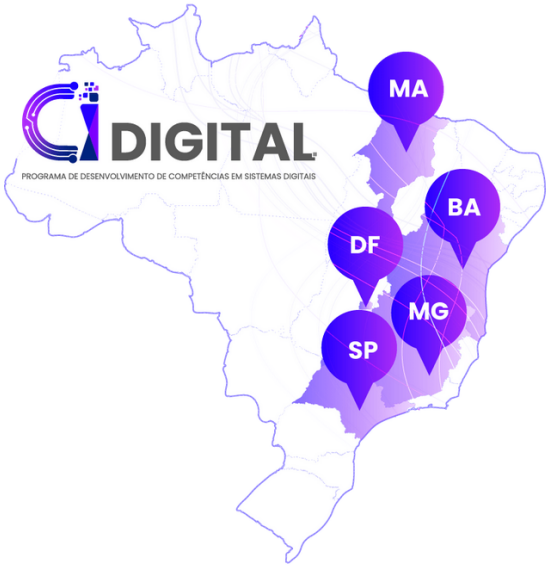


Projeto de Filtro de Kalman em FPGA

Ponto de Controle 3

Luso de Jesus Torres

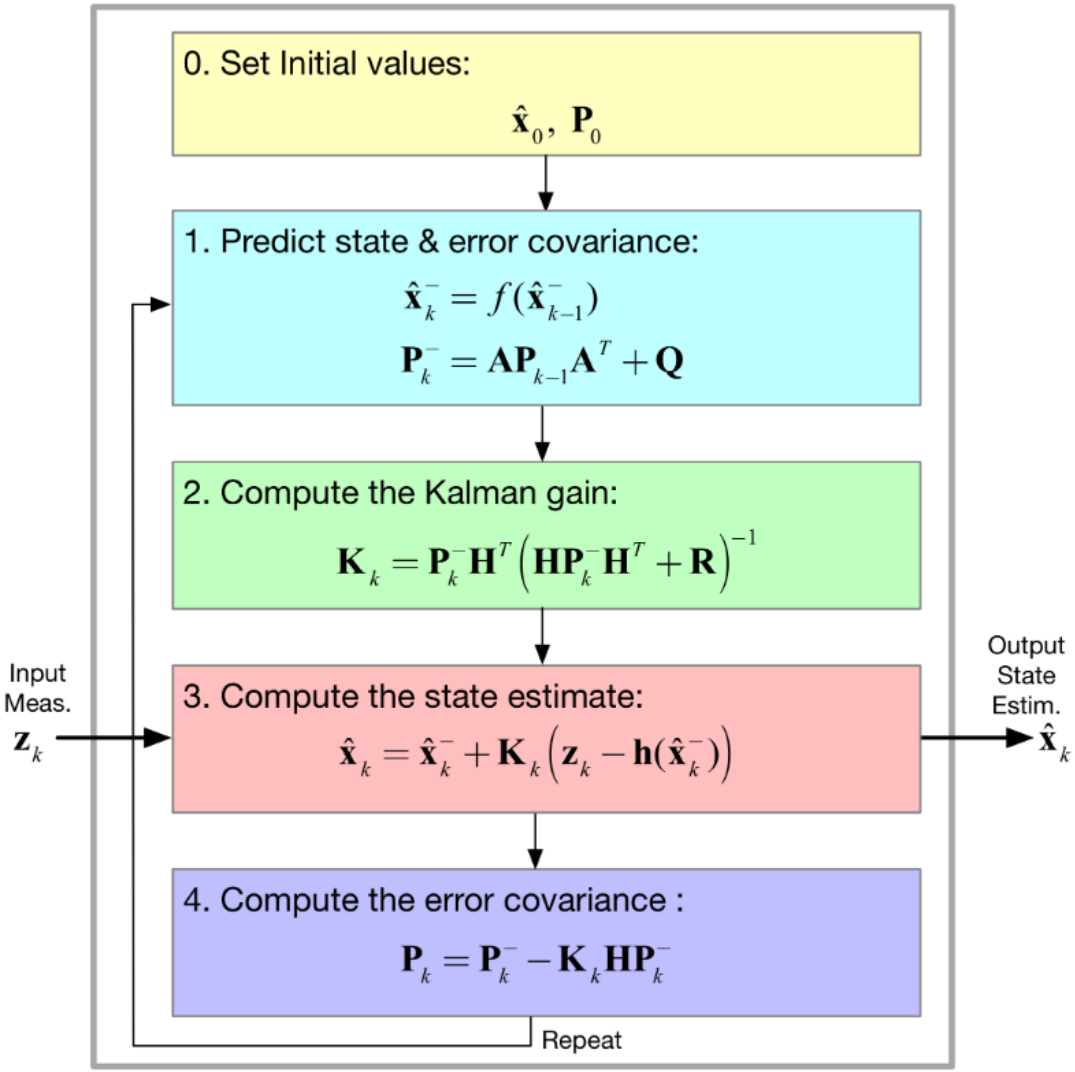
Ministério da Ciência, Tecnologia e Inovação
Programa de Desenvolvimento de Competências em Circuitos Digitais

