UNIVERSIDADE DE BRASÍLIA
FACULDADE GAMA
ENGENHARIA DE SOFTWARE AUTOMOTIVO

LABORATÓRIO 6 – PROCESSAMENTO DISTRIBUÍDO

Wanderson Silva dos Santos

Resumo

Neste laboratório, foi explorada a utilização da comunicação CAN em um processamento distribuído envolvendo três ECUs. A CAN é amplamente utilizada em sistemas automotivos e industriais devido à sua eficiência e robustez, permitindo que múltiplos microcontroladores troquem mensagens de maneira confiável. Nesse experimento, o processamento distribuído foi aplicado para estimar a velocidade de um veículo. O módulo do painel de instrumentos (ICM) exibiu ao usuário informações sobre a marcha atual, rotação do motor e velocidade, demonstrando a eficácia do sistema.

Introdução

O desenvolvimento de sistemas mais complexos e interconectados para o controle dos diversos sistemas e componentes de um carro tem sido impulsionado pelo avanço da tecnologia automotiva. O processamento distribuído de mensagens entre as Unidades de Controle Eletrônico (ECUs) torna-se necessário nessa situação. As ECUs são módulos especializados que regulam sistemas automotivos específicos, incluindo a transmissão, os freios e o gerenciamento do motor (ENGINEER A CAR, 2023).

A coordenação das mensagens entre essas unidades é essencial para garantir o desempenho e a segurança do veículo. No processamento distribuído, cada ECU executa suas funções de forma autônoma, mas em cooperação com outras ECUs através de uma rede de comunicação. Isso permite a descentralização do controle e a distribuição das tarefas, aumentando a eficiência e a robustez do sistema.

Para que a comunicação entre as ECUs ocorra de maneira eficiente, utiliza-se o protocolo Controller Area Network (CAN). O protocolo CAN é um padrão de comunicação robusto que foi desenvolvido pela Bosch na década de 1980, esse protocolo permite a troca de mensagens entre as ECUs através de uma rede de barramento serial. Uma das principais vantagens do protocolo CAN é sua capacidade de operar em ambientes ruidosos, comuns em veículos automotivos, garantindo a integridade das mensagens transmitidas. O protocolo utiliza um esquema de arbitragem para controlar o acesso ao barramento, permitindo que múltiplas ECUs possam enviar e receber mensagens sem conflitos (GALLASSI, 2022).

No contexto do processamento distribuído, a matriz de comunicação e o dicionário de dados são conceitos fundamentais. A matriz de comunicação especifica quais ECUs comunicam entre si e que tipos de mensagens são trocadas, facilitando o gerenciamento rede. Já o dicionário de dados contém uma descrição de todas as mensagens e sinais trocados na rede CAN, incluindo seus identificadores, tamanhos e unidades de medida. Esses documentos são essenciais para garantir a não interrupção e consistência da comunicação entre as ECUs.

O uso do protocolo CAN em sistemas automotivos possibilita a construção de redes de comunicação escaláveis, permitindo que novas funções sejam integradas ao longo do tempo. Assim, o processamento distribuído entre ECUs não apenas melhora o desempenho do veículo, mas também facilita a manutenção e a implementação de melhorias tecnológicas.

Tendo em vista esses conhecimentos, o objetivo deste laboratório foi utilizar o protocolo CAN para aplicar os conceitos de processamento distribuído na estimativa da velocidade de um veículo, segundo um dos métodos de estimação, o qual é dado pela fórmula a seguir:

$$v_x = \frac{R_\omega \omega_e}{n_g n_d} \tag{1}$$

onde R_ω é o raio efetivo do pneu, ω_e a rotação do motor, n_g a taxa de transmissão da transmissão (variável de acordo com a marcha) e n_d a taxa de transmissão do diferencial.

Materiais e métodos

Para o experimento desse laboratório, foi utilizado um Kit de desenvolvimento ANEB v1.0 Figura 1 que consiste em uma placa onde seus componentes estão listados a seguir:

- 04 entradas digitais (*pushbuttons*);
- 02 entradas seletoras de barramento;
- 04 entradas analógicas (potenciômetros);
- 01 saída analógica em loop (ECU1-ECU2);
- 01 saída analógica para LDR (ECU1-ECU1);
- 05 controladores CAN e *transceivers*;
- 02 redes CAN (CAN1 e CAN2) independentes;
- 01 sub-rede automotiva LIN;
- 01 hub FTDI para conexão com as ECUs.
- 09 LEDs
- 01 Buzzer
- 01 Sensor de Luminosidade LDR
- 04 Arduinos Nano (ECUs)
- 01 Cabo USB para conexão serial da placa com o PC

Para esse laboratório, foram utilizados os pushbuttons DIN#1 e DIN#2, a ECU1, ECU2, ECU3 e o potenciômetro AIN#1, a Figura 1 mostra esses componentes. Além disso cada ECU tinha um módulo CAN MCP2515 conectados ao BUS1, que por meio deles eram feitas as comunicações.

