



UNICAMP

Universidade Estadual de Campinas

Faculdade de Engenharia Mecânica

IM420X – Projeto de Sistemas Embarcados de Tempo Real

Novembro de 2024

Docente: Dr. Rodrigo Moreira Bacurau

Discente:

– Gabriel Toffanetto França da Rocha – 289320

Arquitetura HIL para teste de sistemas
embarcados como *vehicle interface* de veículos
autônomos baseados no Autoware

Sumário

1 Resumo	2
2 Motivação	3
3 Documentação	5
3.1 Requisitos	7
3.2 Componentes	8
3.3 Arquitetura	10
3.4 Método de desenvolvimento	12
3.5 Projeto de <i>software</i>	13
3.6 Periféricos	21
4 Manual de utilização	24
5 Problemas identificados e não resolvidos	25
6 Códigos da comunidade	26
Referências	27
Apêndices	28

1 Resumo

2 Motivação



Figura 1: Veículo Autônomo do LMA.

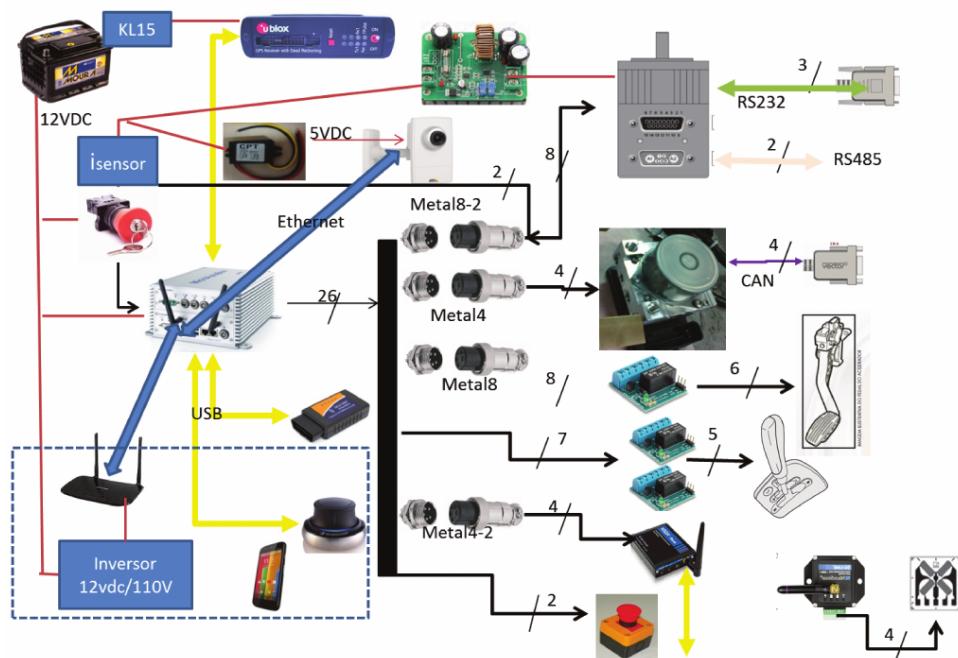


Figura 2: Diagrama de *hardware* do VILMA01 (BEDOYA, 2016).

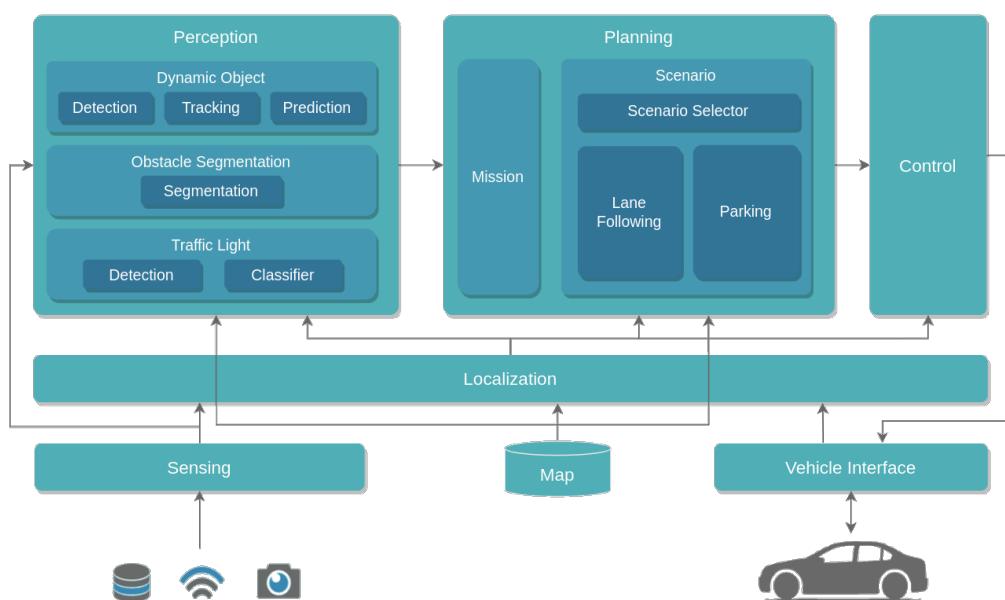


Figura 3: Arquitetura de alto nível (The Autoware Fundation, 2023).

3 Documentação

A partir da problemática de integrar por meio do Autoware todos os módulos de um veículo autônomo, se mantém em aberto como será realizada a integração do sistema de alto nível do veículo com o baixo nível, que é realizado pelo módulo *vehicle interface*, mas que não é especificado no *framework*. Dessa forma, o escopo do projeto se define na implementação de uma interface Autoware – veículo autônomo, como mostrado na Figura 4.

Para fazer tal integração, parte-se da premissa de uma plataforma de baixo nível baseada em microcontroladores STM32, executando um sistema operacional de tempo real. A partir dessa base, propõem-se o uso do *framework* micro-ROS(micro-ROS, 2024), para implementação do módulo *vehicle interface* dentro do sistema embarcado, permitindo a presença direta do Autoware desde o alto-nível até o baixo-nível. A implementação do Autoware no microcontrolador foi batizada microAutoware.