Jul / 2025

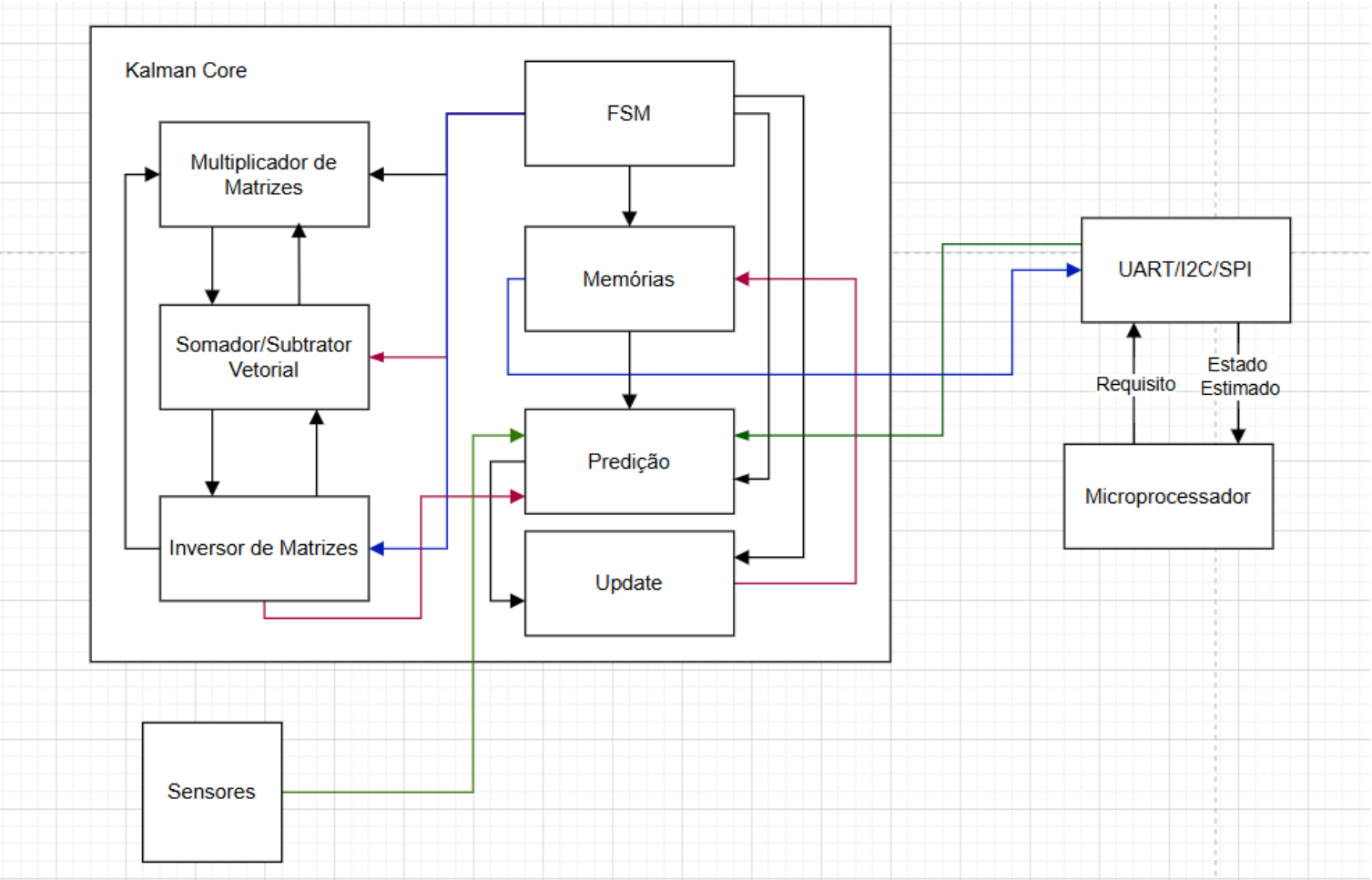
Objetivos

- Projeto de um Filtro de Kalman em Verilog.
- Detalhar módulos de cálculo matricial e controle (FSM).
  - Trabalhar com matrizes de baixa dimensão: 2×2, 2×1, 1×2.
  - Cada estado da FSM representa um estágio de multiplicações e atualizações.
- Apresentar modelo com controlador e acoplamento dos estados.
- Expor relacionamento entre componentes em alto nível.

Esquemático



Arquitetura Modular em HDL



Correção do Módulo: `matrix_inv`

- Entradas: `a, b, c, d` (Q2.14)
- Saídas: `a_inv, b_inv, c_inv, d_inv` (Q2.14)
- Sinal de erro quando $\det = 0$
- Instancia submódulo de recíproco
- Três estágios de pipeline:
  1. Determinante & registros de entrada
  2. Cálculo do recíproco
  3. Multiplicação da adjunta & arredondamento
  4. Registro e Saída

Estágio 1: Cálculo do Determinante

```
        if (reset) begin
            det_q4_28 <= 0;
            det_zero_reg <= 1; // erro no reset
            a1_reg <= 0;
            b1_reg <= 0;
            c1_reg <= 0;
            d1_reg <= 0;
        end else begin
            // det = a*d - b*c  (Q2.14 \times Q2.14 -> Q4.28)
            det_q4_28 <= (a * d) - (b * c);
            // Check if det is zero
            det_zero_reg <= ((a * d) - (b * c)) == 0;
            a1_reg <= a;
            b1_reg <= b;
            c1_reg <= c;
            d1_reg <= d;
        end
    end
```

- Armazena entradas para estágios posteriores
- Sinaliza matriz singular se $\det = 0$

