



**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

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

---

## Sumário

<b>1 Resumo</b>	<b>2</b>
<b>2 Motivação</b>	<b>3</b>
<b>3 Documentação</b>	<b>5</b>
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
<b>4 Manual de utilização</b>	<b>24</b>
4.1 Inicialização . . . . .	24
4.2 Modo AUTOWARE . . . . .	24
4.3 Modo MANUAL . . . . .	24
<b>5 Problemas identificados e não resolvidos</b>	<b>25</b>
<b>Referências</b>	<b>26</b>
<b>Apêndices</b>	<b>27</b>

---

# 1 Resumo

A arquitetura hierárquica de veículos autônomos é comumente dividida em duas partes: alto e baixo nível. O primeiro composto principalmente por um ou mais computadores de propósito geral, que executam algoritmos que exigem alto poder computacional, e é alimentado diretamente por sensores de alto nível como câmeras, LiDARs e radares, além de receber sinais da camada de baixo nível. Esse por sua vez, é normalmente concentrado por um ou mais sistemas embarcados, que realizam a leitura de sensores de baixo nível como encoder, IMU, GPS, e fazem a atuação em motores, realizando o controle dos mesmos conforme o sinal recebido do alto nível. De forma a integrar todos os componentes do alto nível, o Autoware é um *framework* baseado em ROS que vem sendo amplamente utilizado como base para veículos autônomos, onde por meio de um módulo de interface com o veículo, nomeado *vehicle interface*, realiza a troca de dados com o baixo nível de forma transparente à natureza do mesmo.

Porém, por meio do micro-ROS, é possível trazer o ROS para dentro do microcontrolador, dessa forma, integrando alto e baixo nível de forma implícita dentro do ecossistema ROS. Tal capacidade, permite então que a *vehicle interface* do Autoware possa estar posicionada dentro do sistema embarcado, fazendo com que o *framework* esteja em ambas as camadas da hierarquia de automação, e dessa forma, possa ser executado em um sistema operacional de tempo real. Dessa forma, o microAutoware é enunciado como a possibilidade de, com o apoio do micro-ROS, levar o Autoware para dentro do sistema embarcado, se comunicando com os demais nós por meio dos tópicos e serviços do ROS, sem necessidade de interromper sua arquitetura original.

Para validação da utilização da interface com o veículo dentro do sistema embarcado, foram realizadas simulações *Hardware-In-the-Loop*, onde o sistema embarcado, conectado ao Autoware, foi utilizado para controlar um veículo autônomo virtual, simulado no CARLA Simulator. Com isso, foi possível realizar os testes do *hardware* sem a necessidade de um protótipo real, o que principalmente no ramo de veículos autônomos, traz além de redução de custos e tempo de testes, mais segurança para todas as pessoas envolvidas.

Dentre os recursos implementados, além de receber e enviar os dados necessários para o funcionamento do Autoware, o sistema engloba também um modo de controle manual em *hardware*, utilizando um *joystick*, de forma a representar por exemplo os pedais e o volante do veículo, realizando um comando indireto do carro a partir do sistema de automação. Também foram modelados protocolos de segurança para no caso de perda de *deadlines* de recebimentos das mensagens do alto nível ou do veículo (simulador), o sistema embarcado possa de forma independente passar o controle do carro para o motorista, ou entrar em modo de emergência, efetuando uma parada autônoma. Com isso, o microAutoware provou cumprir todos os requisitos de uma *vehicle interface*, apresentando resultados promissores para utilização do mesmo como tal em sistemas reais.

## 2 Motivação

Veículos autônomos estão cada vez mais presentes nas cidades, trazendo a premissa de torná-las mais seguras, inteligentes e sustentáveis, além de trazer mais conforto aos passageiros, onde não é necessário que um deles venha a ser o motorista. A Figura 1 apresenta o VILMA, Veículo Autônomo do Laboratório de Mobilidade Autônoma, que por meio de uma gama de sensores e atuadores, consegue conduzir de forma 100% autônoma.



Figura 1: Veículo Autônomo do LMA.

Parte desses sensores e atuadores são apresentados no diagrama da Figura 2, onde, os sensores de alto nível como câmera são conectados à um computador de propósito geral, e os sensores de baixo nível e atuadores são lidos e comandados por um sistema embarcado de tempo real presente no dSpace MicroAutobox. Tal sistema embarcado é uma solução comercial, de *software* proprietário, ou seja, havendo necessidade da posse de licenças e personalização limitada. Por outro lado, tal sistema embarcado pode ser substituído por sistemas embarcados baseados na arquitetura ARM-Cortex, onde podem ser implementados diferentes sistemas operacionais de tempo real (inclusive de código aberto), dando mais liberdade e versatilidade para o desenvolvimento do baixo-nível de veículos autônomos.

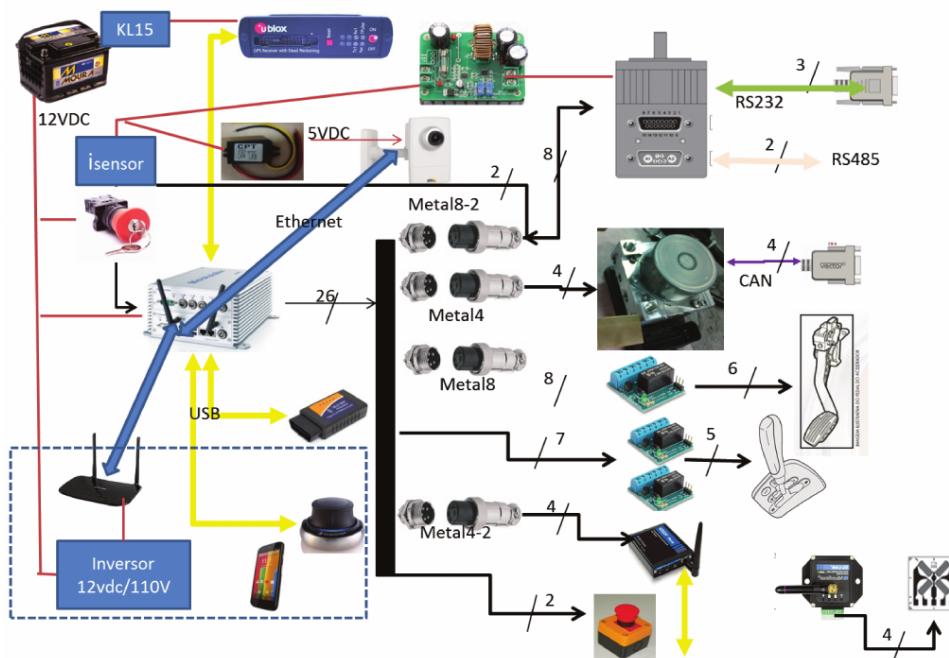


Figura 2: Diagrama de *hardware* do VILMA01 (BEDOYA, 2016).