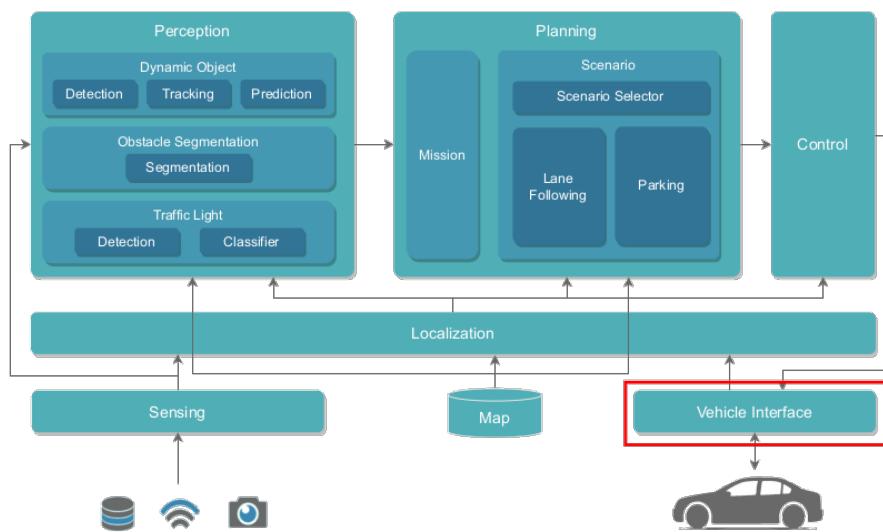
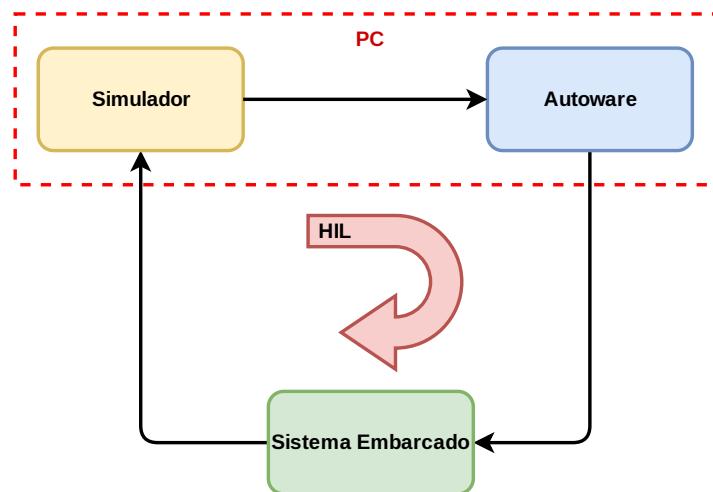


Figura 4: Escopo do projeto na arquitetura Autoware.

Para realização dos testes e validação do microAutoware, foi considerada a utilização do simulador de última geração CARLA(CARLA Team, 2024), que permite que o *hardware* possa ser testado sem a necessidade de um protótipo real do veículo autônomo, poupando assim: custo, risco dos equipamento e principalmente humano, e encurtando o tempo de pesquisa. Dessa forma, a validação do projeto é realizada por meio de uma arquitetura *Hardware-In-the-Loop* (HIL), conforme diagrama da Figura 5. Nesse cenário, o simulador representa o veículo, que alimenta o Autoware com dados de sensores de alto nível utilizados para obtenção do comando de controle que o veículo deve realizar, que é enviado para o sistema embarcado (em *hardware*). Esse por sua vez, é responsável por realizar o controle do veículo no simulador, e realizar a leitura dos sensores de baixo nível do mesmo. Com isso, pode-se realizar o teste do sistema embarcado em uma estrutura que aproxima a aplicação real.

Figura 5: Arquitetura de teste do *hardware*.

Com isso, a Figura 6 ilustra as principais ferramentas que serão somadas ao sistema embarcado baseado em STM32 para viabilização do projeto: Autoware, ROS e micro-ROS são aliados ao CARLA para permitir a implementação e validação do microAutoware. Para integração do CARLA junto ao Autoware foi utilizada a *bridge* produzida por Kaljavesi et al. (2024), realizando algumas modificações para integração com o microAutoware.

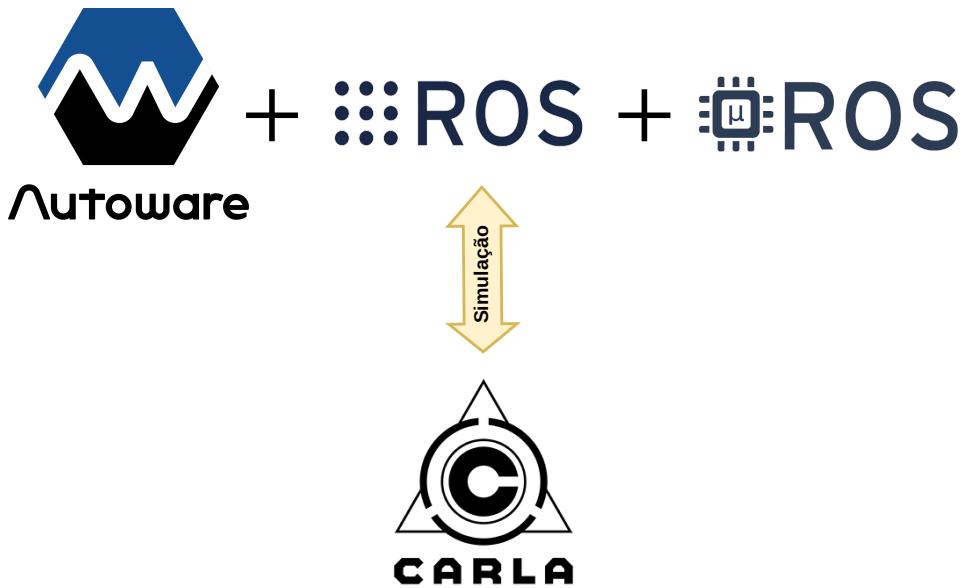


Figura 6: Ferramentas para viabilização do projeto.

A *vehicle interface* é um módulo que mesmo não especificado profundamente necessita de atender a alguns requisitos de comunicação, sendo eles os dados que serão recebidos (*subscribers*), os dados que devem ser reportados (*publishers*) e ainda um serviço de troca de modo de controle. A implementação da interface se dá por meio de um nó do ROS, que deve obrigatoriamente receber o **comando de controle** do veículo, e devolver o **modo de controle atual**, o **ângulo de esterçamento atual** e as **velocidades atuais** do veículo. Ainda, deve prover um serviço que dada uma **solicitação de troca de modo de controle**, responde se a mesma foi feita com sucesso ou não. A Figura 7 explicita na forma de um diagrama as entradas e saídas

da interface Autoware – veículo.

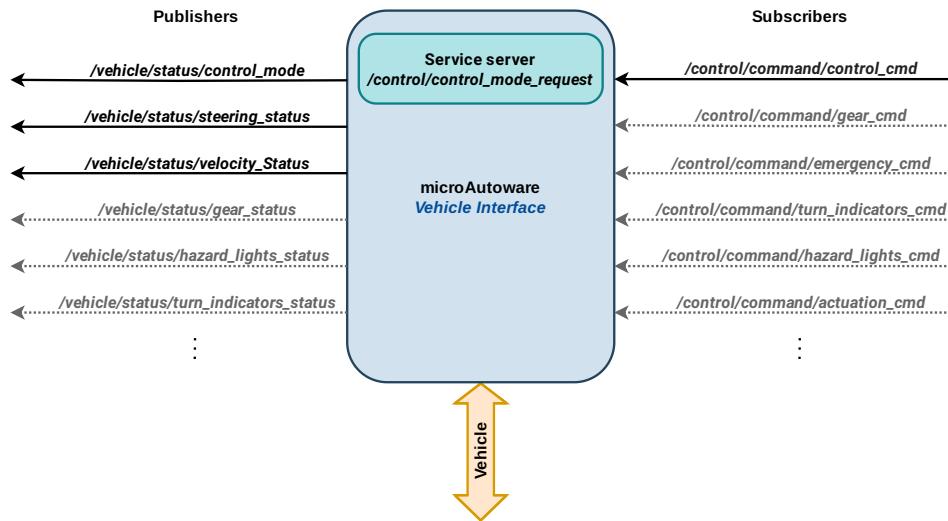


Figura 7: Diagrama de tópicos da *vehicle interface*.