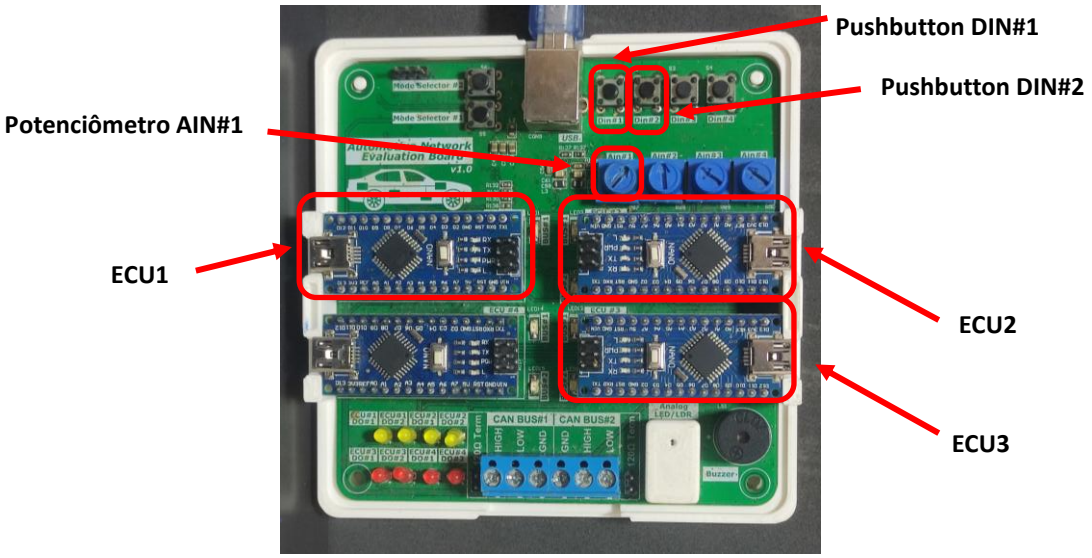

Figura 1 – Componentes utilizados no laboratório.

O método de estimação da velocidade do veículo foi dada pela Eq.1 e o diagrama da Figura 2 mostra como ocorre o fluxo de dados através das ECUs. ω_e é a rotação do motor, g é a marcha atual, TCM é o módulo de controle da transmissão, ECM o módulo de controle do motor, ICM o módulo do painel de instrumentos, M1 é a mensagem da marcha atual, M2 da rotação do motor e M3 da velocidade do veículo.

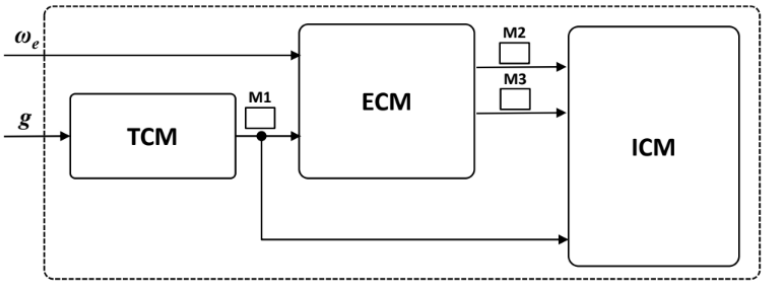

Figura 2 – Fluxo de dados.

A sequência das ações se deu em três partes. Na primeira, a ECU1 TCM lia a marcha atual, encapsulava em uma mensagem (M1) e transmitia na rede a cada 200ms. A mudança de marcha foi implementada com a utilização dos pushbuttons, onde o DIN#1 diminuía a marcha e o DIN#2 aumentava, de forma que a marcha mínima era 0 e a máxima 5. O código dessa ECU está no Anexo como Código Laboratório 6 – ECU1.

Na segunda parte, a ECU2 ECM lia a marcha atual (M1) e desencapsulava o sinal. Após isso, ela lia a rotação do motor, a qual era fornecida pelo potenciômetro AIN#1, encapsulava em uma mensagem (M2) e transmitia na rede a cada 100ms com base nas restrições da Tabela 1. Por conseguinte, através da marcha atual e rotação do motor, a

ECU2 calculava a velocidade do veículo utilizando a Eq.1, encapsulava em uma mensagem (M3) e transmitia na rede a cada 250ms. O código dessa ECU está no Anexo como Código Laboratório – ECU2.

c	Rotação máxima (rpm)	Taxa de transmissão (ng)
0 (Lenta)	8000	0
1	4000	3,83
2	4800	2,36
3	5600	1,69
4	6400	1,31
5	7200	1

Tabela 1 – Restrições de rotação.

Na terceira parte, a ECU3 ICM lia as mensagens M1, M2 e M3, desencapsulava-as e imprimia os valores de rotação, velocidade e marcha atual a cada 500ms no seguinte formato: Rotação: XXXX rpm, Velocidade: YYY km/h, Marcha atual: Z. O código dessa ECU está no Anexo como Código Laboratório 6 – ECU3.

Para melhor compreensão desse processamento distribuído, na Tabela 2 e Tabela 3 estão a Matriz de Comunicação e o Dicionário de Dados, respectivamente.

ECU	Mensagem	Sinal	TCM ECU1	ECM ECU2	ICM ECU3
TCM	M1	Marcha atual	S	R	R
ECM	M2 M3	Rotação Velocidade		S S	R R
ICM					

Tabela 2 – Matriz de Comunicação.

