UFPR – Bacharelado em Ciência da Computação
CI1210 – Projetos Digitais e Microprocessadores
Trabalho Semestral — 2021-ere-2 26 de março de 2021

Vocês foram contratados pela Carcará Sistemas Bélicos e sua tarefa é implementar uma parte do sistema de controle de direção do veículo aéreo não tripulado (VANT) Urubu.

O Urubu pode permanecer no ar por várias horas, e seu computador de bordo deve manter o proa do VANT na direção correta e aplicar correções de rota sempre que ocorrer algum desvio, possivelmente provocado por vento de través.
O dispositivo que mede a direção de proa do VANT entrega um número inteiro que representa a direção apontada pela proa do Urubu.
O controlador (humano) de voo determina a direção de proa ao definir o *set-point*. Se a proa apontar para uma direção diferente do *set-point* porque há um erro de direção, então o controlador (automático) deve atuar para corrigir o direção da proa. O controlador deve atuar sempre que o erro de direção for diferente de zero.
Erros negativos indicam que o rumo desejado está à bombordo (esquerda) do VANT, e desvios positivos indicam que o rumo desejado está à estibordo (direita) do VANT.
A Figura 1 mostra um diagrama de blocos do controlador de voo e do sistema controlado. A cada intervalo de medição, a direção é determinada pelo sinal σ_t (*set-point*), e a medida de direção da proa π_t é fornecido pelo sistema de pilotagem. O erro de direção δ_t é a diferença entre a direção desejada e a direção medida: $\delta_t = \sigma_t - \pi_t$.

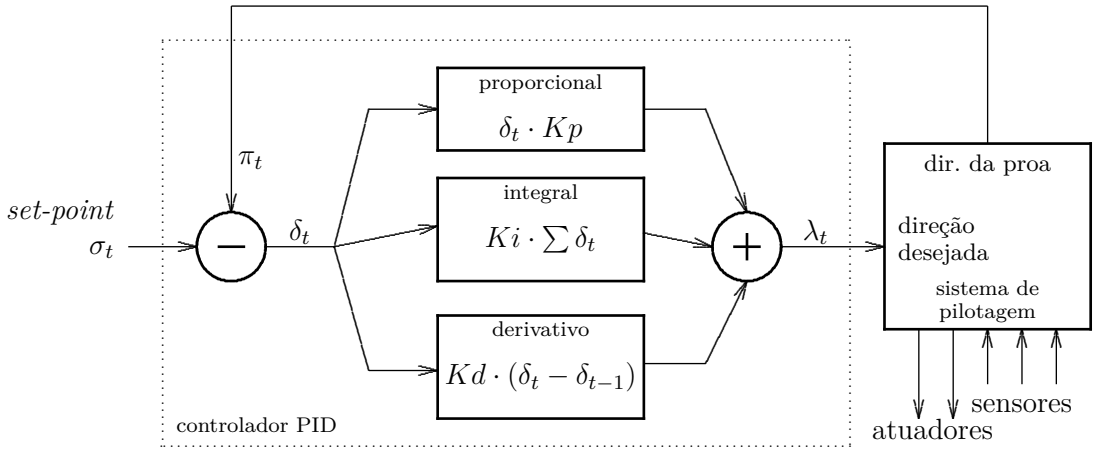

Figura 1: Diagrama de blocos do controlador de direção.

A diferença δ_t entre o *set-point* e a posição medida é usada para computar a correção λ_t. A saída do controlador é portanto a correção que deve ser aplicada ao sistema de pilotagem para recolocar o Urubu na proa desejada.

O sinal de correção é a soma de três componentes, um que é proporcional ao erro, um que é proporcional à integral do erro, e um que é proporcional à derivada do erro. Este tipo de controlador é chamado de PID por causa dos três componentes do sinal de correção: proporcional, integral e derivativo.

O componente proporcional consiste apenas de um multiplicador, que multiplica o erro por uma constante positiva Kp.

O componente integral acumula os erros ao longo do tempo e a integral do erro é multiplicada pela constante Ki. A integral deve ser aproximada por uma soma porque o controlador trabalha com sinais discretizados e não com sinais contínuos.