3.1 Requisitos

Requisitos funcionais

- Comunicação com o Autoware;
- Controle da aceleração, frenagem e direção do veículo;
- Controle dos faróis e luzes de sinalização (seta) do veículo;
- Teleoperação do veículo por um *joystick* em *hardware*;
- Troca do modo de operação por meio da *switch* do *joystick*;
- Subscrição por meio do micro-ROS em todos os tópicos necessários do Autoware;
- Publicação a partir micro-ROS em todos os tópicos necessários do Autoware;
- Comunicação a partir micro-ROS em todos os serviços necessários do Autoware.

Requisitos não funcionais

- A *vehicle interface* deve ser construída na forma de um pacote portável para outros microcontroladores STM32;
- O interfaceamento com o veículo deve ser intercambiável com diferentes configurações;
- Deve-se garantir sincronização de *timestamp* entre o Autoware e o microcontrolador;
- O sistema embarcado deve abstraír o veículo como um sistema *Drive-By-Wire* (DBW) para o Autoware.

3.2 Componentes

Placa de desenvolvimento NUCLEO-H753ZI

- Microcontrolador STM32H753ZI;
- ARM Cortex-M7;
- 1 MB RAM;
- 2 MB Flash;
- *Clock* máximo de 480 MHz;
- DMA;
- Comunicação:
 - UART/USART;
 - Ethernet;
 - USB.
- Custo: US\$ 27,00.



Figura 8: NUCLEO-753ZI.

Joystick

- Tensão de operação: 3V3 – 5V;
- Saída analógica referente ao eixo *x*;

- Saída analógica referente ao eixo y ;
- Saída digital referente ao eixo z ;
- Custo: R\$ 10,00.



Figura 9: Joystick 2 eixos.

Estação de trabalho

- Ubuntu 22.04 LTS;
- CPU Intel Core I7 5960X;
- 4x Memória RAM DDR4 8 GB;
- 2x GPU NVIDIA GEFORCE GTX TITAN X 12 GB;
- Custo: R\$ 10000,00.

Módulo FTDI

- Tensão de operação: 3V3 – 5V;
- Circuito integrado: FT232RL;
- *Baudrate* mínima: 183,1 bps;
- *Baudrate* máxima: 3 Mbps;
- Custo: R\$ 20,00.

3.3 Arquitetura

A arquitetura HIL é baseada no uso de dois sistemas, um computador, executando o CARLA e o Autoware, e um sistema embarcado executando o FreeRTOS. A Figura 10 mostra os blocos presentes em cada sistema e como eles se comunicam. Observa-se que o CARLA não recebe dados diretamente do Autoware, e que existem duas linhas comunicação *full-duplex*, entre o Autoware e a tarefa microAutoware, e entre o simulador e a tarefa TaskControle.

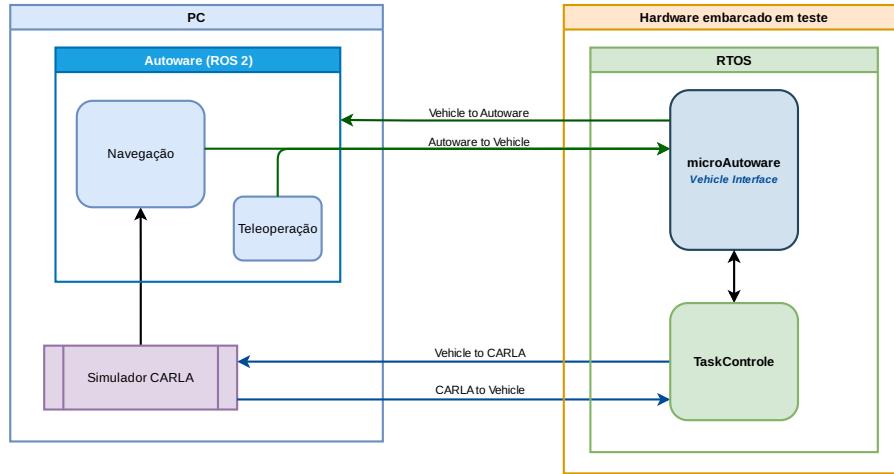


Figura 10: Diagrama de blocos em alto nível da arquitetura HIL.

Especificando como a comunicação é feita, cada linha de comunicação é dada por uma porta de comunicação serial assíncrona (UART), sendo uma responsável pela comunicação com o Autoware, utilizando o agente do micro-ROS para o interfaceamento Serial-ROS, enquanto a outra se comunica com o simulador, por meio do pacote Carla-Serial-Bridge¹. A recepção e transmissão via UART é feita utilizando DMA, de forma a preservar o CPU e reduzir o tempo dentro da rotina de interrupção, sendo utilizado um módulo DMA para cada UART para evitar sobrecarga. Adicionalmente, para leitura do joystick são utilizados duas entradas analógicas e uma entrada digital, que é atrelada à um canal de interrupções externas (EXTI).

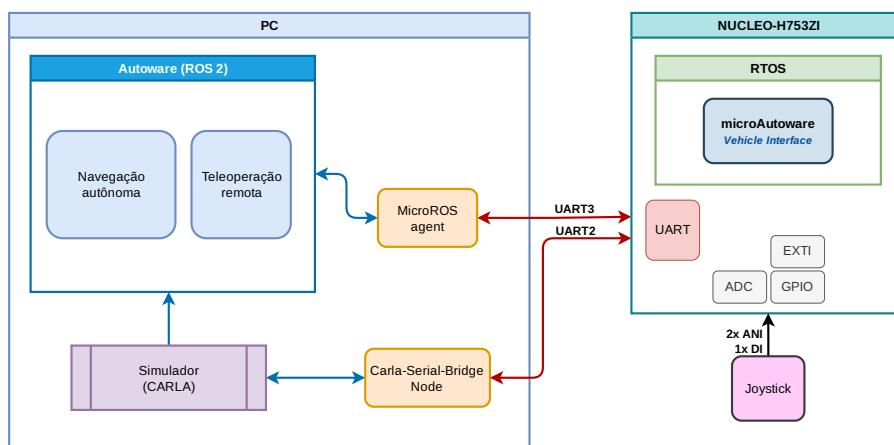


Figura 11: Diagrama de blocos do sistema embarcado.

¹Disponível em: github.com/LMA-FEM-UNICAMP/carla_serial_bridge

A ligação física se dá como mostrado no esquemático da Figura 12, onde a alimentação do circuito é feita por meio da porta USB conectada ao ST-LINK. Um detalhe importante de montagem é que o pino de alimentação 3,3 V do módulo FTDI não foi conectado à placa NUCLEO, com intuito de evitar correntes parasitas no circuito de alimentação devido a desparidade de referência causada pelos conversores da NUCLEO e do FTDI. A montagem física é mostrada na Figura 13.

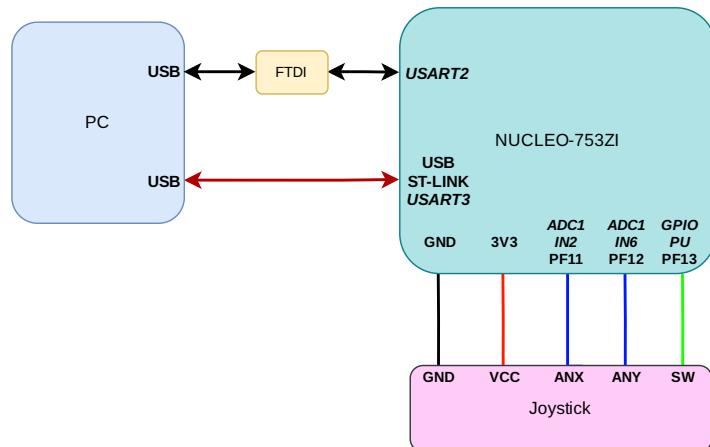


Figura 12: Esquemático de ligações elétricas.

