (add / subtract)

Okay, now the same game only this time I will think of a number from 0 to 90. You tell me to add or subtract a value from the number I'm thinking of, and I will tell you if the result is positive or negative. You are trying to get the number to equal zero. Then the sum of the adds and subtracts would be the original value you were trying to guess.

If you want to make the number zero in the least number of guesses, what should be your first "guess" ?
You should tell me to "subtract 45".

Why ? Because that will tell you if the number is between 0 and 45 or 45 to 90. If the number goes negative after subtracting 45, then it must have been 0 and 45. If it remains positive then it must have been 45 and 90.

Let's assume I say the number is now negative (so it must have been between 0 and 45). What should be your second "guess" ? You should tell me to "add 22".

Now I say the number is positive (so it must have been between 22 and 45). Your next guess should be to subtract 11.

And we will keep going, your values being half of the previous value. And the operation (add or subtract) depending on if the number is now negative or positive. If the number is negative, you will tell me to add the value. If the number is positive you will tell me to subtract the value. This only makes sense since we are trying to get the number to equal zero after all.

If you keep track of each operation you can get pretty close to the knowing what the original value was.

Here is a demonstration:
Unknown value is 43

Subtract 45	(-2) Negative
Add 22	(+20) Positive
Subtract 11	(+9) Positive
Subtract 5	(+4) Positive
Subtract 2	(+2) Positive
Subtract 1	+1 Positive
Subtract 1	0

Sum = -45+22-11-5-2-1-1 = -43 to get number to zero

Guessing game (Find ATAN using SIN & COS)

Now let's use this guessing game technique to calculate the angle of a point. Let's say you have a point at x=5; y=10. What is the angle from 0,0 to 5,10 ? To find the angle you use the arctangent function like this angle=arctangent(y/x).

So the answer is 63.435 degrees. Since we want to find degrees, instead of simply adding and subtracting a value, we need to add or subtract degrees. So really we will be rotating the point clockwise (+) or counter-clockwise(-).

Here are the classic formulas to rotate a point:

Counter clockwise:

```
Xnew = X * COS(angle) - Y * SIN(angle)
Ynew = Y * COS(angle) + X * SIN(angle)
Angle = Angle / 2
```

Clockwise:

```
Xnew = X * COS(angle) + Y * SIN(angle)
Ynew = Y * COS(angle) - X * SIN(angle)
Angle = Angle / 2
```

With the classic way to rotate a point you need to use the SIN and COS functions. Now you might ask "How in the heck are we going to calculate SIN and COS on a microcontroller ?". We don't need to. Because we know ahead of time what angles we need to use (45, 22, 11, etc) we can simply pre-calculate the SIN and COS of those angle and put them in a table.

```
CosTable = Cos(45), Cos(22), Cos(11), Cos(5), Cos(2), Cos(1), Cos(1)
SinTable = Sin(45), Sin(22), Sin(11), Sin(5), Sin(2), Sin(1), Sin(1)
CurAngle = 45
SumAngle = 0
LoopNum = 0
```

Counter clockwise:

```
Xnew = X * CosTable(LoopNum) – Y * SinTable(LoopNum)
Ynew = Y * CosTable(LoopNum) + X * SinTable(LoopNum)
SumAngle = SumAngle + CurAngle
CurAngle = CurAngle / 2
LoopNum = LoopNum + 1
```

Clockwise:

```
Xnew = X * CosTable(LoopNum) + Y * SinTable(LoopNum)
Ynew = Y * CosTable(LoopNum) - X * SinTable(LoopNum)
SumAngle = SumAngle - CurAngle
CurAngle = CurAngle / 2
LoopNum = LoopNum + 1
```

As you can see even if we use tables for SIN(angle) and COS(angle), there is still quite a bit of multiplication that we need to do. But this method will work for a microcontroller.

When you rotate a point to zero degrees, the sum of all the rotations is equal to the original angle, and the ending X coordinate will be equal to the radius (hypotenuse) of the original point. And the Y coordinate will be zero.

