

Angle Recoding 2. Wu

3. MVR

20181003 Wed

Copyright (c) 2015 - 2018 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

② MVR (Modified Vector Rotational)

two modifications

① **repetition** of elementary angles

each micro-rotation of elementary angle
can be performed repeatedly

- more possible combinations
- smaller ξ_m

② **confinement** of total micro-rotation number

confine the iteration number

in the micro-rotation phase
to **R_m** ($R_m \ll W$)

The role of **R_m** is quite similar
to the **number of non-zero digit**
 N_D in CSD recoding scheme

MV2

* repetition of elementary angles

$$\theta = a(2) * 2 = \tan^{-1}(2^2) * 2$$

AR

$$+ a(1) + a(5) - a(8) - a(10) - a(14)$$

k= 0 theta= 0.4899573 ik= 1 uik=+1 a[1]= 0.4636476, new theta= 0.0263097
k= 1 theta= 0.0263097 ik= 5 uik=+1 a[5]= 0.0312398, new theta=-0.0049301
k= 2 theta=-0.0049301 ik= 8 uik=-1 a[8]= 0.0039062, new theta=-0.0010239
k= 3 theta=-0.0010239 ik=10 uik=-1 a[10]= 0.0009766, new theta=-0.0000473
k= 4 theta=-0.0000473 ik=14 uik=-1 a[14]= 0.0000610, new theta= 0.0000137

Conventional
CORDIC

$$+ a(0) - a(1) + a(2) - a(3) + a(4) - a(5) + a(6) + a(7) \\ - a(8) - a(9) - a(10) - a(11) + a(12) + a(13) - a(14) - a(15)$$

k= 0 theta= 0.4899573 u[0]=+1 a[0]= 0.7853982, new theta=-0.2954408
k= 1 theta=-0.2954408 u[1]=-1 a[1]= 0.4636476, new theta= 0.1682068
k= 2 theta= 0.1682068 u[2]=+1 a[2]= 0.2449787, new theta=-0.0767719
k= 3 theta=-0.0767719 u[3]=-1 a[3]= 0.1243550, new theta= 0.0475831
k= 4 theta= 0.0475831 u[4]=+1 a[4]= 0.0624188, new theta=-0.0148357
k= 5 theta=-0.0148357 u[5]=-1 a[5]= 0.0312398, new theta= 0.0164041
k= 6 theta= 0.0164041 u[6]=+1 a[6]= 0.0156237, new theta= 0.0007804
k= 7 theta= 0.0007804 u[7]=+1 a[7]= 0.0078123, new theta=-0.0070319
k= 8 theta=-0.0070319 u[8]=-1 a[8]= 0.0039062, new theta=-0.0031257
k= 9 theta=-0.0031257 u[9]=-1 a[9]= 0.0019531, new theta=-0.0011726
k=10 theta=-0.0011726 u[10]=-1 a[10]= 0.0009766, new theta=-0.0001960
k=11 theta=-0.0001960 u[11]=-1 a[11]= 0.0004883, new theta= 0.0002923
k=12 theta= 0.0002923 u[12]=+1 a[12]= 0.0002441, new theta= 0.0000481
k=13 theta= 0.0000481 u[13]=+1 a[13]= 0.0001221, new theta=-0.0000740
k=14 theta=-0.0000740 u[14]=-1 a[14]= 0.0000610, new theta=-0.0000129
k=15 theta=-0.0000129 u[15]=-1 a[15]= 0.0000305, new theta= 0.0000176

data : W -bit word length

the iteration number : N $N \leq W$

the restricted iteration number : R_m $R_m \ll W$

$$i \in [0, R_m - 1]$$

$$R_m < W$$

the angle quantization error

$$\xi_{m, \text{MVR}} \triangleq \theta - \sum_{i=0}^{R_m-1} \alpha(i) a(s(i))$$

the rotational sequence

$$s(i) \in \{0, 1, \dots, W-1\} \quad * \text{repetition allowed}$$

the micro-rotation angle
in the i -th iteration

the directional sequence

$$\alpha(i) \in \{-1, 0, +1\}$$

the direction of the i -th
micro-rotation of $a(s(i))$

$$\alpha(i) a(s(i)) = \tilde{\theta}(i)$$

the rotational sequence

$$s(i) \in \{0, 1, \dots, W-1\} \quad [0, 3, 6, 7]$$

the directional sequence

$$\alpha(i) \in \{-1, 0, +1\} \quad [1, -1, -1, 1]$$

$$\text{atan}(2^0) - \text{atan}(2^{-3}) - \text{atan}(2^{-6}) + \text{atan}(2^{-7})$$

