ARTHUR SANTIAGO NETO

IMPLEMENTAÇÃO DE TELEMETRIA NO VEÍCULO DA EQUIPE CERRADO BAJA SAE

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

2021

ARTHUR SANTIAGO NETO

IMPLEMENTAÇÃO DE TELEMETRIA NO VEÍCULO DA EQUIPE CERRADO BAJA SAE

Trabalho de conclusão de curso apresentado ao Curso de Graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção do título de BACHAREL em **ENGENHARIA MECATRÔNICA**.

Área de concentração: Eletrônica e Redes

Orientador: Prof. Dr. Pedro Augusto Queiroz de Assis

UBERLÂNDIA - MG

2021

IMPLEMENTAÇÃO DE TELEMETRIA NO VEÍCULO DA EQUIPE CERRADO BAJA SAE

Projeto de conclusão de curso **APROVADO** pelo Colegiado do Curso de Graduação em Engenharia Mecatrônica da Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia.

BANCA EXAMINADORA

_____

Prof. Dr. Pedro Augusto Queiroz de Assis
Universidade Federal de Uberlândia

_____

Prof. Dr. Daniel Dall'Onder dos Santos
Universidade Federal de Uberlândia

_____

Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares
Universidade Federal de Uberlândia

UBERLÂNDIA - MG

2021

Santiago, Arthur N. **Implementação De Telemetria No Veículo Da Equipe Cerrado Baja SAE.** 2021. 68. Projeto de Conclusão de Curso – Engenharia Mecatrônica, Universidade Federal de Uberlândia, Uberlândia, Brasil.

RESUMO

Este trabalho envolve a implementação de um sistema de sensoriamento e telemetria no veículo construído pela equipe Cerrado BAJA SAE da Universidade Federal de Uberlândia. Inicialmente são analisados dois módulos de comunicação disponíveis. Mais especificamente, comparam-se os módulos Xbee S2 e LoRa Ra-02 para realização da comunicação entre o veículo e o *box.* Os testes realizados envolveram a aferição da distância máxima de comunicação, confiabilidade e integridade dos dados, e o custo. Como resultado desse estudo comparativo, adotou-se o LoRa RA-02 para realização da telemetria. Consideram-se sensores para aferição de temperatura do motor e CVT (*Continuously Variable Transmission*), velocidade, nível de combustível e inclinação do veículo. Os circuitos eletrônicos e os códigos para leitura desses sensores foram desenvolvidos ao longo do presente trabalho. As informações coletadas são tratadas e transmitidas em uma interface desenvolvida no *software* LabView. A interface possui uma tela de apresentação das informações em tempo real e que possibilita a gravação de um arquivo de texto contendo os dados recebidos. A interface também conta com uma tela para análise gráfica do desempenho do veículo a partir do arquivo de texto. Resultados experimentais mostram o funcionamento do sistema de telemetria desenvolvido.

Palavras-chave: Telemetria, Eletrônica, LoRa, Protocolo CRC, Arduino, LabView.

Santiago, Arthur N. **Implementation of Telemetry in the Vehicle of the Cerrado Baja SAE Team**. 2021. 68. Course Completion Project - Mechatronics Engineering, Federal University of Uberlândia, Uberlândia, Brazil.

ABSTRACT

This work involves the implementation of a sensing and telemetry system in the vehicle built by the Cerrado BAJA SAE team at the Federal University of Uberlândia. Initially, two available communication modules are analyzed. More specifically, the Xbee S2 and LoRa Ra-02 modules are compared to carry out communication between the vehicle and the box. The tests carried out involved the measurement of the maximum communication distance, reliability and data integrity, and the cost. As a result of this comparative study, the LoRa RA-02 was adopted to perform telemetry. Sensors are used to measure engine temperature and CVT (Continuously Variable Transmission), speed, fuel level, and vehicle inclination. Electronic circuits and codes for reading these sensors were developed throughout the present work. The collected information is treated and transmitted in an interface developed in the LabView software. The interface has a screen for presenting information in real-time, which allows the recording of a text file containing the received data. The interface also has a screen for graphical analysis of the vehicle's performance from the text file. Experimental results show the functioning of the developed telemetry system.

