Implementation of Particle Filter-based Target Tracking

V. Rajbabu
rajbabu@ee.iitb.ac.in
VLSI Group Seminar

Dept. of Electrical Engineering
IIT Bombay

November 15, 2007
Outline

1. Introduction
2. Target Tracking
3. Bayesian Estimation
4. Particle Filter
5. Implementation
Outline

1. Introduction
2. Target Tracking
3. Bayesian Estimation
4. Particle Filter
5. Implementation
### Introduction

#### Problem scenario
- Multi-target tracking - batch measurements [Volkan Cevher]
- Particle filters - computational complexity
- Sensors operating under constrained environment

#### Problem Illustration

#### Objective
Efficient digital implementation of particle filters
- Real-time
- Low-power
Outline

1. Introduction
2. Target Tracking
3. Bayesian Estimation
4. Particle Filter
5. Implementation
What is tracking?

- **Tracking** - process measurements to sequentially estimate hidden states
  - Target tracking
  - Visual tracking

Applications

- Surveillance and monitoring
- Medical imaging
- Robotics
- Motion in sports
We consider multi-target tracking

- State vector - target or vehicle kinematic characteristics, ex., 2-D position and velocity, or motion parameters in polar coordinates
- Measurements - range or angle with respect to sensor
State-space Formulation

- Dynamic state-space problem - sequential
- State-space model depends on physics of the problem

**System transition equation**

\[ x_t = f_t(x_{t-1}, u_t), \quad u_t \text{ – system noise} \]

\[ f_t() \text{ – system evolution function (possibly nonlinear)} \]

**Observation equation**

\[ y_t = g_t(x_t, v_t), \quad v_t \text{ – measurement noise} \]

\[ g_t() \text{ – measurement function (possibly nonlinear)} \]
Bearings-only Tracking

Automated estimation of a moving target’s state using angle measurements (bearings)

- State update equation

\[ X_t = FX_{t-1} + Gu_t, \]

where \( X_t = \begin{bmatrix} x & v_x & y & v_y \end{bmatrix}_T \), \( u_t = \begin{bmatrix} u_x & u_y \end{bmatrix}_T \),

\( F = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \) and \( G = \begin{pmatrix} 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0 & 1 \end{pmatrix} \).

- Measurement equation

\[ z_t = \arctan \left\{ \frac{y_t}{x_t} \right\} + r_t \]
Batch-based Tracking – State Vector

- Multiple target state vector - independent partitions
- Constant velocity during batch period $T$

$$X_t = [x_1^T(t), x_2^T(t), \ldots, x_k^T(t)]^T$$

$$x_k(t) = \begin{bmatrix} \theta_k(t) \\ R_k(t) \\ v_k(t) \\ \phi_k(t) \end{bmatrix}, k - target index$$

- $\theta_k(t)$ — direction-of-arrival (DOA)
- $R_k(t) \triangleq \log \text{range}$ (range)
- $v_k(t)$ — velocity
- $\phi_k(t)$ — heading direction
Batch-based Tracking – Observation Model

- Image template-matching - batch measurements
  - DOA measurements from acoustic sensor - beamforming
  - Range measurements from radar sensor
- Observability - batch of minimum three measurements

\[ y_t = \{ y_{t+m\tau}(p) \}_{m=0}^{M-1} \]

Figure: Template-matching
State-Space Model

Batch-based range tracking

Non-linear state transition equation

\[
\begin{bmatrix}
\theta_k(t+T) \\
R_k(t+T) \\
v_k(t+T) \\
\phi_k(t+T)
\end{bmatrix}
= \begin{bmatrix}
\tan^{-1}\left(\frac{e^{R_k} \sin \theta_k + T v_k \sin \phi_k}{e^{R_k} \cos \theta_k + T v_k \cos \phi_k}\right) \\
\frac{1}{2} \log \left\{ e^{2R_k + T^2 v_k^2 + 2T e^{R_k} v_k \cos(\theta_k - \phi_k)} \right\} \\
v_k \\
\phi_k
\end{bmatrix} + u_k(t)
\]

