UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROJETO E DESENVOLVIMENTO DE SISTEMAS EMBARCADOS - DAS410056

Projeto em AADL de Câmera Estabilizada com Gimbal para Aeronave Remotamente Pilotada

Autores:

Afonso da Fonseca Braga

Angela Crepaldi

Rafael Azambuja da Silva

Professor:

Leandro Buss Becker, Dr.

Florianópolis, 7 de outubro de 2019.

1 INTRODUÇÃO

No desenvolvimento de sistemas cyber-físicos (*cyber-physical systems*) foi constatado que muitos problemas não são identificados antes da sua entrada em operação. Para reduzir e evitar que esses problemas sejam encontrados num estágio avançado do projeto, foi então desenvolvida a linguagem de projeto e análise de arquitetura (*Architecture Analysis & Design Language,* AADL). [1]

A AADL é uma linguagem de modelagem que permite a análise da arquitetura de um sistema num estágio inicial, como avaliação de desempenho, escalonabilidade e confiabilidade, e pode ser usada para modelar e analisar sistemas que estão em uso e projetar e integrar novos sistemas. Além disso, a AADL é baseada em modelos, o que permite reuso de código. Ela é especialmente efetiva para especificação de sistemas embarcados de tempo real e vem ganhando espaço na indústria e é definida na Sociedade dos Engenheiros Automotivos (*Society of Automotive Engineers*, SAE) padrão AS5506. [1]

Dada as vantagens e aplicações da AADL, ela foi escolhida para modelar e analisar um sistema embarcado composto por uma câmera estabilizada com gimbal montada numa aeronave remotamente pilotada (ARP), cujo projeto é apresentado neste documento.

Primeiro são levantados os requisitos do sistema no capítulo 2. O capítulo 3 descreve as funções que o sistema deve realizar. No capítulo 4, tem-se o detalhamento do sistema em seus subcomponentes. O capítulo 5 apresenta o processo com suas threads e restrições consideradas para o projeto. E, por fim, o capítulo 6 mostra os resultados das análises executadas no software AADL Inspector.

2 REQUISITOS

1- O sistema da câmera estabilizada deve se comunicar com o controlador da aeronave remotamente pilotada (ARP).

2- A posição da câmera deve ser remotamente controlada.

3- Capturas de imagens devem ser salvas em um cartão SD.

3 FUNÇÕES

1- Ler a posição da câmera.

2- Movimentar a câmera nos três eixos ortogonais (*roll*, *pitch*, *yaw*) a partir de comandos recebidos pelo rádio do ARP.

3- Estabilizar a imagem da câmera.

4- Salvar as imagens da câmera.

5- Enviar imagens de baixa resolução para a ARP.

6- Monitorar funcionamento do sistema.

4 SISTEMA

O sistema desenvolvido para realizar as funções requeridas é composto por um controlador (*controller*) conectado ao controlador da ARP (*unmanned aircraft vehicle*, UAV), um dispositivo de armazenamento de dados (*SD card*) e um gimbal contendo uma câmera, três servo-motores e uma unidade de medida inercial (*inertial measurement unit*, IMU) como mostrado na Figura 1.

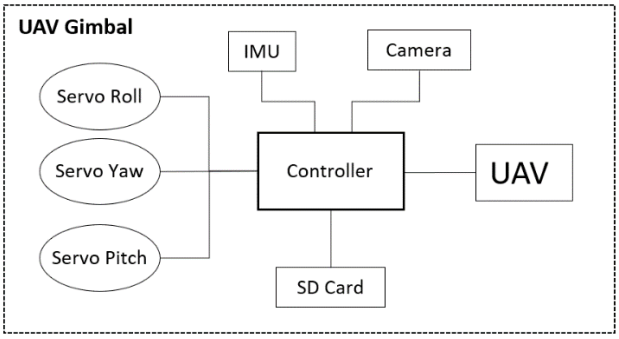


Figura 1: Sistema de controle da câmera estabilizada.