Estágio 2: Módulo de Recíproco

```
1     entity divNRDA_FSM is
2     generic (num_bits : integer range 0 to 32:=32);
3     Port ( reset : in STD_LOGIC;
4            clk : in STD_LOGIC;
5            --dividend : in STD_LOGIC_VECTOR (num_bits-1        downto 0);
6            det_q4_28: in STD_LOGIC_VECTOR (num_bits-1 downto 0);
7            start : in STD_LOGIC;
8            quotient : out STD_LOGIC_VECTOR (num_bits-1 downto 0);
9            --remainder : out STD_LOGIC_VECTOR (num_bits-1 downto 0);
10           error : out STD_LOGIC;
11           ready : out STD_LOGIC);
12 end divNRDA_FSM;
13
14 architecture Behavioral of divNRDA_FSM is
15
16 type estado is (espera,inicio,desloca,subtrai,compara_op,compara_i,somau,somad);
17 signal estado_atual, proximo_estado : estado := espera;
18 signal divisor : STD_LOGIC_VECTOR (num_bits-1 downto 0);
19 signal i : integer range 0 to num_bits := 0;
20 signal regQ: signed(num_bits-1 downto 0) := (others=>'0');
21 signal regA, regM : signed(num_bits downto 0) := (others=>'0');
22 signal A0: std_logic:='0';
23 signal dividend: std_logic_vector (num_bits-1 downto 0) := std_logic_vector(to_unsigned(65536,
       num_bits));
24 constant zero: std_logic_vector(num_bits-1 downto 0) := (others =>'0');
```

Estágio 3: Multiplicação & Arredondamento

```
1   // Estagio 3: Arredondamento e Multiplicar pela Adjuta
2   wire signed [31:0] a_s = { {16{d1_reg[15]}}, d1_reg };  // +d
3   wire signed [31:0] b_s = -{ {16{b1_reg[15]}}, b1_reg };  // -b
4   wire signed [31:0] c_s = -{ {16{c1_reg[15]}}, c1_reg };  // -c
5   wire signed [31:0] d_s = { {16{a1_reg[15]}}, a1_reg };  // +a
6
7   // Produtos em Q18.30
8   wire signed [MUL_I+MUL_F-1:0] prod_a = a_s * inv_det;
9   wire signed [MUL_I+MUL_F-1:0] prod_b = b_s * inv_det;
10  wire signed [MUL_I+MUL_F-1:0] prod_c = c_s * inv_det;
11  wire signed [MUL_I+MUL_F-1:0] prod_d = d_s * inv_det;
```

```
1   function signed [15:0] round_q18_30_to_q2_14;
2       input signed [MUL_I+MUL_F-1:0] in_prod;
3       reg   signed [MUL_I+MUL_F-1:0] tmp;
4       begin
5           tmp = in_prod + ROUND_OFF;
6           round_q18_30_to_q2_14 = tmp[BIT_H:BIT_L]; // Extract [47:32]
7       end
8   endfunction
```

- Elementos da matriz adjunta: $\{d, -b, -c, a\}$
- Multiplica cada elemento por `inv_det` (Q18.30)
- Arredonda para Q2.14:

Estágio 4: Saida

```
1     always @(posedge clk or posedge reset) begin
2        if (reset) begin
3            a_inv_reg <= 0;
4            b_inv_reg <= 0;
5            c_inv_reg <= 0;
6            d_inv_reg <= 0;
7            error_reg <= 1;
8        end else begin
9            if rec_done then
10               a_inv_reg <= round_q18_30_to_q2_14(prod_a);
11               b_inv_reg <= round_q18_30_to_q2_14(prod_b);
12               c_inv_reg <= round_q18_30_to_q2_14(prod_c);
13               d_inv_reg <= round_q18_30_to_q2_14(prod_d);
14               error_reg <= det_zero_stage1 | error_recip;
15           end if;
16       end
17    end
```