Mensagem	ID	Byte	Bit	Comprimento (Bits)	Descrição	Resolução	Limites	Offset
M1	0x101	1	1	8	Marcha atual	1 marcha/bit	0 – 5	0
M2	0x102	1	1	16	Rotação do motor	0,125 rpm/bit	0 – 8000	0
M3	0x103	1	1	8	Velocidade do veículo	1 (km/h)/bit	0 – 250	0

Tabela 3 – Dicionário de Dados.

Para garantir o correto funcionamento das sequências de ações, o laboratório foi dividido em experimentos que consistiu em quatro verificações:

1. Na ECU1: verificar se estava mandando a mensagem de marcha atual (M1) no barramento;
2. Na ECU2: verificar se estava recebendo a mensagem da marcha atual (M1);
3. Ainda na ECU2: verificar se as mensagens de rotação (M2) e velocidade (M3) estavam sendo enviadas na rede de acordo com a marcha recebida;

4. Na ECU3: verificar se estava recebendo e desencapsulando todas as mensagens.

Resultados e Discussão

Nesta seção serão analisadas e discutidas as quatro verificações citadas na seção anterior. Para a primeira verificação, a Figura 3 mostra a confirmação de que a mensagem de marcha (M1) foi transmitida pela ECU1 e, logo em seguida, mostra qual o valor de marcha relativa a cada transmissão.

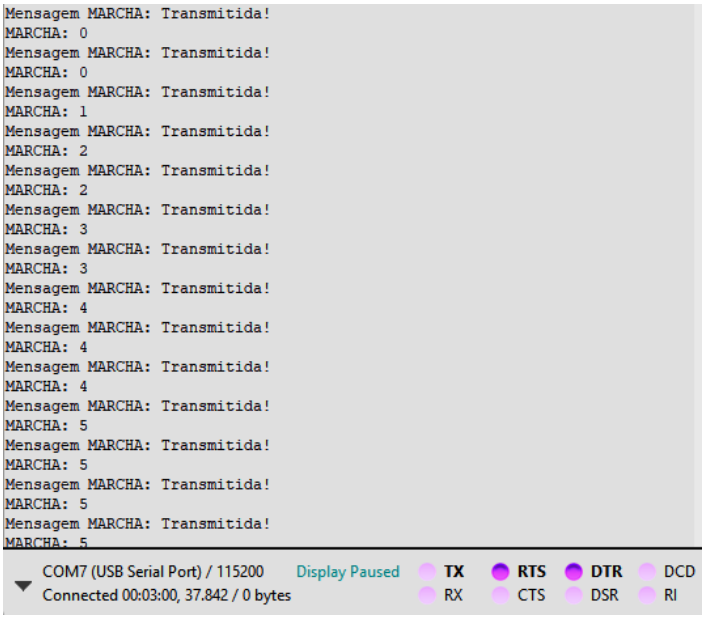

Figura 3 – Mensagem de marcha (M1) transmitida.

Para a segunda verificação, a Figura 4 mostra que a mensagem de marcha (M1) estava sendo recebida pela ECU2. À esquerda são vistas as marchas sendo transmitidas e, à direita, as marchas que foram recebidas, desencapsuladas e imprimidas na serial da ECU2.

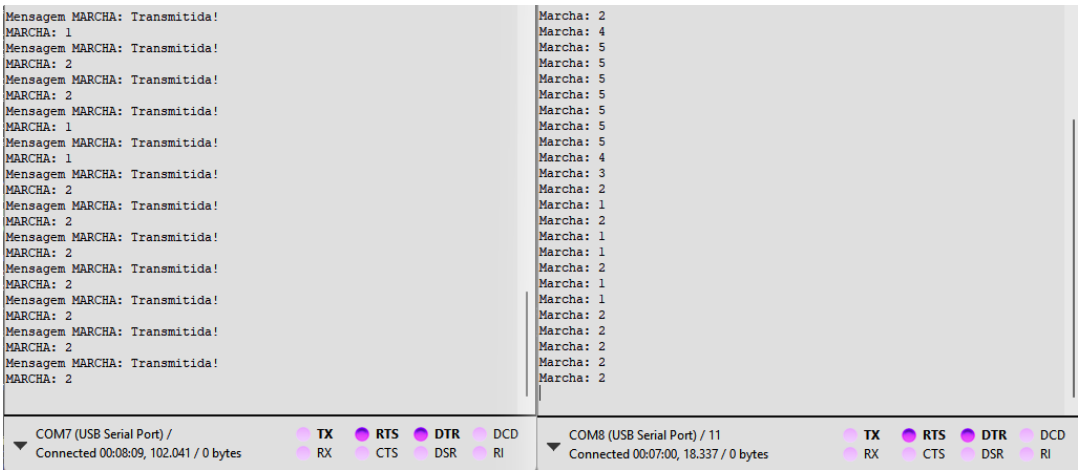

Figura 4 – Mensagem M1 transmitida pela ECU1 (TCM) e recebida pela ECU2 (ECM).

A terceira verificação incluiu seis sub-verificações. A primeira sub-verificação teve como objetivo determinar se, em marcha lenta (marcha 0), a velocidade do veículo permanecia em 0 km/h mesmo com a rotação do motor variando entre 0 e 8000 rpm. Este fato pode ser comprovado na Figura 5, onde, à esquerda, observa-se a marcha 0 que a ECU1 está enviando, enquanto à direita, nota-se a rotação variando (em ciano) e a velocidade permanecendo em 0 km/h (em laranja).