where \( u_k(t) \) — Gaussian process noise

Observation likelihood

\[
p(y(t)|x_k(t)) \propto \prod_{m=0}^{M-1} \left\{ 1 + \frac{1-\kappa}{\sqrt{2\pi \sigma^2_f \kappa \lambda}} \sum_{p_i} \exp \left\{ -\frac{(h_{m\tau}(x_k(t))-y_{t+m\tau}(p_i))^2}{2\sigma^2_f} \right\} \right\}
\]
Outline

1. Introduction
2. Target Tracking
3. Bayesian Estimation
4. Particle Filter
5. Implementation
Objective - estimate hidden state $x_t$ from observations $y_t$

Probabilistic description - using pdfs

- prior distribution - $p(x_0)$
- state transition - $p(x_t|x_{t-1})$
- data likelihood - $p(y_t|x_t)$

Bayesian estimation - minimum mean square estimate

Conditional mean $E(x_t|y_t)$ of the posterior distribution

$$p(x_t|y_t) \propto p(y_t|x_t) \cdot p(x_t|x_{t-1})$$
Sequential Representation

Cumulative state up to time $t$: $X_t = \{x_j, j = 0, \ldots, t\}$

Cumulative measurement up to time $t$: $Y_t = \{y_j, j = 0, \ldots, t\}$

Bayesian estimation

Estimate $x_t$ based on all available measurements up to $t$ by constructing the posterior $p(X_t|Y_t)$
Recursive Filter

- Consists of two steps
  - **Prediction step:** \( p(x_{t-1}|Y_{t-1}) \rightarrow p(x_t|Y_{t-1}) \)
  - **Update step:** \( p(x_t|Y_{t-1}), y_t \rightarrow p(x_t|Y_t) \)

- Given the *pdf*s
  - **Prediction step:**
    \[
    p(x_t|Y_{t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|Y_{t-1})dx_{t-1}
    \]
  - **Update step:**
    \[
    p(x_t|Y_t) = \frac{p(y_t|x_t)p(x_t|Y_{t-1})}{p(y_t|Y_{t-1})}
    \]

  where, \( p(y_t|Y_{t-1}) = \int p(y_t|x_t)p(x_t|Y_{t-1}) \)
Recursive Filter

- **Recursive** application of **prediction** and **update** steps provides the optimal Bayesian solution

- Minimum mean square estimate (MMSE) is the conditional mean - $E(x_t | Y_t)$

- Analytically **intractable** - integrals and *pdf*s
Optimal solution for the recursive problem exists

- Kalman filter - optimal solution if
  - state and measurement models - linear, and
  - state and measurement noises - Gaussian

- Extended Kalman filter (EKF) - extension of Kalman filter
  - state and/or measurement models - nonlinear, and
  - state and measurement noises - Gaussian
Outline

1. Introduction
2. Target Tracking
3. Bayesian Estimation
4. Particle Filter
5. Implementation
Particle Filter

- Suboptimal filter - for nonlinear systems and non-Gaussian noise
  - handles nonlinearity as such - without linearisation
  - handles multimodal distributions
- Based on Monte Carlo methods
  - Monte Carlo - “randomly chosen”
- Other names
  - Sequential Monte Carlo (SMC) methods
  - Bootstrap filter
  - Sequential Importance Sampling (SIS) filter
  - CONDENSATION - CONditional DENSity propogATION
To numerically evaluate: \( I = \int q(x) dx \) where \( x \in \mathbb{R}^n \)

**Monte Carlo (MC) method**

Factorize: \( q(x) = f(x) \cdot \pi(x) \), s.t., \( \pi(x) \) is a pdf. We can draw \( N \) samples \( \{ x^i; i = 1, \ldots, N \} \)

MC estimate of \( I = \int f(x)\pi(x) dx \) is the sample mean

\[
I_N = \frac{1}{N} \sum_{i=1}^{N} f(x^i)
\]

What if it is difficult to draw samples from \( \pi(x) \)?
Importance Sampling

- Choose an **Importance distribution** \( g(x) \), such that:

\[
\pi(x) > 0 \Rightarrow g(x) > 0 \text{ for all } x \in \mathbb{R}^n.
\]

and is easy to draw samples from \( g() \).