Sinal de Erro & Pipeline

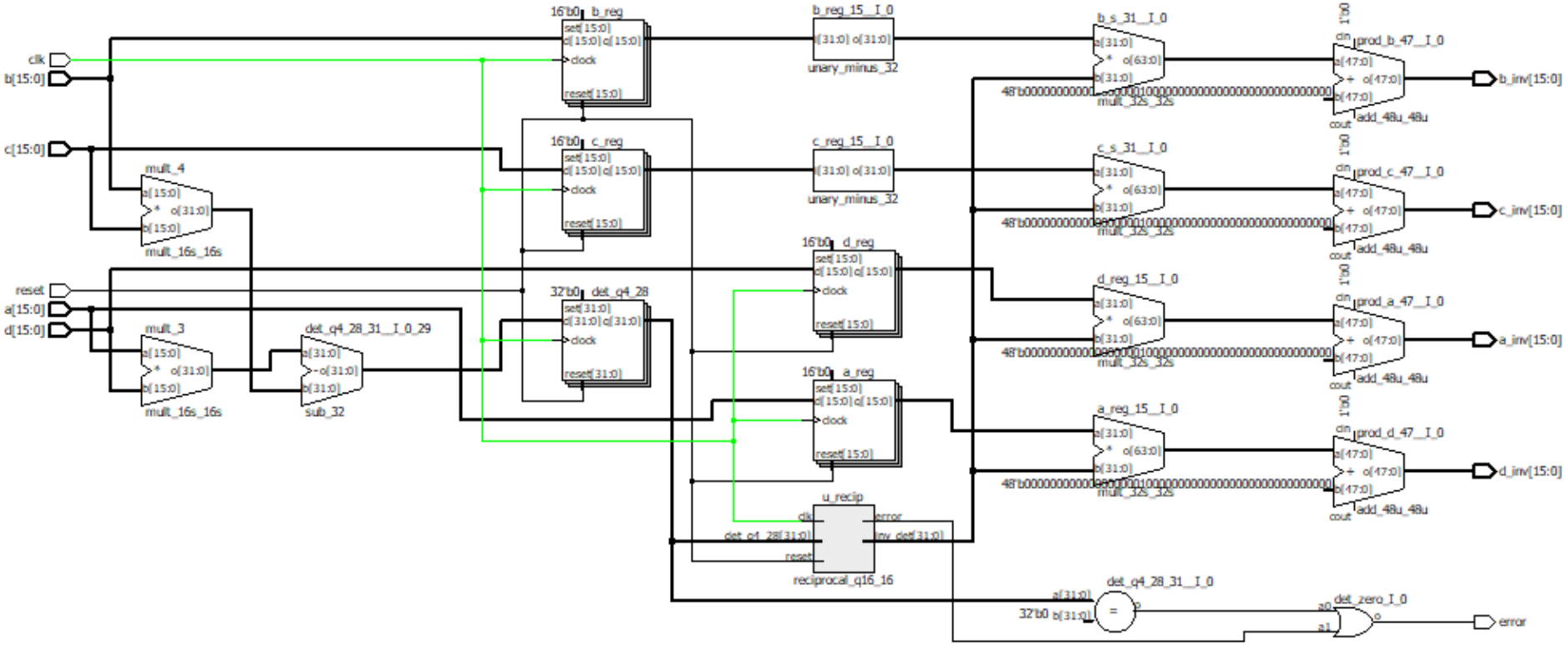
Sinal de Erro

- `det_zero` OU erro do recíproco
- Propagado ao módulo principal

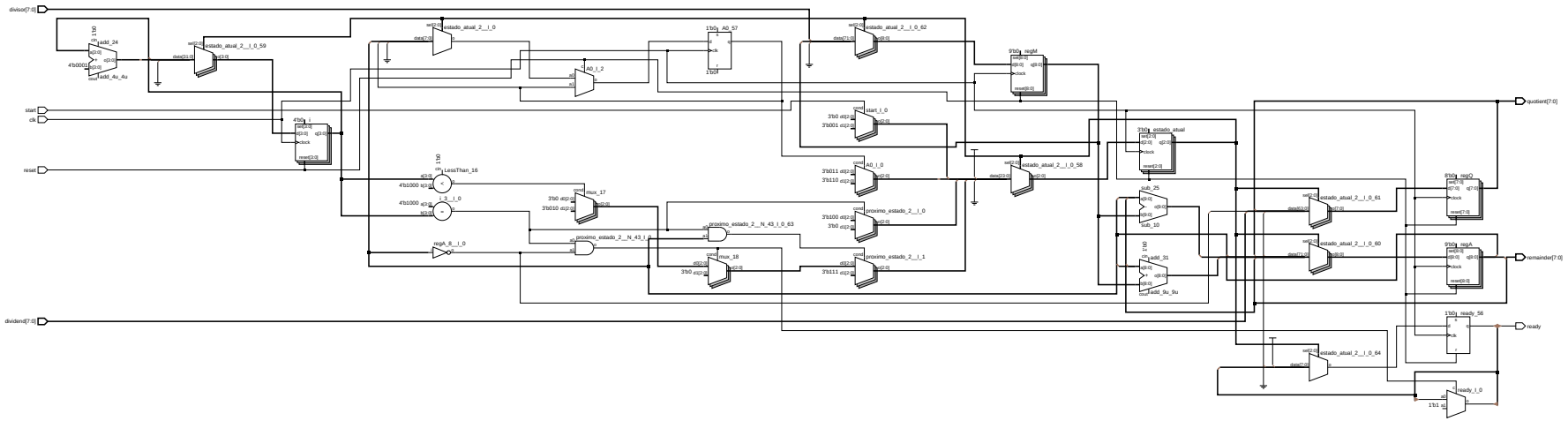
Estágios de Pipeline

1. Determinante & registros
2. Recíproco (multiciclo)
3. Multiplicação & arredondamento
4. Registro

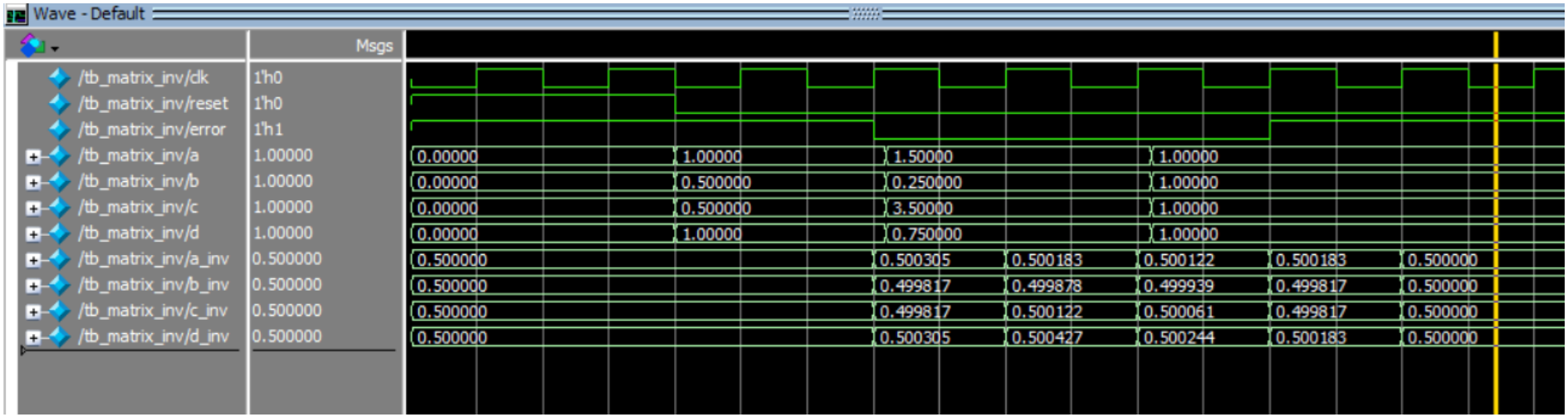
Representação RTL



Reciproco a partir do NRDA



Simulação


Wave - Default
Msgs
/tb_matrix_inv/clk 1'h0
/tb_matrix_inv/reset 1'h0
/tb_matrix_inv/error 1'h1
/tb_matrix_inv/a 1.00000 0.00000 1.00000 1.50000 1.00000
/tb_matrix_inv/b 1.00000 0.00000 0.500000 0.250000 1.00000
/tb_matrix_inv/c 1.00000 0.00000 0.500000 3.50000 1.00000
/tb_matrix_inv/d 1.00000 0.00000 1.00000 0.750000 1.00000
/tb_matrix_inv/a_inv 0.500000 0.500000 0.500305 0.500183 0.500122 0.500183 0.500000
/tb_matrix_inv/b_inv 0.500000 0.500000 0.499817 0.499878 0.499939 0.499817 0.500000
/tb_matrix_inv/c_inv 0.500000 0.500000 0.499817 0.500122 0.500061 0.499817 0.500000
/tb_matrix_inv/d_inv 0.500000 0.500000 0.500305 0.500427 0.500244 0.500183 0.500000

