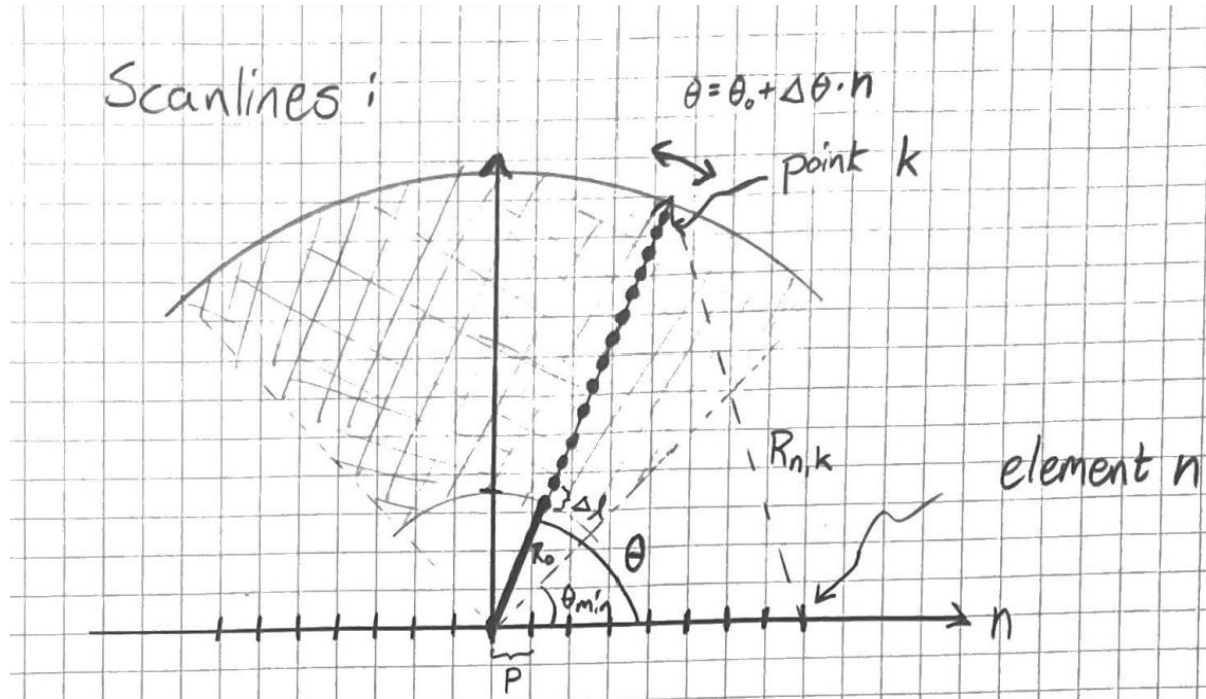


Status på prosjektoppgave

Gustav Kollstrøm

Systematisk scanning ved bruk av scanlines:

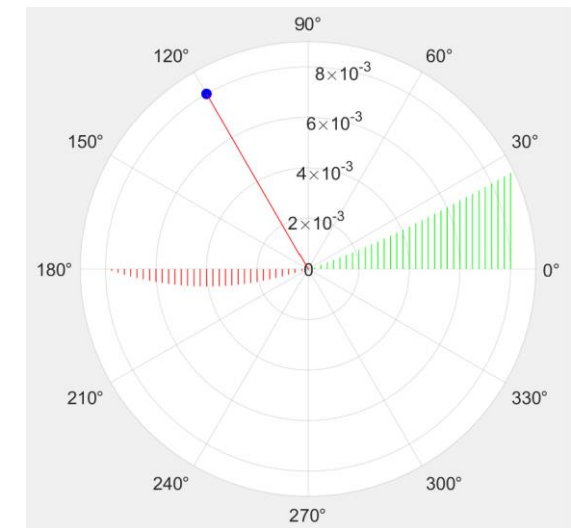


Distanse fra hvert element n til scanpunkt k :

$$R_{n,k}^2 = (np)^2 + R_0^2 - 2npR_0 \cos \theta + k(k\Delta l^2 + 2R_0\Delta l - 2np\Delta l \cos \theta)$$

Delay in clock cycles can be written:

$$N_{n,k}^2 = \left(\frac{f_s}{V_s}\right)^2 R_0^2$$



Algoritme for å regne ut delay mest mulig effektivt

1. $N_{0,0}^2 = \left(\frac{f_s}{v_s}\right)^2 R_0^2$
2. $N_{n+1,0}^2 = N_{n,0}^2 + A_0(2n+1) - C_0$
3. $N_{n,k+1}^2 = N_{n,k}^2 + 2k+1 + B_n$
4. Repetér for neste vinkel

Delay i referansepunkt/element (origo, $n = 0, k = 0$)