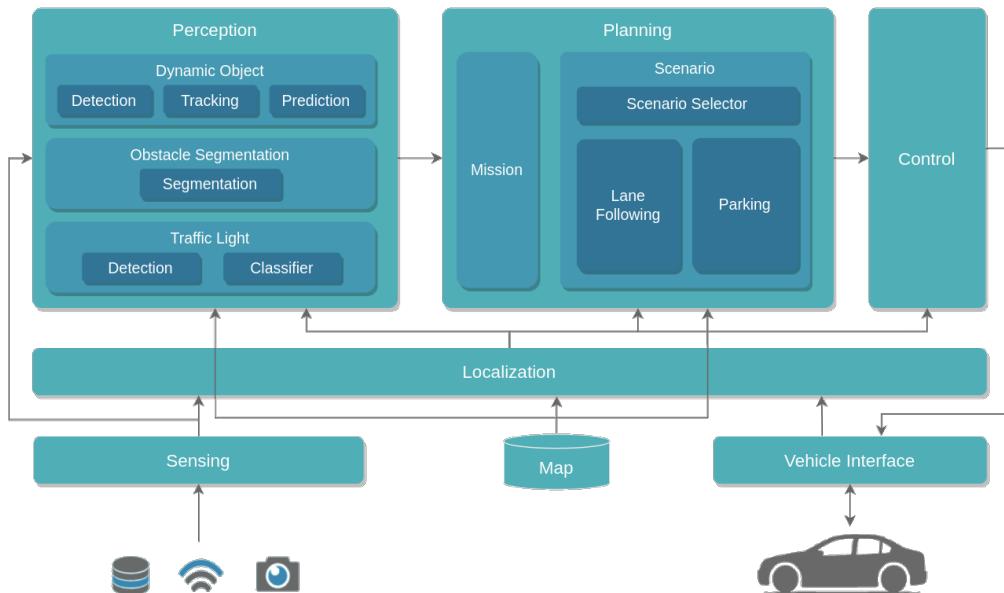


Figura 3: Arquitetura de alto nível (The Autoware Fundation, 2023).

Todavia, para possibilitar a condução autônoma do veículo, os itens mostrados na Figura 2 e muitos outros que podem ser integrados devem ser interligados à arquitetura de automação do veículo, que normalmente é composta por módulos como percepção, planejamento, controle, localização e mapeamento. Nesse âmbito, o Autoware é um *software open-source* que traz a arquitetura completa e todas as funcionalidades básicas para o guiamento de um veículo autônomo. Sua arquitetura é mostrada na Figura 3, onde se percebe que a partir das entradas fornecidas pelos sensores e pelo mapa, o mesmo realiza o comando do carro a partir da *vehicle interface*, que pode ser implementada de diversas formas, como por exemplo, protocolo serial ou rede CAN.

Dessa forma, o presente trabalho busca explorar a comunicação do Autoware com o veículo, por meio da utilização de sistemas embarcados de tempo real baseados na arquitetura ARM-Cortex, visando a presença do *framework* em alto e baixo nível na hierarquia do veículo autônomo.

### 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.

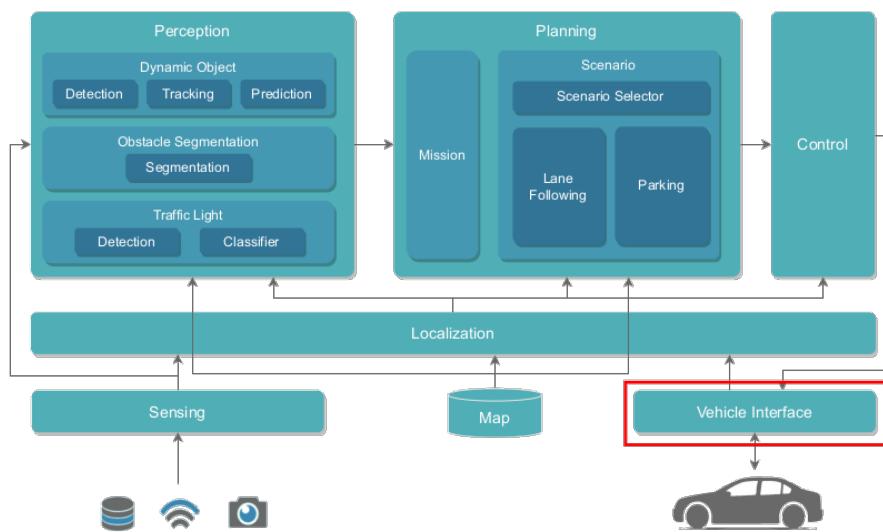
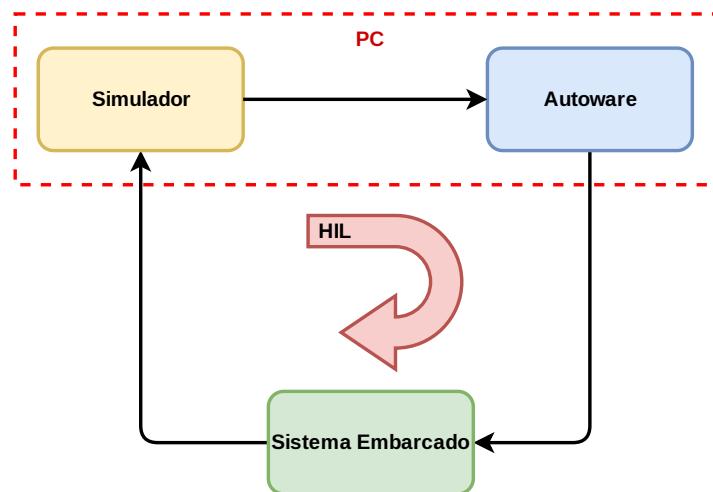


Figura 4: Escopo do projeto na arquitetura Autoware.