O componente derivativo computa o a diferença entre o erro anterior e o erro atual, e a 'derivada' do erro é multiplicada pela constante Kd. A derivada deve ser aproximada por uma subtração de duas amostras consecutivas porque o controlador trabalha com sinais discretizados.

Sempre que ocorrer uma mudança no *set-point*, ou alguma perturbação externa que altere a direção do Urubu, o controlador PID deve gerar um sinal de correção para recolocar a proa na direção definida pelo *set-point*, no menor tempo possível.
A soma das três componentes (λ_t) é aplicada ao sistema de pilotagem. A soma deve ser reduzida com uma divisão para que o valor de λ_t se ajuste ao esperado pelo sistema de pilotagem.
Por conta de limitações de peso e energia, as três constantes Kp, Ki e Kd devem ser potências de dois.
O modelo do Urubu tem um atraso de 8 ciclos entre a amostragem de λ_t e seu reflexo em π_t, e este atraso modela a inércia nas reações do VANT.

A Figura 2 mostra o caso trivial em que o Urubu segue uma linha reta e não há nenhuma perturbação. Nesse caso, a linha verde (λ) se sobrepõe à linha preto (σ).

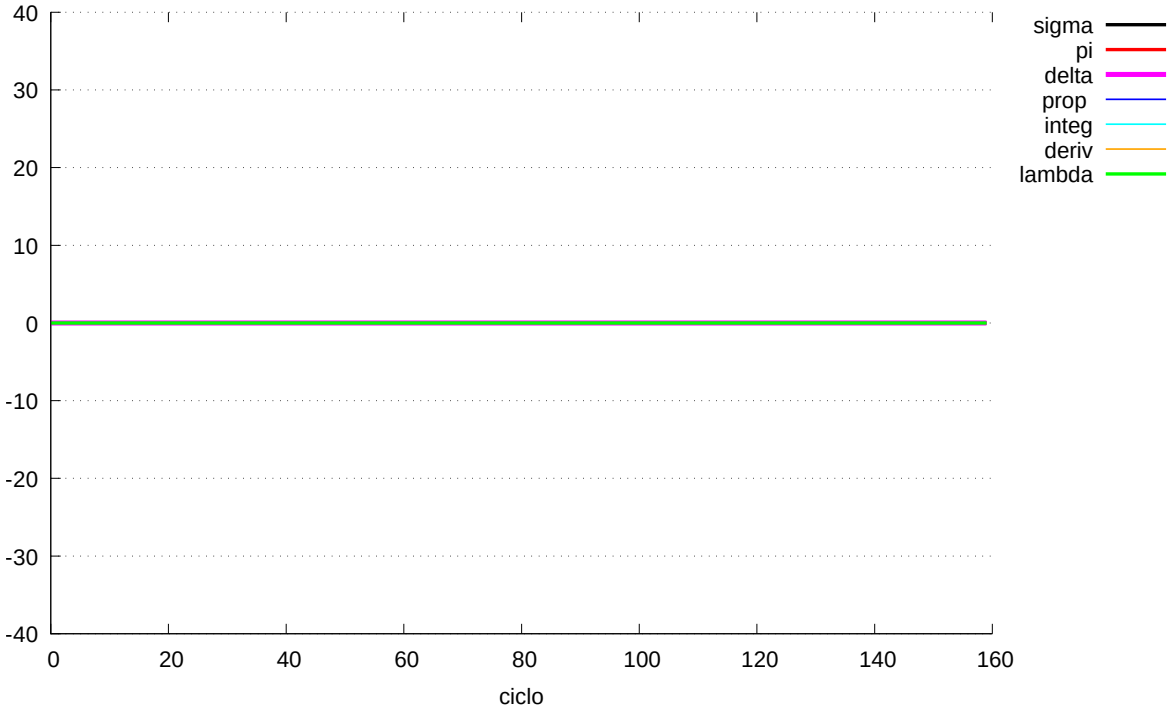

Figura 2: trajetória em linha reta.

Veja o vetor `RETA` no arquivo `genData.c`. Para gerar este teste, diga
`./run.sh -R`
O resultado da simulação é mantido no arquivo `res.pdf`.