Key-words: Telemetry, Electronics, LoRa, CRC Protocol, Arduino, LabView.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE FLUXOGRAMAS

LISTA DE ABREVIAÇÕES

SAE - *Society of Automotive Engineers* – Sociedade de Engenheiros Automotivos

CRC – *Cyclic Redundancy Check* - Verificação Cíclica de Redundância

ECU - *Electronic Control Unit* - Unidade de Controle Eletrônico

CVT - *Continuously Variable Transmission* - Transmissão Continuamente Variável

RF – Rádio Frequência

MIPS – Milhão de Instruções Por Segundo

USB - *Universal Serial Bus* – Porta Serial Universal

DC - *Direct Current* – Corrente Contínua

AC - *Alternating Current* – Corrente Alternada

IMU - *Inertial Measurement Unit* – Unidade de Medida Inercial

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

1.1. Visão geral

A palavra telemetria é derivada da junção de duas palavras gregas: *tele,* que significa distante e *metron*, que significa medida, ou seja, obtenção de dados a distância (DIAS, 2010). A telemetria consiste na transmissão de dados, de forma confiável e segura, partindo de um sistema remoto para uma base, onde os dados poderão ser apresentados em tempo real. Os primeiros sistemas de telemetria foram desenvolvidos em meados do século XIX e no início do século XX para transmissão de dados militares, e monitoramento atmosférico e sísmico (QUEIRÓS, 2011). Nos anos 70 e início de 80, os fabricantes de automóveis começaram a utilizar meios eletrônicos para diagnosticar em tempo real problemas no motor, atendendo os padrões de emissão de poluentes da agência *U.S. Environmental Protection Agency* (EPA). Na área automotiva a telemetria permite diagnosticar falhas operacionais em tempo real (TEIXEIRA, 2014).

Os sensores são responsáveis por captar informações do veículo para a Unidade de Controle Eletrônica ou *Electronic Control Unit* (ECU) processar e analisar (NUNES, 2016). Sendo assim, a escolha da forma de comunicação entre sensores e ECU é uma etapa importante no desenvolvimento de um sistema de telemetria. Neste trabalho, dois módulos estão disponíveis: Xbee S2 e LoRa Ra-02. O Xbee S2, fabricado pela *Digi International,* comunica via rádio frequência (RF) no padrão ZigBee IEEE 802.15.4 na frequência de 2,4 GHz. (DIGI, 2018). Já o LoRa Ra-02, fabricado pela *Ai-Thinker,* comunica no padrão LoRaWAN na faixa de 410 - 525 MHz (AI-THINKER, 2017). Então, a primeira parte do trabalho envolve a realização de testes para definição do módulo a ser

empregado. Tais testes avaliam a distância máxima de comunicação, a confiabilidade e integridade dos dados, e o custo.

Posteriormente desenvolver-se-ão *hardware* e *software* para realização da telemetria. Em particular, sensores para medir velocidade, nível de combustível, temperatura do motor e da transmissão CVT (*Continuously Variable Transmission*), e o ângulo de rotação do veículo em torno dos eixos longitudinal e transversal (Y e Z). Visando facilitar o monitoramento dos dados, uma interface gráfica será desenvolvida para possibilitar uma rápida atuação da equipe em situações iminentes de falha. Mais ainda, a telemetria diminuirá a necessidade de paradas de verificação do veículo, que são tipicamente realizadas em provas longas. Isso auxiliará o desempenho da equipe nas competições em que participa.

1.2. Objetivos

Este trabalho tem como objetivo principal realizar a telemetria do veículo da equipe Cerrado Baja SAE. Para realizar essa tarefa, os seguintes objetivos específicos devem ser alcançados:

- Comparar e escolha da tecnologia de comunicação a ser utilizada;
- Transmitir e receber os dados de maneira correta e confiável;
- Coletar velocidade, nível de combustível, temperatura do motor e da CVT, e ângulo de rotação dos eixos longitudinal e transversal (*Y* e *Z*);
- Desenvolver uma interface que permita a equipe acompanhar o desempenho do veículo durante e após as provas.

CAPÍTULO II

COMPARAÇÃO ENTRE DISPOSITIVOS PARA COMUNICAÇÃO DE RADIOFREQUÊNCIA

Neste capítulo descrever-se-ão os módulos disponíveis para o envio das informações obtidas através da instrumentação do veículo. Também será realizada uma comparação para definição de qual hardware adotar.

2.1. Xbee S2

O Xbee S2, Figura 2.1, é uma placa de circuito impresso que se comunica via RF no padrão ZigBee IEEE 802.15.4. Este permite comunicações robustas operando na frequência ISM (*Industrial Scientific and Medical*) de 2.4 Ghz (16 canais) e potência de 2 mW. Este suporta comunicação ponto-a-ponto ou multiponto sem roteamento, podendo hospedar milhares de dispositivos em uma rede. Isso permite que vários módulos se comuniquem entre si em uma rede *wireless* robusta e confiável e que possui perda de pacote, conjunto de informações, quase nula (DIGI, 2018).

Os módulos RF padrão ZigBee foram criados para economizar o máximo de energia possível, isso porque estes dispositivos quando não estão transmitindo ou recebendo dados, entram em um estado de dormência ou em "*Sleep*", consumindo o mínimo de energia. As especificações deste módulo estão dispostas na Tabela 2.1 (MAX STREAM, 2007).