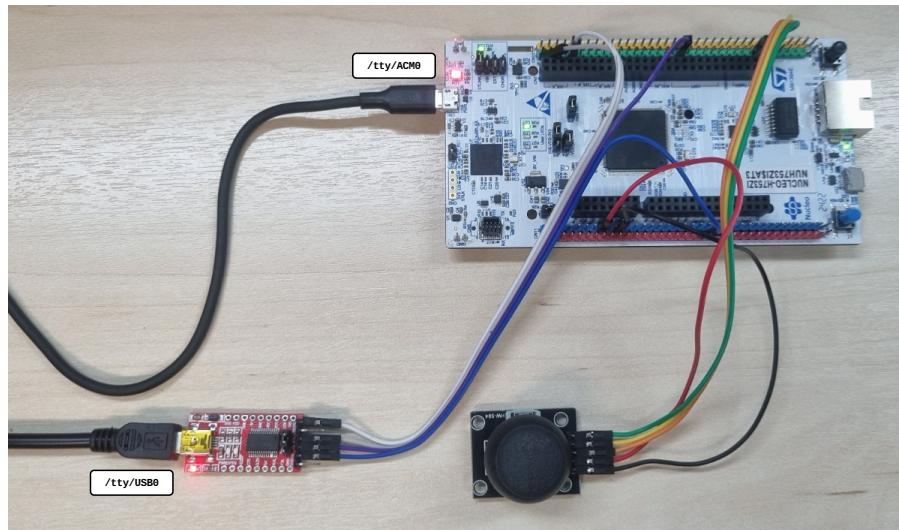


Figura 13: Montagem do circuito.

Periféricos necessários

- Direct Memory Access
- UART
- ADC
- GPIO
- EXTI

3.4 Método de desenvolvimento

Para realização das atividades necessárias para o desenvolvimento do projeto, foi selecionado o método V, onde, como mostrado na Figura 14, as tarefas de criação são feitas em paralelo com as de validação, logo, permite a entrega final de forma confiável e sem a necessidade de re-manufaturar grandes módulos.

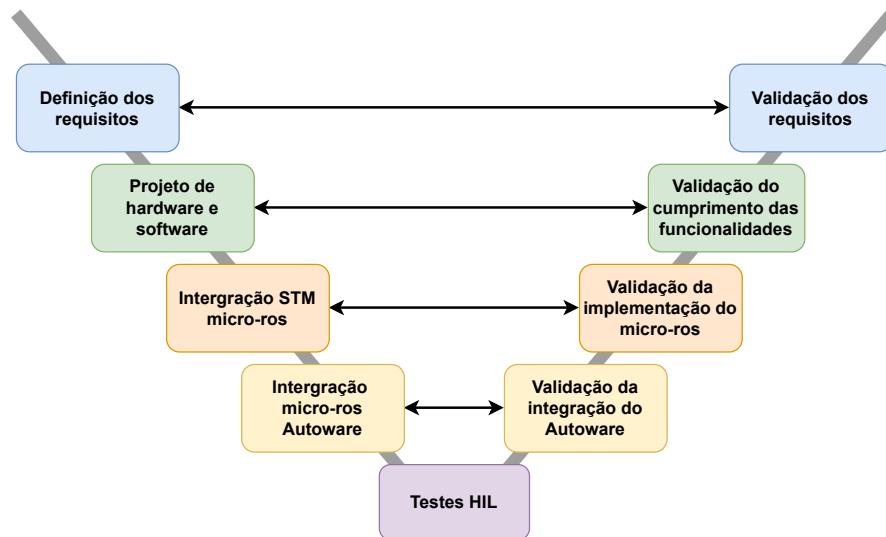


Figura 14: Modelo de execução das atividades do projeto.

O projeto teve sua realização em um horizonte de 9 semanas, como mostrado na Tabela 1, sendo realizadas entregas parciais nas semanas 2, 4, 7 e 9.

Atividade/Semana	1	2	3	4	5	6	7	8	9
Proposta do projeto		■							
Projeto de <i>hardware e software</i>			■	■					
Integração do STM com o micro-ROS			■						
Integração do micro-ROS com o Autoware				■	■				
Implementação das tarefas do sistema embarcado					■	■	■	■	
Construção do ambiente de testes					■	■	■	■	
Realização dos testes						■	■	■	
Escrita do relatório		■	■	■	■	■	■	■	

Tabela 1: Cronograma de atividades.

- **Semana 2:** Apresentação Etapa 1
- **Semana 4:** Apresentação Etapa 2
- **Semana 7:** Apresentação Etapa 3
- **Semana 9:** Apresentação Final

3.5 Projeto de *software*

Estados do sistema

O sistema pode se encontrar em três estados durante seu funcionamento: **AUTOWARE**, **MANUAL** ou **EMERGÊNCIA**, sendo o primeiro escolhido na inicialização. No primeiro, o veículo é conduzido pelos comandos do Autoware, e caso seja solicitado pelo Autoware ou pelo botão do *joystick*, ou ainda se houver perda da *deadline* de recebimento dos comandos do Autoware, o estado do sistema se altera para **MANUAL**. Neste segundo modo, o veículo é comandado pelo *joystick*, podendo voltar à ser autônomo por comandos do próprio controle ou do Autoware. Para os dois estados iniciais, caso haja a perda da janela de tempo de recebimento dos sinais do simulador, o sistema entra em modo **EMERGÊNCIA**, onde o veículo efetua uma parada de emergência. Caso a comunicação com o CARLA seja restaurada, o sistema retorna ao estado **MANUAL**.

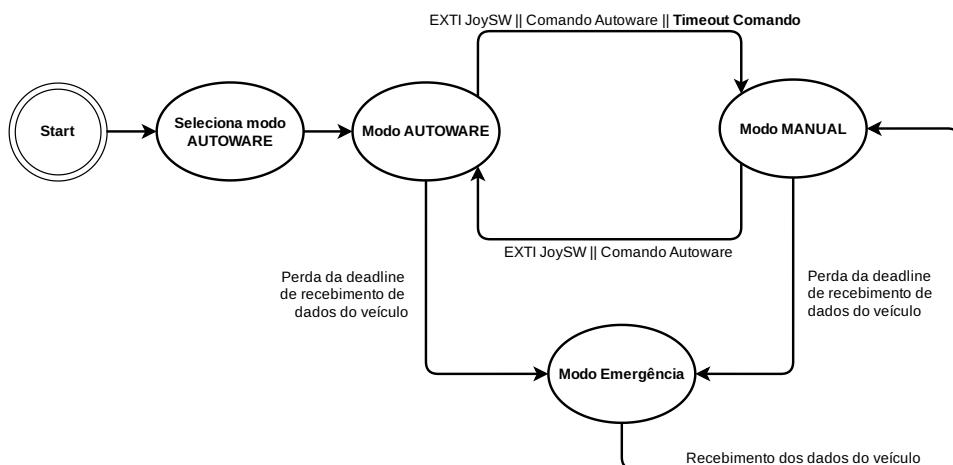


Figura 15: Máquina de estados do sistema.