Na AADL, foi criado um sistema chamado `uavcam` com os seguintes subcomponentes:

- Processador: `cpu`

- Processo: `sw`

- Memória RAM: `ram`

- Barramento de memória: `mem_bus`

- Servo-motor para ângulo de rolagem: `servo_roll`

- Servo-motor para ângulo de arfagem: `servo_pitch`

- Servo-motor para ângulo de guinada: `servo_yaw`

- Unidade de medida inercial: `imu`

- Câmera: `camera`

- SD card: `sd_card`

- Controlador da ARP: `uav`

- Barramento I2C para comunicação com a IMU: `i2c_bus`

- Barramento USB para comunicação com a câmera: `usb_cam`

- Barramento SPI para comunicação com o SD card: `spi_bus`

- Barramento UART para comunicação com o controlador da ARP: `uart_bus`

- Barramento PWM para comunicação com os servo-motores: `pwm_bus`

O diagrama do sistema gerado automaticamente pelo software OSATE é mostrado na Figura 2. As linhas que cortam o processo *sw* representam os fluxos do sistema.

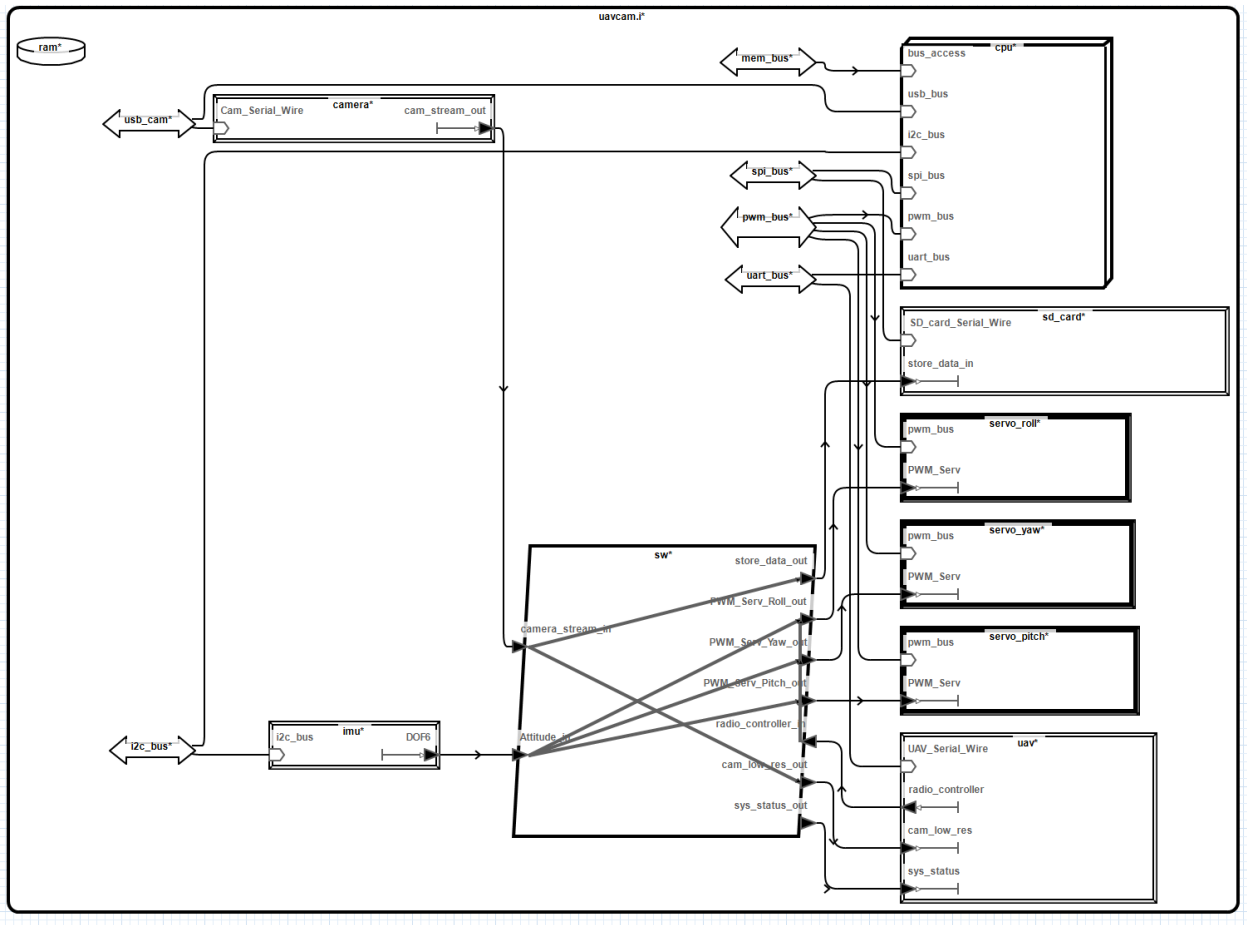


Figura 2: Diagrama do sistema *uavcam*.

O processador possui as seguintes propriedades temporais:

```
Clock_Period => 1 ns;
Scheduling_Protocol => (POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL);
Priority_Range => 0 .. 255;
Preemptive_Scheduler => true;
```

Foi escolhido uma frequência de 1 GHz devido ao processamento de imagem, o protocolo de escalonamento mais alta prioridade primeiro, onde 255 é a thread de mais alta prioridade e escalonador preemptivo.

5 PROCESSO

O sistema possui apenas um processo chamado de *sw* e pode ser visto na Figura 3 com as threads, as conexões e os fluxos das threads. Na Figura 4, são mostrados os fluxos do processo.