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 equipamentos e principalmente humano, além de encurtar 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.

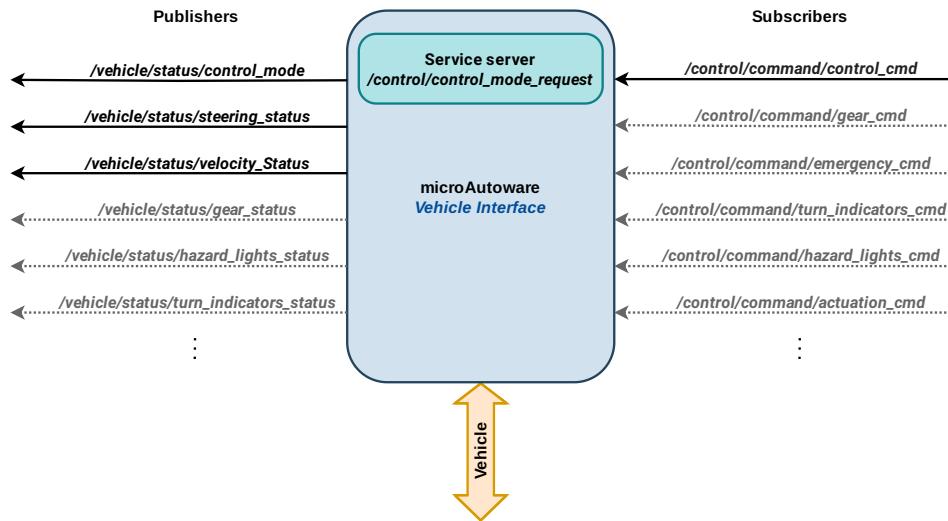


Figura 7: Diagrama de tópicos da *vehicle interface*.

### 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.

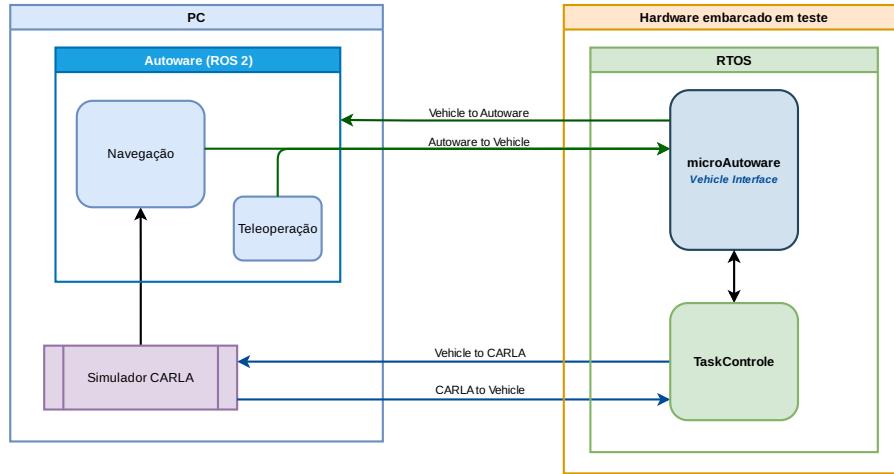


Figura 10: Diagrama de blocos em alto nível da arquitetura HIL.

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-Bridge<sup>1</sup>. 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).

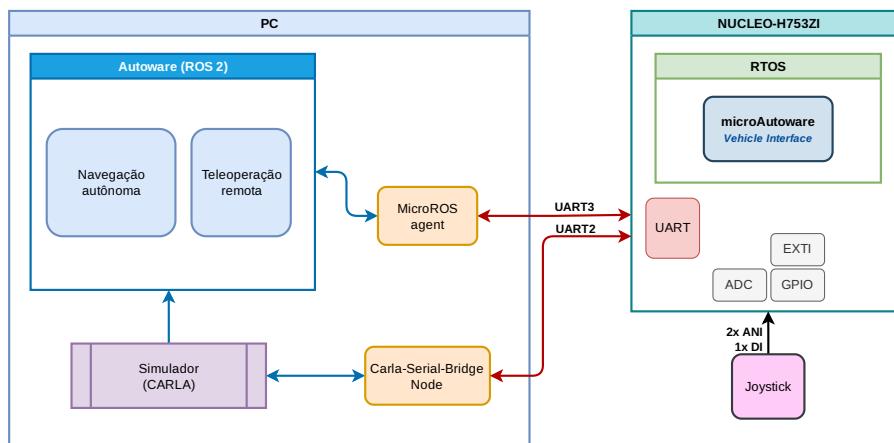


Figura 11: Diagrama de blocos do sistema embarcado.

<sup>1</sup>Disponível em: [github.com/LMA-FEM-UNICAMP/carla\\_serial\\_bridge](https://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.

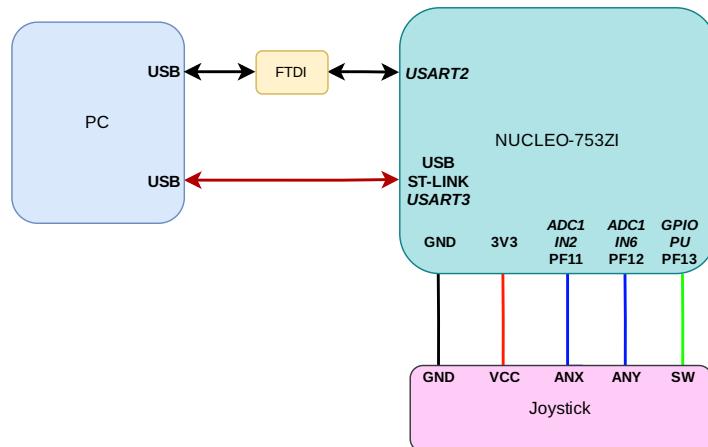


Figura 12: Esquemático de ligações elétricas.