$$\alpha(i) a(s(i)) = \tilde{\theta}(j)$$

MVR-CORDIC Algorithm with $R_n = 4$	Greedy Algorithm	3	$\vec{\alpha} = [1 \ -1 \ -1 \ -1]$ $\vec{s} = [0 \ 3 \ 6 \ 7]$	$5.2891 \cdot 10^{-4}$
	Semi-greedy Algorithm ($D = 2$)	4	$\vec{\alpha} = [1 \ -1 \ -1 \ 1]$ $\vec{s} = [0 \ 3 \ 5 \ 7]$	$5.2033 \cdot 10^{-4}$
	TBS Algorithm	5	$\vec{\alpha} = [1 \ 1 \ -1 \ -1]$ $\vec{s} = [1 \ 2 \ 4 \ 7]$	$2.5911 \cdot 10^{-4}$

```
>> s = [0, 3, 6, 7]
>> alpha = [1, -1, -1, -1]
>> sum(atan(2.^(-s) .* alpha))
ans = 0.63761
>>
```

```
>> s = [0, 3, 5, 7]
>> alpha = [1, -1, -1, 1]
>> sum(atan(2.^(-s) .* alpha))
ans = 0.63762
```

```
>> alpha = [1, 1, -1, -1]
>> s = [1, 2, 4, 7]
>> sum(atan(2.^(-s) .* alpha))
ans = 0.63840
```

```
0 7.85398163397448e-01
1 4.63647609000806e-01
2 2.44978663126864e-01
3 1.24354994546761e-01
4 6.24188099959574e-02
5 3.12398334302683e-02
6 1.56237286204768e-02
7 7.81234106010111e-03
8 3.90623013196697e-03
9 1.95312251647882e-03
10 9.76562189559319e-04
11 4.88281211194898e-04
12 2.44140620149362e-04
13 1.22070311893670e-04
14 6.10351561742088e-05
15 3.05175781155261e-05
```

$w=16$

0	7.85398163397448e-01	s(0)
1	4.63647609000806e-01	.
2	2.44978663126864e-01	.
3	1.24354994546761e-01	s(1)
4	6.24188099959574e-02	.
5	3.12398334302683e-02	.
6	1.56237286204768e-02	s(2)
7	7.81234106010111e-03	s(3)
8	3.90623013196697e-03	.
9	1.95312251647882e-03	.
10	9.76562189559319e-04	.
11	4.88281211194898e-04	.
12	2.44140620149362e-04	.
13	1.22070311893670e-04	.
14	6.10351561742088e-05	.
15	3.05175781155261e-05	.

0	7.85398163397448e-01	s(0)
3	1.24354994546761e-01	s(1)
6	1.56237286204768e-02	s(2)
7	7.81234106010111e-03	s(3)

$R_m = 4$

$s(0)=0$
 $s(1)=3$
 $s(2)=6$
 $s(3)=7$

0	7.85398163397448e-01
1	4.63647609000806e-01
2	2.44978663126864e-01
3	1.24354994546761e-01
4	6.24188099959574e-02
5	3.12398334302683e-02
6	1.56237286204768e-02
7	7.81234106010111e-03
8	3.90623013196697e-03
9	1.95312251647882e-03
10	9.76562189559319e-04
11	4.88281211194898e-04
12	2.44140620149362e-04
13	1.22070311893670e-04
14	6.10351561742088e-05
15	3.05175781155261e-05

AQ & MVR CORDIC

$$\xi_{m, MVR} \triangleq \theta - \left[\sum_{j=0}^{R_m-1} \alpha(j) a(s(j)) \right]$$

the rotational sequence $s(j)$

$$j = 0, 1, 2, \dots, R_m-1$$

$s(j) \in \{0, 1, \dots, W-1\}$ rotational sequence

determines the micro-rotation angle $a(s(j))$
in the j -th iteration

the directional sequence $\alpha(j)$

$$\alpha(j) \in \{-1, 0, +1\}$$

controls the direction of the j -th
micro-rotation of $a(s(j))$

$$\alpha(j) a(s(j)) = \tilde{\theta}(j)$$

$$\begin{array}{ll} i = 0, 1, 2, 3, \dots, W-1 & \\ s(j) = 0, 1, 2, 3, \dots, W-1 & \text{rotational sequence} \\ \alpha(j) = -1, 0, 0, +1, \dots, -1 & \text{directional sequence} \\ j = 0, -, -, 1, \dots, R_m-1 & \text{effective iteration number} \\ R_m \ll W & \end{array}$$

i	j	$S(j)$
0	0	$S(0) = 0$

1

2

3

4	1, 4	$S(1) = 4, S(4) = 4$
---	------	----------------------

repetition allowed

5	2	$S(2) = 5$
---	---	------------

6

7

8	3	$S(3) = 8$
---	---	------------

9

10

11

12

13

14

$W-1 = 15$

effective
iteration
number

rotational
sequence

i	Conventional	j	$S(j)$
①	$S(0) = 0$	0	$S(0) = 0$
1	$S(1) = 1$		
2	$S(2) = 2$		
3	$S(3) = 3$		
④	$S(4) = 4$	1	$S(1) = 4$
⑤	$S(5) = 5$	2	$S(2) = 5$
6	$S(6) = 6$		
7	$S(7) = 7$		
⑧	$S(8) = 8$	3	$S(3) = 8$
9	$S(9) = 9$		
10	$S(10) = 10$		
11	$S(11) = 11$		
12	$S(12) = 12$		
13	$S(13) = 13$		
14	$S(14) = 14$		
15	$S(15) = 15$		