Protocolo de comunicação serial

A comunicação serial entre o sistema embarcado e o simulador é realizada por meio de um protocolo implementado via máquina de estados, onde a Figura 16 representa o envio de dados do microcontrolador para o CARLA, enquanto a Figura 17 o caminho contrário.

1. CARLA → RTOS (USART3)

- Informações:
 - Velocidade longitudinal do veículo – `float fLongSpeed;`
 - Velocidade lateral do veículo – `float fLatSpeed;`
 - Taxa de guinada do veículo (velocidade angular no eixo z) – `float fHeadingRate;`
 - Ângulo de esterçamento da roda virtual – `float fSteeringStatus;`
- Padrão da mensagem: `#A%c%c%cB%c%c%cC%c%c%cD%c%ccc$`
- Tamanho da mensagem: 22 bytes;
- Envia uma *ThreadFlag* ao receber a mensagem inteira.

2. RTOS → CARLA (USART2)

- Informações:
 - Ângulo de esterçamento desejado do volante – `float fSteeringAngle;`
 - Velocidade de esterçamento do volante – `float fSteeringVelocity;`
 - Velocidade linear desejada para o veículo – `float fSpeed;`
 - Aceleração linear desejada para o veículo – `float fAcceleration;`
 - Jerk linear desejada para o veículo – `float fJerk;`
 - `unsigned char ucControlMode;`
- Padrão da mensagem: `#S%c%c%c%cW%c%c%cV%c%c%cA%c%c%cJ%c%c%cM%c$`
- Tamanho da mensagem: 30 bytes.

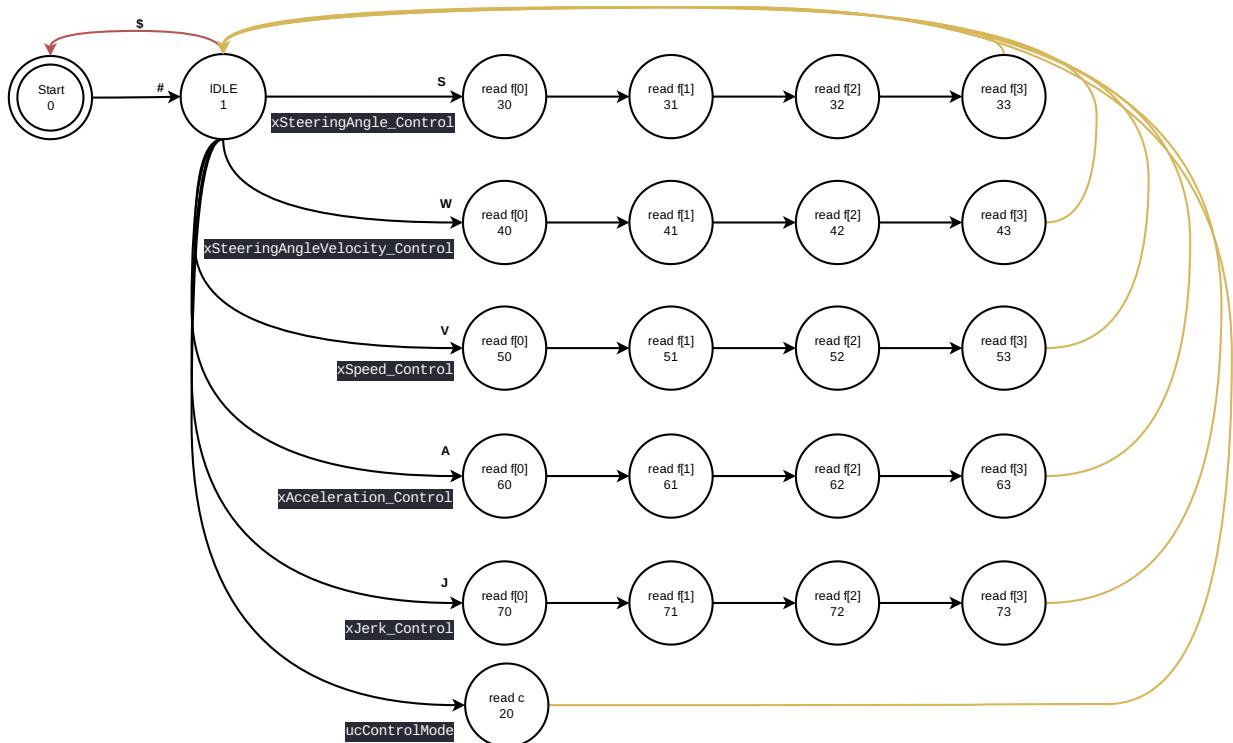


Figura 16: Máquina de estados da comunicação serial do RTOS para o CARLA.

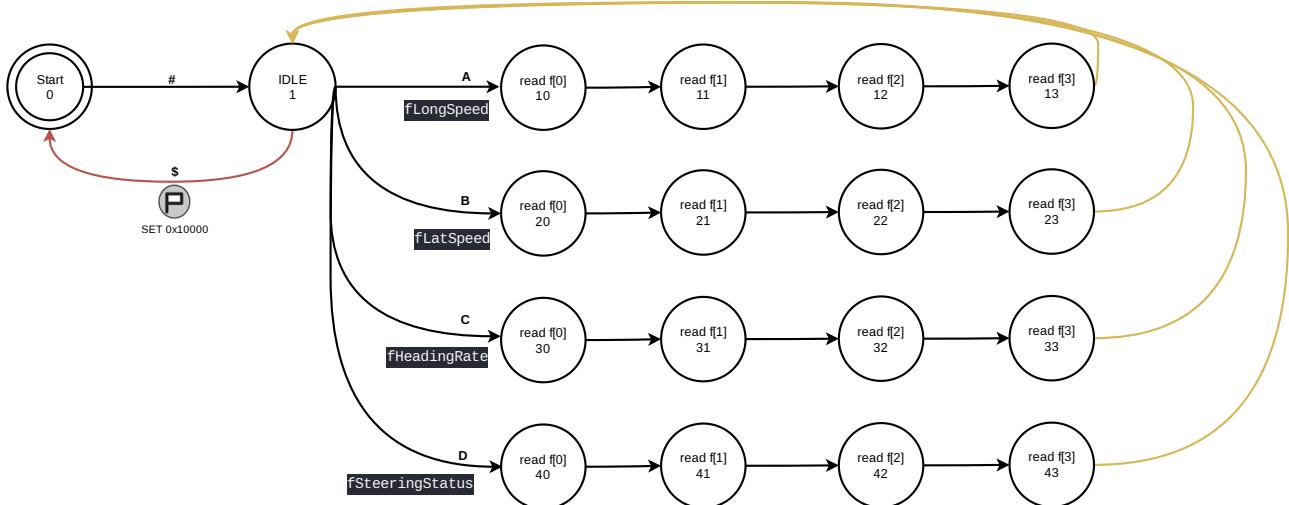


Figura 17: Máquina de estados da comunicação serial do CARLA para o RTOS.