Guessing game (Find ATAN using TAN (radius gets scaled))

To reduce the number of multiply operations we can simplify the formulas to use the TAN function instead of SIN and COS. The only side effect is that the hypotenuse does not keep it's scale. But the effect is constant for each loop so a single multiply at the very end will get the value back to what it should be.

Counter Clockwise:

Xnew = X - Y * TAN(angle)
Ynew = Y + X * TAN(angle)
Angle = Angle / 2

Clockwise

Xnew = X + Y * TAN(angle)
Ynew = Y - X * TAN(angle)
Angle = Angle / 2

Again in a microcontroller the TAN(angle) values will be stored in a table.

TanTable = Tan(45), Tan(22), Tan(11), Tan(5), Tan(2), Tan(1), Tan(1)
CurAngle = 45
SumAngle = 0
LoopNum = 0

Counter Clockwise:

Xnew = X - Y * TanTable(LoopNum)
Ynew = Y + X * TanTable(LoopNum)
SumAngle = SumAngle + CurAngle
CurAngle = CurAngle / 2
LoopNum = LoopNum + 1

Clockwise

Xnew = X + Y * TanTable(LoopNum)
Ynew = Y - X * TanTable(LoopNum)
SumAngle = SumAngle – CurAngle
CurAngle = CurAngle / 2
LoopNum = LoopNum + 1

The first value in the TanTable would be Tan(45). And it just so happens that Tan(45) = 1. So that is easy.

Our next angle is 22, so Tan(22) = 0.40402 Hmmm too bad it isn't 0.5, we could do that with a simple shift operation instead of a full multiply. (hint…hint).

The next angle would be 11, so Tan(11) = 0.19438 Hmm too bad that isn't 0.25, we could do that with a simple shift operation (hint…hint).

Guessing game (Find ATAN using TAN(angle) = 1/(2^N))

As we saw in the last example we still needed to do some multiplication. But what if we didn't use the optimum angle values of 45, 22, 11, etc. What if we used angle value that gave us multipliers that we could perform with shifts (like 0.5, 0.25, 0.125, etc). Well the first angle of 45 is really easy because it is 1.0.

So what angle would give us Tan(Angle) = 0.500. It turns out to be 26.56505118. Now this isn't exactly half of 45, but it doesn't really matter. As long as it is smaller than 45 and larger than or equal to 22.5 it will work.

Continuing on, the angle that would give us 0.25, turns out to be 14.03624347 and this value is between 22.5 and 11.25. If we keep going at this finding the angles for 0.125, 0.0625, etc. We can greatly simplify the operations now to just add, subtract and shift. But since the angles are special values we need to keep a table of what angles we are rotating by when multiply by 0.5, 0.25, 0.125 etc. By adding or subtracting the angle values in the table we will know what the original angle was. And that is what we are trying to find.

AngTable = 45, 26.565, 14.036, 7.125, 3.576, 1.790, 0.895, 0.448
SumAngle = 0
LoopNum = 0

If Y is positive:

```
Xnew = X + (Y >> LoopNum)
Ynew = Y – (X >> LoopNum)
SumAngle = SumAngle + AngTable[LoopNum]
LoopNum = LoopNum + 1
```

If Y is negative:

```
Xnew = X - (Y >> LoopNum)
Ynew = Y + (X >> LoopNum)
SumAngle = SumAngle – AngTable[LoopNum]
LoopNum = LoopNum + 1
```

The above is the CORDIC method.