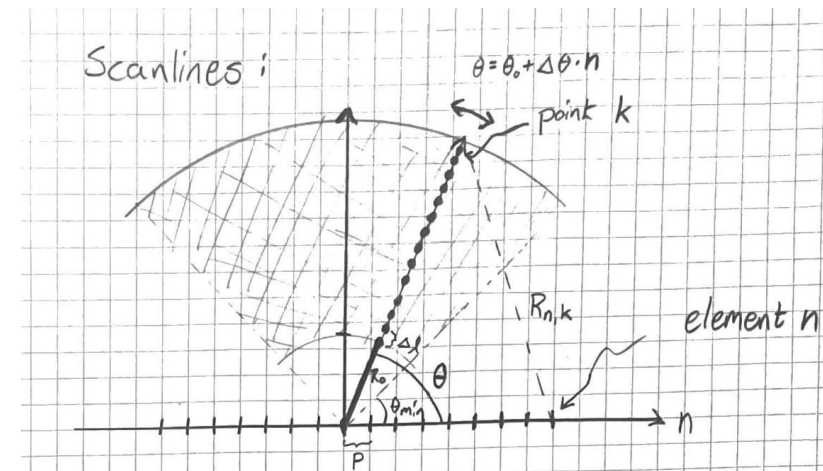
Delay i neste transducer element for punkt $k=0$, iterativt uttrykt vha forrige element

Delay i neste scanpunkt for element n , iterativt uttrykt vha forrige element

$$\left(\begin{aligned} A_0 &= \left(\frac{f_s}{v_s} P\right)^2 \\ C_0 &= \left(\frac{f_s}{v_s}\right)^2 \cdot 2pR_0 \cos\theta \\ B_n &= 2R_0 \frac{v_s}{f_s} - 2np \frac{v_s}{f_s} \cos\theta \end{aligned} \right) \quad \Delta l = \frac{v_s}{f_s}$$

Alle verdier konstante utenom $\cos(\theta)$

flipp v_s og f_s



Fordeler og utfordringer ved denne metoden

- Fordeler

- Simplifiserer utregningen i stor grad
- Skalérbart: Lett å implementere flere parallelle blokker i hvert steg for å øke throughput
- Bare den første utregningen for hvert iterative uttrykk er avhengig av forrige steg
 - Alle stegene kan gjøres parallelt, ingen trenger å vente på forrige steg

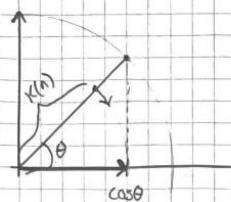
- Utfordringer

- Avhengig av $\cos(\theta)$, multiplikasjon og kvadratroten

Potensielle løsninger

CORDIC

To calculate $\cos \theta$, CORDIC algorithm can be used:



$$x_{i+1} = x_i - \sigma_i y_i 2^{-i} \quad x_0 = 1$$

$$y_{i+1} = y_i + \sigma_i x_i 2^{-i} \quad y_0 = 0$$

$$\theta_{i+1} = \theta_i - \sigma_i \gamma_i \quad \theta_0 = \theta$$

$$\sigma_i = \begin{cases} 1 & \theta_i > 0 \\ -1 & \theta_i < 0 \end{cases} \quad \sigma_0 = \begin{cases} 1 & \theta_0 > 0 \\ -1 & \theta_0 < 0 \end{cases}$$