Tarefas

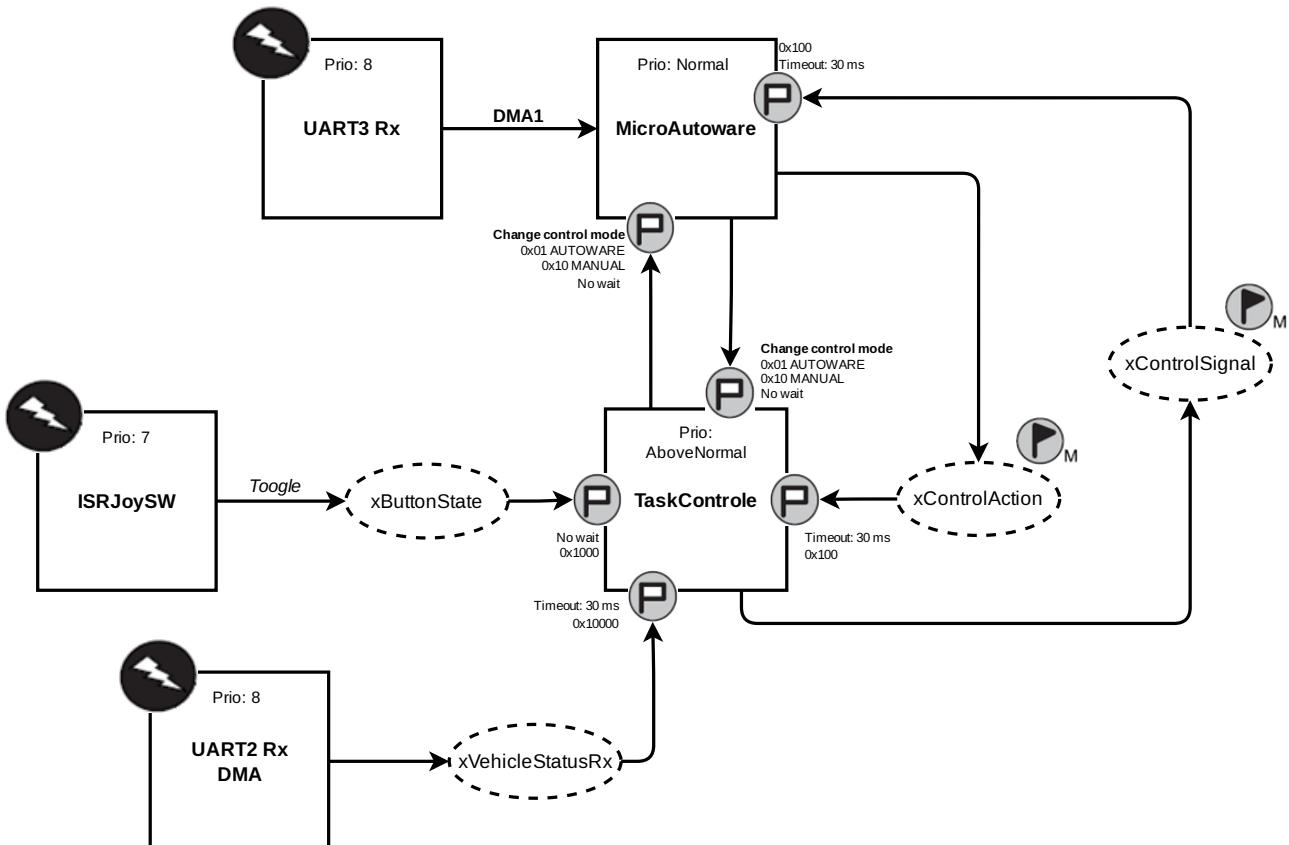


Figura 18: Diagrama do sistema embarcado.

Nome	MicroAutoware
Prioridade	Normal
Tamanho da stack	3500 kB
Detalhes	Leitura dos <i>subscribers</i> Autoware, leitura dos <i>subscribers</i> CARLA, envio das informações de controle e modo de operação para a TaskControle, recebimentos das informações de controle da TaskControle, escrita dos <i>publishers</i> Autoware, escrita dos <i>publishers</i> CARLA.

Tabela 2: Especificaçõe da tarefa MicroAutoware.

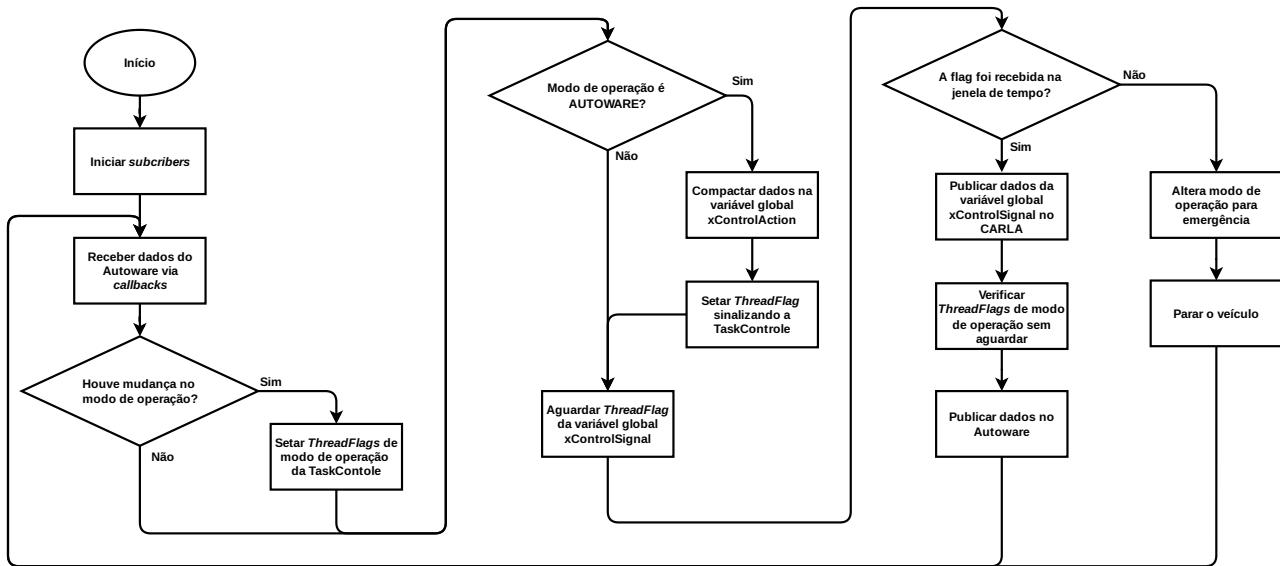


Figura 19: Fluxograma da tarefa MicroAutoware.

Nome	TaskControle
Prioridade	AboveNormal
Tamanho da stack	500 kB
Detalhes	Realiza o controle do veículo utilizando a referência dada pelo <i>joystick</i> ou pelo Autoware, dado o modo de operação, podendo ser MANUAL ou AUTOWARE, respectivamente. A alteração do modo é feita por <i>ThreadFlag</i> , gerada por ISR ou pelo Autoware. Em caso do modo de operação AUTOWARE, os sinais de controle são recebidos por variável global e sincronizados por <i>ThreadFlag</i> , com tempo de 30 ms, onde caso não receba, entra em algum modo de segurança. Em caso de operação MANUAL, o <i>joystick</i> é lido por DMA, aguardando 20 ms antes de cada leitura, convertendo os valores analógicos em sinais de controle, onde também caso haja algum erro, o modo de emergência é acionado. O sinal de controle é enviado para o MicroAutoware por uma variável global e sincronizado por <i>ThreadFlag</i> .

Tabela 3: Especificação da tarefa TaskControle.