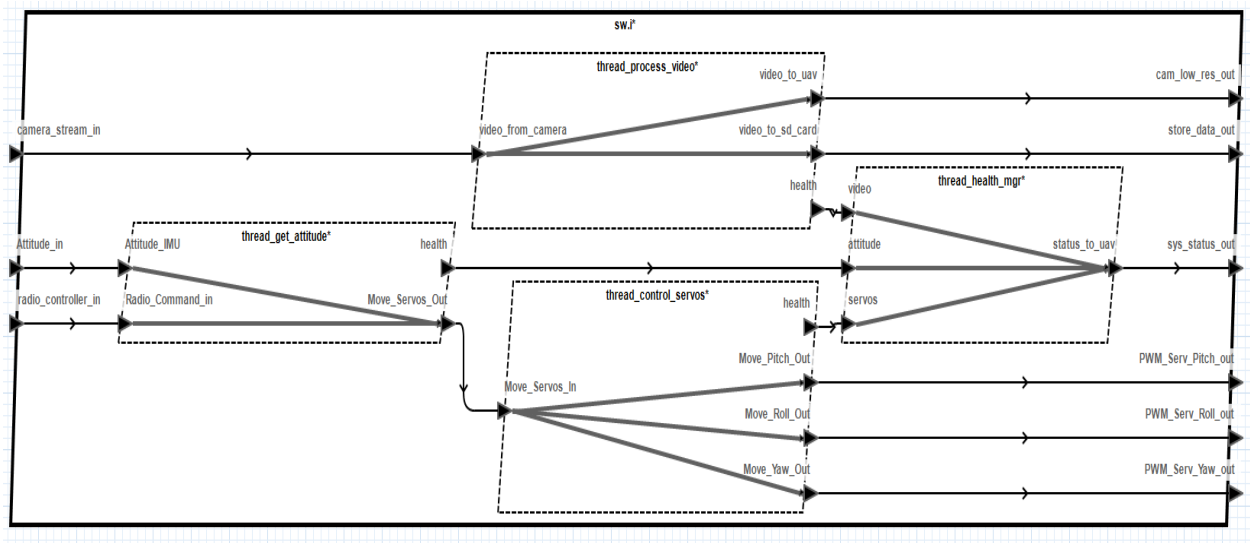


Figura 3: Processo

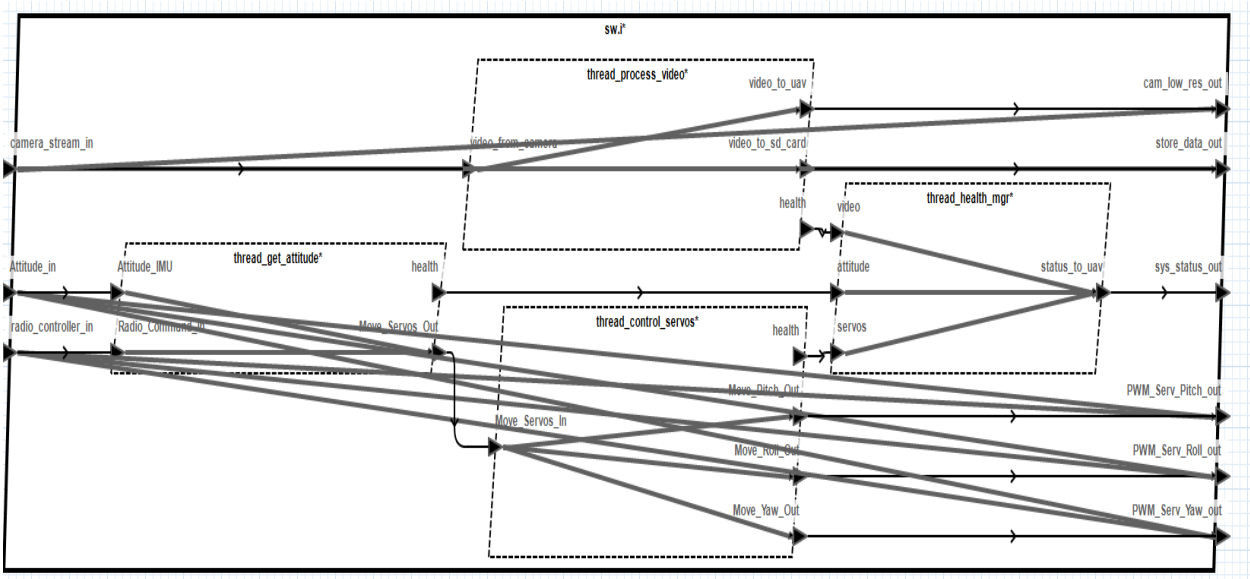


Figura 4:Fluxos do processo.

As funções do sistema foram divididas em quatro threads: *get_attitude*, *control_servos*, *process_video* e *health_mgr*. As características de tempo das threads são mostradas na Tabela 1, onde a tarefa de mais alta prioridade é a de maior valor, quatro.

Threads	Período	Tempo de execução	Deadline	Prioridade
get_attitude	40 ms	2 ms a 4 ms	40 ms	4
control_servos	100 ms	4 ms	100 ms	3
process_video	50 ms	10 ms a 20 ms	50 ms	2
health_mgr	1000 ms	2 ms	1000 ms	1

Tabela 1: Características temporais das threads do sistema.

5.1 get_attitude

A thread *get_attitude* é responsável por receber os dados dos sensores inerciais da IMU (`Attitude_IMU`) e o comando de posicionamento do operador remoto (`Radio_Command_in`) que é recebido pelo controlador da ARP. Essa thread gera um valor de posicionamento para o comando dos servo-motores (`Move_Servos_Out`) e envia dados de monitoramento de falha de software e hardware (`health`).

Foi considerada como restrição temporal que o período da thread *get_attitude* fosse pelo menos metade do período da thread que usa os dados de saída da get_attitude, respeitando o Teorema de Nyquist.

5.2 control_servos

A thread *control_servos* trata os dados de posicionamento (`Move_Servos_In`) e comanda os servo-motores por PWM (`Move_Roll_Out, Move_Pitch_Out, Move_Yaw_Out`). Essa thread também envia dados de monitoramento de falha de software e hardware relativos ao seu funcionamento (`health`).