De mesma forma foi verificado para as outras marchas: a Figura 6 mostra que, para a marcha 1, a rotação máxima era de 4000 rpm (conforme as restrições vista na seção anterior) e a velocidade máxima de 36 km/h. A Figura 7 mostra que, para a marcha 2, a rotação máxima era de 4800 rpm e velocidade máxima de 70 km/h. A Figura 8 mostra que, para a marcha 3, a rotação máxima era de 5600 rpm e velocidade máxima de 114 km/h. A Figura 9 mostra que, para a marcha 4, a rotação máxima era de 6400 rpm e velocidade máxima de 169 km/h. Por último, a Figura 10 mostra que, para a marcha 5, a rotação máxima era de 7200 rpm e velocidade máxima de 250 km/h.

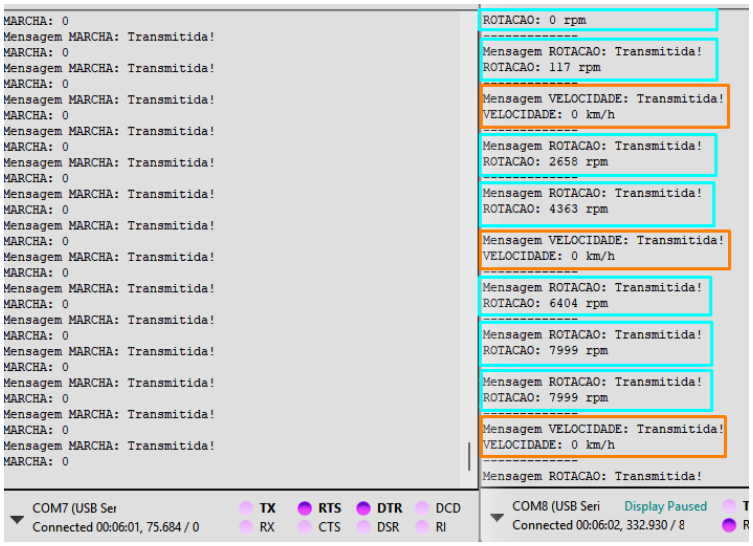

Figura 5 – Rotação e velocidade máximas para marcha 0.

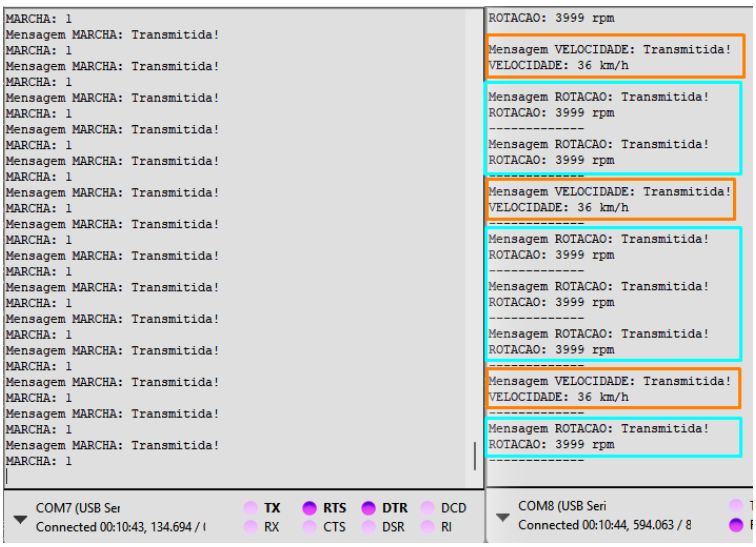

Figura 6 – Rotação e velocidade máximas para marcha 1.

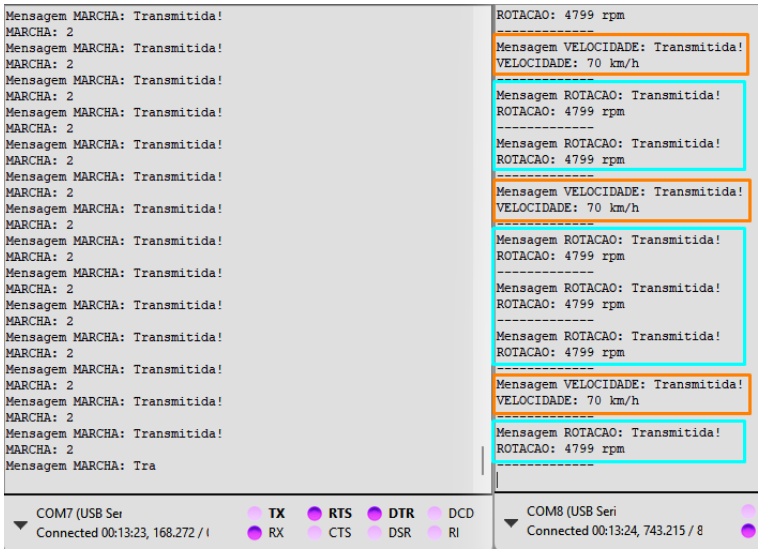

Figura 7 – Rotação e velocidade máximas para marcha 2.