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

$\cos(\theta) \approx K(n) \cdot x_n$

$$\sin(\theta) \approx K(n) \cdot y_n$$

$\sim 10-20$ iterations needed

Tilnærming av kvadratroten

Square root approximation

$$y = \sqrt{x}$$

$$= \sqrt{x + 2^m - 2^m}$$

$$= \sqrt{2^m \left(\frac{x}{2^m} + 1 - 1 \right)}$$

$$= 2^{\frac{m}{2}} \sqrt{\frac{x - 2^m}{2^m} + 1}$$

$$= 2^{\frac{m}{2}} \sqrt{x' + 1}$$

$$x' = \frac{x - 2^m}{2^m} < 1$$

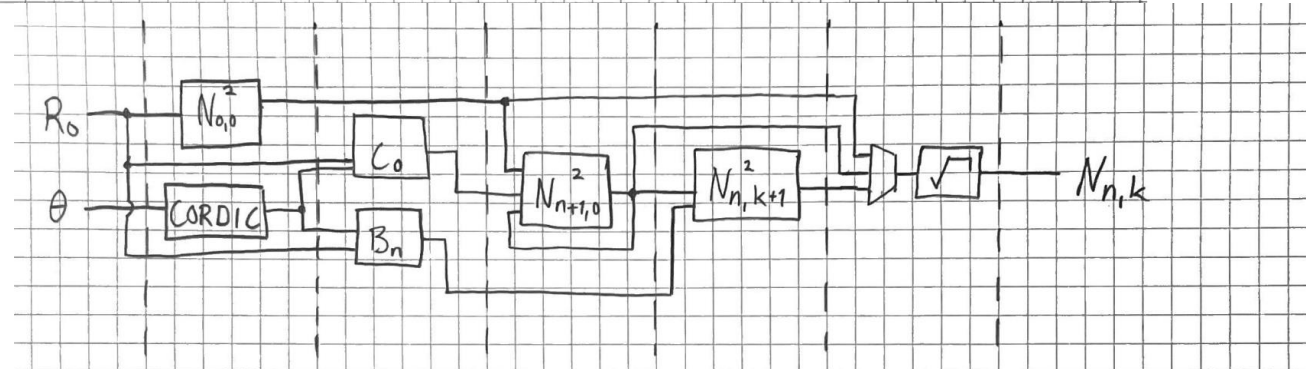
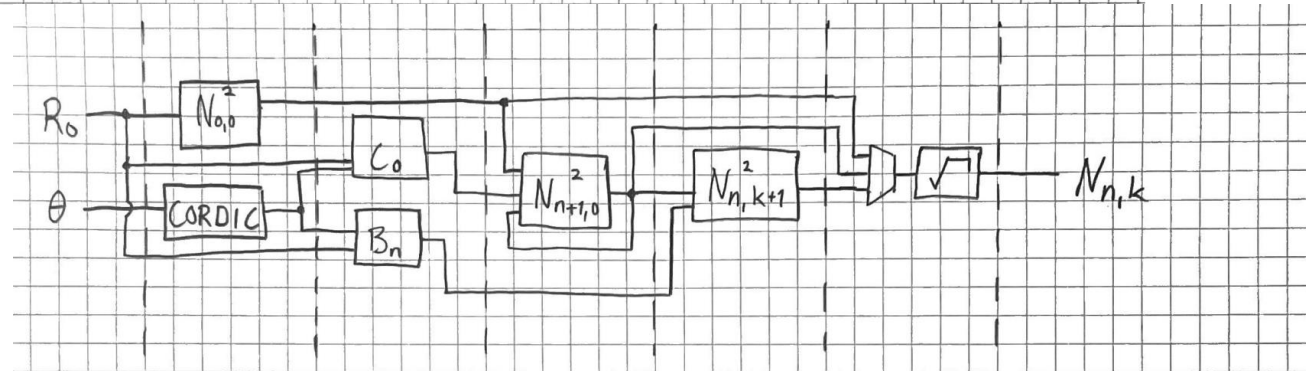
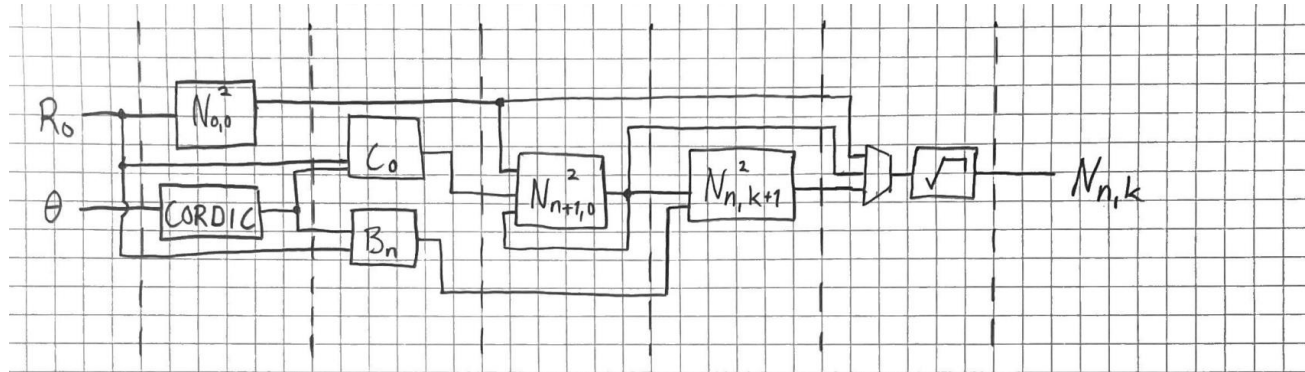
$$\sqrt{x' + 1} \approx p + q \cdot x' - \frac{x'^2}{r}$$

Kommentarer

CORDIC løser problemet med $\cos(\theta)$ og potensielt multiplikasjon ettersom $K(n)$ kan skaleres inn i x_0 før utregning.

Denne metoden for tilnærming av kvadratroten blir nok den tregeste delen av systemet ettersom den trenger en del iterasjoner for hver utregning, og den må skje for hver delay-verdi. Det gjenstår å se hvor mange iterasjoner som kreves for tilstrekkelig presisjon.

Proposed system diagram + pipelining iterations of θ



Status og veien videre

- Status
 - Algoritme som reduserer kompleksitet og øker effektivitet på utregningene
 - Areal, power og speed
- Veien videre
 - Se på muligheter for å unngå/simplifisere kvadratroter
 - Implementere algoritmen i Python/MATLAB og sammenligne med teoretisk utregning
 - Forberedelser før implementasjon
 - Tilpasse systemet slik at det er innenfor tidsfrister med minst mulig parallelle blokker
 - Se på utvidelse av algoritmen til 3D?