Um caso mais interessante é mostrado na Figura 3, no qual o Urubu deve fazer dois desvios, o primeiro para estibordo e o segundo à bombordo.

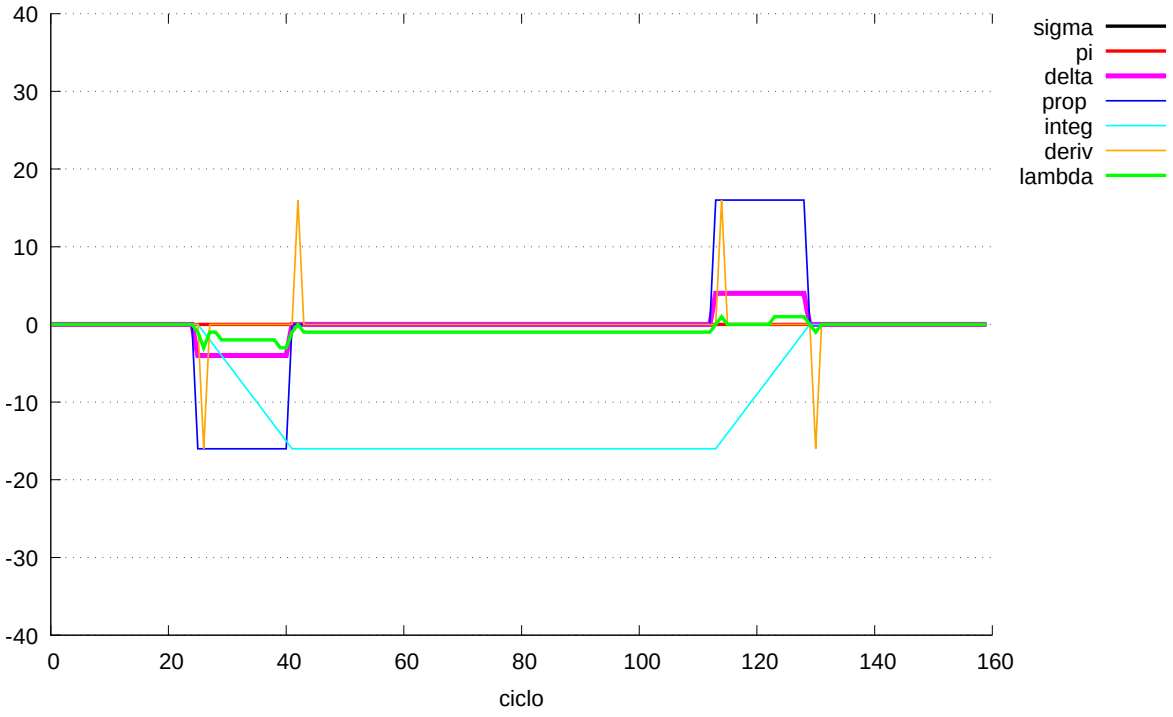

Figura 3: Dois degraus, estibordo e bombordo.

Infelizmente, a linha σ (preto) fica escondida sob as outras. O diagrama mostra a atuação do controle proporcional (linha azul) que é um múltiplo do erro medido (δ_i), e também do controle derivativo (linha laranja), que é um múltiplo da diferença entre o erro atual (δ_i) e o erro anterior (δ_{i-1}).

O controle integral (linha ciano) deve ter um efeito atenuado no Urubu porque, do contrário, causaria instabilidade no controle de proa. A linha ciano indica a acumulação do erro medido (rampa), seguido da manutenção do valor da integral, e finalmente outra rampa com o desconto do erro, que é o erro na direção contrária. Por causa da integral, a linha λ (verde) está ligeiramente a estibordo do que o necessário. Esse é um problema de precisão nos cálculos e que vocês deverão resolver.

Veja o vetor `STEP` no arquivo `genData.c`. Para gerar este teste, diga
`./run.sh -D`

A Figura 4 mostra o comportamento do controlador do Urubu ao longo de uma curva de raio constante. O comando de desvio ocorre no instante 8 e se vê a atuação dos controles derivativos (triângulo laranja) e proporcional (degrau azul). O comando proporcional atua durante todo o intervalo da manobra. Ao final da curva, o controlador derivativo atua novamente para compensar a curva à bombordo.