Let’s try an example:

X = 200
Y = 100

‘ Y is positive	
Xnew = 200 + 100 >> 0	Xnew = 300
Ynew = 100 – 200 >> 0	Ynew = -100
SumAngle = SumAngle + AngTable[0]	SumAngle = 45
‘ Y is negative	
Xnew = 300 - (-100) >> 1	Xnew = 300 + 50 = 350
Ynew = -100 + 300 >> 1	Ynew = -100 + 150 = 50
SumAngle = SumAngle - AngTable[1]	SumAngle = 45 - 26.565 = 18.483
‘ Y is positive	
Xnew = 350 + 50 >> 2	Xnew = 250 + 12 = 362
Ynew = 50 – 350 >> 2	Ynew = 50 – 87 = -37
SumAngle = SumAngle + AngTable[2]	SumAngle = 18.483 + 14.036 = 32.519
‘ Y is negative	
Xnew = 362 - (-37) >> 3	Xnew = 362 + 4 = 366
Ynew = -37 + 362 >> 3	Ynew = -37 + 45 = 8
SumAngle = SumAngle – AngTable[3]	SumAngle = 32.519 – 7.125 = 25.394
‘ Y is positive	
Xnew = 366 + 8 >> 4	Xnew = 366 + 0 = 366
Ynew = 8 - 366 >> 4	Ynew = 8 - 22 = -14
SumAngle = SumAngle + AngTable[4]	SumAngle = 25.394 + 3.576 = 28.97
‘Y is negative	
Xnew = 366 – (-14) >> 5	Xnew = 366 + 0 = 366
Ynew = -14 + 366 >> 5	Ynew = -14 + 11 = -3
SumAngle = SumAngle + AngTable[5]	SumAngle = 28.97 - 1.790 = 27.18
‘Y is negative	
Xnew = 366 - (-3) >> 6	Xnew = 366 + 0 = 366
Ynew = -3 + 366 >> 6	Ynew = -3 + 5 = 2
SumAngle = SumAngle – AngTable[6]	SumAngle = 27.18 – 0.895 = 26.285
‘ Y is positive	
Xnew = 366 + 2 >> 7	Xnew = 366 + 0 = 366
Ynew = 2 – 366 >> 7	Ynew = 2 – 2 = 0
SumAngle = SumAngle + AngTable[7]	SumAngle = 26.285 + 0.448 = 26.733

‘ Finished
‘ SumAngle = 26.733 (26.565 is the real angle)
‘ X = 366 scale by 0.60726 for any point as long as 8 loops are used = 222.25 (223.6 is actual hypotenuse)
‘ Y = 0

Using CORDIC to find Hypotenuse

In order to keep the math theory to a minimum I kind of glossed over how we got from using SIN/COS to using TAN. So here is a more detailed explanation.

Since TAN(angle) = SIN(angle) / COS(angle) we can rewrite the formulas:

Xnew = X * COS(angle) - Y * SIN(angle)
Xnew = X * COS(angle) - Y * SIN(angle) * COS(angle) / COS(angle) **' COS(angle) / COS(angle) = 1**
Xnew = COS(angle) * [X - Y * SIN(angle) / COS(angle)] **' Factor out * COS(angle)**
Xnew = COS(angle) * [X - Y * TAN(angle)] **' Replace SIN(angle) / COS(angle) with TAN(angle)**

In CORDIC we only use the X - Y * TAN(angle) and ignore the "COS(angle) *" part. So the values get scaled by 1/COS(angle). Now the wonderful thing about this is that COS(angle) = COS(-angle). This means that it doesn't matter if we rotate clockwise or counter-clockwise the scale is the same. If we multiply the COS(angle) of each of the angles that we are going to use in CORDIC we can just multiply the final X value by that value to get the scale correct.

COS(45)*COS(26.565)*COS(14.036)*COS(7.125)*COS(3.576)*COS(1.790)*COS(0.895)*COS(0.448) = 0.60726

This value is called the CORDIC “Gain”. And depends on how many angles you use. For 8 angles it is 0.60726.

Using CORDIC to find SIN and COS

In the previous pages we have used cordic to rotate a point at an unknown angle to get it to zero degrees. By keeping track of how we have moved it we know what it must have been to start with. For example if we had to rotate a total of -53 degrees to get the point to zero degrees (Y=0), then the point MUST have been at +53 to start with.

With finding SIN and COS we have the opposite situation. We take some point at zero degrees (Y=0) and rotate it by the number of degrees given. When we do, the final X and Y position of the point will be equal to the SIN and COS of the angle. We can only rotate by the angles in the CORDIC table (45, 26.565, 14.036, ect). Each angle is either added or subtracted to reach the specified angle. Let's say you want to find SIN and COS of angle 30.