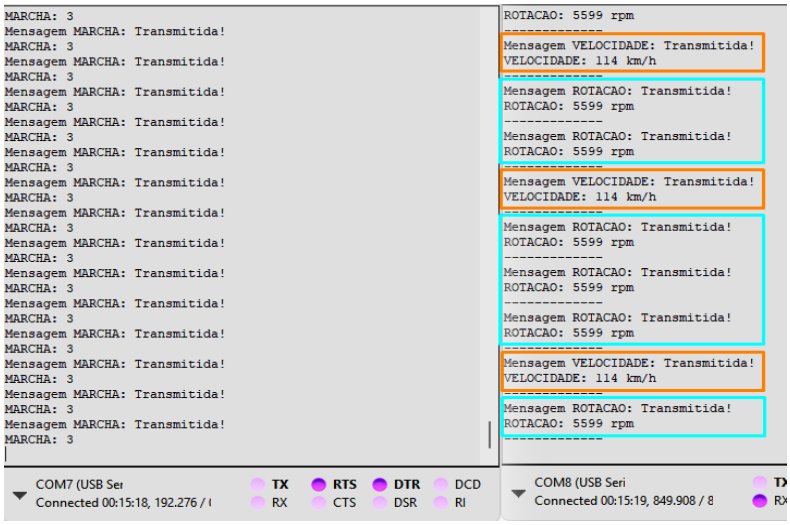

Figura 8 – Rotação e velocidade máximas para marcha 3.

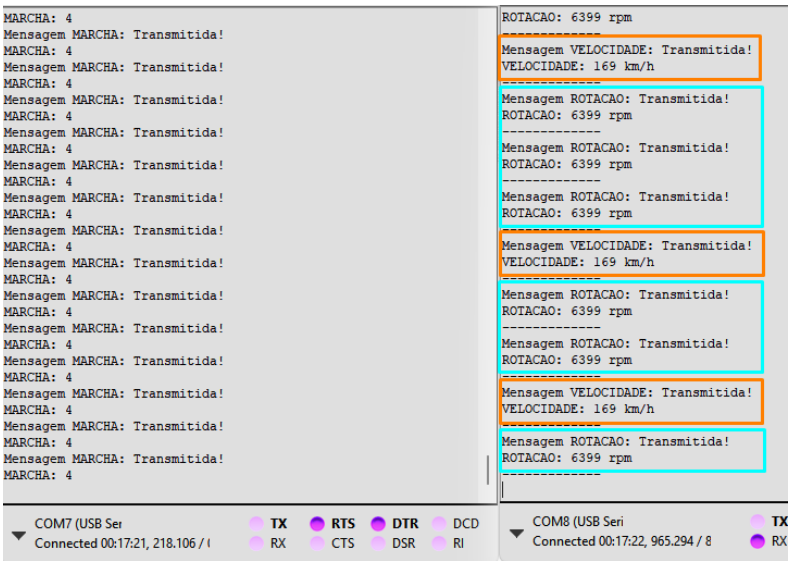

Figura 9 – Rotação e velocidade máximas para marcha 4.

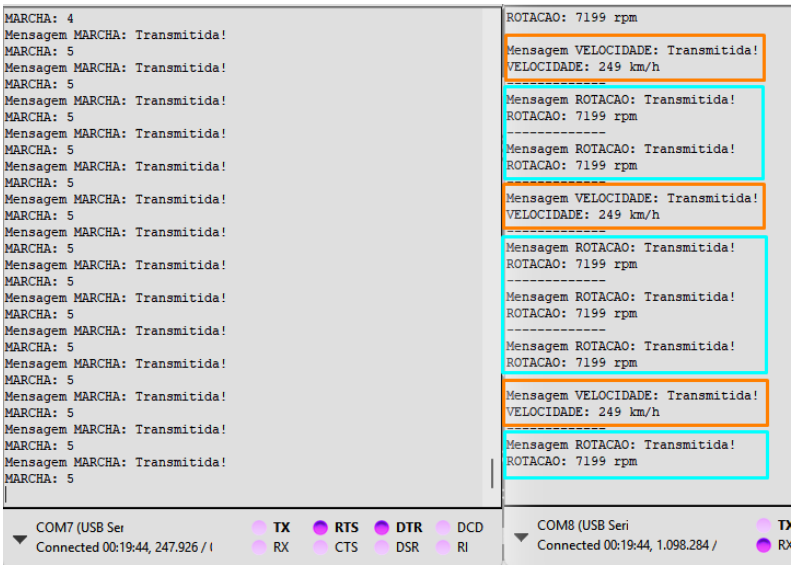

Figura 10 – Rotação e velocidade máximas para marcha 5.

Para a quarta e última verificação, a Figura 11 mostra que as mensagens M1, M2 e M3 estavam sendo recebidas, desencapsuladas e imprimidas na serial da ECU3 (ICM). À esquerda, é exibida a marcha enviada pela TCM; ao centro, as rotações e velocidades enviadas pela ECM; e à direita, as mensagens de marcha, rotação e velocidade recebidas pela ICM.