A atuação do controle é eficaz porque a curva λ (verde) se mantém sobre a curva σ (preto).

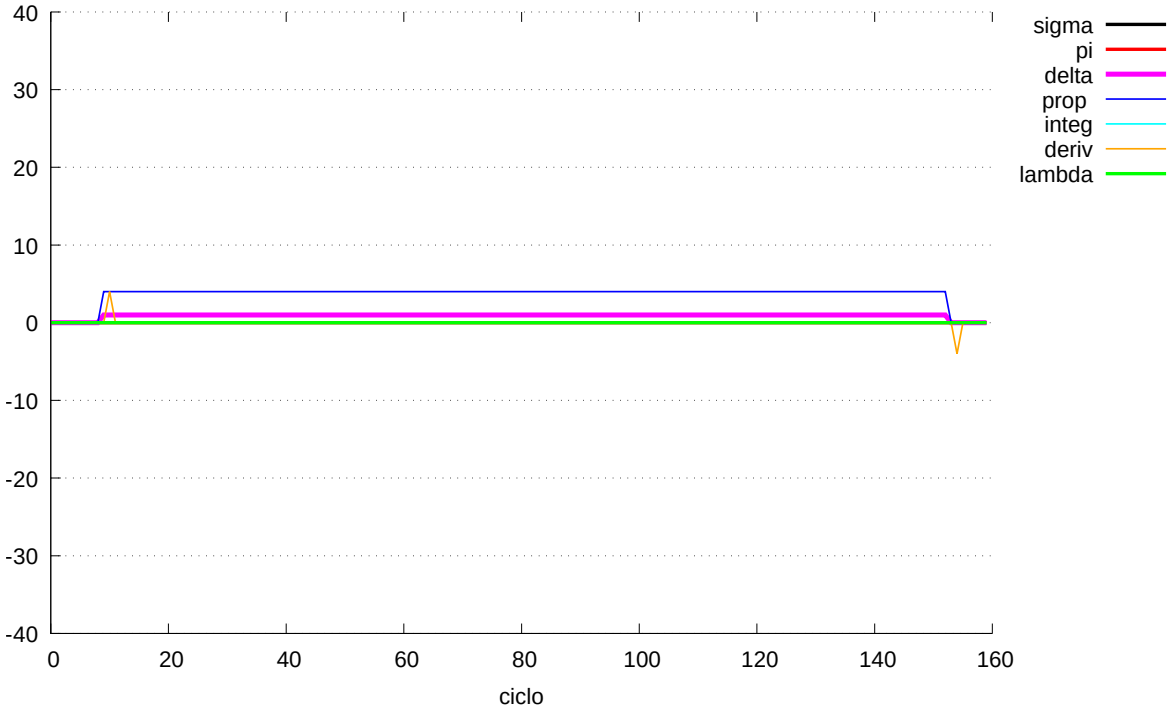

Figura 4: Curva a bombordo.

Veja o vetor `CURVA` no arquivo `genData.c`. Para gerar este teste, diga
`./run.sh -C`

Espaço em branco proposital.

Uma trajetória menos bem comportada é mostrada na Figura 5. A trajetória contém um desvio a bombordo numa curva que fica mais e mais acentuada (a correção proporcional – azul – cresce, e a manobra coloca a proa, abruptamente, na direção original, o que causa uma forte atuação do controle derivativo (laranja). Por volta do ciclo 35 nota-se uma diferença entre σ (preto) e λ (verde).