We start at angle zero with Y=0 and X=starting value. If the current angle is less than the desired angle, then we add the CORDIC angle. If the desired angle is less than the current angle, then we subtract the CORDIC angle.

0 + 45 - 26.565 + 14.036 - 7.125 + 3.576 + 1.790 - 0.895 + 0.448 = 30.265

As long as we choose the correct starting X value, the final X and Y values will be the SIN and COS of the angle. Since SIN and COS return values of 1.0 or less, we need to scale up the values by some factor (otherwise we will only get 1 or 0 as answers. Basically we need to specify a value that will be multiplied by the SIN and COS. Let’s say you would like to get 256 * SIN(angle) and 256 * COS(angle). You would start with Y=0 and X=256 * “cordic gain”. We saw in the previous section that for 8 angles the cordic gain is 0.60726. So we would start the SIN, COS cordic process with Y=0 and X = 155.

Using CORDIC on a microcontroller

Most microcontrollers do not have floating point math. And those that do can usually perform integer math MUCH faster. To implement CORDIC in integer math normally we will the idea of "fixed point" arithmetic. Fixed point is kind of like changing the units that the values represent. For example with money we could say that the value "1" is one dollar. But we could also say that the value "100" representing pennies (or 1/100 of a dollar) is also 1 dollar.

We do the same thing with fixed point. Normally the scales will be powers of two since they work nicely with microcontrollers. For a 32 bit core like the propeller we might use units of 1/256 of a degree. So 45 degrees would be the value 45*256 or 11,520. Using this method the TAN angle values are:

45.000 =	11520
26.565 =	6801
14.036 =	3593
7.125 =	1824
3.576 =	916
1.790 =	458
0.895 =	229
0.448 =	115
0.224 =	57
0.112 =	28
0.056 =	14
0.028 =	7
0.014 =	4
0.007 =	2
0.003 =	1

You can see by the table that our answer will have a resolution of about 0.003 degrees, because that is the smallest angle we can rotate by.

CORDIC only works for angles zero to 90. If the angle is not in that range, then some preconditioning (and possibly post-conditioning) of the value is needed.

On the following pages are two program that show how to use the CORDIC method. They are written in spin for the Parallax Propeller processor.

First is a program to calculate the ATAN2 and Hypotenuse of an X, Y coordinate.

The second is a program to calculate the SIN and COS of an angle.

Propeller CORDIC Program to Calculate ATAN2 and HYP

```
' CORDIC Demo program to compute ATAN2 and Hypotenuse of X,Y coordinates
'
' Notice the X, Y starting values and the angle, hyp values are ALL in 1/256 units.
'  The Angle is in 1/256 of a degree. If radians or brads are wanted, just scale the ATAN_Table
values
'
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

OBJ
  Debug : "FullDuplexSerial"

VAR
  LONG angle, hyp
  LONG X, Y
  LONG Xnew, Ynew
  LONG i

PUB Start
  X := -76800  ' 300 * 256 (value * 256)
  Y := -102400 ' 400 * 256 (Value * 256)

  Debug.Start(31, 30, 0, 115200)
  WaitCnt(clkfreq * 4 + cnt) ' Wait four seconds for user to start PST

  Debug.Str(string(16, "CORDIC Test", 13))
  Debug.Str(string(13, "X = "))
  Dec256(X)
  Debug.Str(string(13, "Y = "))
  Dec256(Y)
  Debug.Tx(13)

  ' Start of CORDIC routine
  angle := 0
  IF X < 0
    angle := 180 * 256
    X := -X
    Y := -Y
  ELSEIF Y < 0
    angle := 360 * 256
  REPEAT i FROM 0 TO 14
    IF Y < 0
      ' Rotate counter-clockwise
      Xnew := X - (Y ~> i)
      Ynew := Y + (X ~> i)
      angle := angle - ATAN_Table[i]
    ELSE
      ' Rotate clockwise
      Xnew := X + (Y ~> i)
      Ynew := Y - (X ~> i)
      angle := angle + ATAN_Table[i]
    X := Xnew
    Y := Ynew
  hyp := (X ~> 1) + (X  ** $1B74_EDA9) ' hyp := x * 0.607252935
  ' End of CORDIC routine

  Debug.Str(string("Angle = "))
  Dec256(angle)
  Debug.Str(string(13, "Hyp = "))
  Dec256(hyp)
  Debug.Str(string(13, "Finished...", 13))
```

```
PUB Dec256(given) | temp
  if given < 0
    given := -given
    Debug.Tx("-")
  temp := given ~> 8
  Debug.Dec(temp)
  temp := given & 255
  temp := temp * 1000
  temp := temp / 256
  Debug.Tx(".")
  if temp < 100
    Debug.Tx("0")
    if temp < 10
      Debug.Tx("0")
  Debug.Dec(temp)
  RETURN


DAT
  ' ATAN_Table is the values of ATAN(1/(2^i)) * 256 in degrees (NOT RADIANS)
  ATAN_Table LONG 11520, 6801, 3593, 1824, 916, 458, 229, 115, 57, 28, 14, 7, 4, 2, 1
```