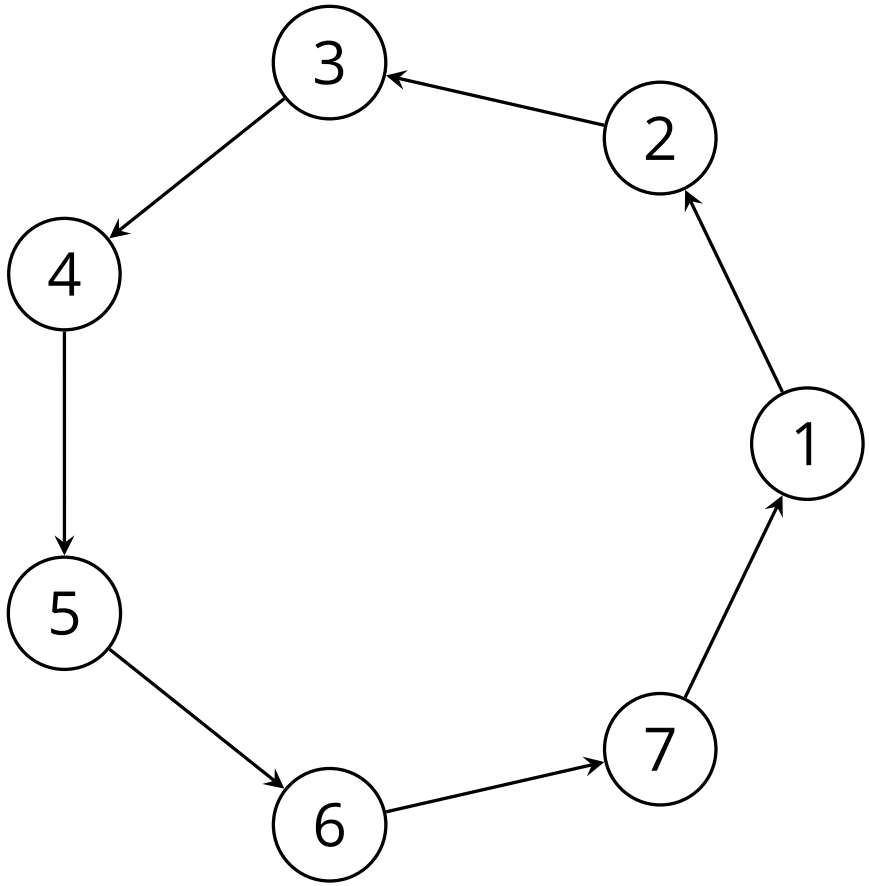
Multiplicadores de Matrizes

- Operações matriciais do Filtro de Kalman e suas dimensões (caso 2 variáveis):

Operação	**Dimensões**
$\hat{x}^- = A \cdot x_k + B \cdot u_k + w_k$	$2 \times 2 \cdot 2 \times 1 = 2 \times 1$
$P^- = A \cdot P \cdot A^T + Q$	$2 \times 2 \cdot 2 \times 2 \cdot 2 \times 2 = 2 \times 2$
$K = P \cdot H^T \cdot (H \cdot P \cdot H^T + R)^{-1}$	$2 \times 2 \cdot 2 \times 1 = 2 \times 1$
$\hat{x} = \hat{x}^- + K \cdot (z - H \cdot \hat{x}^-)$	$2 \times 1 + 2 \times 1 = 2 \times 1$
$P = (I - K \cdot H) \cdot P$	$2 \times 2 \cdot 2 \times 2 = 2 \times 2$

Máquina de Estados do Filtro de Kalman

- **1 – IDLE**: Espera por novo dado ou comando.
- **2 – PREV_STATE**: Calcula previsão do estado.
- **3 – PREV_COV**: Calcula previsão da covariância.
- **4 – GAIN_CALC**: Calcula o ganho de Kalman.
- **5 – UPDATE_STATE**: Atualiza o vetor de estado.
- **6 – UPDATE_COV**: Atualiza a matriz de covariância.
- **7 – DONE**: Finaliza ciclo e retorna ao IDLE.