5.3 process_video

A thread *process_video* captura os frames da câmera de vídeo (`video_from_camera`), faz o processamento de imagem, envia a imagem processada para armazenar no cartão de memória (`video_to_sd_card`) e envia uma versão da imagem com baixa resolução para o controlador do ARP (`video_to_uav`) para que o operador tenha uma amostra da imagem. Essa thread também envia dados de monitoramento de falhas do software e dos dispositivos de hardware relacionados a ela.

O período da thread de 50 ms foi baseado numa taxa de 20 frames por segundo para captura das imagens da câmera.

5.4 health_mgr

A thread *health_mgr* executa o monitoramento do sistema e das demais threads (`attitude, vídeo, servos`) e envia o status para o operador (`status_to_uav`) a fim de que ele possa tomar uma decisão caso haja alguma falha no sistema prejudicando a missão.

Julgou-se que um período de 1 segundo é suficiente para atualizar os dados de saúde do sistema.

6 RESULTADOS

Os resultados foram obtidos através do AADL Inspector [2].

Em primeiro lugar, foi executada uma análise estática no AADL Inspector (*Parse and Instantiate, Check Consistency Rules, Check Legality Rules, Checking Naming Rules*) onde nenhum erro foi reportado. Posteriormente, realizou-se uma simulação do sistema e análise teórica.

6.1 Simulação

A Figura 5 mostra a linha de tempo com as tarefas e as suas entradas e saídas. O período em que as tarefas estão em execução é mostrado em preto e o período em que as tarefas estão aguardando execução ou

são preemptadas estão em laranja. A troca de mensagens é mostrada com as setas em cinza e considera-se que são transmitidas ao final das threads.

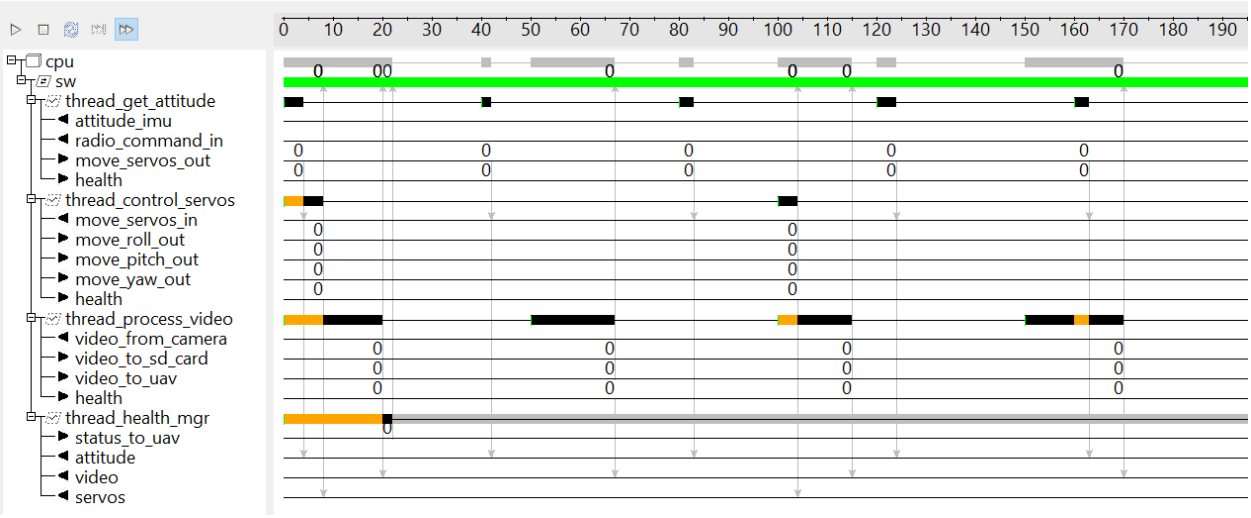
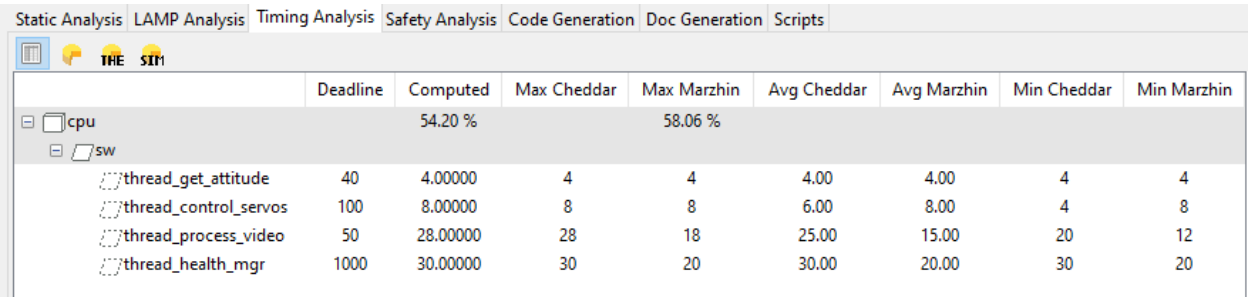