Figura 2.1 – Módulo Xbee Series 2 (elaborada pelo autor).

Tabela 2.1 – Especificações técnicas do módulo Xbee S2 (elaborada pelo autor).

Descrição	**Valor**
Tamanho	24,38 mm x 27,61 mm x 6,85 mm
Interface de dados	UART e SPI
Tensão de alimentação	2,8 V a 3,4 V (DC)
Frequência	2,4 GHz
Taxa de dados máxima	250 Kbps
Potência de transmissão	2 mW
Temperatura de operação	-40°C a 80°C
Peso	3,25 g

2.2. LoRa Ra-02

LoRa Ra-02 (Figura 2.2) permite comunicação a longas distâncias com um consumo mínimo de energia utilizando uma faixa de frequência ISM de 410-525 MHz. Uma rede LoRa adota uma topologia em estrela, ou seja, cada dispositivo da rede é conectado a um ponto central de acesso (AUGUSTIN et al., 2016).

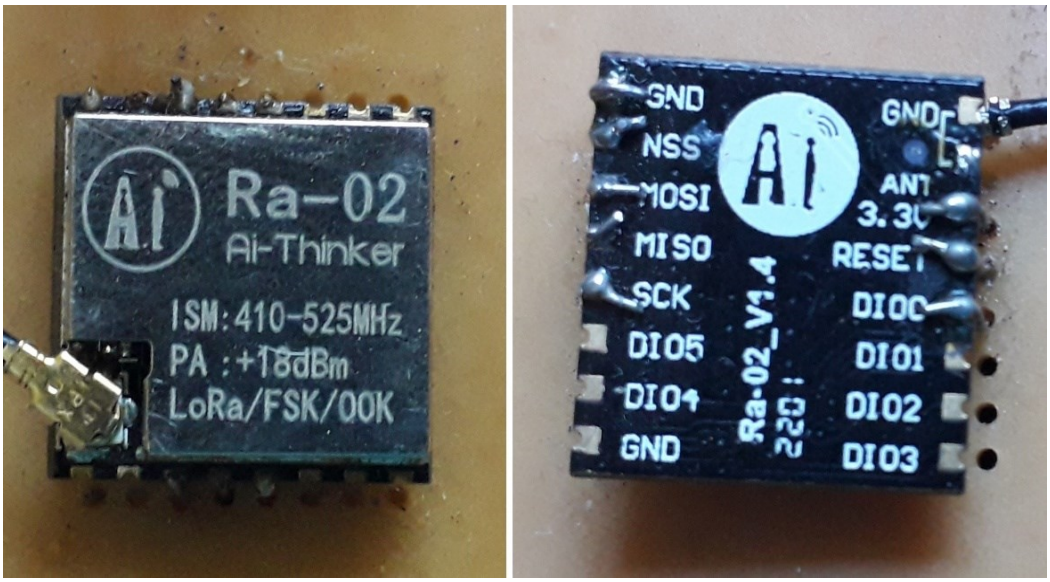


Figura 2.2 – Módulo transceptor LoRa Ra-02 (elaborada pelo autor).

Os dispositivos com a tecnologia LoRa podem realizar transmissões em áreas urbanas a uma distância de 3 a 4 quilômetros e em áreas rurais até 15 quilômetros, utilizando uma potência na ordem de 20 dbm ou 100 mW (LORAWAN, 2020). As especificações deste módulo estão dispostas na Tabela 2.2 (AI-THINKER, 2017).

Tabela 2.2 – Especificações técnicas do módulo LoRa Ra-02 (elaborada pelo autor).

Descrição	**Valor**
Tamanho	17,0 mm x 16,0 mm x 3,2 mm
Interface de dados	SPI
Tensão de alimentação	2,5 V a 3,7 V (DC)
Frequência	410 a 525 MHz
Taxa de dados máxima	300 Kbps
Potência de transmissão	63,1 mW
Temperatura de operação	-30°C a 85°C
Peso	0,45 g

2.3. Comparação entre os módulos Xbee S2 e LoRa Ra-02

Para a escolha do dispositivo a ser utilizado na implementação da telemetria do veículo, a subárea de eletrônica da equipe Cerrado Baja SAE estabeleceu os seguintes requisitos:

- Distância máxima de comunicação;
- Perda de pacotes inferior a 5%;
- Confiabilidade e integridade dos dados;
- Custo.

Todos os testes e experimentos foram realizados no Campus Glória (UFU), mais precisamente na pista de testes da equipe Cerrado, que oferece condições semelhantes às encontradas nas competições que participa.