effective
iteration
number

rotational
sequence

$W-1 =$

$$\text{sub-angle } (\alpha(j) a(s(j))) \sim \tilde{\theta}(j)$$

$$\begin{aligned} \xi_{m,AR} &= \theta - \left[\sum_{j=0}^{N'-1} \tilde{\theta}(j) \right], \quad \tilde{\theta}(j) = \tan^{-1}(\alpha(j) \cdot 2^{-s(j)}) \\ &= \theta - \left[\sum_{j=0}^{N'-1} \tan^{-1}(\alpha(j) \cdot 2^{-s(j)}) \right] \end{aligned}$$

$$N' \triangleq \sum_{j=0}^{N-1} |\mu(j)| \quad \text{the effective iteration number}$$

EAS formed by MVR-CORDIC
is the same as AR
also performs AQ

the EAS consists of all possible values of $\tilde{\theta}(j)$

the EAS S_1 in AR

$$S_1 = \{ \tan^{-1}(\alpha^* \cdot 2^{-s^*}) : \alpha^* \in \{-1, 0, +1\}, s^* \in \{0, 1, \dots, N-1\} \}$$

The major difference

1) the total number of sub-angles N_A

the total iteration number

in the micro-rotation phase

is kept fixed to a pre-defined value of R_m

$$N_A = R_m$$

2) the sub-angle θ_i corresponds to $\alpha(j) \alpha(s(j))$

$$\theta_j = \alpha(j) \alpha(s(j)) = \tilde{\theta}_j$$

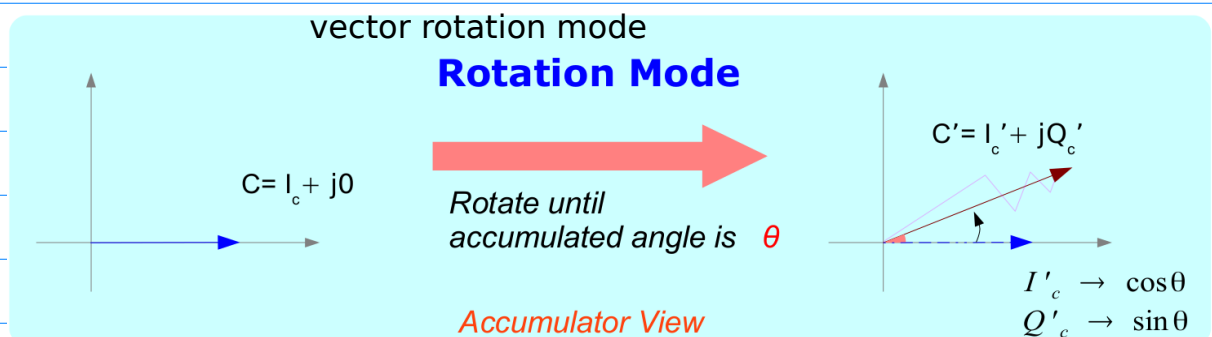
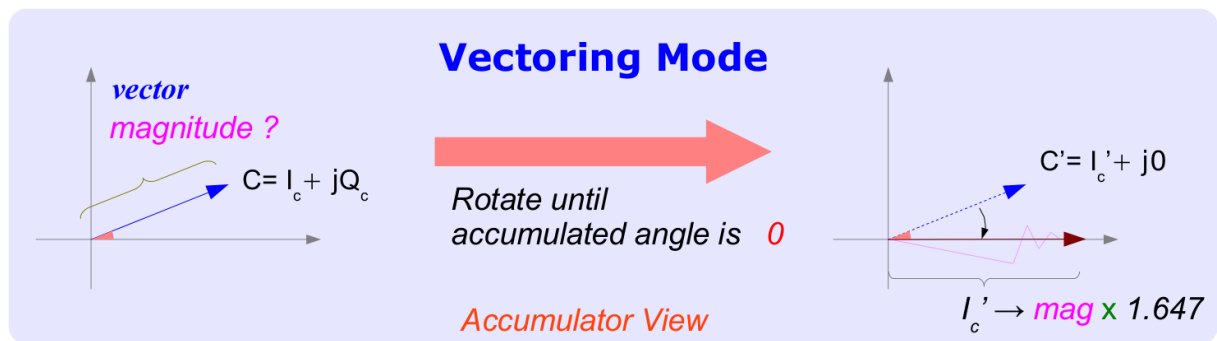
MVR (Modified Vector Rotation)

1) Repeat of Elementary Angles θ_i, θ_i

2) fixed total micro-rotation Number R_m

* Vector Rotation Mode

* and the rotation angles are known in advance



Modified Vector Rotational MVR CORDIC

- reduce the iteration number
- maintaining the SQNR performance
- modifying the basic microrotation procedure

Three Searching Algorithm

- ① the selective prerotation
- ② the selective scaling
- ③ iteration-tradeoff scheme

Optimization Problem

EAS point of view

Given θ , find the combination of R_m elementary angles from EAS S_i , such that the angle quantization error $|\xi_{m,nur}|$ is minimized.