Figura 5: Simulação do sistema.

6.2 Análise Temporal

A análise temporal foi realizada utilizando o plugin Cheddar [3].

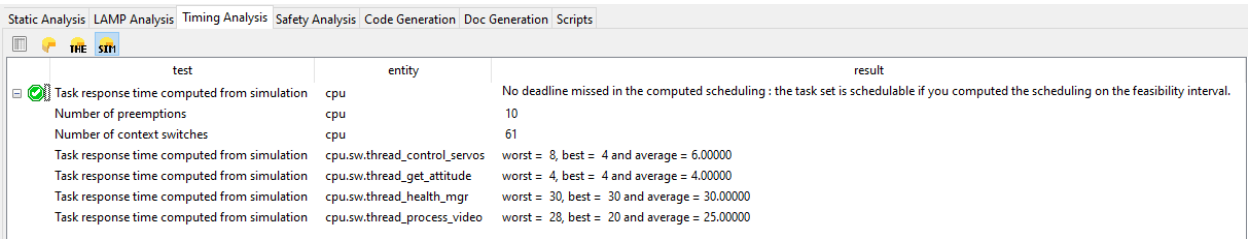
A Figura 6 apresenta um relatório da utilização do processador e a análise de tempo de resposta de cada thread, para o método do Cheddar e do Marzhin.



	Deadline	Computed	Max Cheddar	Max Marzhin	Avg Cheddar	Avg Marzhin	Min Cheddar	Min Marzhin
cpu		54.20 %		58.06 %				
sw								
thread_get_attitude	40	4.00000	4	4	4.00	4.00	4	4
thread_control_servos	100	8.00000	8	8	6.00	8.00	4	8
thread_process_video	50	28.00000	28	18	25.00	15.00	20	12
thread_health_mgr	1000	30.00000	30	20	30.00	20.00	30	20

Figura 6: Relatório de Análise temporal.

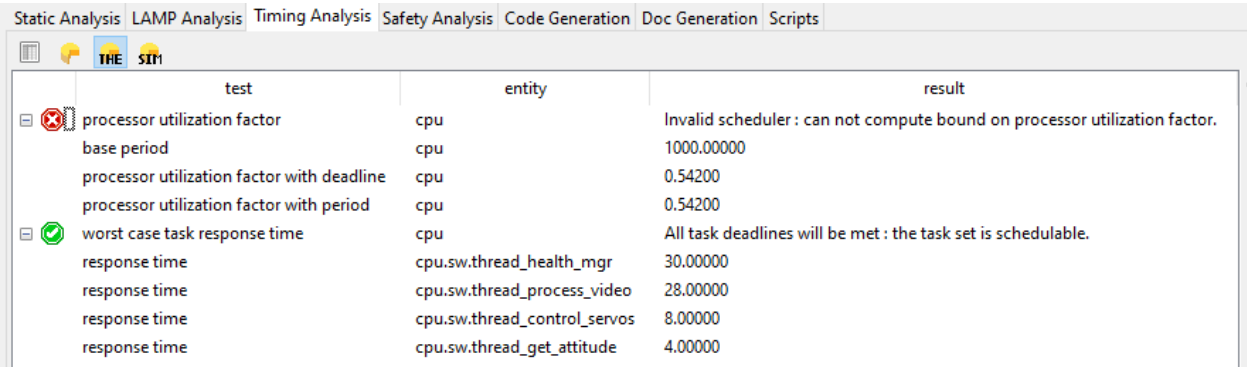
A Figura 7 mostra o resultado do teste simulado onde nenhum deadline foi perdido. Também são mostrados o tempo de resposta calculado para cada tarefa no pior caso, no melhor caso e a média.