2.3.1. Distância máxima de comunicação

Como exposto anteriormente, a Equipe Cerrado Baja SAE participa de 2 competições por ano, sendo a etapa Nacional a de maior porte. Com a finalidade de atender as duas competições, a dimensão da pista da etapa Nacional foi considerado como requisito de distância máxima a ser transmitida. Tal pista encontra-se localizada próxima a FATEC – Faculdade de Tecnologia de São José dos Campos em São Paulo.

Durante a competição, são utilizados dois *Boxes*. O *Box* 1 durante todas as provas da competição e o *Box* 2 somente para a prova final (enduro). As Figuras 2.3 e 2.4 mostram as distâncias entre o *Box* 1 e 2 e o ponto mais extremo da pista. Nota-se que o dispositivo escolhido deve permitir a comunicação de até 535 m.

Figura 2.3 – Distância entre o *box* 1 e o ponto mais extremo da pista na FATEC em São José dos Campos – SP (Google Maps).

Figura 2.4 – Distância entre o box 2 e o ponto mais extremo da pista na FATEC em São José dos Campos – SP (Google Maps).

Para determinar o alcance máximo de cada módulo, o pacote de dados 19325A4E8E3A1825A41A587C de tamanho e valor fixo, foi enviado repetidamente em um intervalo de 1 segundo. O receptor foi fixado em um ponto e o transmissor foi colocado a uma distância inicial de 20 metros. Então, essa distância foi aumentada gradualmente até ocorrer perda de comunicação. Como resultado, a distância máxima obtida utilizando o Xbee S2 foi de 146 metros. Com o LoRa Ra-02 uma distância de 632 metros foi alcançada. Isso indica que somente o LoRa Ra-02 atende o requisito de alcance máximo. Mesmo assim, procedeu-se para uma verificação da confiabilidade dos dados recebidos.

2.3.2. Confiabilidade e integridade dos dados

Ao se criar uma rede de comunicação, é fundamental analisar a confiabilidade e integridade dos dados transmitidos. Isso auxilia a minimizar o risco de perder informações ou de encontrar dados modificados durante o processo de comunicação (KUROSE et al., 2006). Diferentes ferramentas para avaliar a confiabilidade de dados em sistemas de comunicação podem ser encontradas na literatura. Por exemplo, métodos de repetição, paridade, soma de verificação (*checksum*) e CRC (*Cyclic Redundancy Check*). Na sequência, tais técnicas são brevemente detalhadas.

2.3.2.1. Método da repetição

O método da repetição consiste em enviar várias vezes a mesma informação com o caractere “\0” no fim de cada transmissão, como apresentado na Tabela 2.3. Desta forma, o receptor pode captar várias vezes a mesma informação e compará-las. Essa técnica é comumente adotada no envio de pacotes pequenos devido à baixa velocidade da transferência de dados (KUROSE et al., 2006). Dependendo da origem do erro, a informação pode chegar ao receptor sempre com o mesmo erro. Como resultado, tomar-se-ia tal informação errada como correta.

Tabela 2.3 – Representação de uma mensagem recebida várias vezes para a detecção de erros com o método da repetição (elaborada pelo autor).

Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Envio 1	T	e	s	t	e		d	e		T	r	a	n	s	m	i	s	s	a	o	\0
Envio 2	T	e	s	t	e		d	e		X	r	a	n	s	m	i	s	s	a	o	\0
Envio 3	T	e	s	t	e		d	e		T	r	a	n	s	m	i	X	s	a	o	\0
Envio 4	T	e	s	t	e		d	e		T	r	a	n	s	m	i	s	s	a	o	\0

2.3.2.2. Paridade

Semelhante ao método da repetição, a técnica da paridade de verificação de erro é normalmente utilizada quando se trabalha com pacotes pequenos, por exemplo 8 bits. A ideia fundamental consiste em adicionar um *bit* de redundância após um determinado número de *bits*. Podem ser empregados *bits* de paridade par ou ímpar. Na primeira, adiciona-se um *bit* 0 ao final da mensagem caso o número de *bits* 1 for par. Senão, um *bit* 1 é adicionado. Já na segunda, tem-se uma situação invertida. Isto é, um número ímpar de bits 1 é indicado por 0 no final do pacote e um número par por 1 (KUROSE et al., 2006). A ideia do *bit* de paridade é ilustrada na Tabela 2.4.

Tabela 2.4 – Representação de como é calculado o *bit* de paridade no caso da paridade par e ímpar (elaborada pelo autor).

Paridade Par		
Carácter	*Bit* de Paridade	Sequência a transmitir
1000100	0	10001000
1110000	1	11100001

Paridade Ímpar		
Carácter	Bit de Paridade	Sequência a transmitir
1000100	1	10001001
1110000	0	11100000

O método da paridade consegue identificar quando há um *bit* errado. Contudo, pode ser que múltiplos bits errados passem.