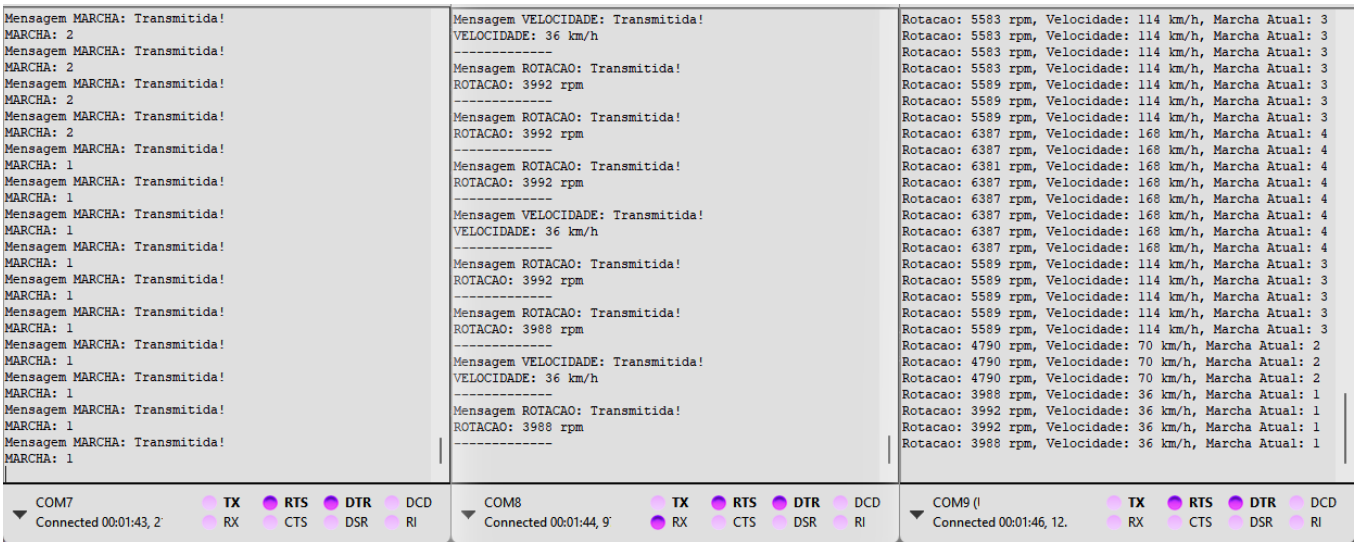

Figura 11 – Mensagens M1, M2 e M3 lidas e imprimidas pela ICM (à direita).

Assim, pode-se verificar que a sequência de ações para a estimação da velocidade do veículo concluiu com êxito a partir desse processamento distribuído utilizando o protocolo CAN para a comunicação entre as ECUs.

Conclusões

O experimento de estimação de velocidade do veículo, utilizando a comunicação CAN entre três ECUs em um sistema distribuído, mostrou como o protocolo é eficaz para

troca rápida e precisa de dados. A implementação do protocolo CAN revelou como um processamento distribuído pode ser complexo; contudo, é eficiente e vantajoso, o que explica sua vasta utilização atualmente.

Por fim, é essencial monitorar o tempo das mensagens no barramento CAN, pois qualquer atraso ou problema de sincronização pode comprometer a integridade dos dados e a segurança do sistema.

Referências Bibliográficas

ENGINEER A CAR. **What is a Car ECU?** 2023. Disponível em: https://www.engineeracar.com/what-is-car-ecu/. Acesso em: 3 jun. 2024.

GALLASSI, Juliana. **Rede CAN: Tudo sobre o que você precisa saber!** 2022. Disponível em: https://tl.trimble.com/blog/rede-can/. Acesso em: 3 jun. 2024.

Anexo

Código Laboratório 6 – ECU1:

```
// Laboratorio 6 - Processamento distribuido
// ECU1 (TCM)

#include "mcp_can.h"
#include "mcp_can_dfs.h"
#include <SPI.h>
#include "Board.h"

// Set CS to pin 10
MCP_CAN CAN0(ECU1_CAN1_CS);

// ID das mensagens
#define CAN_ID_MARCHA 0x101

// Iniciando campo de dados para rotacao e velocidade
byte data_marcha[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}; // Array armazenar a marcha

// Iniciando contador de marcha
byte marcha = 0;

// Variavel controle de tempo
int inter_envio = 0;
unsigned long ultimo_envio = 0;

// Variaveis para controle dos botoes
int estado1;
int estado2;
int estadoant1;
int estadoant2;


void setup()
{
  Serial.begin(115200);

  // Inicializando o controlador CAN
  // CAN baudrate = 500kbps, MCP_8MHZ
  while((CAN0.begin(MCP_ANY, CAN_500KBPS, MCP_8MHZ) != CAN_OK))
  {
```

```
    Serial.println("Erro para inicializar o controlador CAN!");
    delay (200); //Aguarda 200ms
  }

  Serial.println("Can send inicializado com sucesso!");

  // Configuracao para transmissao e recepcao de mensagens
  CAN0.setMode(MCP_NORMAL);

  // Inicializar os potenciometros
  pinMode(ECU1_DIN1, INPUT); // Definindo Botao DIN#1 como entrada
  pinMode(ECU1_DIN2, INPUT); // Definindo Botao DIN#2 como entrada

  estadoant1 = HIGH; // Iniciando o estado do botao DIN#1 como desligado
  estadoant2 = HIGH; // Iniciando o estado do botao DIN#2 como desligado


}


void loop()
{
    estado1 = digitalRead(ECU1_DIN1); // Lendo estado do botao DIN#1
    estado2 = digitalRead(ECU1_DIN2); // Lendo estado do botao DIN#2

    // Botao DIN#1 descrementa a marcha
    if(estado1 == LOW && estadoant1 == HIGH){
        if(marcha > 0){
            marcha--;
        }
    }

    estadoant1 = estado1; // Atualizando ultimo estado

    // Botao DIN#2 incrementa a marcha
    if(estado2 == LOW && estadoant2 == HIGH){
        if(marcha < 5){
            marcha++;
        }
    }

    estadoant2 = estado2; // Atualizando ultimo estado

    // Atualizando campos de dados
    data_marcha[0] = marcha;

  // ------------------- TRANSIMITINDO MENSAGEM M1: MARCHA ------------------- //


  inter_envio = millis() - ultimo_envio; // Calculando intervalo de envio da mensagem

  /* Se o intervalo de envio entre o tempo atual
    e a última mensagem for >= que 200ms, enviar mensagem na CAN */

  if(inter_envio >= 200){

        byte envio_marcha = CAN0.sendMsgBuf(CAN_ID_MARCHA, 0, 8,data_marcha);
        if(envio_marcha == CAN_OK){
            Serial.println("Mensagem MARCHA: Transmitida!");
            /* Para debud
            Printando rotacao em decimal
            Serial.print("MARCHA: ");
            Serial.println(data_marcha[0]);*/

            ultimo_envio = millis(); // Atualizando tempo de envio da mensagem
        }
        else{
            Serial.println("Erro na transmissao da mensagem!");
        }
```

```
  }
}
```