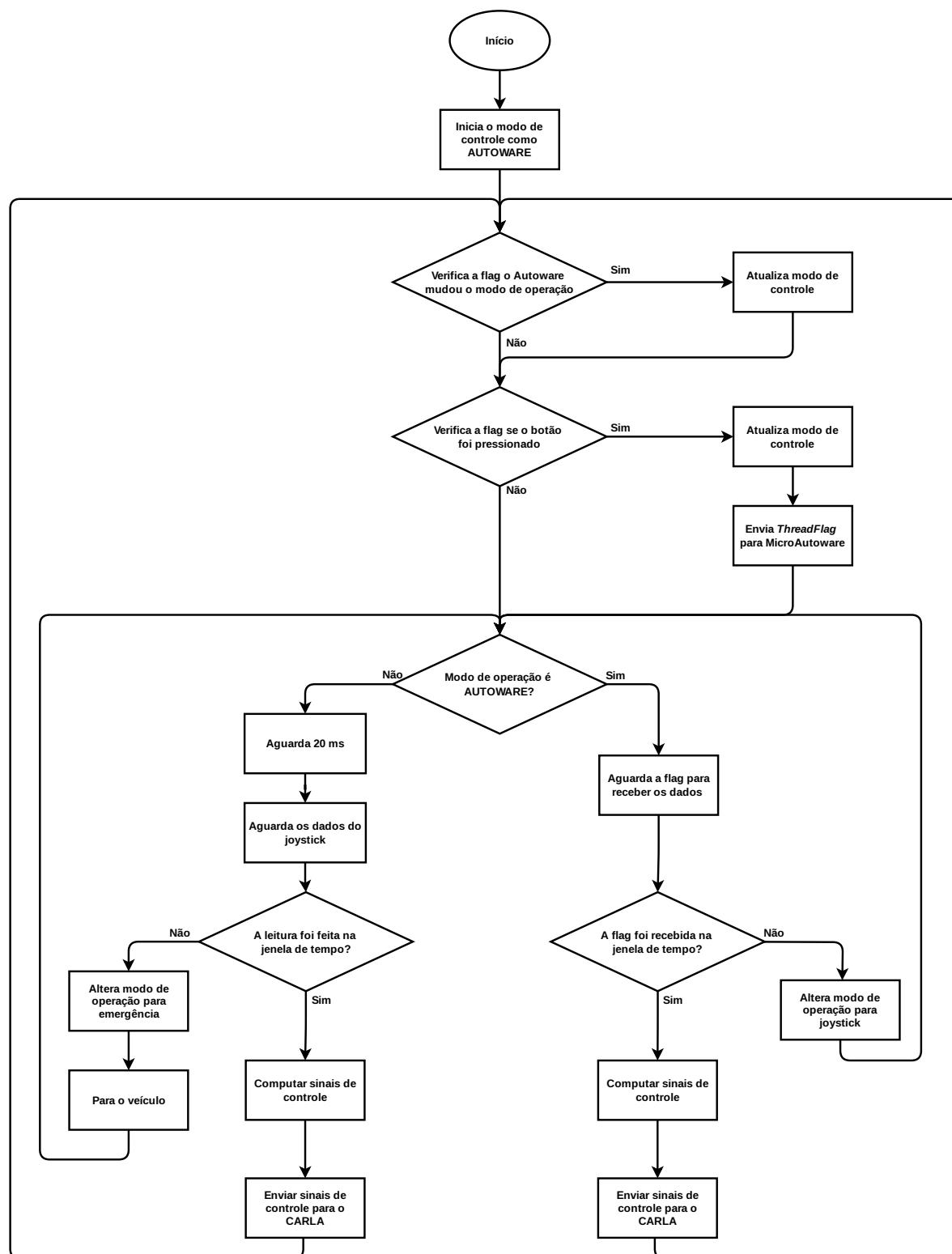


Figura 20: Fluxograma da tarefa TaskControle.

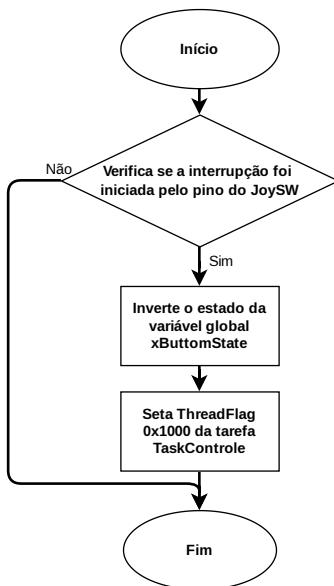


Figura 21: Fluxograma da ISR JoySW.

Sinalização xButtonState

- **Objeto:** *ThreadFlag*
- **Flag:** 0x1000
- **Modo:** *No wait*
- **Descrição:** Sinaliza ocorrência da interrupção do botão JoySW.

Sinalização xControlAction

- **Objeto:** *ThreadFlag*
- **Flag:** 0x0100
- **Modo:** *Timeout* 30 ms
- **Descrição:** Sinaliza o recebimento de dados pela variável global *xControlAction*.

Sinalização xControlSignal

- **Objeto:** *ThreadFlag*
- **Flag:** 0x0100
- **Modo:** *Timeout* 30 ms
- **Descrição:** Sinaliza o recebimento de dados pela variável global *xControlSignal*.

Alteração do modo de condução por interrupção JoySW

- **Objeto:** *ThreadFlag*
- **Flags:**
 - Modo de controle alterado para AUTOWARE: 0x01
 - Modo de controle alterado para MANUAL: 0x10
- **Modo:** *No wait*
- **Descrição:** Realiza a sincronização do modo de operação da tarefa TaskControle para a MicroAutoware.

Alteração do modo de condução pelo Autoware

- **Objeto:** *ThreadFlag*
- **Flags:**
 - Modo de controle alterado para AUTOWARE: 0x01
 - Modo de controle alterado para MANUAL: 0x10
- **Modo:** *No wait*
- **Descrição:** Realiza a sincronização do modo de operação da tarefa MicroAutoware para a TaskControle.

Proteção de recursos

Variável global `xControlSignal`

- Protegida por MUTEX.
 - `MutexControlSignal`

Variável global `xControlAction`

- Protegida por MUTEX.
 - `MutexControlAction`

Padronização de código ROS

- Subscriber: `nome_subscriber_sub_`
- Publisher: `nome_subscriber_pub_`
- Service server: `nome_subscriber_server_`
- Mensagem: `nome_mensagem_msg_`
- Node: `nome_do_node`
- Callback: `nome_do_topico_callback`

3.6 Periféricos

Joystick

1. Eixos x e y

- ADC de 16 bits *multi-channel*;
- Leitura contínua por DMA;
- Modo de *clock* assíncrono dividido por 256;
- Tempo de amostragem 387,5 ciclos;
- Interrupções desativadas para preservar a CPU.

2. Botão

- GPIO em modo *pull-up*;
- Interrupção EXTI.

Processamento de sinais

Para conversão do valor discreto de leitura analógica obtido por meio do conversor analógico-digital (*analog-digital converter – ADC*), foi realizada uma etapa de calibração, onde foram tomadas as medidas dos valores máximos, mínimos e de zero para os dois eixos do *joystick*. Tomado esses valores, foi realizado o processo de linearização por partes, onde o valor discreto obtido, é mapeado no intervalo $(-1, 1)$, considerando uma banda morta B ao redor do zero para evitar flutuação de zero e melhorar a usabilidade dos eixos de forma independente.