2.3.2.3. Soma de verificação (Checksum)

A soma de verificação é um método que se baseia no envio das somas dos dados enviados e recebidos. Ilustrar-se-á esse método por meio de um exemplo. Seja um pacote com dados 1010 1100. Primeiramente é realizado a

inversão dos *bits*, ficando 0101 0011. Posteriormente é realizada a soma dos *bits*. A Figura 2.5 ilustra os procedimentos de inversão, soma e o pacote final.

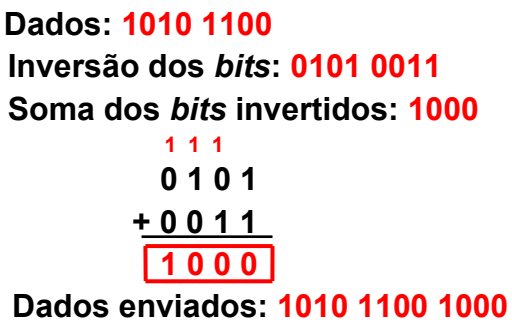


Figura 2.5 – Representação do método *checksum* (elaborada pelo autor).

O pacote que chega no receptor é invertido e somado (incluindo o *checksum*), como apresentado na Figura 2.6. O resultado da soma das palavras deve ser uma palavra com todos bits iguais a 1. Caso isso ocorra, a informação recebida está correta. Senão, os dados estão corrompidos. Nesse cenário, uma possível solução é realizar uma nova transmissão da informação.

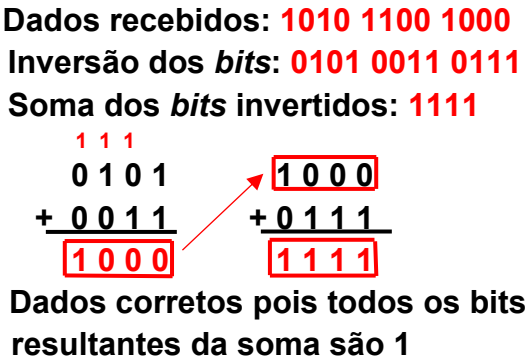


Figura 2.6 – Representação da inversão de *bits* e da soma realizada no receptor (elaborada pelo autor).

Essa técnica permite a detecção de múltiplos erros na informação, e, portanto, é dita mais robusta (KUROSE et al., 2006).

2.3.2.4 Verificação de redundância cíclica (CRC)

O método de verificação de redundância cíclica (CRC - *Cyclic Redundancy Check*) destaca-se pela simplicidade de implementação e capacidade de detecção de erros. Essa técnica é amplamente utilizada em diversos padrões e protocolos de redes de computadores (KUROSE et al., 2006). Na sequência, o CRC será detalhado.

2.3.2.4.1. Fundamentos teóricos CRC

De acordo com Ross e Kurose (2006), as operações matemáticas e lógicas empregadas neste método tem como base a aritmética polinomial, uma vez que pode-se considerar uma sequência de *bits* um polinômio constituído de potências de x com coeficientes 1 e 0.

Qualquer sequência de *bits* (0 e 1) pode-se ser transformada em um polinômio que varia *desde* x^{k-1} *até* x^0, onde *k* é número de *bits* desta sequência (TANENBAUM, 2003). As equações (1) e (2) mostram a representação em polinômios das sequências 10101 e 1101100. Desse modo a comunicação ocorre por meio da troca de polinômios entre o receptor e o transmissor.

$$F(x) = 10101 = 1x^4 + 0x^3 + 1x^2 + 0x^1 + 1x^0 = x^4 + x^2 + 1 \qquad (1)$$

$$F(x) = 1101100 = 1x^6 + 1x^5 + 0x^4 + 1x^3 + 1x^2 + 0x^1 + 0x^0 \qquad (2)$$
$$= x^6 + x^5 + x^3 + x^2$$

O CRC se baseia na divisão de um polinômio por outro. Para realizar a divisão dos polinômios uma operação lógica XOR é realizada para cada par de coeficientes de acordo com o grau correspondente (PETERSON et al., 2004). Considerando os polinômios $p_1(x) = x^4 + x^3 + x$ e $p_2(x) = x^4 + x^2 + x + 1$ como exemplo para demonstrar a operação XOR. Primeiramente, $p_1(x)$ e $p_2(x)$ são representados em termos de um número binário:

$$p_1(x) = x^4 + x^3 + x = 11010$$

$$p_2(x) = x^4 + x^2 + x + 1 = 10111$$

Então, a operação XOR *bit* a *bit* é aplicada a esses polinômios.

$$\begin{array}{r} \oplus\ 11010 \\ \underline{\quad 10111} \\ 01101 \end{array}$$

Para que se entenda melhor, seja $F(x) = 10011101$ a mensagem, e $G(x) = 101101$ o nosso gerador, um exemplo é apresentado em seguida.