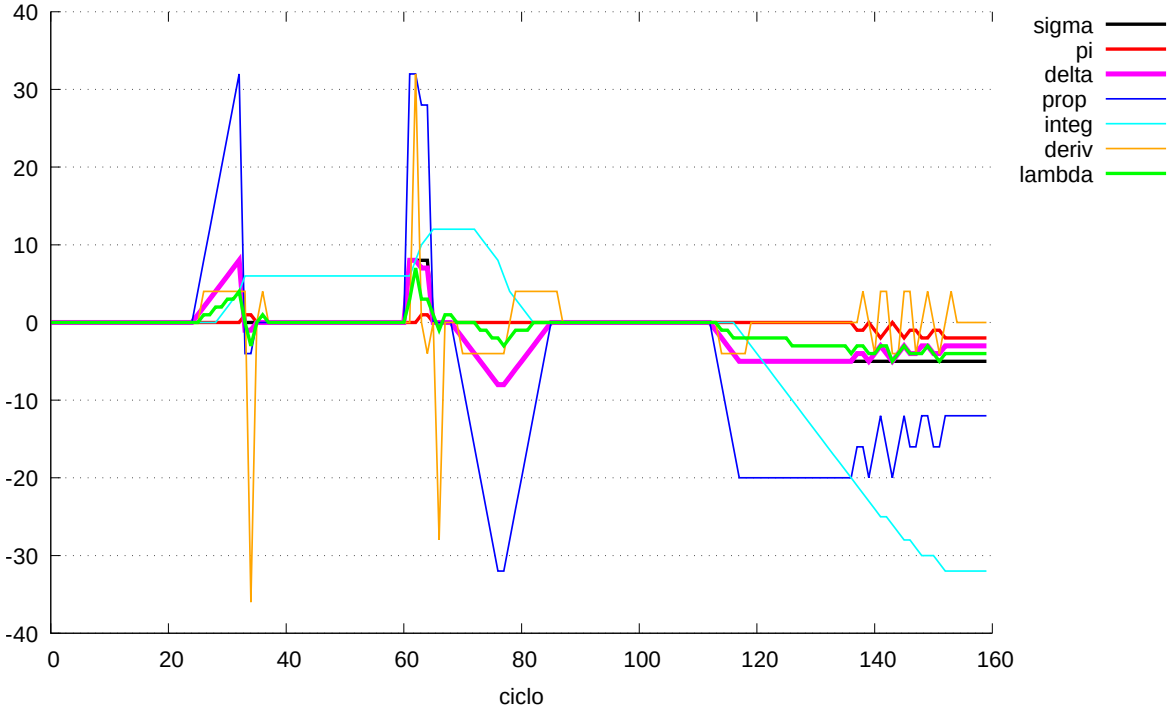

Figura 5: Trajetória semi-aleatória.

A segunda manobra é uma curva fechada a estibordo, seguida de outra a bombordo. Por volta do ciclo 65 nota-se o atraso na correção da trajetória porque λ (verde) está atrasada com relação a σ (preto).

A terceira manobra é uma curva fechada para estibordo e a atuação do controle não é suficientemente rápida. A inércia do Urubu atrasa sua resposta do ciclo 115 (aprox.) até o ciclo 135 (aprox.) quando a proa fica alternando sua direção. Aqui há instabilidade no controle: as componentes da derivada e proporcional seguem variando em função da oscilação do erro e a atuação do controlador integral (ciano) não é suficiente para atenuar as oscilações em lambda (verde). A proa não aponta na direção desejada até depois do ciclo 160, quando a simulação se encerra.

Veja o vetor `SEMI` no arquivo `genData.c`. Para gerar este teste, diga
`./run.sh -S`

Espaço em branco proposital.

Finalmente temos um teste de fogo, mostrado na figura 6. A trajetória inicia com uma curva violenta para bombordo e uma correção, também violenta, para estibordo.