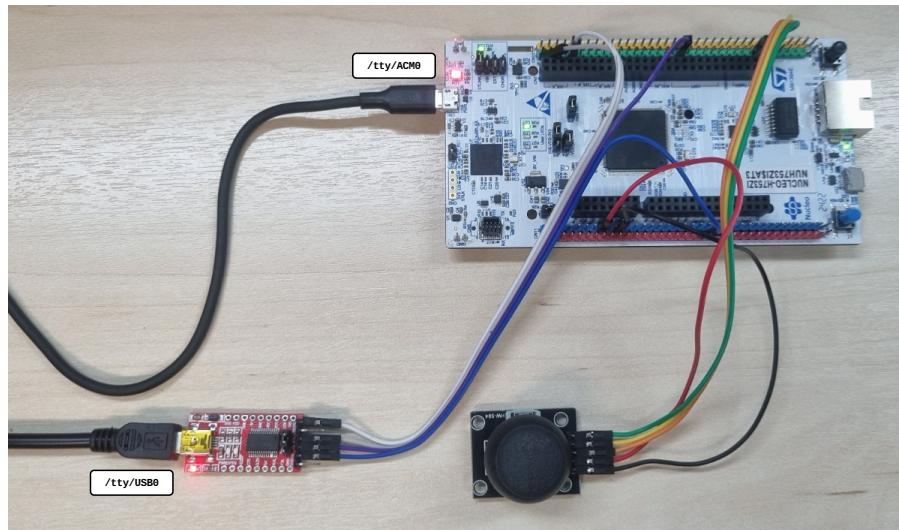


Figura 13: Montagem do circuito.

## 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.

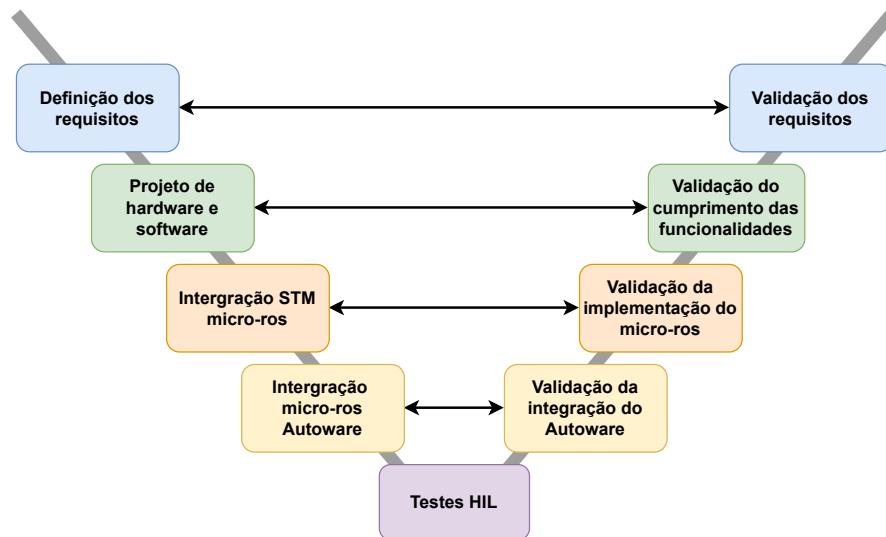


Figura 14: Modelo de execução das atividades do projeto.

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**.

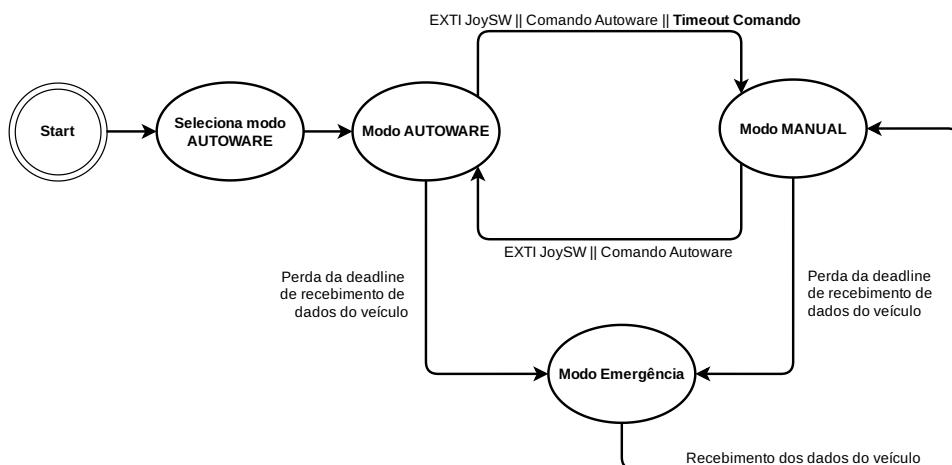


Figura 15: Máquina de estados do sistema.

#### 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%c$c$c$`
- 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`;
  - Modo de controle – `unsigned char ucControlMode`;
- Padrão da mensagem: `#S%c%c%c%cW%c%c%c%cV%c%c%c%cA%c%c%c%cJ%c%c%c%cM%c$`
- Tamanho da mensagem: 30 bytes.

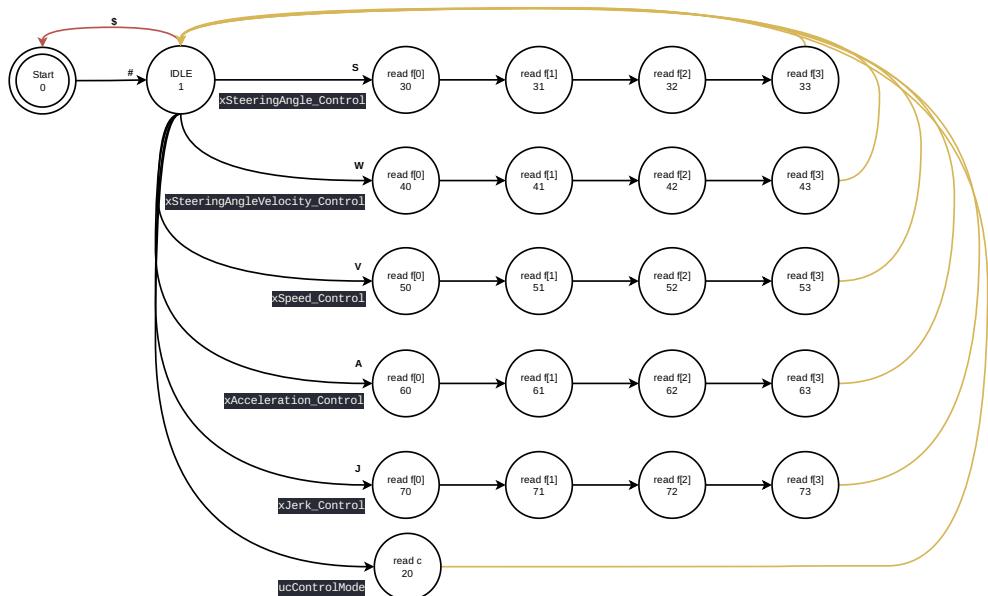


Figura 16: Máquina de estados da comunicação serial do RTOS para o CARLA.