- Codificação da mensagem: O primeiro passo é acrescentar uma quantidade *n* de bits nulos ao final da mensagem, sendo que *n* é o grau do polinômio gerador. O grau do polinômio gerador é 5, logo acrescenta-se 5 bits nulos ao final da mensagem que passa a ser 1001110100000;
- Codificação da mensagem: Realiza-se a divisão de 1001110100000 por 101101 (Figura 2.7), obtendo-se o CRC = 11001.

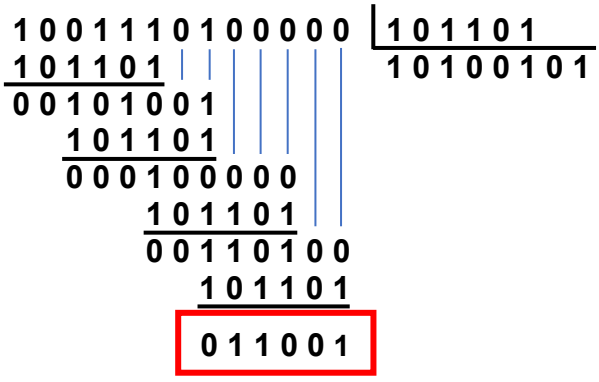


Figura 2.7 – Representação de como é feita a divisão no módulo transmissor (elaborada pelo autor).

- Codificação da mensagem: Monta-se a mensagem completa a ser enviada concatenando o CRC, 1001110111001.
- Decodificação da mensagem: Ao chegar à informação no receptor, a mensagem completa recebida é dividida pelo gerador. Se o resto desta divisão for uma sequência de *bits* nulos (Figura 2.8), a mensagem enviada foi recebida corretamente. Caso ocorra algum *bit* 1 no resto da divisão, a mensagem recebida está corrompida.

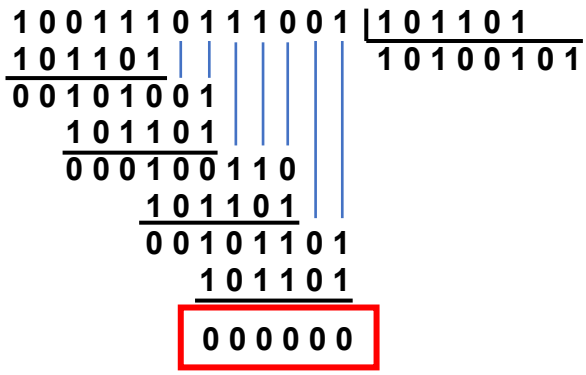
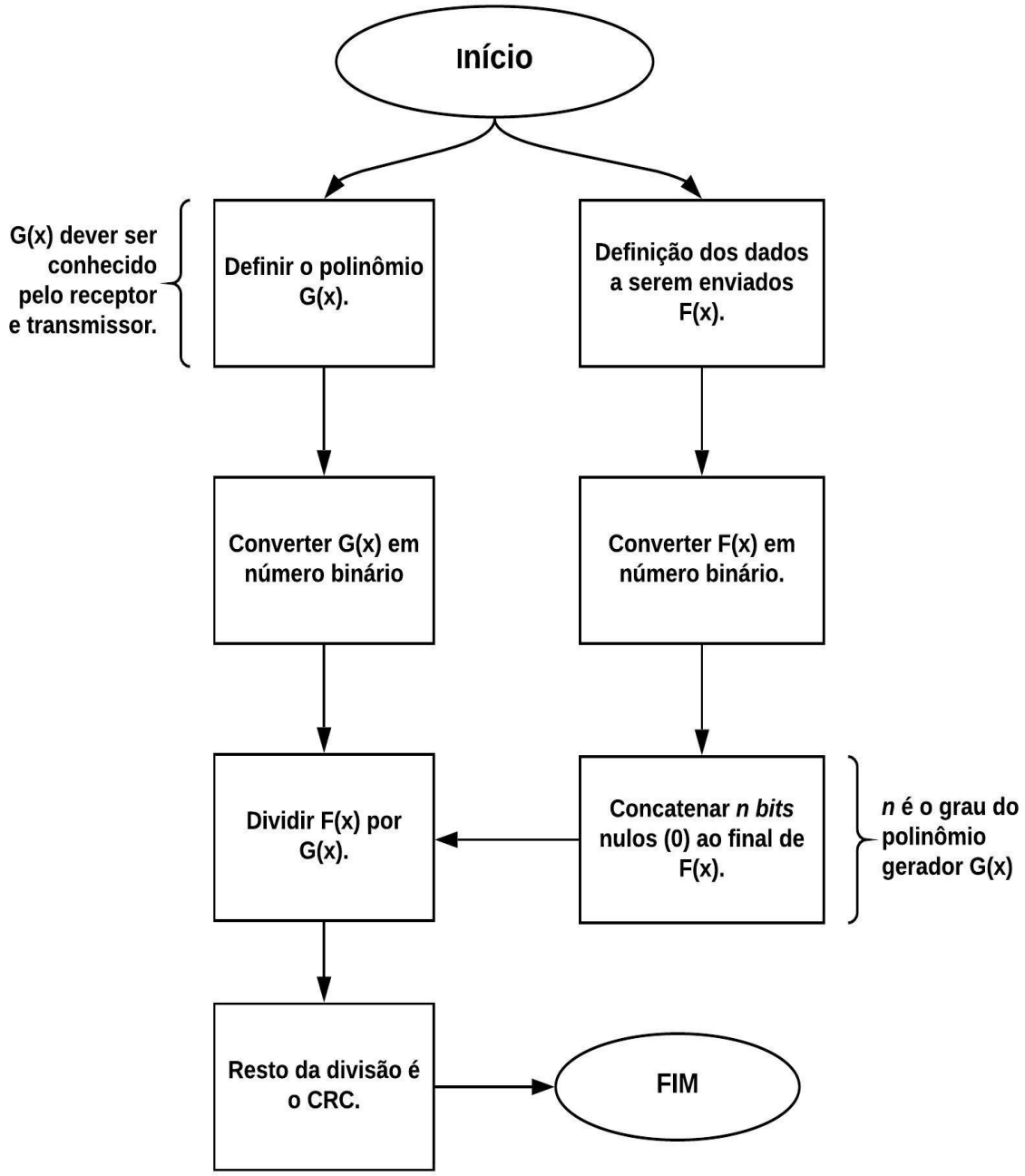