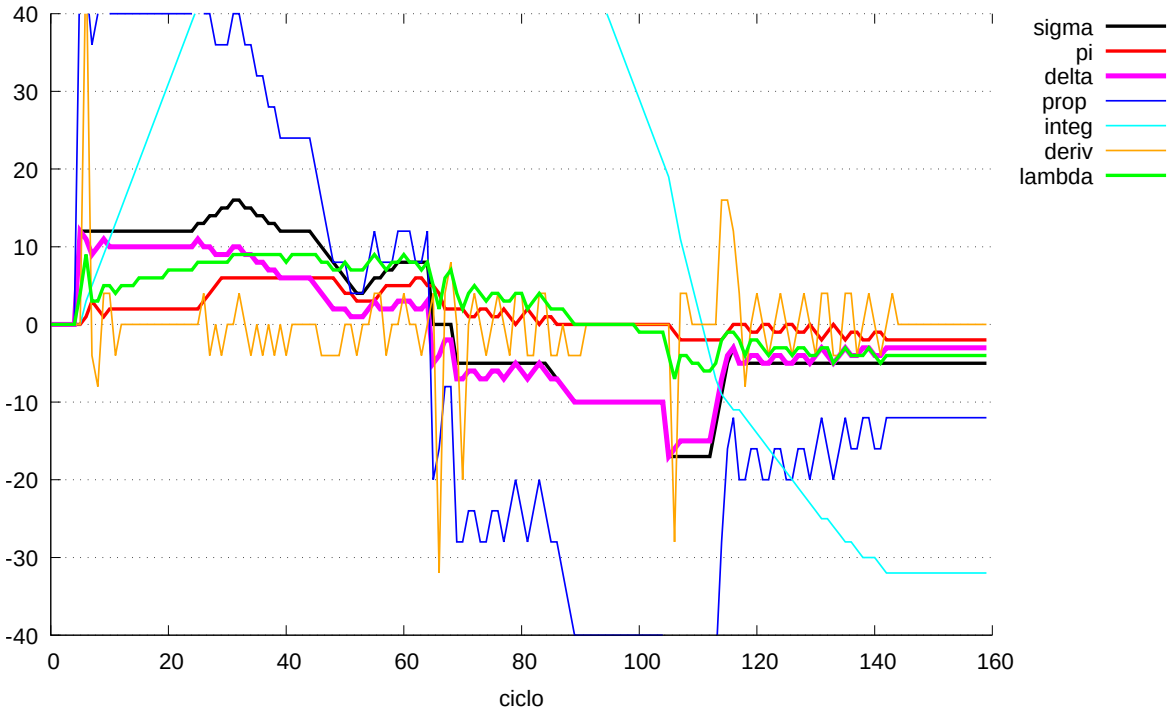

Figura 6: Trajetória aleatória.

Entre os ciclos 5 e 40, o controlador proporcional "puxa" λ (verde) para σ (preto), e o atraso na resposta do Urubu é notável. Entre os ciclos 110 e 120 há um atraso considerável entre a direção da proa (vermelho) a rota desejada (preto). As constantes K_p, K_i e K_d necessitam de ajustes para compensar a inércia do Urubu. A metodologia para o ajuste dessas constantes é estudada na disciplina Servomecanismos, que é parte do currículo de Engenharia Elétrica.

Veja o vetor `LOUCA` no arquivo `genData.c`. Para gerar este teste, diga
`./run.sh -L`

Espaço em branco proposital.

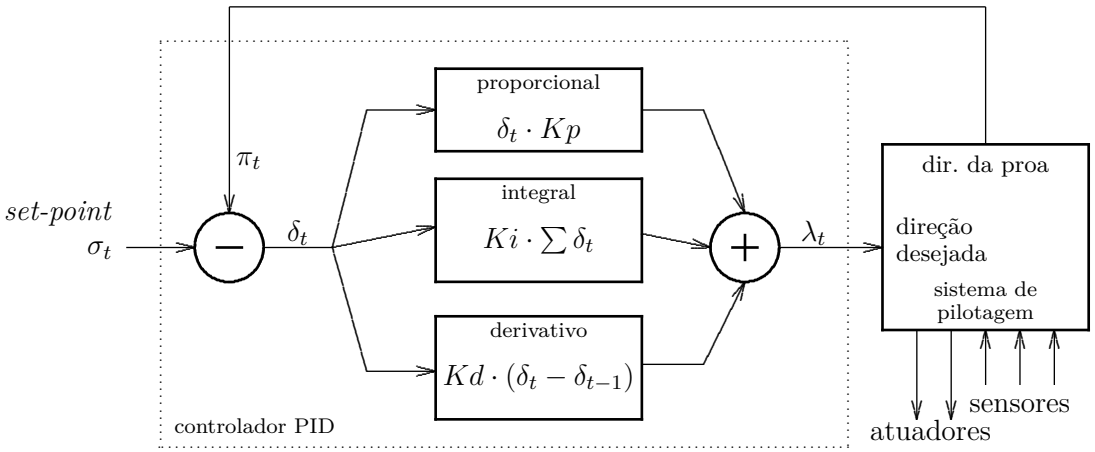

Figura 7: Sinais do controlador de direção.