- **MC estimate** - generate samples \( \{x^{(i)}\} \sim g(x) \)

\[
I_N = \frac{1}{N} \sum_{i=1}^{N} f(x^{(i)}) \tilde{w}(x^{(i)})
\]

where, \( \tilde{w}(x^{(i)}) = \frac{\pi(x^{(i)})}{g(x^{(i)})} \).
Importance Sampling

Target distribution \( \pi(x) \)

Proposal distribution \( g(x) \)

Random Support: \( x^{(j)} \sim \mathcal{N}(\mu, \sigma^2) \)

Unnormalized weights: \( w_u^j = \frac{\pi(x^{(j)})}{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(x^{(j)}-\mu)^2}{2\sigma^2} \right\} } \)

Target distribution's expectation:

\[
E_{\pi}\{F(x)\} \approx \frac{\sum_{j=1}^{20} F(x^{(j)}) w_u^j}{\sum_{j=1}^{20} w_u^j}
\]
Particle Filter - (1/2)

Use randomly chosen “particles” to represent posterior distribution

\[
\left\{ x_t^{(i)}, w_t^{(i)} \right\}_{i=1}^{N},
\]

- \( x_t^{(i)} \): support points
- \( w_t^{(i)} \): associated weights

\( N \) – number of particles

Discrete weighted approximation to the posterior

\[
p(x_t | Y_t) \approx \sum_{i=1}^{N} w_t^{(i)} \delta(X - X_t^{(i)})
\]
Discrete approximation of distribution using **Importance Sampling**
- Particles - determine the 'support' region
- Weights - proportional to probabilities
- Proposal or importance function plays a critical part

Recursive update by propagating 'particles' and 'weights'
Use updated distributions to obtain estimates
Degeneracy of particles - after a few iterations most particles have negligible weights

Resampling
  - Eliminate or replicate particles depending on their importance weights

Image adapted from Dr. Volkan Cevher
Sequential Importance Resampling (SIR) Particle Filter

Given the observed data $y_k$ at $k$, do

1. For $i = 1, 2, \ldots, N$,
   Sample particles: $x_k^{(i)} \sim g(x_k|x_{k-1}^{(i)}, y_k)$.

2. For $i = 1, 2, \ldots, N$,
   Calculate the importance weights:
   \[
   w_k^{(i)} = \frac{p(y_k|x_k^{(i)})p(x_k^{(i)}|x_{k-1}^{(i)})}{g(x_k^{(i)}|x_{k-1}^{(i)}, y_k)}.
   \]
   For $i = 1, 2, \ldots, N$,
   Normalize the weights:
   \[
   \tilde{w}_k^{(i)} = \frac{w_k^{(i)}}{\sum_{j=1}^{N} w_k^{(j)}}.
   \]

3. Calculate the state estimates:
   \[
   E\{f(x_t)\} = \sum_{i=1}^{N} \tilde{w}_t^{(i)} f(x_t^{(i)}).
   \]

4. Resample \(\left\{x_k^{(i)}, \tilde{w}_k^{(i)}\right\}\) to obtain new set of particles
   \(\left\{x_k^{(j)}, w_k^{(j)} = \frac{1}{N}\right\}\).
PF Flow - Suboptimal Proposal Function

- **Initialization**
- **Propose Particles**
- **Evaluate Weights**
  \[ w^*(i) = w^{(i)} \cdot p(y_t | X_t^{(i)}) \]
- **Normalize Weights**
- **Resample Particles**
- **Measurements**
- **Estimate States**
  \[ \sum_{i=1}^{N} w_t^{(i)} X_t^{(i)} \]
- **Output**