Figura 2.8 – Representação da divisão no receptor (elaborada pelo autor).

O cálculo do CRC é realizado segundo o algoritmo mostrado no Fluxograma 2.1.



Fluxograma 2.1 – Fluxograma do método CRC (elaborada pelo autor).

A partir de uma quantidade redundante de *bits* o método CRC detecta erros em dados com milhares de *bytes* de extensão. Por esse motivo, o CRC é amplamente utilizado em diversos padrões e protocolos de redes de computadores (KUROSE et al., 2006). O código implementado no Arduino para o CRC encontra-se no Apêndice A.

2.3.2.4.2. Resultados CRC

Para testar a confiabilidade e integridade dos dados, um pacote de tamanho fixo 19325A4E8E3A1825A41A587C, foi enviado 100 vezes ao receptor com intervalo entre pacotes de 1 segundo, para as distâncias de 80, 100 e 120 metros.

Com a implantação do CRC, pode-se verificar a quantidade de erros de transmissão que ocorriam com cada um dos módulos disponíveis (Xbee e LoRa). A Tabela 2 apresenta o comparativo para diferentes distâncias.

Tabela 2.5 - Resultados do teste para verificação de dados corrompidos enviados de cada módulo (elaborada pelo autor).

Módulos	**Distâncias**		
	80 m	**100 m**	**120 m**
Xbee S2	0	0	0
LoRa Ra-02	1	2	2

Nota-se que o Xbee não apresentou nenhum pacote de mensagem contendo *bits* errados, enquanto o LoRa apresentou no máximo 2 pacotes. Analisando em detalhes os problemas apresentados pelo LoRa notou-se que em todos os testes, os primeiros pacotes continham erros. Os pacotes posteriores não apresentavam erro. Então, pode-se concluir que, excetuando os pacotes iniciais, ambos os módulos apresentam desempenho semelhante.

2.3.3. Custo

Um fator importante em todo projeto de engenharia é o custo dos componentes. Nesse ponto, o LoRa apresenta uma vantagem em relação ao Xbee. O transceptor LoRa Ra-02 Sx1278 433 Mhz custa, em média, R$ 40,00 e o Xbee 2 mW Serie 2 custa, em média, R$ 200,00.

2.3.4. Escolha do módulo

Nos testes realizados o LoRa se destacou em dois requisitos, distância máxima e custo. O Xbee se mostrou melhor na confiabilidade dos dados. Vale relembrar que, após o envio dos dois primeiros pacotes, o desempenho dos dispositivos foi equivalente. Também cabe salientar que somente com o LoRa

Ra-02 foi possível cobrir a distância total da pista (535 m). Com efeito, escolheu-se o LoRa para a realização da telemetria, assim como o método CRC para detecção de erros.