Código Laboratório 6 – ECU2:

```
// Laboratorio 6 - Processamento distribuido
// ECU2 (ECM)

#include "mcp_can.h"
#include "mcp_can_dfs.h"
#include <SPI.h>
#include "Board.h"

// Set CS to pin 10
MCP_CAN CAN0(ECU2_CAN1_CS);

// Pino 2 interrupt
#define CAN0_INT 2

// ID das mensagens
#define CAN_ID_MARCHA 0x101
#define CAN_ID_ROTACAO 0x102
#define CAN_ID_VELOCIDADE 0x103

long unsigned int rxId;
unsigned char len = 0;
unsigned char rxBuf[8];
char msgString[128];                        // Array to store serial string

// Iniciando variaveis
unsigned char marcha = 0;
unsigned char velocidade = 0;
unsigned int rotacao = 0;
float nd = 3.55;    // Taxa de transmissao diferencial
float ng = 0.0;     // Taxa de trasminssao da transmissao
float pi = 3.14159265;
float Rw = 0.326;   // Raio da Roda
int pot1; // Guardar posicao do potenciometro

// Variaveis contadoras
int interM2 = 0;
unsigned long ult_M2 = 0;
int interM3 = 0;
unsigned long ult_M3 = 0;

byte data_rotacao[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};    // Array para rotacao
byte data_velocidade[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}; // Array para velocidade

void setup()
{
    Serial.begin(115200);

  // Initialize MCP2515 running at 8MHz with a baudrate of 500kb/s
  // and the masks and filters disabled.
   while((CAN0.begin(MCP_ANY, CAN_500KBPS, MCP_8MHZ) != CAN_OK))
  {
    Serial.println("Erro para inicializar o controlador CAN!");
    delay (200); //Aguarda 200ms
  }

  Serial.println("Can iniciado com sucesso!");

  CAN0.setMode(MCP_NORMAL);    // Configuracao para modo normal

  pinMode(CAN0_INT, INPUT);    // Configurando PIN para Interrupt como Input

  Serial.println("MCP2515 Library Receive Example...");
```

```
  pinMode(ECU2_AIN1,INPUT);     // Configurando pino do potenciomêtro como entradada

}

void loop()
{
    if(!digitalRead(CAN0_INT)){         // If CAN0_INT pin is low, read receive buffer

        // Lendo mensagens na CAN
        CAN0.readMsgBuf(&rxId, &len, rxBuf);   // Read data: len = data length, buf = data byte(s)
    }

    // Armazenando valor da marcha atraves do rxBuf
    unsigned char marcha = rxBuf[0];

    // Definindo ROTACAO com base na marcha e posicao do potenciometro
    if(marcha == 0) {
        rotacao = (int)((7.82013)*analogRead(ECU2_AIN1));
        ng = 0.0;
    }else if(marcha == 1) {
        rotacao = (int)((3.91006)*analogRead(ECU2_AIN1));
        ng = 3.83;
    }else if(marcha == 2) {
        rotacao = (int)((4.69208)*analogRead(ECU2_AIN1));
        ng = 2.36;
    }else if(marcha == 3){
        rotacao = (int)((5.47409)*analogRead(ECU2_AIN1));
        ng = 1.69;
    }else if(marcha == 4) {
        rotacao = (int)((6.256109)*analogRead(ECU2_AIN1));
        ng = 1.31;
    }else if(marcha == 5) {
        rotacao = (int)((7.03812)*analogRead(ECU2_AIN1));
        ng = 1.00;
    }

    /*////// Para debug
    Serial.print("Marcha: ");
    Serial.println(marcha);
    */

    // Armazenando a rotacao no array data rotacao
    data_rotacao[0] = (rotacao >> 8);
    data_rotacao[1] = (rotacao & 0x00FF);

    // Calculo VELOCIDADE
    if (marcha == 0){
        velocidade = 0;
    }else if(marcha > 0){
        // Velocidade me km/h - Fator de conversao 0.001*2*pi*60
        velocidade  = (Rw*rotacao*0.001*2*pi*60)/(ng*nd);
    }

    // Armazenando valor da velocidade
    data_velocidade[0] = velocidade;

    // ------------------- TRANSIMITINDO MENSAGEM M2: ROTACAO ------------------- //

    /* Se o intervalo de envio entre o tempo atual
    e a última mensagem for >= que 100ms, enviar mensagem na CAN */
    interM2 = millis()-ult_M2;

    if(interM2 >= 100){
        byte envio_rotacao = CAN0.sendMsgBuf(CAN_ID_ROTACAO, 0, 8,data_rotacao);

        if(envio_rotacao == CAN_OK){
            /* Para debug
            Serial.println("Mensagem ROTACAO: Transmitida!");
```

```
            Serial.print("ROTACAO: ");
            Serial.print(data_rotacao[0] << 8 | data_rotacao[1]);
            Serial.println(" rpm");
            Serial.println("-------------");*/

            ult_M2 = millis(); // Atualizando tempo de envio da mensagem
        }
        else{
            Serial.println("Erro na transmissao da ROTACAO!");
        }
    }

    // ------------------- TRANSIMITINDO MENSAGEM M3: VELOCIDADE ------------------- //

    /* Se o intervalo de envio entre o tempo atual
    e a última mensagem for >= que 250ms, enviar mensagem na CAN */
    interM3 = millis()-ult_M3;

    if(interM3 >= 250){
        byte envio_velocidade = CAN0.sendMsgBuf(CAN_ID_VELOCIDADE, 0, 8,data_velocidade);

        if(envio_velocidade == CAN_OK){
            /* Para debug
            Serial.println("Mensagem VELOCIDADE: Transmitida!");
            Serial.print("VELOCIDADE: ");
            Serial.print(data_velocidade[0]);
            Serial.println(" km/h");
            Serial.println("-------------");*/

            ult_M3 = millis(); // Atualizando tempo de envio da mensagem
        }
        else{
            Serial.println("Erro na transmissao da VELOCIDADE!");
        }
    }
}
```