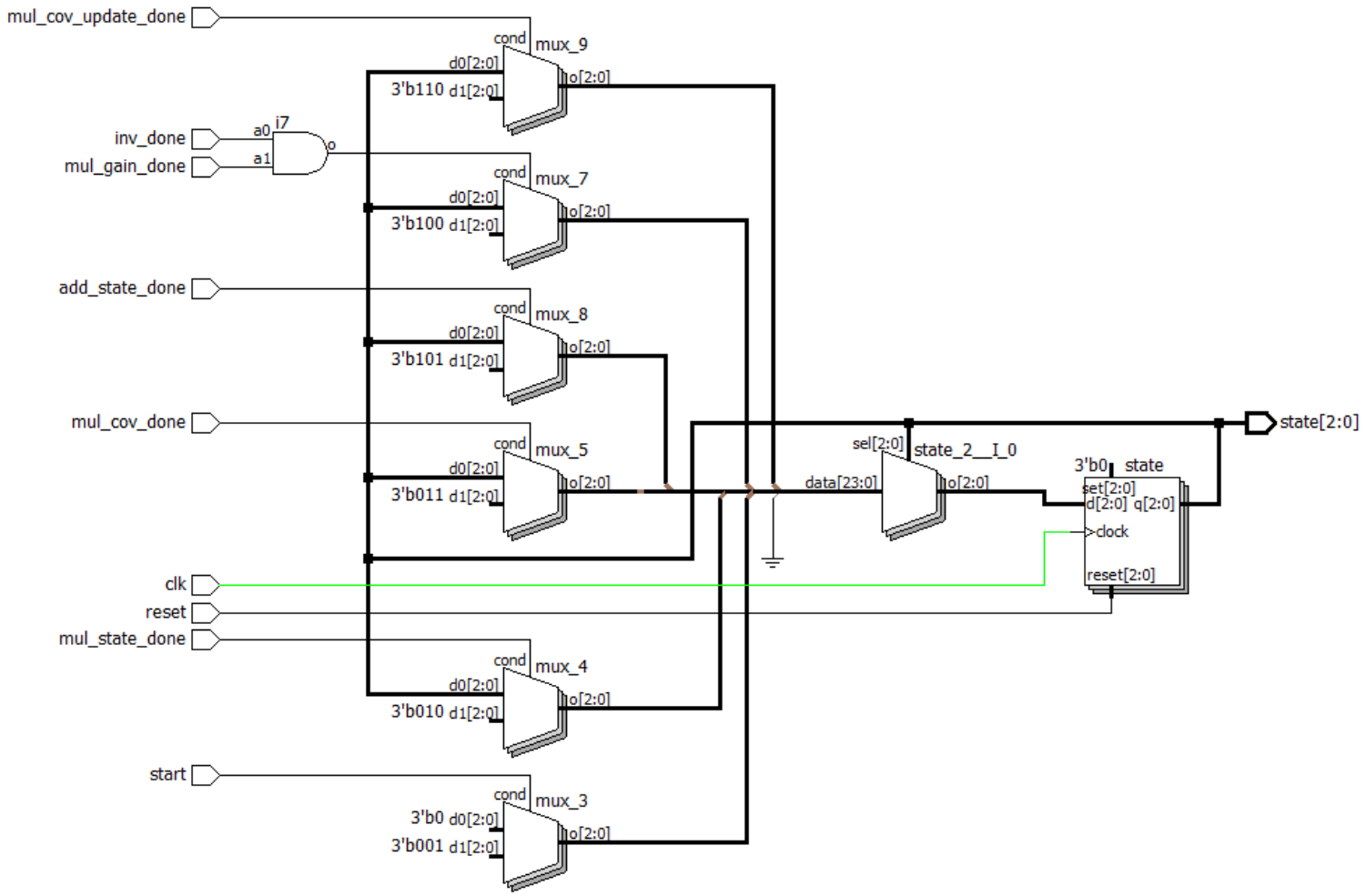


Trecho do Código Verilog

```
always @(posedge clk or posedge reset) begin
        if (reset) begin
            state <= IDLE;
        end else begin
            case (state)
                IDLE: begin
                    if (start)
                        state <= PREDICT_STATE;
                    else
                        state <= IDLE;
                end

                PREDICT_STATE: begin
                    if (mul_state_done)
                        state <= PREDICT_COV;
                end

                PREDICT_COV: begin
                    if (mul_cov_done)
                        state <= GAIN_CALC;
                end
```

```
                GAIN_CALC: begin
                    if (inv_done && mul_gain_done)
                        state <= UPDATE_STATE;
                end

                UPDATE_STATE: begin
                    if (add_state_done)
                        state <= UPDATE_COV;
                end

                UPDATE_COV: begin
                    if (mul_cov_update_done)
                        state <= DONE;
                end

                DONE: begin
                    state <= IDLE;  // or loop in DONE
                end

                default: state <= IDLE;
            endcase
        end
    end
```

Representação RTL



Modelo de Estado

Vetor de estado:

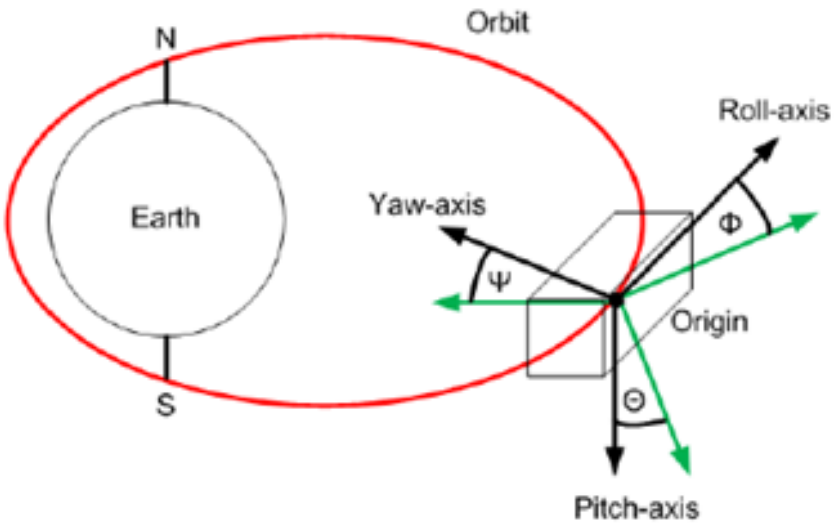
$$\mathbf{x} = \begin{bmatrix} \phi \\ \theta \end{bmatrix}$$

Dinâmica do sistema:

$$\mathbf{x}_{k+1} = A \cdot \mathbf{x}_k + B \cdot u_k + \mathbf{w}_k$$

$$A = \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \Delta t \end{bmatrix}$$

Onde $\alpha = 0{,}03$ representa o acoplamento entre pitch e roll.