semi-greedy algorithm
trade offs between computational complexities
and performance

key issue in the MVR-CORDIC
is to find the best sequences of
 $s(i)$ and $\alpha(i)$ to minimize $|\xi_m|$
subject to the constraint that
the total iteration number is confined to R_m

- 1) Greedy Algorithm
- 2) Exhaustive Algorithm
- 3) Semigreedy Algorithm

data : W -bit word length

the iteration number : N $N \leq W$

the restricted iteration number : R_m $R_m \ll W$

Hu's greedy algorithm

$$\theta(0) = \theta, \{\mu(i) = 0, 0 \leq i \leq N-1\}, k=0$$

repeat until $|\theta(k)| < \alpha(N-1)$ do

choose $i_k, 0 \leq i_k \leq N-1$

$$|\theta(k) - a(i_k)| = \min_{0 \leq i \leq N-1} |\theta(k) - a(i)|$$

$$\theta(k+1) = \theta(k) - \mu(i_k) a(i_k)$$

$$\mu(i_k) = \text{sign}(\theta(k))$$

$$J(i) = |\theta(i) - \alpha(i) a(i)| \text{ is minimized}$$

1) Greedy Algorithm

given θ , W , R_m

try to approach the target rotation angle, θ , step by step
in each step, decisions are made on $\alpha(i)$ and $s(i)$
by choosing the best combination of $\alpha(i)$ and $s(i)$
so as to minimize $|\xi_m|$

$\alpha(i)$ and $s(i)$ are determined such that

the error function $J(i) = |\theta(i) - \alpha(i)\alpha(s(i))|$ is minimized

$\theta(i)$: the residue angle in the i -th step

$$\theta(i) = \theta - \sum_{m=0}^{i-1} \alpha(m)\alpha(s(m))$$

the searching is terminated

$\left\{ \begin{array}{l} \text{if no further improvements can be found} \\ J(i) \geq J(i-1) \\ \text{if the iteration } (i) \text{ reaches } R_m - 1 \end{array} \right.$

$\alpha(R_m - 1)$ and $s(R_m - 1)$ are determined

at the end of the searching

the greedy algorithm terminates

Only when the residue angle error
cannot be further reduced.

Initialization:

given θ angle

W wordlength

R_m restricted iteration number

$$\theta(i) = \theta - \sum_{n=0}^{i-1} \alpha(n) a(s(n))$$

Select $\alpha(i) \in \{-1, 0, +1\}$

$s(i) \in \{0, 1, 2, \dots, W-1\}$

to minimize $J(i) = \theta(i) - \alpha(i) a(s(i))$

$s(m)$ repetition allowed

N

$J(i) < J(i-1)$

Y

$\theta(i+1) = \theta(i) - \alpha(i) a(s(i))$

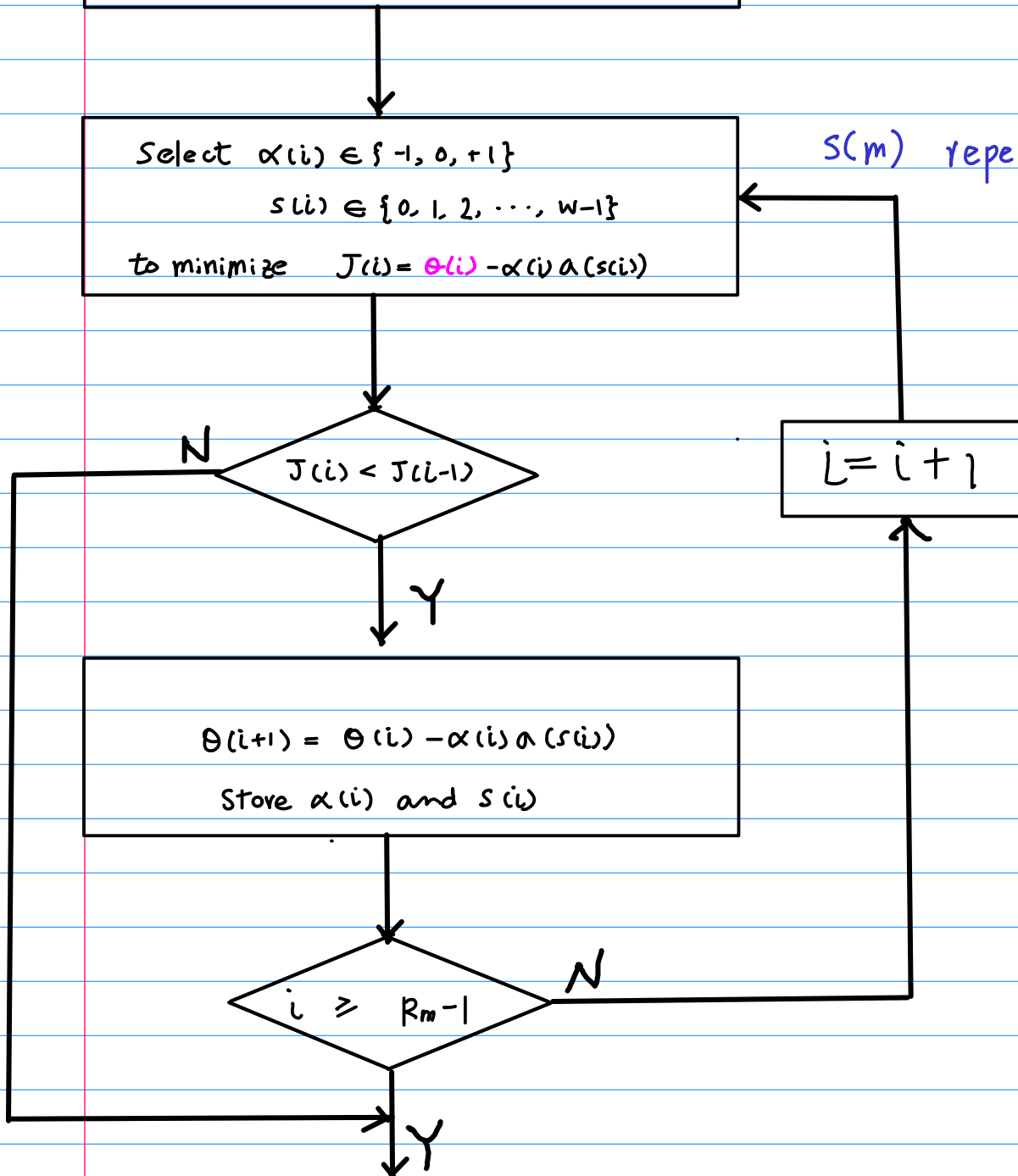
Store $\alpha(i)$ and $s(i)$

$i = i + 1$

$i \geq R_m - 1$

N

Y



2) Exhaustive Algorithm

search for the entire solution space

$$\begin{array}{ccc} \alpha(i) & a(s(i)) & i \\ \{-1, 0, +1\} & \{s(0), s(1), \dots, s(W-1)\} & \{0, 1, \dots, R_m-1\} \\ 3 & W & R_m \end{array} \Rightarrow (3W)^{R_m}$$

all possible combinations of

$$\sum_{i=0}^{R_m-1} \alpha(i) a(s(i))$$

in a single step

decisions for $\alpha(i)$ and $s(i)$, $0 \leq i \leq R_m-1$
by minimizing the error function

$$J = \left| 0 - \sum_{i=0}^{R_m-1} \alpha(i) a(s(i)) \right|$$

global optimal solution

Initialization:

given θ, W, R_m

let $\theta(0) = \theta,$

$i = 0$

$J(-1) = \infty$



Select $\alpha(i) \in \{-1, 0, +1\}$

$s(i) \in \{0, 1, 2, \dots, W-1\}$

for $0 \leq i \leq R_m - 1$

to minimize $J(i) = \theta - \sum_{i=0}^{R_m-1} \alpha(i) a(s(i))$

$$(3 \cdot W) \cdot (3 \cdot W) \dots (3 \cdot W) \\ = 3^{R_m} \cdot W^{R_m}$$



Store $\alpha(i)$ and $s(i)$

for $0 \leq i \leq R_m - 1$

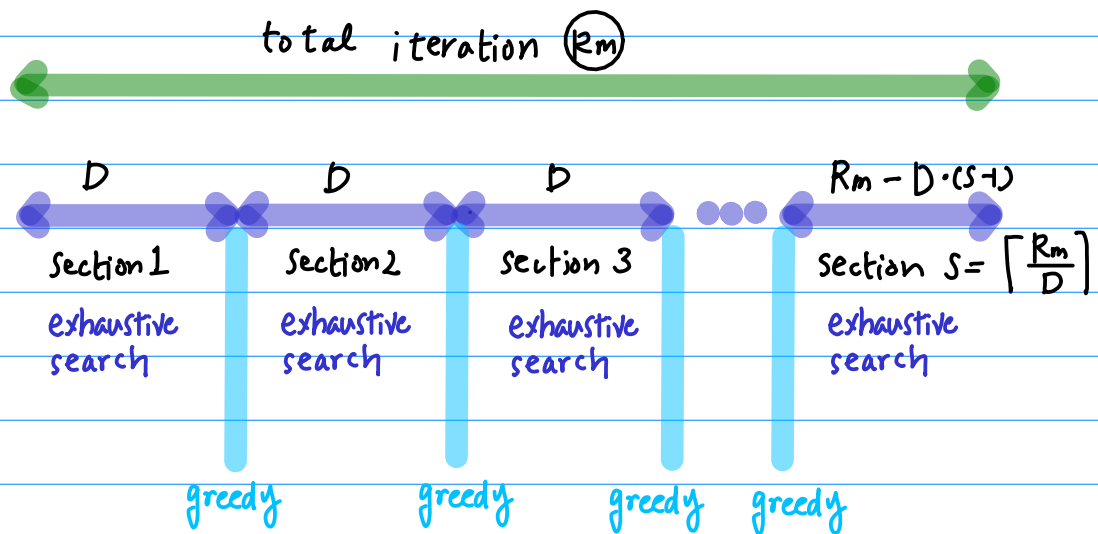
3) Semi-greedy Algorithm

a combination of greedy and exhaustive algorithm

the search space of $\alpha(i)$ and $s(i)$ for $0 \leq i \leq R_m - 1$
are divided into several sections