Código Laboratório 6 – ECU3:

```
// Laboratorio 6 - Processamento distribuido
// ECU3 (ICM)

#include "mcp_can.h"
#include "mcp_can_dfs.h"
#include <SPI.h>
#include "Board.h"

// Set CS to pin 10
MCP_CAN CAN0(ECU2_CAN1_CS);

// Pino 2 interrupt
#define CAN0_INT 2

// ID das mensagens
#define CAN_ID_MARCHA 0x101
#define CAN_ID_ROTACAO 0x102
#define CAN_ID_VELOCIDADE 0x103

long unsigned int rxId;
unsigned char len = 0;
unsigned char rxBuf[8];
char msgString[128];                        // Array to store serial string

// Iniciando variaveis
unsigned char marcha = 0;
unsigned char velocidade = 0;
unsigned int rotacao = 0;
```

```
// Variaveis contadoras
int interMsg = 0;
unsigned long ult_Msg = 0;

void setup()
{
    Serial.begin(115200);

  // Initialize MCP2515 running at 8MHz with a baudrate of 500kb/s
  // and the masks and filters disabled.
   while((CAN0.begin(MCP_ANY, CAN_500KBPS, MCP_8MHZ) != CAN_OK))
  {
    Serial.println("Erro para inicializar o controlador CAN!");
    delay (200); //Aguarda 200ms
  }

  Serial.println("Can iniciado com sucesso!");

  CAN0.setMode(MCP_NORMAL);    // Configuracao para modo normal

  pinMode(CAN0_INT, INPUT);    // Configurando PIN para Interrupt como Input

  Serial.println("MCP2515 Library Receive Example...");

}

void loop(){

    if(!digitalRead(CAN0_INT)){       // If CAN0_INT pin is low, read receive buffer

        // Lendo mensagens na CAN
        CAN0.readMsgBuf(&rxId, &len, rxBuf); // Read data: len = data length, buf = data byte(s)
    }

    // ------------------- DESENCAPSULANDO M1, M2 E M3 ------------------- //

    if(rxId == CAN_ID_ROTACAO){
        rotacao = (rxBuf[0] << 8) | rxBuf[1];
    }else if(rxId == CAN_ID_VELOCIDADE){
        velocidade = rxBuf[0];
    }else if(rxId == CAN_ID_MARCHA){
        marcha = rxBuf[0];
    }
    // -------------- MOSTRANDO MENSAGENS M1, M2 E M3 NA SERIAL A CADA 500MS ----------- //

    interMsg = millis() - ult_Msg;

    if(interMsg >= 500){
        Serial.print("Rotacao: ");
        Serial.print(rotacao);
        Serial.print(" rpm, ");

        Serial.print("Velocidade: ");
        Serial.print(velocidade);
        Serial.print(" km/h, ");

        Serial.print("Marcha Atual: ");
        Serial.print(marcha);
        Serial.println("");

        // Serial.println(interMsg); // Printando intervalo de tempo que a mensagem foi enviada

        ult_Msg = millis();

    }
}
```