Representação dos ângulos de orientação: Roll, Pitch e Yaw

Medições e Incertezas

Medição simulada com acelerômetro:

$$\mathbf{z}_k = \begin{bmatrix} \phi_{acc} \\ \theta_{acc} \end{bmatrix} = H \cdot \mathbf{x}_k + \mathbf{v}_k, \quad H = I_{2\times 2}$$

Covariâncias:

$$Q = \sigma_w^2 I, \quad R = \sigma_v^2 I$$

- Q: incerteza no modelo
- R: incerteza na medição

Filtro de Kalman - Etapas

1. Previsão (Predict)

$$\hat{x}^- = A\hat{x} + Bu, \quad P^- = APA^\top + Q$$

2. Atualização (Update)

$$K = P^- H^\top (HP^- H^\top + R)^{-1}$$

$$\hat{x} = \hat{x}^- + K(z - H\hat{x}^-), \quad P = (I - KH)P^-$$

Modulo Predição Estados

```
1 // predição
2 // B = [0; dt], u = [0; omg*theta]
3 // x_pred = A*x + B*u
4 wire signed [15:0] u2;  // omg*theta em Q2.14
5 // x1_pred = 1*x1 + alpha*x2 + 0*u2
6 assign x1_pred = x1 + ((A12 * x2) >>> 14);
7 // x2_pred = 0*x1 + 1*x2 + dt*u2
8 assign x2_pred = x2 + ((dt * u2) >>> 14);
```

Modulo Predição Covariancia

```
// A = [1 alpha; 0 1], A^T = [1 0; alpha 1]
wire signed [15:0] A11 = 16'd16384;      // 1 em Q2.14
wire signed [15:0] A12 = 16'd491;       // alpha=0.03*2^14=491
wire signed [15:0] A21 = 16'd0;
wire signed [15:0] A22 = 16'd16384;
```

```
// primeiro calcula M = P * A_T
mat2x2_mult mult1 (
  .A11(P11), .A12(P12), .A21(P21), .A22(P22),
  .B11(A11), .B12(A21), .B21(A12), .B22(A22),
  .C11(tmp11), .C12(tmp12), .C21(tmp21), .C22(tmp22)
);

// entao Pp = A * M + Q
mat2x2_mult mult2 (
  .A11(A11), .A12(A12), .A21(A21), .A22(A22),
  .B11(tmp11), .B12(tmp12), .B21(tmp21), .B22(tmp22),
  .C11(Pp11), .C12(Pp12), .C21(Pp21), .C22(Pp22)
);
```

Modulo do Ganho

```
// S = H*Pp*H^T + R = Pp + R  (H = I)
wire signed [15:0] S11 = Pp11_q + R11;
wire signed [15:0] S12 = Pp12_q + R12;
wire signed [15:0] S21 = Pp21_q + R21;
wire signed [15:0] S22 = Pp22_q + R22;

// K = Pp * inv(S)
// Para 2\times2, inv(S) = (1/det) * [S22 -S12; -S21 S11]
matrix_inv U1 (
        .clk(clk),
        .reset(reset),
        .a(S11),
        .b(S12),
        .c(S21),
        .d(S22),
        .a_inv(S_inv11),
        .b_inv(S_inv12),
        .c_inv(S_inv21),
        .d_inv(S_inv22),
        .error(error)
    ); // modulo inversor com divisor NRDA

wire signed [15:0] K11;
wire signed [15:0] K12;
wire signed [15:0] K21;
wire signed [15:0] K22;
```

Modulo do Ganho

```
always @(posedge clk) begin
    if (error) begin
        K11= 0;
        K12= 0;
        K21= 0;
        K22= 0;
    end else
        K11 = (Pp11_q * S_inv22 - Pp12_q * S_inv12) >>> 28;
        K12 = (-Pp11_q * S_inv12 + Pp12_q * S_inv11) >>> 28;
        K21 = (Pp21_q * S_inv22 - Pp22_q * S_inv12) >>> 28;
        K22 = (-Pp21_q * S_inv12 + Pp22_q * S_inv11) >>> 28;
    end
```

Modulo Atualização de Estados

```
1      // Predição y = z - x_pred
2  wire signed [15:0] y1 = phi_acc - x1_pred;
3  wire signed [15:0] y2 = theta_acc - x2_pred;
4
5  // x = x_pred + K * y
6  wire signed [15:0] x1_upd = x1_pred + ((K11 * y1 + K12 * y2) >>> 14);
7  wire signed [15:0] x2_upd = x2_pred + ((K21 * y1 + K22 * y2) >>> 14);
8
9  // P = (I - K*H)*Pp = (I-K)*Pp
10 wire signed [15:0] M11 = 16'd16384 - K11;
11 wire signed [15:0] M12 =          - K12;
12 wire signed [15:0] M21 =          - K21;
13 wire signed [15:0] M22 = 16'd16384 - K22;
```