with D iterations as a segment
 $\downarrow$ block length $\downarrow$ block

the segmentation scheme



in the i -th block

decision of $\alpha(k)$ and $s(k)$ for $iD \leq k \leq (i+1)D-1$

$$\text{minimizes } J = \left| \theta(i) - \sum_{k=iD}^{(i+1)D-1} \alpha(k) a(s(k)) \right|$$

$$\text{where } \theta(i) = \theta - \sum_{m=0}^{i-1} \left[\sum_{k=mD}^{(m+1)D-1} \alpha(k) a(s(k)) \right]$$

the residue angle in the i -th step

$$s = \left\lceil \frac{R_m}{D} \right\rceil$$

$$\theta(i) = \theta - \left[\sum_{k=0D}^{1D-1} \alpha(k) a(s(k)) + \sum_{k=1D}^{2D-1} \alpha(k) a(s(k)) + \dots + \sum_{k=(i-1)D}^{iD-1} \alpha(k) a(s(k)) \right]$$

Initialization:

given Θ , W , R_m

let $\Theta(0) = \Theta$,

$i = 0$

$J(-1) = \infty$

Select $\alpha(k) \in \{-1, 0, +1\}$

$s(k) \in \{0, 1, 2, \dots, W-1\}$

for $iD \leq k \leq (i+1)D - 1$

to minimize $J(i) = \Theta(i) - \sum_{k=iD}^{(i+1)D-1} \alpha(k) a(s(k))$

N

$J(i) < J(i-1)$

$\Theta(i+1) = \Theta(i) - \sum_{k=iD}^{(i+1)D-1} \alpha(k) a(s(k))$

Store $\alpha(k)$, $s(k)$

$i \geq \lceil \frac{R_m}{D} \rceil - 1$

N

Y

$i = i + 1$

```

#include <stdio.h>
#include <math.h>

#define N 16
#define Rm 8

//-----
// conventional cordic
// input initial angle : angle[0]
// ouput residue angles : angle[1] .. angle[N]
//-----
void conventional_cordic(double a[], double angle[]) {
    int k = 0, uk;

    printf("* conventional cordic ... \n");

    for (k=0; k<N; ++k) {
        uk = (angle[k] >= 0) ? +1 : -1;

        printf("k=%2d angle[%2d]=%10.7f ", k, k, angle[k]);
        printf("uk=%+d ", uk);

        angle[k+1] = angle[k] - uk * a[k];

        printf("a[%2d]=%10.7f ", k, a[k]);
        printf("angle[%2d]=%+10.5f \n", k+1, angle[k+1]);
    }
}

```

```

void angle_repeat_cordic(double a[], double angle[]) {
    int k, m, mm, km, i, uk;
    double residue, minval;

    for (k=0; k<N; ++k) {

        mm = -1;
        minval = 1e+100;

        // perform conventional cordic from k-th step
        for (i=k; i<N; ++i) {
            uk = (angle[i] >= 0) ? +1 : -1;
            angle[i+1] = angle[i] - uk * a[i];
            // printf("angle[%2d]=%10f \n", i+1, angle[i+1]);
        }

        uk = (angle[k] >= 0) ? +1 : -1;

        // m: repeating depth
        for (m=1; m+k<N; ++m) {

            // expected residue angle by repeating a[k+m] (m+1) times
            residue = angle[k] - uk * (m+1)*a[k+m];

            // find the best angle repetition
            if (fabs(angle[k+m+1]) >= fabs(residue)) {
                if (minval >= fabs(residue)) {
                    printf("m=%2d residue=%f angle[%2d]=%10f \n",
                        m, residue, k+m+1, angle[k+m+1]);
                    minval = fabs(residue);
                    mm = m;
                }
            }
        }
    }
}

```

```

m = mm;

if (m > 0) {
    km = k+m;
    for (i=0; i<=m; i++) {
        printf("angle[%2d]=%10.7f ", k, angle[k]);
        printf("i=%2d uk=%+d m=%2d ", i, uk, m-i);

        angle[k+1] = angle[k] - uk * a[km];

        printf("a[%2d]=%10.7f, ", km, a[km]);
        printf("angle[%2d]=%+10.5e \n", k+1, angle[k+1]);

        k++;
    }
    k--;
} else {
    printf("angle[%2d]=%10.7f ", k, angle[k]);
    printf("i=%2d uk=%+d m=%2d ", 0, uk, m);

    angle[k+1] = angle[k] - uk * a[k];

    printf("a[%2d]=%10.7f, ", k, a[k]);
    printf("angle[%2d]=%+10.5e \n", k+1, angle[k+1]);
}

} // end of k

}