Vejamos os sinais do controlador do Urubu na Figura 8: (i) a linha preta (sigma) corresponde ao *set-point* (σ_t); (ii) a linha vermelha representa a direção da proa (pi, π_t); (iii) a linha magenta é a diferença entre o *set-point* e o erro ($\delta_t = \sigma_t - \pi_t$); e (iv) a linha verde representa o sinal de comando ao sistema de pilotagem (lambda, λ_t).
A linha azul representa o componente proporcional, a linha ciano o integral, e a laranja o derivativo. O sistema de controle deve manter o erro δ_t (linha magenta) o mais próximo possível de zero, o que equivale a aproximar σ_t de π_t.

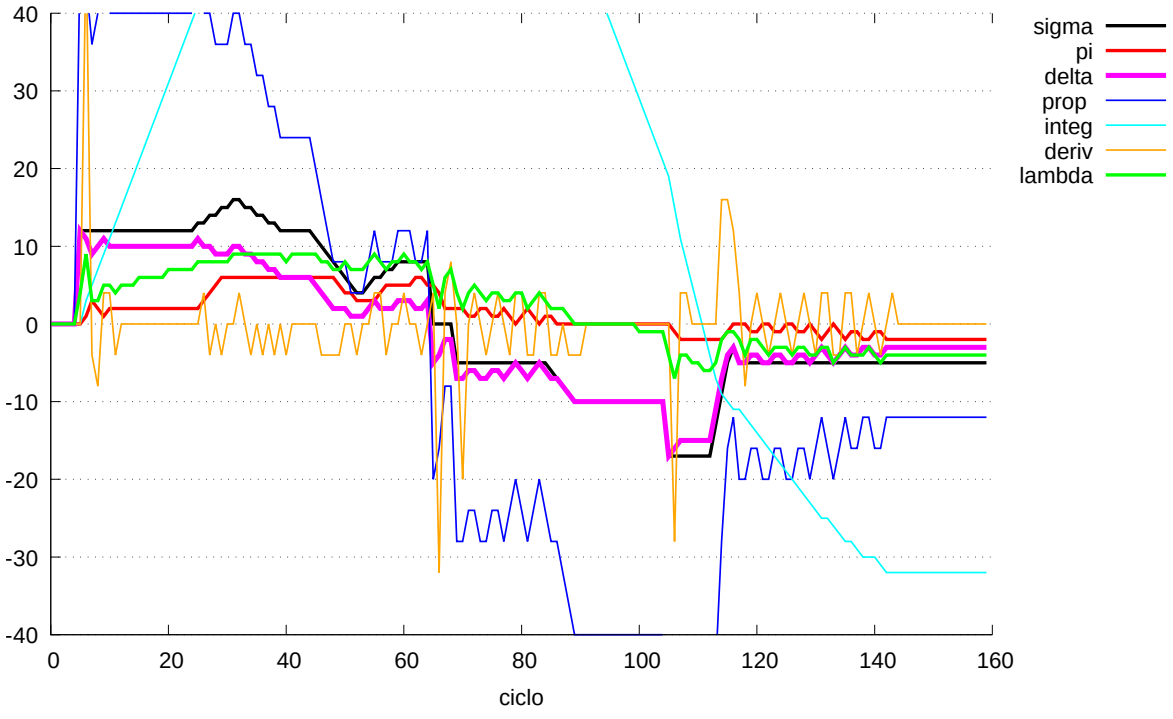

Figura 8: Relação entre os sinais no controlador do Urubu.

Da tarefa Sua tarefa é implementar, em VHDL estrutural, o modelo do controlador PID do Urubu. Você receberá um modelo do comportamento do Urubu, e deve traduzir o código VHDL funcional (a especificação) do PID para código VHDL estrutural (a implementação).

Dos programas auxiliares O programa `genData.c` gera o arquivo binário que é a entrada para a simulação. Caso você queira gerar um outro vetor de voo, edite `genData.c` e diga:
`gcc -Wall genData.c && ./a.out > input.data`
O simulador `tb_pid` espera encontrar `input.data` no diretório corrente.