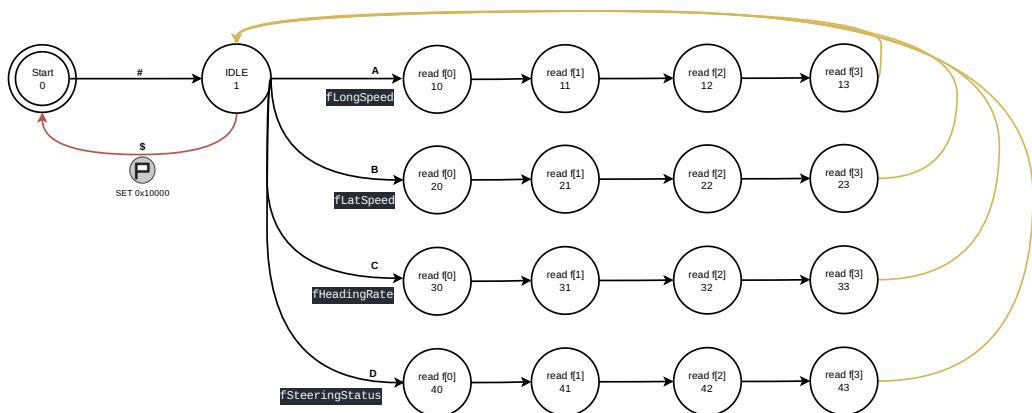


Figura 17: Máquina de estados da comunicação serial do CARLA para o RTOS.

## Tarefas

O sistema embarcado de tempo real foi estruturado por meio de duas tarefas: microAutoware, responsável pela comunicação com o Autoware e TaskControle, incubida pelo controle do veículo. Essas tarefas se comunicam por meio de variáveis globais protegidas por *MUTEX* e são sincronizadas por *ThreadFlags*. São empregadas também interrupções para comunicação serial e leitura do botão do *joystick*, como representado no diagrama da Figura 18.

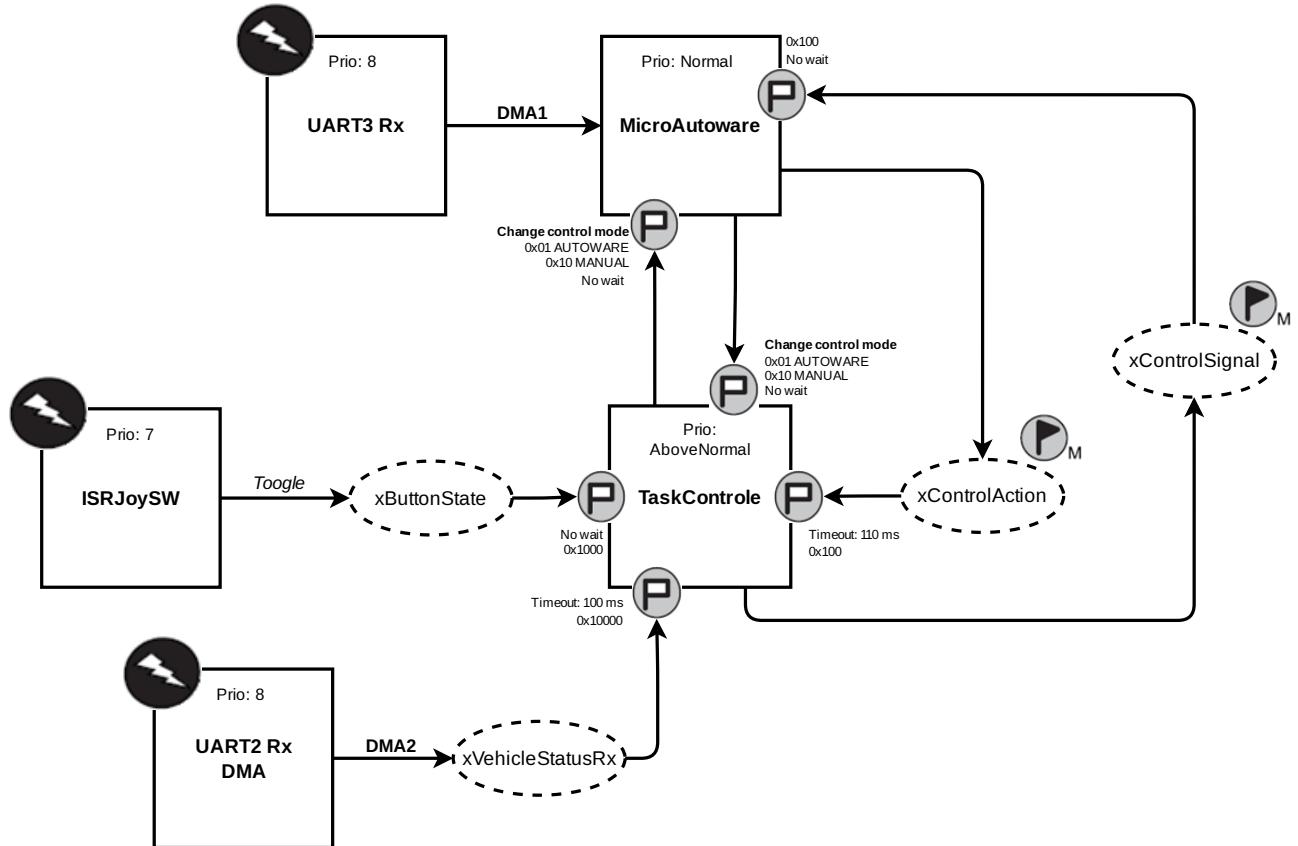


Figura 18: Diagrama do sistema embarcado.

## MicroAutoware

<b>Nome</b>	MicroAutoware
<b>Prioridade</b>	Normal
<b>Tamanho da stack</b>	3500 kB
<b>Detalhes</b>	Realiza a leitura dos <i>subscribers</i> do Autoware executando o executor por um certo tempo, envia das informações de controle e modo de operação para a TaskControle, e ao receber as informações de controle da TaskControle, às escreve nos <i>publishers</i> do Autoware.