```

```

int main(void) {
    double a[N], angle[N+1];
    // double theta = 0.63761;
    // double theta = 0.27623;
    double theta; // = 4*atan(pow(2,-5));

    int i, s;

    for (i=0; i<N; ++i) {
        a[i] = atan(1./pow(2, i));
    }

    for (s=2; s<3; ++s) {
        angle[0] = theta = s * atan(pow(2, -6));
        printf("theta= %2d * atan(pow(2,-6) = %10g \n", s, theta);
        conventional_cordic(a, angle);
        angle_repeat_cordic(a, angle);
    }
}

```

```

theta= 2 * atan(pow(2,-6) = 0.0312475
* conventional cordic ...
k= 0 angle[ 0]= 0.0312475 uk=+1 a[ 0]= 0.7853982 angle[ 1]= -0.75415
k= 1 angle[ 1]=-0.7541507 uk=-1 a[ 1]= 0.4636476 angle[ 2]= -0.29050
k= 2 angle[ 2]=-0.2905031 uk=-1 a[ 2]= 0.2449787 angle[ 3]= -0.04552
k= 3 angle[ 3]=-0.0455244 uk=-1 a[ 3]= 0.1243550 angle[ 4]= +0.07883
k= 4 angle[ 4]= 0.0788306 uk=+1 a[ 4]= 0.0624188 angle[ 5]= +0.01641
k= 5 angle[ 5]= 0.0164118 uk=+1 a[ 5]= 0.0312398 angle[ 6]= -0.01483
k= 6 angle[ 6]=-0.0148281 uk=-1 a[ 6]= 0.0156237 angle[ 7]= +0.00080
k= 7 angle[ 7]= 0.0007956 uk=+1 a[ 7]= 0.0078123 angle[ 8]= -0.00702
k= 8 angle[ 8]=-0.0070167 uk=-1 a[ 8]= 0.0039062 angle[ 9]= -0.00311
k= 9 angle[ 9]=-0.0031105 uk=-1 a[ 9]= 0.0019531 angle[10]= -0.00116
k=10 angle[10]=-0.0011573 uk=-1 a[10]= 0.0009766 angle[11]= -0.00018
k=11 angle[11]=-0.0001808 uk=-1 a[11]= 0.0004883 angle[12]= +0.00031
k=12 angle[12]= 0.0003075 uk=+1 a[12]= 0.0002441 angle[13]= +0.00006
k=13 angle[13]= 0.0000634 uk=+1 a[13]= 0.0001221 angle[14]= -0.00006
k=14 angle[14]=-0.0000587 uk=-1 a[14]= 0.0000610 angle[15]= +0.00000
k=15 angle[15]= 0.0000023 uk=+1 a[15]= 0.0000305 angle[16]= -0.00003
angle[ 0]= 0.0312475 i= 0 uk=+1 m=-1 a[ 0]= 0.7853982, angle[ 1]=-7.54151e-01
angle[ 1]=-0.7541507 i= 0 uk=-1 m=-1 a[ 1]= 0.4636476, angle[ 2]=-2.90503e-01
m= 1 residue=-0.041793 angle[ 4]= 0.078831
angle[ 2]=-0.2905031 i= 0 uk=-1 m= 1 a[ 3]= 0.1243550, angle[ 3]=-1.66148e-01
angle[ 3]=-0.1661481 i= 1 uk=-1 m= 0 a[ 3]= 0.1243550, angle[ 4]=-4.17931e-02
angle[ 4]=-0.0417931 i= 0 uk=-1 m=-1 a[ 4]= 0.0624188, angle[ 5]=+2.06257e-02
angle[ 5]= 0.0206257 i= 0 uk=+1 m=-1 a[ 5]= 0.0312398, angle[ 6]=-1.06141e-02
angle[ 6]=-0.0106141 i= 0 uk=-1 m=-1 a[ 6]= 0.0156237, angle[ 7]=+5.00960e-03
angle[ 7]= 0.0050096 i= 0 uk=+1 m=-1 a[ 7]= 0.0078123, angle[ 8]=-2.80274e-03
angle[ 8]=-0.0028027 i= 0 uk=-1 m=-1 a[ 8]= 0.0039062, angle[ 9]=+1.10349e-03
angle[ 9]= 0.0011035 i= 0 uk=+1 m=-1 a[ 9]= 0.0019531, angle[10]=-8.49636e-04
m= 1 residue=0.000127 angle[12]= -0.000361
m= 2 residue=-0.000117 angle[13]= -0.000117
angle[10]=-0.0008496 i= 0 uk=-1 m= 2 a[12]= 0.0002441, angle[11]=-6.05496e-04
angle[11]=-0.0006055 i= 1 uk=-1 m= 1 a[12]= 0.0002441, angle[12]=-3.61355e-04
angle[12]=-0.0003614 i= 2 uk=-1 m= 0 a[12]= 0.0002441, angle[13]=-1.17214e-04
m= 1 residue=0.000005 angle[15]= -0.000056
angle[13]=-0.0001172 i= 0 uk=-1 m= 1 a[14]= 0.0000610, angle[14]=-5.61793e-05
angle[14]=-0.0000562 i= 1 uk=-1 m= 0 a[14]= 0.0000610, angle[15]=+4.85585e-06
angle[15]= 0.0000049 i= 0 uk=+1 m=-1 a[15]= 0.0000305, angle[16]=-2.56617e-05