**Proposal function - state update**
Introduction
Target Tracking
Bayesian Estimation
Particle Filter
Implementation

PF Flow - Optimal Proposal Function

- Proposal function - full-posterior

\[ w^*(i)_t = w(i)_t \frac{p(y_t|x_t(i))p(x_t(i)|x_{t-T})}{\prod_k g_k(x_k^{(i)}(t)|y_t,x_k^{(i)}(t-T))} \]

Initializaton \rightarrow Propose Particles \rightarrow Evaluate Weights \rightarrow Normalize Weights \rightarrow Resample Particles

Measurements \rightarrow Estimate States

\[ \sum_{i=1}^{N} w_t(i) X_t^{(i)} \]

Output
Example: 1D Estimation

Estimate states of a nonlinear, non-stationary state space model

State dynamic equation

\[ x_k = 0.5x_{k-1} + \frac{25x_{k-1}}{1 + x_{k-1}^2} + \cos(1.2(k - 1)) + w_k \]

Measurement equation

\[ y_k = \frac{x_k^2}{20} + v_k \]

\( w_k \) and \( v_k \) are zero-mean, Gaussian white noise.
Example - 1D Estimation

State \( x_k \) of a 1D model

- True value
- Posterior mean estimate

Posterior using Particle Filters

- Posterior density
Target Tracking Results

Tracking result
Outline

1. Introduction
2. Target Tracking
3. Bayesian Estimation
4. Particle Filter
5. Implementation
Design Hierarchy

Various stages in developing and implementing a particle filter algorithm

SYSTEM LEVEL
- Mutliple Target Tracking

ALGORITHM LEVEL
- Particle Filters
- Floating-point to Fixed-point
- Newton search
- Proposal Fcn.

ARCHITECTURE LEVEL
- Digital
- Mixed-mode
- Analog
- DSP
- FPGA
- ASIC
- FPAA

CIRCUIT LEVEL
- MITEs
Various stages in developing and implementing a particle filter algorithm
DSP Implementation

- Floating-point DSP – TI C6713
- Internal memory (IRAM) or External memory (SDRAM)
  - Speed
  - Size
- Sampling rate of \( \sim 0.3 \) seconds, for \( K = 1, N = 1000, \) 225 MHz

**Table:** Memory sizes in the C6713-DSK - Single-target, \( N = 1000 \)

<table>
<thead>
<tr>
<th>Type</th>
<th>Section</th>
<th>Available</th>
<th>Occupied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal</td>
<td>IRAM</td>
<td>192 KB</td>
<td>190.16 KB</td>
</tr>
</tbody>
</table>
FPGA Realization

- Matlab simulation
  - Finite word length identified from Matlab fixed-point simulation

- Xilinx System generator - Simulink based tool
  - Xilinx blocks - bit and cycle true FPGA code
  - Nonlinear functions using CORDIC algorithm
  - Device: Xilinx Virtex II Pro

- Xilinx Embedded Development Kit (EDK)
  - To use MicroBlaze soft-core and Power PC hard-core
For each particle $i=1....N$

$$\begin{bmatrix} \theta_k^{(i)}(t) \\ \varphi_k^{(i)}(t) \\ \dot{\varphi}_k^{(i)}(t) \end{bmatrix}$$

DOA UPDATE

$$h_{mv}(x_k^{(i)}(t))$$

Uses exp, atan, sin, cos

$$\begin{bmatrix} \theta_k^{(i)}(1), \ldots, \theta_k^{(i)}(M) \end{bmatrix}$$

$$\exp\left(-\frac{(\cdot)^2}{2\sigma^2}\right)$$

A

$$\sum_p$$

$$\prod_m$$

P\left(y|x_k^{(1)}\right)