Tabela 2: Especificaçõe da tarefa MicroAutoware.

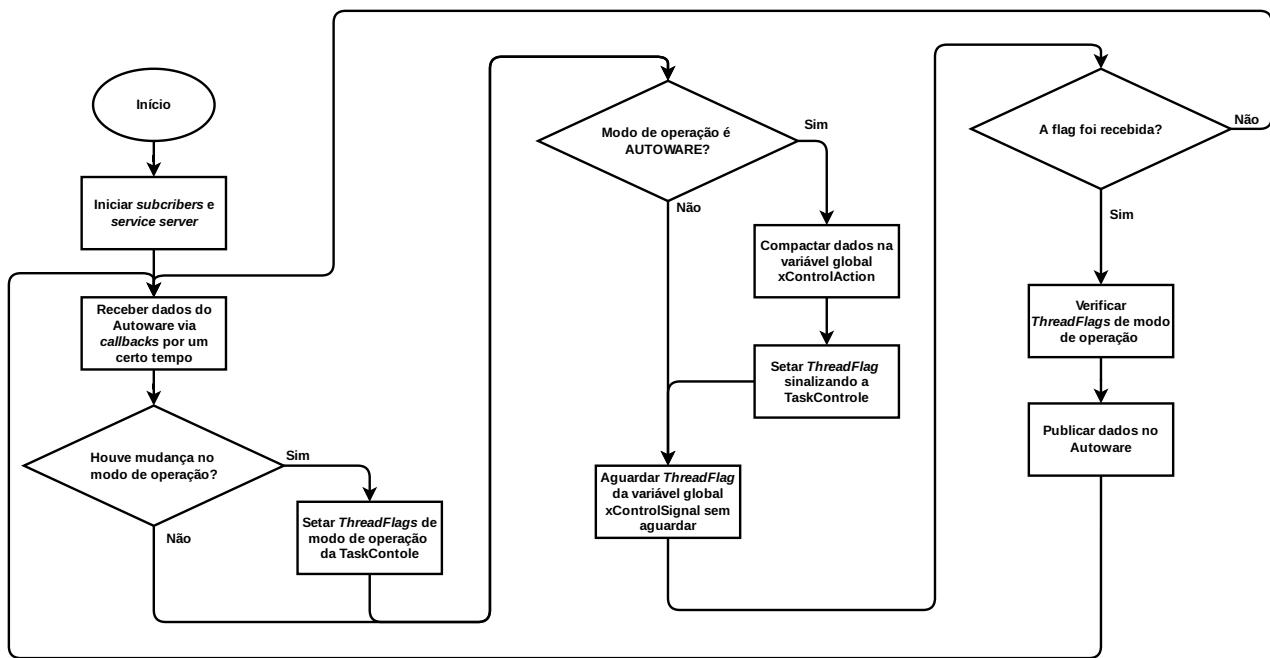


Figura 19: Fluxograma da tarefa MicroAutoware.

### TaskControle

<b>Nome</b>	TaskControle
<b>Prioridade</b>	AboveNormal
<b>Tamanho da stack</b>	500 kB
<b>Detalhes</b>	<p>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 60 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>.</p>

Tabela 3: Especificação da tarefa TaskControle.

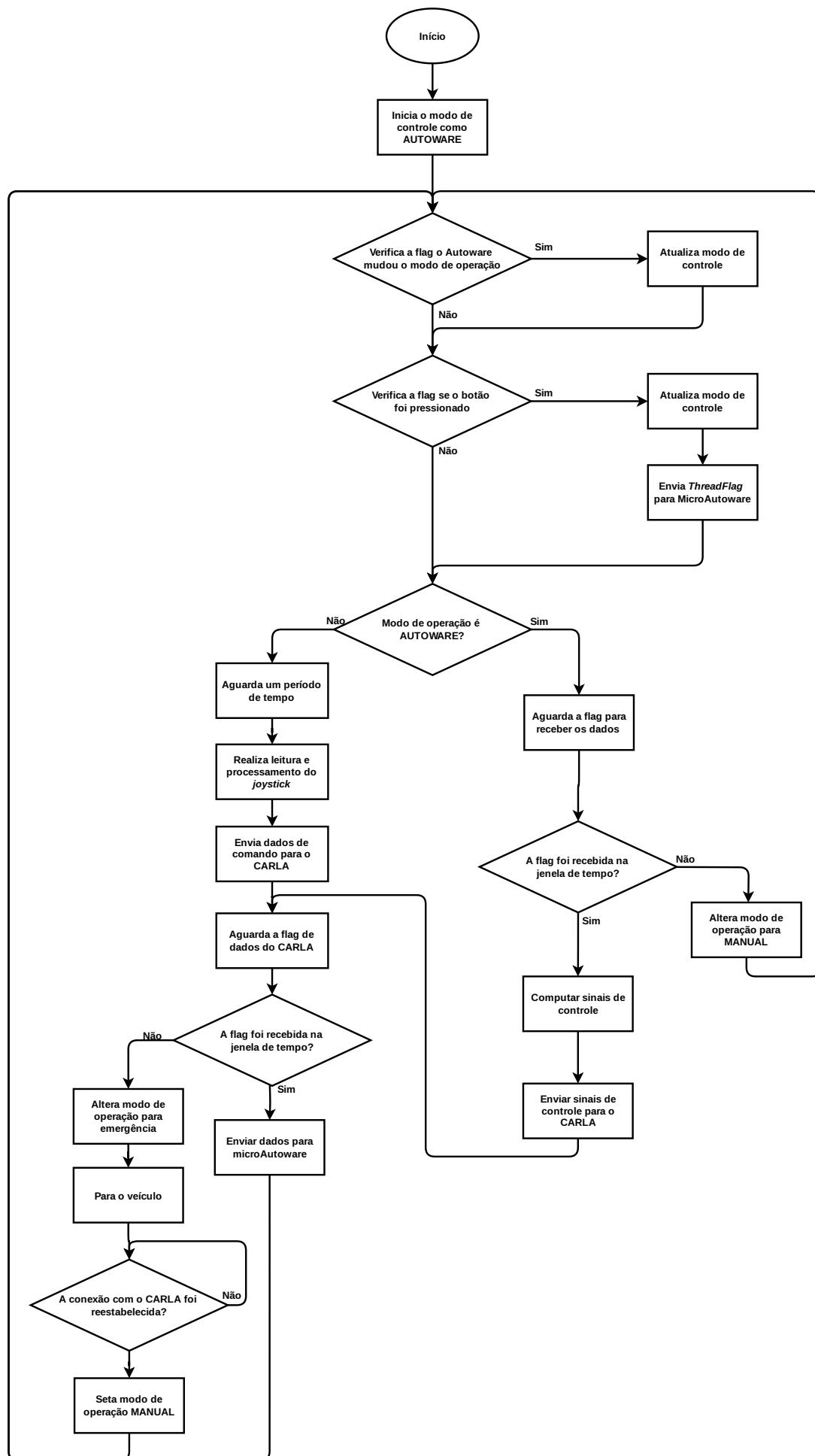


Figura 20: Fluxograma da tarefa TaskControle.