O a interpolação é realizada por meio de (1), resultando nos gráficos mostrados nas Figuras 22 e 23, para o mapeamento nos eixos x e y , respectivamente.

O eixo x é alocado como a variável de esterçamento do volante, enquanto o eixo y representa os sinais de acelerador e freio. Para valores positivos do eixo y , obtém-se a medida do acelerador no intervalo $(0, 1)$, enquanto para valores negativos o freio é mapeado entre 0 e 1.

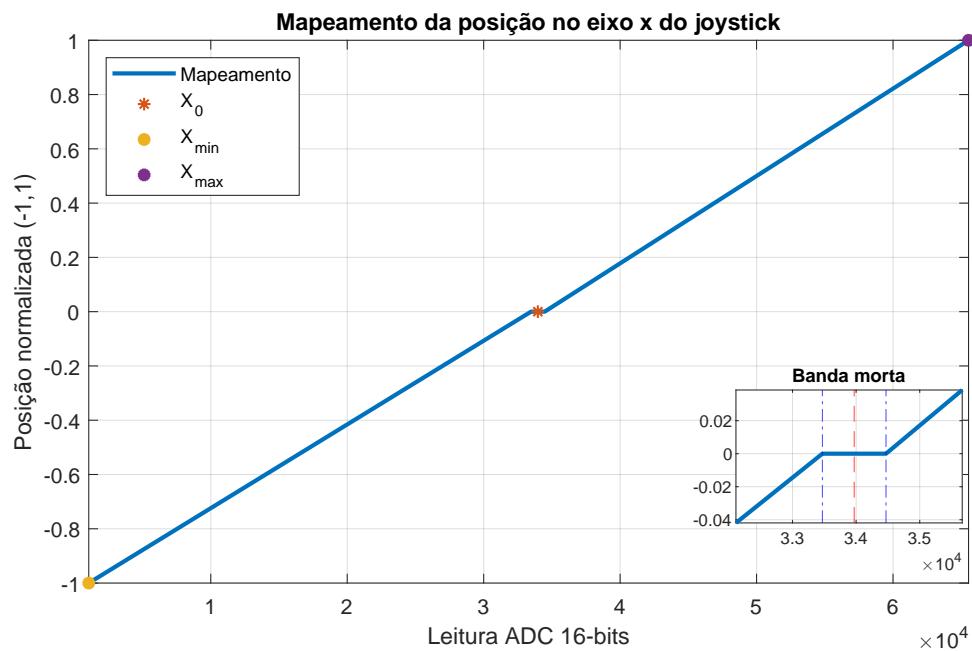


Figura 22: Mapeamento da posição no eixo x normalizada do *joystick* de acordo com a leitura analógia do ADC.

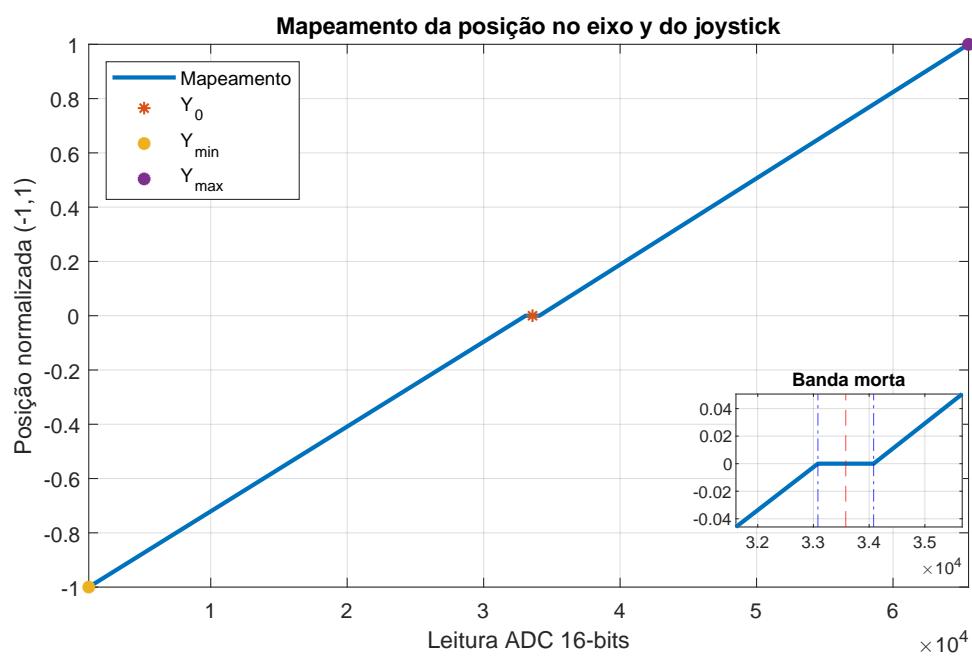


Figura 23: Mapeamento da posição no eixo y normalizada do *joystick* de acordo com a leitura analógia do ADC.

$$p(v) = \begin{cases} 0, & \text{se } -B \leq v - V_0 \leq B \\ \frac{v - V_0 - B}{V_{max} - V_0 - B}, & \text{se } v > V_0 + B \\ \frac{v - V_0 + B}{V_0 - V_{min} - B}, & \text{se } v < V_0 - B \end{cases} \quad (1)$$

Comunicação serial – UART

1. UART2 – Comunicação com o simulador.

- **Baudrate:** 921600 bps
- **Modo de leitura:** DMA única;
- **Modo de escrita:** DMA única;
- **Interface física:** Módulo FTDI.

2. UART3 – Comunicação com o Autoware.

- **Baudrate:** 921600 bps
- **Modo de leitura:** DMA circular;
- **Modo de escrita:** DMA única;
- **Interface física:** ST-LINK.

4 Manual de utilização

5 Problemas identificados e não resolvidos

6 Códigos da comunidade

Referências

BEDOYA, O. G. *Análise de risco para a cooperação entre o condutor e sistema de controle de veículos autônomos*. Tese (Doutor em Engenharia Mecânica) — Universidade Estadual de Campinas, Campinas, SP, fev. 2016. Disponível em: <<https://repositorio.unicamp.br/Busca/Download?codigoArquivo=471471>>.

CARLA Team. *CARLA: Open-source simulator for autonomous driving research*. 2024. Disponível em: <<https://carla.org>>.

KALJAVESI, G. et al. CARLA-Autoware-Bridge: Facilitating Autonomous Driving Research with a Unified Framework for Simulation and Module Development. In: *2024 IEEE Intelligent Vehicles Symposium (IV)*. [S.l.: s.n.], 2024. p. 224–229. ISSN 2642-7214.

micro-ROS. *micro-ROS puts ROS 2 onto microcontrollers*. 2024. Disponível em: <<https://micro.ros.org>>.

The Autoware Fundation. *Architecture overview*. 2023. Disponível em: <<https://autowarefoundation.github.io/autoware-documentation/main/design/autoware-architecture/>>.

Apêndices

Carla-Autoware-Bridge

Repositório: <github.com/LMA-FEM-UNICAMP/Carla-Autoware-Bridge> (derivado do repositório original <github.com/TUMFTM/Carla-Autoware-Bridge>).

carla_serial_bridge

Repositório: <https://github.com/LMA-FEM-UNICAMP/carla_serial_bridge>.