Group 2
Joni Korpela
Henri Launonen

T1.

Parameters for group 2:

ϕ [deg]	-164.9
x	5.25
y	2.75

a)

For the initialization step we used formulae 10 and 11 from section 2.2.5 of the intro pdf:

```
phi_rad = deg2rad(phi);

if phi_rad < 0
    d_init = -1;
else
    d_init = 1;
end

x0 = -dinit * y;
y0 = dinit * x;
z0 = phi_rad - dinit * pi / 2;
```

b)

i	z_i [deg]	d_i	$atan(2^{-i})$ [deg]	z_{i+1} [deg]
0	-74.9000	-1	45	-29.9
1	-29.9	-1	26.5651	-3.3349
2	-3.3349	-1	14.0362	10.7013
3	10.7013	+1	7.12502	3.5763
4	3.5763	+1	3.57633	approx. 0

c)

i_i	x_iA_i	y_iA_i	d_i	$x_{i+1}A_{i+1}$	$y_{i+1}A_{i+1}$
0	2.75	-5.25	-1	-2.5	-8
1	-2.5	-8	-1	-6.5	-6.75
2	-6.5	-6.75	-1	-8.1875	-5.125
3	-8.1875	-5.125	+1	-7.54688	-6.14844
4	-7.54688	-6.14844	+1	-7.1626	-6.62012

As i grows, the term atan(2^-i) shrinks. This means that more fraction bits are needed as i grows to represent the angle precisely.

d)

The CORDIC gain:

$$A_N = \prod_{i=0}^{4} \sqrt{1 + 2^{-2i}} = \frac{5\sqrt{113594}}{1024} = 1.645689$$

e)

The gain-compensated x and y are:

$$x' = x_4 A_4 / A_N = -4.35234$$

$$y' = y_4 A_4 / A_N = -4.0227$$

Sanity check:

$$\sqrt{x'^2 + y'^2} = 5.92663$$

$$\sqrt{{x_0}^2 + {y_0}^2} = 5.92663$$

The original and transformed vectors' magnitudes are similar.

Let's compare the results to Givens transform:

$${x_{Givens}}' = x\,cos(\phi) - y\,sin(\phi) = -4.35234$$

$${y_{Givens}}' = y\,cos(\phi) - x\,sin(\phi) = -4.0227$$

$$x' = {x_{Givens}}', \quad y' = {y_{Givens}}'$$

Everything is in order.

Group 2
Joni Korpela
Henri Launonen

T2.

Base angle ϕ_0 [degrees]	8
Number of angles *N*	10
Maximum error δ_{max}	1.0×10^{-3}

Points for which to produce samples:

$$t = \phi_0 + n\Delta, \quad n = 0, 1, \dots, N - 1, \quad \Delta = 360 / N$$

With group's parameters plugged in:

$$t = 8 + 36n, \quad n = 0, 1, \dots, 9$$

For finding the iteration count and sp.n format, we used the MATLAB scripts test_wordlength.m and test_wordlength_caller.m.

Through floating-point testing, it was found that minimum CORDIC iterations needed to pass the maximum error criterion is N_iter = 11. The testing was done by changing the iteration count by hand and finding the minimum iteration count that passes for all angles with the given maximum error.

With N_iter = 11, it was found that a sufficient sp.n format for x, y and z is 14.12. The testing was done by calling the ucordic function with a range of xy_p, xy_n, z_p and z_n values, and finding the minimum value combination that passes for all angles with the given maximum error.

T3.

Block to design	CORDIC 3
Max rotation error [deg]	<= 0.2
Angle	-33.75°

We didn't use initialization since the angle is small enough that default CORDIC is enough, also we didn't reuse use angles.

We used two approaches to find a good sequence of rotations for the CORDIC block. The first approach was to use breadth first search to find shortest possible rotation strings that yielded wanted rotation and satisfied error constraint. From these strings we chose one that minimized rotation count and gave lowest error. The best rotation string is given in table 1.