```

```

void exhaustive_choice(double a[], double ntheta[][N], theta) {
    int u, i;

    printf("* exhaustive searching cordic... \n");

    for (u=0; u<3; u++) {
        for (i=0; i<N; ++i) {

            ntheta[u][i] = theta - (u-1) * a[i];
        }
    }
}

```

```

void exhaustive_cordic(double a[], double angle[], int uk[], int ak[]) {
    int k, u, i;

    printf("* exhaustive greedy searching cordic ... \n");

    for (k=0; k<N; k++) {
        theta = angle[k]

        minval = 1e+100;
        for (u=0; u<3; u++) {
            for (i=0; i<N; ++i) {
                ntheta = theta - (u-1) * a[i];
                if (minval > ntheta) {
                    angle[k+1] = ntheta
                    uk[k] = u;
                    ak[k] = i;
                }
            }
        }
    }
}

```

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define N 2
#define Rm 8

//-----
// (2N+1)-ary tree node
// 1st N choices -a(0), -a(1), ..., -a(N-1)
// 2nd N choices +a(0), +a(1), ..., +a(N-1)
// last choice    zero
//-----
typedef struct node {
    double theta;
    int depth;

    struct node * child[2*N+1];
    struct node * parent;
} nodetype;

//-----
// queue node type
// used for breadth first search traversal
//-----
typedef struct qnode {
    struct node * node;
    struct qnode * next;
} qnodetype;

```



```

//-----
// create a node for the angle tree
//-----
nodetype * create_node() {
    nodetype * p = (nodetype *) malloc (sizeof(nodetype));

    if (p == NULL) {
        perror("node creation error \n");
        exit(1);
    }
    else {
        // for (i=0; i<2N+1; ++i)
        //   child[i] = NULL;
        // child[N] = NULL;
        return p;
    }
}

//-----
// create a node for the queue in order to BFS the angle tree
//-----
qnodetype * create_qnode() {

    qnodetype * q = (qnodetype *) malloc (sizeof(qnodetype));

    if (q == NULL) {
        perror("qnode creation error \n");
        exit(1);
    }
    else {
        // node = NULL;
        // next = NULL;
        return q;
    }
}

```

```

//-----
// create (2N+1) children node to the current node pointed by p
//-----
void expand_node(double a[], nodetype *p) {
    nodetype *np;
    int i;
    double ntheta, theta;

    // printf("* expanding a node... \n");

    theta = p->theta;

    for (i=0; i<2*N+1; ++i) {
        if (i < N)          ntheta = theta + 1 * a[i];
        else if (i < 2*N)  ntheta = theta - 1 * a[i];
        else if (i == 2*N) ntheta = theta + 0 * a[i];

        np = create_node ();
        p->child[i] = np;
        np->parent  = p;
        np->theta   = ntheta;
        np->depth   = p->depth +1;
    }
}

```

```

//-----
// BFS tree traversal to expand all (2N+1) nodes
//-----
qnodetype *head =NULL;
qnodetype *tail =NULL;

void enqueue(qnodetype *q) {
    // printf("* enqueue ... \n");
    if (head == NULL && tail == NULL) head = q;
    if (tail != NULL) tail->next = q;
    tail = q;
}

qnodetype * dequeue() {
    // printf("* dequeue ... \n");
    qnodetype *q;

    if (head != NULL) {
        q = head;

        if (head != tail) head = head->next;
        else head = tail = NULL;

        return q;
    }
    else {
        return NULL;
    }
}

```

```

void tree_traverse(double a[], nodetype *p) {
    qnodetype *q, *nq;
    int i, k = 0;

    printf("* tree traversing ... \n");

    q = create_qnode();
    q->node = p;
    enqueue(q);

    while (head != NULL) {
        printf("* node %d to be expanded \n", k);

        q = dequeue();
        k++;

        if ((q->node)->depth >= N) break;

        if (q != NULL) expand_node(a, q->node);

        for (i=0; i<2*N+1; ++i) {
            nq = create_qnode();
            nq->node = (q->node)->child[i];
            enqueue(nq);
        }
    }
}

```

```

int main(void) {
    double a[N];
    // double angle[N+1];
    // double theta = 0.63761;
    // double theta = 0.27623;
    double theta; // = 4*atan(pow(2,-5));

    int i, s;

    nodetype *p;

    for (i=0; i<N; ++i) {
        a[i] = atan(1./pow(2, i));
    }

    for (s=2; s<3; ++s) {

        theta = s * atan(pow(2, -6));
        // angle[0] = theta;

        printf("theta= %2d * atan(pow(2,-6) = %10g \n", s, theta);

        // conventional_cordic(a, angle);

        p = create_node();
        p->theta = theta;
        p->depth = 0;
        tree_traverse(a, p);
    }
}

```