O arquivo `output.txt` é o resultado da simulação e contém uma lista com os valores produzidos pelo simulador. O *script* `plota.gp` usa o programa `gnuplot` para produzir gráficos como o mostrado na Figura 8.

O script `run.sh` gera os arquivos de entrada, executa a simulação, e produz o arquivo `res.pdf` com o resultado da simulação. `res.pdf` é similar às figuras deste texto. Diga
`./run.sh -h` para ver as opções de simulação.

Da execução Sua tarefa é substituir o código funcional do modelo do controlador PID (entidade `pid` no arquivo `pid.vhd` por código estrutural. Para tanto você deve substituir as somas, multiplicações e divisões (deslocadores) de inteiros por somadores e multiplicadores de vetores de bits. As variáveis do modelo do controlador `pid` devem ser substituídas por registradores, quando e onde for necessário.

Note que a parte que gera os inteiros com a saída da simulação não pode ser alterada. Além disso, seu modelo deve transformar os sinais representados por vetores de bits nos inteiros necessários para gerar os gráficos. Há um exemplo dessa transformação de tipos no arquivo `pid.vhd`.

Você pode usar quaisquer dos circuitos empregados nos laboratórios, além daqueles que você deverá projetar especificamente para este trabalho. Os tempos de propagação dos componentes (portas lógicas e FFs) devem ser todos zero.

Você deve ter reparado nos gráficos que nem sempre λ (verde) está em cima de zero e isso é causado pela baixa precisão empregada no modelo. Por conta disso, seu modelo deve ser implementado com 32 bits sendo 24 bits para a parte inteira e 8 bits para a parte fracionária. As constantes usadas no código funcional não são potências de dois. Issi significa que as curvas geradas pelo seu controlador PID serão (ligeiramente) diferentes das curvas mostradas nesse texto. As diferenças são esperadas mas devem ser relativamente pequenas.

1. O trabalho pode ser efetuado em duplas;
2. o código no modelo deve, necessariamente, ser escrito como um modelo estrutural, com elementos em RTL;
3. o arquivo com os produtos deve ser nomeado `xx-yy.tgz` sendo `xx` e `yy` os *usernames* dos componentes do grupo, e todos os arquivos relevantes deverão estar abaixo do diretório `xx-yy`. Por favor execute "`ghdl --remove`" antes de empacotar seu projeto para envio;
4. copie o arquivo `www.inf.ufpr.br/roberto/ci210/trab20-2.tgz` para sua área de trabalho. O arquivo `LEIAME` contém descrição dos conteúdos da *tarball*.
5. estude o *script* `run.sh` e código VHDL antes de usá-los (`./run.sh -h`);
6. PLÁGIO NÃO SERÁ TOLERADO. É do interesse de todos que vocês conversem sobre o projeto mas **cada grupo deve escrever seu próprio código**.

Dos produtos

1. Arquivo enviado por e-mail para seu professor contendo somente o arquivo fonte `pid.vhd`;
2. todos os programas serão compilados antes de simulados na avaliação, e os arquivos de teste serão alterados para a verificação do trabalho;
3. presença dos membros do grupo na data e hora marcadas para a apresentação.

Sugestões

1. Assegure-se de que entendeu a especificação antes de iniciar a codificação do modelo;
2. use o gráfico para verificar, tanto a correção quanto a qualidade da sua solução – se o seu controlador minimiza δ_t;
3. vocês só precisam alterar o código que está no início de `pid.vhd`, que é a arquitetura da entidade `pid`;
4. provavelmente vocês precisarão de um somador (tempo é irrelevante);
5. **necessariamente** vocês deverão fazer algum (bom, razoável, *scientifically sound*) tipo de arredondamento dos 8 bits de fração para inteiro;
6. experimente com outras formas de onda para o *set-point* porque na apresentação os valores de teste serão diferentes daqueles das figuras neste texto.

Histórico das Revisões:
26mar2021: correção apontada por Todt: π_t é a direção da proa e não o erro;
17mar2021: instrução para gerar as curvas, atualização dos gráficos;
03mar2021: revisão e atualização no texto, inclusão de 7 gráficos;
05nov2015: adição da Fig. 8, comandos para simular;
21out2015: primeira versão.