Group 2
Joni Korpela
Henri Launonen

Table 1

Rotation String (d_0, d_1, d_2, d_3)	CORDIC Gain A_N	Error
(0, -1, 0, -1)	1.1267347735824966	0.05993247401905557°

The search was implemented using python file named **task3.py** and the file is attached to this return. In figure 1 the result of rotation strings search is shown.

```
Finding rotation strings for angle: -33.75°

Shortest rotation strings and their errors are:
[0, -1, 0, -1], error = 0.05993247402021229

Best string (least rotations / highest zero count and lowest error) is : [0, -1, 0, -1]
CORDIC gain for the best string: 1.1267347735824966
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Testing the rotation with values (x = -9.617923237055809, y = 6.797315078405326)

Result of the rotation using CORDIC with best string:    (x_cord = -4.232123852514929,     y_cord = 10.990771918568264)
Result of the rotation using Givens:                     (x_giv = -4.220624983053131,      y_giv = 10.995192786915663)

Angle between CORDIC rotated and Givens rotated vector: 0.059932474012974316°
--------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Figure 1

In the second approach we simulated a 3-valued CORDIC with MATLAB, and used the following heuristic to determine the best move:

```
% determine the rotation dir that gets us closest
c = atan(2^(-k));
errs = [abs(z - c), abs(z + c), abs(z - 0)];
[~, idx] = min(errs);
if idx == 1
    d = 1;
elseif idx == 2
    d = -1;
else
    d = 0;
end
```

The heuristic finds the rotation that leaves the least absolute distance to the desired rotation angle. It is basically a greedy algorithm that expects that the optimal choice at each step is the immediate action that brings us the closest to the goal. While it's guaranteed to converge eventually, it might not find the most optimal rotation sequence, as we observe in table 2.

Group 2
Joni Korpela
Henri Launonen

We used the following script to find the minimum number of iterations to pass the max error criterion:

```
N_iter_max = 0;
N_iter_min = 100;

for x = -10:0.1:10
    for y = -10:0.1:10
        N_iter = 5;
        pass = CORDIC_3(x, y, N_iter);
        while pass == false
            N_iter = N_iter + 1;
            pass = CORDIC_3(x, y, N_iter);
        end
        if N_iter < N_iter_min
            N_iter_min = N_iter;
        end
        if N_iter > N_iter_max
            N_iter_max = N_iter;
        end
    end
end

disp('Lowest N_iter to pass: '); disp(N_iter_min);
disp('Minimum N_iter required to pass all'); disp(N_iter_max);
```

At first, we found the minimum number of iterations to be 10, but this was actually not the case. The reason for this long rotation sequence is that MATLAB's arctan function returns degrees in range +-180, and this iteration of the loop had an unfortunately positioned vector, which was positioned just at the threshold after rotating, leading to a wraparound error:

```
Rotation directions:
    -1     0     1     0    -1     0     1     0     0

original angle:
 -146.3099

true rotated angle:
  179.9401

CORDIC rotated angle:
 -179.9548

Angular error:
359.89491720
Maximum error exceeded!
```

Actually, the error was well within the error criterion.

Group 2
Joni Korpela
Henri Launonen

When limiting the vector to positive (x,y) values we got more sensible results:

```
for x = 0:0.1:10
    for y = 0:0.1:10
```

```
Rotation directions:
    -1     0     1     0    -1     0     1

original angle:
    45

true rotated angle:
   11.2500

CORDIC rotated angle:
   11.3551

Angular error:
0.10508280
```

Checking the sequence by hand we can see that the rotations add up to around -33.645, which is well within the bounds with a reasonable amount of rotations.

The CORDIC gain with this sequence is as follows:

$$A_N = \prod_{k=0}^{N-1} \sqrt{1 + 2^{-2k}} = \sqrt{1 + 2^0} \cdot \sqrt{1 + 2^{-4}} \cdot \sqrt{1 + 2^{-8}} \cdot \sqrt{1 + 2^{-12}}$$

$$= 1.460761$$

Table 2

Rotation String $(d_0, d_1, d_2, d_3, d_4, d_5, d_6)$	CORDIC Gain A_N	Errors
(-1, 0, 1, 0, -1, 0, 1)	1.460761	0.10508280° 0.79009091°

As can be seen in table 2, the greedy heuristic loses to the breadth first search in terms of rotation count and error.

The files used for this method are CORDIC_3.m and CORDIC_3_caller.m, and they are included in the return files.