## Interrupções

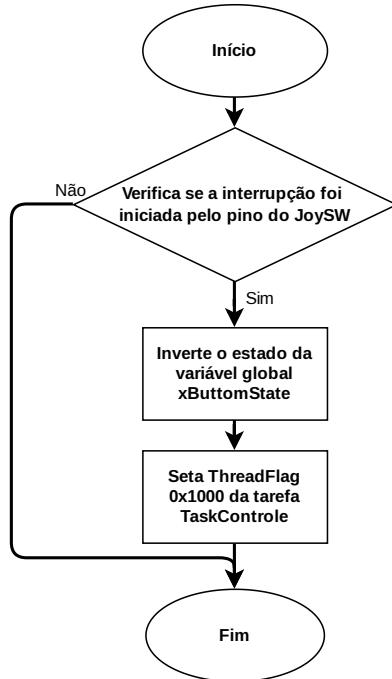


Figura 21: Fluxograma da ISR JoySW.

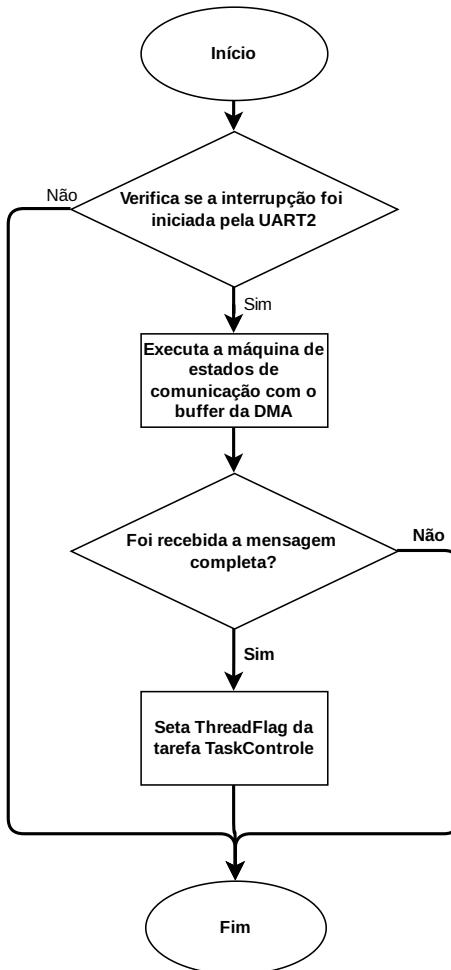


Figura 22: Fluxograma da ISR UART\_RxCpltCallback.

### 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* 110 ms
- **Descrição:** Sinaliza o recebimento de dados pela variável global **xControlAction**.

### Sinalização xControlSignal

- **Objeto:** *ThreadFlag*
- **Flag:** 0x0100
- **Modo:** *No wait*
- **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

O ecossistema ROS possuí uma padronização de código própria, para identificação das suas entidades, como tópicos, nós e serviços. Dessa forma, essa padronização será adotada de forma excepcional para os atributos do micro-ROS, de acordo com o mostrado abaixo.

- 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 de 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 (*analogue-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 23 e 24, 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.

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

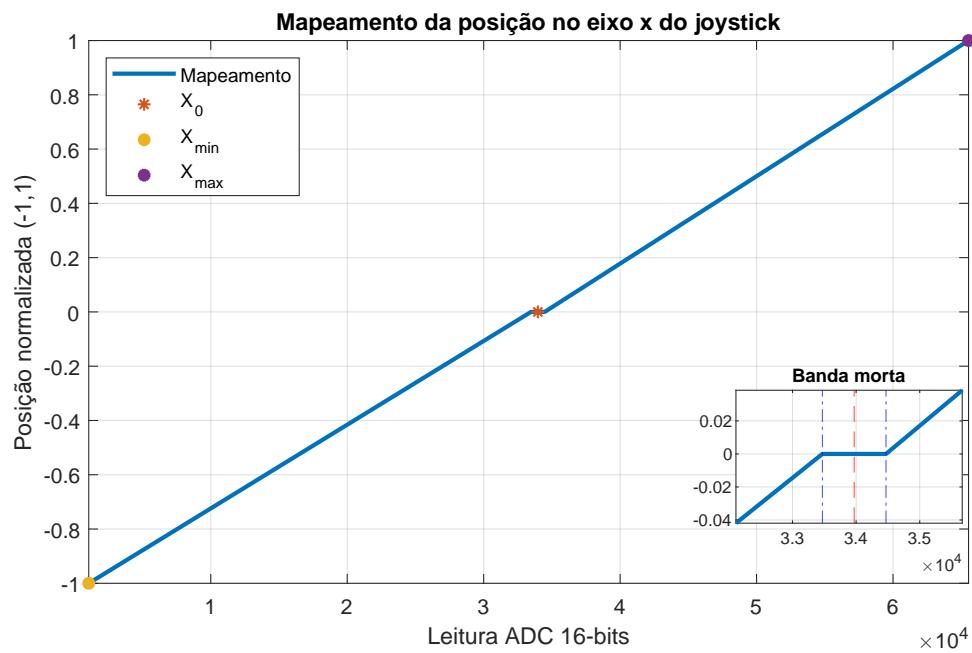


Figura 23: Mapeamento da posição no eixo  $x$  normalizada do joystick de acordo com a leitura analógica do ADC.