NEWTON'S SEARCH

B

X_m, \Sigma_y

\begin{bmatrix} \theta_k^{(i)}(t-T) \\ \varphi_k^{(i)}(t-T) \\ \dot{\varphi}_k^{(i)}(t-T) \end{bmatrix}

STATE UPDATE

$$h_T(x_k^{(i)}(t-T))$$

Uses exp, atan, log functions

OTHER PARTICLES

GENERATE A GAUSSIAN SAMPLE

\begin{bmatrix} \mu_g, \Sigma_g \end{bmatrix}

PROPOSED PARTICLE STATE

X_t^{(i)}

TO PARTICLE WEIGHTING STAGE

\begin{bmatrix} y_{11} & \cdots & y_{1M} \\ \vdots & \ddots & \vdots \\ y_{p1} & \cdots & y_{pM} \end{bmatrix}$$

DOA MEASUREMENT
For each particle $i=1,\ldots,N$

- **PARALLEL or SEQUENTIAL PROCESSING**
- **DOA UPDATE**
  - $\theta_{k}^{(i)}(t)$
  - $Q_{k}^{(i)}(t)$
  - $\phi_{k}^{(i)}(t)$
  - Uses $\exp$, $\tan$, $\sin$, $\cos$

- **DOA MEASUREMENT**
  - $\begin{bmatrix} y_{11} \\ \vdots \\ y_{1M} \\ \vdots \\ y_{p1} \\ \vdots \\ y_{pM} \end{bmatrix}$

- **$\prod_{m=1}^{M} P(y|x_{k}^{(1)})$**

- **$\prod_{m=1}^{M} P(y|x_{k}^{(N)})$**

- **NEWTON'S SEARCH**

- **FPGA soft-core or hard-core processor**

- **A**

- **B**

- **OTHER PARTICLES**

- **$\sum_{p}$**
Particle Weight Evaluation Stage

\[ X^{(i)} \] [3 \times 1]
\[ \mathbf{x}^{(i)} \]
\[ \mathbf{\mu} \mathbf{g} \] [3 \times 1]

\[ \Sigma_{-1} \] [3 \times 3]
\[ \Sigma_{\mathbf{u}} \] [3 \times 3]
\[ \Sigma_{\mathbf{g}} \] [3 \times 3]

\[ p(y_t|x_t^{(i)}) \]

\[ w^{(i)}(t) \]
\[ w^{(i)}(t + T) \]
Reampling Stage

Resampling

- Compute number of replications based on particle weight
- Control circuitry and logic functions

Figure: Residual Systematic Resampling

1: \( U \sim U[0, 1] \)
2: \( K = M/W_n \)
3: \( ind_r = 0, ind_d = M - 1 \)
4: \( \text{for } m=1 \text{ to } M \text{ do} \)
5: \( \text{temp} = w_n^{(m)} \cdot K - U \)
6: \( r^{ind_r} = \left\lceil \text{temp} \right\rceil \)
7: \( U = \text{temp} - r^{ind_r} \)
8: \( \text{if } r^{ind_r} > 0 \text{ then} \)
9: \( i_r^{(ind_r)} = m, \text{ind}_r = \text{ind}_r + 1 \)
10: \( \text{else} \)
11: \( i_d^{(ind_d)} = m, \text{ind}_d = \text{ind}_d - 1 \)
12: \( \text{end if} \)
13: \( \text{end for} \)
Approximation for Data Likelihood

- Data likelihood

\[
p(y(t)|x_k(t)) \propto \prod_{m=0}^{M-1} \left( 1 + \frac{1 - \kappa}{\sqrt{2\pi\sigma_\theta^2}} \sum_{\rho_i} \exp \left\{ - \frac{(h_{m\tau}^\theta(x_k(t)) - y_t + m\tau(\rho_i))^2}{2\sigma_\theta^2} \right\} \right)
\]