test	entity	result
Task response time computed from simulation	cpu	No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.
Number of preemptions	cpu	10
Number of context switches	cpu	61
Task response time computed from simulation	cpu.sw.thread_control_servos	worst = 8, best = 4 and average = 6.00000
Task response time computed from simulation	cpu.sw.thread_get_attitude	worst = 4, best = 4 and average = 4.00000
Task response time computed from simulation	cpu.sw.thread_health_mgr	worst = 30, best = 30 and average = 30.00000
Task response time computed from simulation	cpu.sw.thread_process_video	worst = 28, best = 20 and average = 25.00000

Figura 7: Teste simulado.

A Figura 8 apresenta os resultados do teste teórico e, assim como o teste simulado, afirma que todas as tarefas cumprem com os deadlines exigidos e que o conjunto de tarefas é escalonável. O período base é de 1000 ms e o fator de utilização da CPU é de 54,2% tanto em relação ao período quando ao deadline. No entanto, não foi possível identificar o que causou a mensagem: "*Invalid scheduler: can not compute bound on processor utilization fator*."



test	entity	result
processor utilization factor	cpu	Invalid scheduler : can not compute bound on processor utilization factor.
base period	cpu	1000.00000
processor utilization factor with deadline	cpu	0.54200
processor utilization factor with period	cpu	0.54200
worst case task response time	cpu	All task deadlines will be met : the task set is schedulable.
response time	cpu.sw.thread_health_mgr	30.00000
response time	cpu.sw.thread_process_video	28.00000
response time	cpu.sw.thread_control_servos	8.00000
response time	cpu.sw.thread_get_attitude	4.00000

Figura 8: Teste teórico.

Apesar de incluir fluxo de dados ponta-a-ponta, não se conseguiu analisar o tempo em que uma mensagem atravessa todo o sistema.

7 CONCLUSÃO

A AADL é uma ferramenta que permite modelar os itens de um sistema de software embarcado em dispositivos e software bem como incluir propriedades temporais. É possível conferir vários detalhes ao sistema, utilizar bibliotecas pré-definidas, incluir as instruções de programação em subprogramas e gerar o código.

Com a AADL, conseguiu-se projetar o sistema da câmera estabilizada com gimbal para aeronave remotamente pilotada garantindo a escalonabilidade do sistema. No entanto, sabe-se que esta é primeira etapa do projeto e se faz necessário programar as instruções das threads e executar mais testes para garantir que o sistema todo funcione de acordo com as restrições de tempo.

As ferramentas para modelagem da arquitetura apresentaram alguns desafios. Não foi possível gerar análises pelo Osate, apenas no AADL Inspector. Já o AADL Inspector tem poucas features características de IDEs que apontam erros e *warnings*, permitem a localização da declaração de tipos e objetos, etc. O Osate é uma IDE com mais recursos e permite a criação de diagramas que facilitam a visualização do sistema. O AADL Inspector detectou a presença dos fluxos de dados, mas não foi encontrada a análise ponta-a-ponta.

8 REFERÊNCIAS BIBLIOGRÁFICAS

[1] FEILER, Peter H.; GLUCH, David P.; HUDAK, John J. *The Architecture Analysis & Design Language (AADL): An Introduction*. Carnegie Mellon University, Pennsylvania: 2006.

[2] ELLIDISS TECHNOLOGIES. *AADL Inspector 1.7 – Quick Start Guide*.

[3] ELLIDISS TECHNOLOGIES. *CHEDDAR 3.0 AADL Inspector Plugin Manual User Manual*.