Modulo Atualização Covariancia

```
1  // P = (I - K*H)*Pp
2  // Como H = identidade 2x2, então I - K*H = I - K
3  wire signed [15:0] M11 = 16'd16384 - K11;  // 1.0 in Q2.14 is 16384
4  wire signed [15:0] M12 =           - K12;
5  wire signed [15:0] M21 =           - K21;
6  wire signed [15:0] M22 = 16'd16384 - K22;
7
8  // P_upd = M * Pp
9  wire signed [15:0] P11_upd = ((M11 * Pp11_q + M12 * Pp21_q) >>> 14);
10 wire signed [15:0] P12_upd = ((M11 * Pp12_q + M12 * Pp22_q) >>> 14);
11 wire signed [15:0] P21_upd = ((M21 * Pp11_q + M22 * Pp21_q) >>> 14);
12 wire signed [15:0] P22_upd = ((M21 * Pp12_q + M22 * Pp22_q) >>> 14);
```

Implementação em Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Constants
dt = 0.1
Q = np.eye(2) * 5e-4     # Process noise cov.
R = np.eye(2) * 5e-2     # Meas. noise cov
N = 200

# Initial state
x = np.zeros((2, 1))     # [roll; pitch]
P = np.eye(2)

phi_est = []
theta_est = []

# Simulated motion
true_phi = np.zeros(N)
true_theta = np.zeros(N)

for i in range(1, N):
    # Simulate true system (with coupling)
    true_phi[i] = true_phi[i-1] + 0.01 + 0.05 * true_theta[i-1] * dt
    true_theta[i] = true_theta[i-1] + np.sin(i * 0.05) * 0.02

```

```python
for i in range(N):
    # Control input (only pitch rate measurement from gyro)
    omega_theta = (true_theta[i] - true_theta[i-1]) / dt if i > 0 else 0.0
    u = np.array([[omega_theta]])  # control only affects theta

    # Dynamics with coupling
    A = np.array([
        [1.0, 0.03],  # roll affected by pitch
        [0.0, 1.0]    # pitch evolves independently
    ])

    # Control matrix (only pitch gets control input)
    B = np.array([
        [0.0],
        [dt]
    ])

```

Implementação em Python

```
# Predict
    x_pred = A @ x + B @ u
    P = A @ P @ A.T + Q

    # Simulated measurements (accelerometer)
    phi_acc = true_phi[i] + np.random.randn()
       * 0.05
    theta_acc = true_theta[i] + np.random.
      randn() * 0.05
    z = np.array([[phi_acc], [theta_acc]])

    # Update
    H = np.eye(2)
    y = z - H @ x_pred
    S = H @ P @ H.T + R
    K = P @ H.T @ np.linalg.inv(S)
    x = x_pred + K @ y
    P = (np.eye(2) - K @ H) @ P

    phi_est.append(x[0, 0])
    theta_est.append(x[1, 0])
```

```
# Plot
plt.subplot(2,1,1)
plt.plot(true_phi, label='True Roll')
plt.plot(phi_est, label='Estimated Roll')
plt.legend()
plt.subplot(2,1,2)
plt.plot(true_theta, label='True Pitch')
plt.plot(theta_est, label='Estimated Pitch')
plt.legend()
plt.show()
```

Resultados

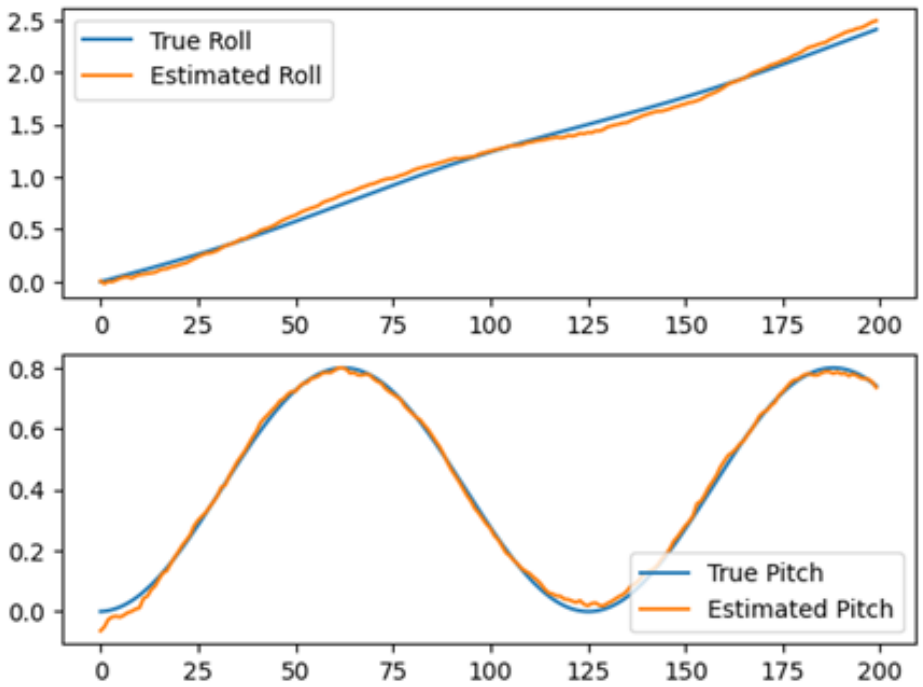


Figure: Simulação no Python

Cronograma de Execução

Semana	Atividade
14/07	Planejamento e Arquitetura Inicial
15–23/07	Implementação do núcleo do filtro: Multiplicador, Inversor e Somador/Subtrator
24–30/07	Ajuste da FSM central do Filtro e incorporação do módulo divisor
31/07–06/08	Comparação com o código em Python/Matlab Correção de erros
28/07–04/08	Documentação, refinamento e caracterização

Conclusão e Atualizações

- Implementação Síncrona dos módulos do Filtro na máquina de estados de controle.
- Comparação do resultado da inversa com dados reais.
- Implementação do controle completo e integração com banco de registradores.