DOA component of nonlinear state transition

\[
h_{m\tau}^\theta(x_k(t)) = \arctan \left( \frac{\sin(\theta_k(t)) + T \cos(\phi_k(t)) e^{(Q_k(t))}}{\cos(\theta_k(t)) + T \sin(\phi_k(t)) e^{(Q_k(t))}} \right)
\]

- Laplace method to approximate distribution
  - Newton search – Jacobians and Hessians
Hardware implementation of square-root and division use Newton-Raphson search

Difficulty in using Newton search to identify mode

- Cost function, Hessian, and Gradient evaluation - nonlinear functions

Software implementation in floating-point - MicroBlaze soft-core or Power PC hard-core processor

- Avoids sensitivity to word length
- Accelerated by configurable hardware
Newton search in FPGA (2/2)

- **MicroBlaze - soft-core IP (processor)**
  - Configurable - has 64 kB RAM for code and data
  - Floating-point unit (FPU) - accelerates floating-point operations

- **Power PC 405 - hard-core IP (processor)**
  - Non-Configurable - has 128 kB for data and 64 kB for code
  - Floating-point operations are emulated in software

- **Newton search for identifying mode** - code size $> 100$ kB
Batch-based tracker utilizes at least four times more resources
  Excluding Newton-search
Requires large, recent FPGA device

Table: Resource utilization - Batch-based tracker

<table>
<thead>
<tr>
<th>Resource</th>
<th>PF Stage</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proposal</td>
<td>Weight evaluation</td>
<td>Resampling</td>
<td>Overall</td>
</tr>
<tr>
<td># Slices</td>
<td>7803</td>
<td>8869</td>
<td>374</td>
<td>17046</td>
</tr>
</tbody>
</table>

†a For comparison, a 16 bit multiplier uses 153 slices

Table: Resource utilization - Bearings-only tracker

<table>
<thead>
<tr>
<th>Resource</th>
<th>PF Stage</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proposal</td>
<td>Weight evaluation</td>
<td>Resampling</td>
<td>Overall</td>
</tr>
<tr>
<td># Slices</td>
<td>2700</td>
<td>1215</td>
<td>374</td>
<td>4635</td>
</tr>
</tbody>
</table>
## FPGA - Latency

**Table:** Update rate using FPGA Clock frequency of 100 MHz

<table>
<thead>
<tr>
<th>Latency</th>
<th>Batch-based tracker</th>
<th>Bearings-only tracker</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3N + 307$</td>
<td></td>
<td>$3N + 50$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update rate</th>
<th>$N = 200$</th>
<th>$N = 1000$</th>
<th>$N = 200$</th>
<th>$N = 1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$9 \mu s$</td>
<td>33 $\mu s$</td>
<td>3.5 $\mu s$</td>
<td>30 $\mu s$</td>
<td></td>
</tr>
</tbody>
</table>

- Update rate of $9 \mu s$ is sufficient to generate estimates every $T = 1$ s
Complete batch-based particle filter algorithm on TI C6713

**Figure:** Target DOAs

**Figure:** Target x-y tracks
FPGA Implementation Results

Propose particles → Evaluate weights → Estimate states

Measurements

Resample particles

Figure: FPGA simulation setup.
FPGA Implementation Results

FPGA implementation of data likelihood evaluated in ModelSim

**Figure:** Target DOAs

**Figure:** Target $x$-$y$ tracks
Comparison of implementation strategies for particle filters

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>DSP</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td><strong>Power dissipation</strong></td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td><strong>Implementation flexibility</strong></td>
<td>High</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Discussion (1/2)

Particle filter characteristics

- Number of particles $N$ can be large - complexity
- Parallel computations for individual particles
  - Resampling can prevent parallelization - parallel resampling
- Nonlinear functions

Constraints

- Real-time operation
- Low power
- Accuracy - word length
Digital implementation of particle filter
- Multiple FPGAs
- SIMD architectures
  - General purpose Graphical processing units (GP-GPUs)
- Random number generators

Analog or mixed-mode implementation of particle filter
- Nonlinear functions in analog - arctan, Gaussian
- Random number or noise generation
Thanks to

- Advisor Prof. James H. McClellan, Georgia Tech.
- Dr. Volkan Cevher, Rice University.