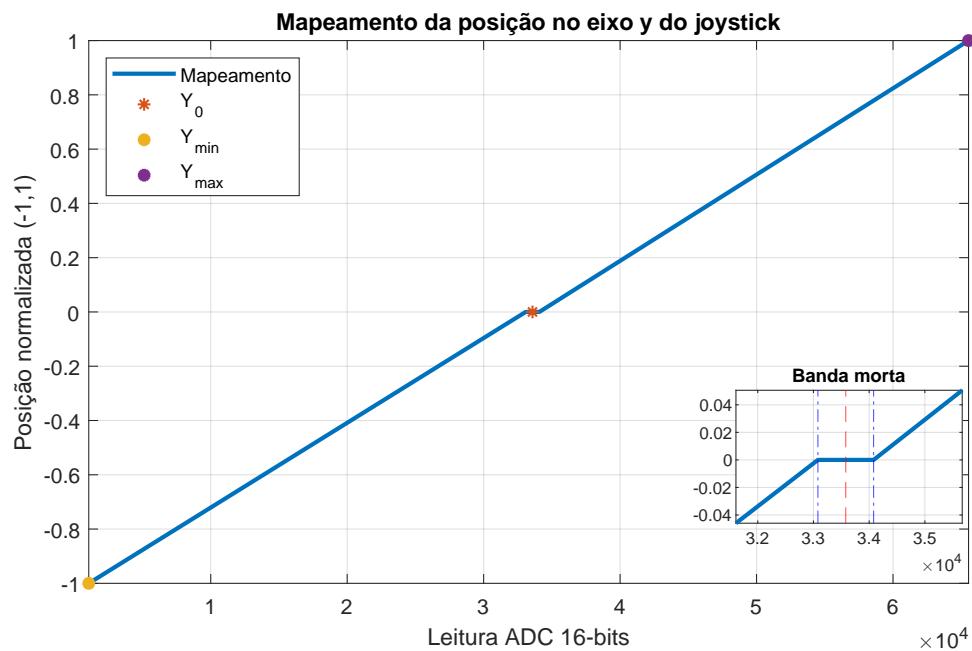


Figura 24: Mapeamento da posição no eixo  $y$  normalizada do joystick de acordo com a leitura analógica do ADC.

## 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 (micro-ROS *agent*).
  - **Baudrate:** 921600 bps
  - **Modo de leitura:** DMA circular;
  - **Modo de escrita:** DMA única;
  - **Interface física:** ST-LINK.

## 4 Manual de utilização

### 4.1 Inicialização

1. Conectar a placa ao computador;
2. Conectar o módulo FTDI ao computador;
3. Iniciar o agente micro-ROS;
4. Iniciar o nó do ROS `carla_serial_bridge_node`;
5. Iniciar o CARLA Simulator;
6. Lançar o Carla-Autoware-Bridge no ROS;
7. Lançar o Autoware em modo simulação.

### 4.2 Modo AUTOWARE

1. Passar uma velocidade limite pela interface do Autoware;
2. Definir o ponto de objetivo no Autoware;
3. Selecionar o modo de controle AUTO;
4. Caso o sistema esteja em modo MANUAL, pressionar o botão do *joystick*;
5. Aguardar o veículo chegar ao destino.

### 4.3 Modo MANUAL

1. Caso o sistema esteja em modo AUTOWARE, pressionar o botão do *joystick*;
2. Mover o analógico do *joystick* para frente para acelerar;
3. Mover o analógico do *joystick* para esquerda e direita para virar o volante;
4. Mover o analógico do *joystick* para trás para freiar.

## 5 Problemas identificados e não resolvidos

### Perda de resposta do serviço de troca de modo de controle

O Autoware faz a utilização do serviço `/control/control_mode_request` para solicitar a *vehicle interface* que faça a troca do modo de controle, realizando o engate do controle via Autoware ou remoção do mesmo. Esse tipo de comunicação ocorre da forma requisição–resposta, onde é feita a requisição de troca para o modo desejado e é respondido o resultado dessa troca (sucesso ou falha).

Porém, constantemente ocorre a falha da resposta desse serviço, ocasionando o travamento do processo de transição de modo de controle. Após testes em diferentes máquinas e pesquisas em fóruns da comunidade, o problema ocorre provavelmente por alguma incompatibilidade do micro-ROS com a CycloneDDS, utilizada pelo Autoware, onde a resposta do servidor é perdida.

### Perda de *deadline* causada pela sobrecarga da estação de trabalho

O sistema embarcado faz uso de um sistema operacional de tempo real, que entre suas premissas, trás o determinismo, onde seu comportamento é conhecido e constante. Já a estação de trabalho utilizada, faz uso de um sistema operacional de propósito geral, que não possui garantias de determinismo. Dessa forma, quando ocorre a execução de algum processo que demanda alto poder computacional, outros processos acabam sendo interferidos, o que resulta no escopo do projeto, principalmente no aumento do período de envio de dados por parte do CARLA e Autoware.

Com isso, o aumento do tempo entre o recebimento de dados, em situações críticas faz com que ocorra a perda de *deadlines* e o sistema embarcado execute alguma rotina de falha, como a mudança para direção manual ou modo de emergência. Para evitar a ocorrência de falsas falhas, a janela de tempo de recebimento de dados foi aumentada, e também foi realizada a prática de somente iniciar o baixo nível uma vez que o alto nível já está funcional.

### Reconexão ao micro-ROS *agent*

Foram implementados procedimentos de verificação de comunicação com o agente do micro-ROS, evitando que o sistema inicie até que o mesmo esteja conectado, porém no caso de desconexão ainda estão ocorrendo falhas para que o sistema embarcado se conecte novamente ao agente sem necessidade de reiniciar a placa.

## 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

## Vídeo de validação do projeto

Link: [youtu.be/7OOIyOqvU\\_o](https://youtu.be/7OOIyOqvU_o).

## Carla-Autoware-Bridge

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

## carla\_serial\_bridge

Repositório: [github.com/LMA-FEM-UNICAMP/carla\\_serial\\_bridge](https://github.com/LMA-FEM-UNICAMP/carla_serial_bridge).

## microautoware\_vehicle\_interface\_stm32

Repositório: [github.com/LMA-FEM-UNICAMP/microautoware\\_vehicle\\_interface\\_stm32](https://github.com/LMA-FEM-UNICAMP/microautoware_vehicle_interface_stm32).

## Repositório do projeto

Repositório: [github.com/toffanetto/im420x/tree/main/project](https://github.com/toffanetto/im420x/tree/main/project).