Propeller CORDIC Program to Calculate SIN and COS

```
' CORDIC Demo program to compute SIN and COS of an angle
'
' Notice the specified angle and the SIN and COS values are ALL in 1/256 units.
'  The Angle is in 1/256 of a degree. If radians or brads are wanted, just scale the ATAN_Table
values
'
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

OBJ
  Debug : "FullDuplexSerial"

VAR
  LONG angle, desiredAngle
  LONG X, Y
  LONG Xnew, Ynew
  LONG i
  LONG cos, sin

PUB Start
  Debug.Start(31, 30, 0, 115200)
  WaitCnt(clkfreq * 4 + cnt) ' Wait two seconds for user to start PST

  Debug.Str(string("CORDIC Test", 13))

  desiredAngle := 30*256 ' Angle * 256

  Debug.Str(string(13, "Angle = "))
  Dec256(desiredAngle)
  Debug.Tx(13)

  ' Start of CORDIC routine
  angle := 0
  Y := 0
  X := 155 ' 256 * CORDIC Gain

  IF desiredAngle > 90*256
    angle := 180*256
  IF desiredAngle > 270*256
    angle := 360*256

  REPEAT i FROM 0 TO 14
    IF desiredAngle > angle
      ' Rotate counter-clockwise
      Xnew := X - (Y ~> i)
      Ynew := Y + (X ~> i)
      angle := angle + ATAN_Table[i]
    ELSE
      ' Rotate clockwise
      Xnew := X + (Y ~> i)
      Ynew := Y - (X ~> i)
      angle := angle - ATAN_Table[i]
    X := Xnew
    Y := Ynew

  IF (desiredAngle > 90*256) AND (desiredAngle < 270*256)
    X := -X
    Y := -Y

  cos := X
  sin := Y
  ' End of CORDIC routine
```

```
  Debug.Str(string("Sin = "))
  Dec256(sin)
  Debug.Str(string(13, "Cos = "))
  Dec256(cos)
  Debug.Str(string(13, "Finished...", 13))

PUB Dec256(given) | temp
  if given < 0
    given := -given
    Debug.Tx("-")
  temp := given ~> 8
  Debug.Dec(temp)
  temp := given & 255
  temp := temp * 1000
  temp := temp / 256
  Debug.Tx(".")
  if temp < 100
    Debug.Tx("0")
    if temp < 10
      Debug.Tx("0")
  Debug.Dec(temp)
  RETURN


DAT
  ' ATAN_Table is the values of ATAN(1/(2^i)) * 256 in degrees (NOT RADIANS)
  ATAN_Table LONG 11520, 6801, 3593, 1824, 916, 458, 229, 115, 57, 28, 14, 7, 4, 2, 1
```