SMC Webpage: http://www-sigproc.eng.cam.ac.uk/smc/

A. Doucet and N. Freitas and N. Gordon
*Sequential Monte Carlo Methods in Practice.*

V. Cevher, R. Velmurugan, and J. H. McClellan
Acoustic Multi-Target Tracking using Direction-of-Arrival Batches
Thanks!

Questions
Sequential Importance Sampling (SIS) Particle Filter

Illustration

Source: Michael Isard’s CONDENSATION demos
http://www.robots.ox.ac.uk/~misard/condensation.html
CORDIC Algorithm

COordinate Rotation DIgital Computer (CORDIC) algorithm

CORDIC iterations

\[
\begin{align*}
x^{(i+1)} &= x^{(i)} - d_i y^{(i)} 2^{-i} \\
y^{(i+1)} &= y^{(i)} + d_i x^{(i)} 2^{-i} \\
z^{(i+1)} &= z^{(i)} - d_i \tan^{-1}(2^{-i})
\end{align*}
\]

Rule: Choose \(d_i \in [-1, 1]\) such that \(z \to 0\)

\[
x_{n+1} = x_n - md_n y_n 2^{-\sigma(n)}
\]

\[
y_{n+1} = y_n + d_n x_n 2^{-\sigma(n)}
\]

\[
z_{n+1} = z_n - w_\sigma(n)
\]

<table>
<thead>
<tr>
<th>Type</th>
<th>(m)</th>
<th>(\sigma_n)</th>
<th>(d_n = \text{sign} z_n)</th>
<th>(d_n = -\text{sign} y_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>circular</td>
<td>1</td>
<td>(n)</td>
<td>(x_n \to K(x_0 \cos z_0 - y_0 \sin z_0))</td>
<td>(x_n \to K \sqrt{x_0^2 + y_0^2})</td>
</tr>
<tr>
<td>tan(^{-1}) 2(^{-k})</td>
<td></td>
<td></td>
<td>(y_n \to K(y_0 \cos z_0 + x_0 \sin z_0))</td>
<td>(y_n \to 0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(z_n \to 0)</td>
<td>(z_n \to z_0 - \tan^{-1} \frac{y_0}{x_0})</td>
</tr>
<tr>
<td>hyperbolic</td>
<td>-1</td>
<td>(n - k)</td>
<td>(x_n \to K'(x_1 \cosh z_1 + y_1 \sinh z_1))</td>
<td>(x_n \to K \sqrt{x_0^2 + y_0^2})</td>
</tr>
<tr>
<td>tanh(^{-1}) 2(^{-k})</td>
<td></td>
<td></td>
<td>(y_n \to K'(y_1 \cosh z_1 + x_1 \sinh z_1))</td>
<td>(y_n \to 0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(z_n \to 0)</td>
<td>(z_n \to z_0 - \tanh^{-1} \frac{y_0}{x_0})</td>
</tr>
</tbody>
</table>
## FPGAs Device Detail

**Table: Xilinx Virtex II Pro FPGA - XC2VP30**

<table>
<thead>
<tr>
<th>Resource</th>
<th># Logic cells&lt;sup&gt;a&lt;/sup&gt;</th>
<th># Slices&lt;sup&gt;b&lt;/sup&gt;</th>
<th>Block RAM</th>
<th>Clock frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30816</td>
<td>13696</td>
<td>2448 kb</td>
<td>100 MHz</td>
</tr>
</tbody>
</table>

<sup>a</sup> Logic cell ≈ one 4-input LUT + one Flip-Flop + Carry logic

<sup>b</sup> Each slice includes two 4-input function generators, carry logic, arithmetic logic gates, wide function multiplexers, and two storage elements.