CAPÍTULO III

SISTEMA DE TELEMETRIA

A telemetria envolve a aquisição, o tratamento e o envio de dados. Neste capítulo, descrever-se-á o sistema de telemetria desenvolvido, os sensores instalados no veículo e a forma de comunicação entre os dispositivos.

3.1 Estrutura do sistema de telemetria

A Figura 3.1 apresenta a estrutura do sistema de telemetria desenvolvido. A primeira etapa é a aquisição das informações dos sensores instalados no veículo. Então, tais informações são tratadas na ECU e transmitidas para a base utilizando o LoRa. Os dados são recebidos por meio de um Arduino UNO e tratados no *software* Labview antes de serem apresentados ao usuário em uma interface. A comunicação entre Arduino UNO e Labview é realizada via USB Serial.

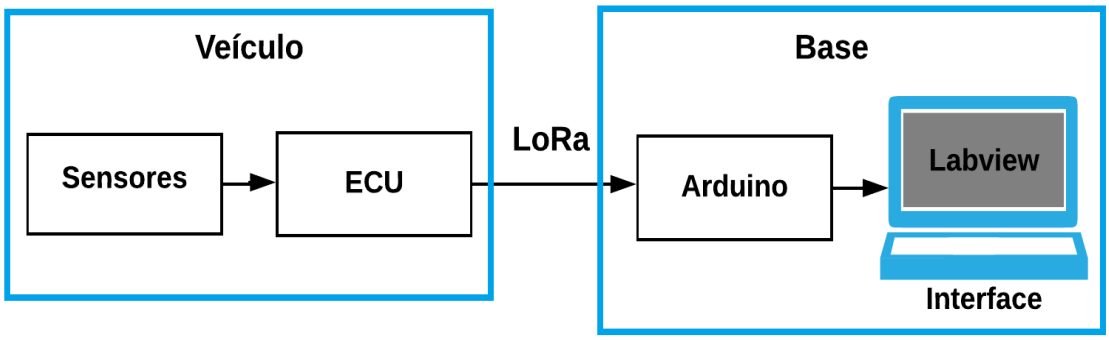


Figura 3.1 – Representação geral da telemetria (elaborada pelo autor).

3.2 Dispositivos instalados no veículo

De acordo com as normas da competição baja SAE, escolheram-se sensores capacitivos, indutivo, temperatura tipo sonda e IMU (Unidade de Medida Inercial ou *Inertial Measurement Unit*) para aquisição das grandezas de interesse, respectivamente, nível de combustível, velocidade, temperatura do motor e da transmissão CVT, e rotação em torno dos eixos longitudinal e lateral do veículo.

3.2.1 Acompanhamento do nível de combustível por meio de sensor capacitivo

O sensor capacitivo adotado gera um campo eletrostático na extremidade. Com a aproximação de qualquer material, sólido (metálico ou não metálico) ou líquido, haverá um aumento de capacitância. A variação de capacitância é detectada por meio de um circuito eletrônico e altera os estados das saídas normal aberta (NA) e normal fechada (NF) (NILSSON, 2015). Esse processo é ilustrado na Figura 3.2.

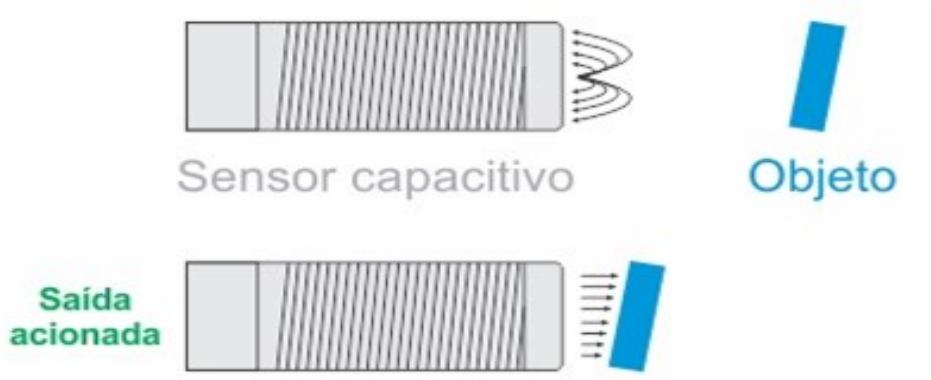


Figura 3.2 – Princípio de funcionamento do sensor capacitivo (Disponível em: http://profcolassante.blogspot.com/2015/06/sensores-capacitivos-e-indutivos.html. Acesso em Abril 2020).

O sensor capacitivo escolhido é do tipo tubular C18-8-DPC (Figura 3.3). As especificações do sensor encontram-se na Tabela 3.1 (METALTEX, 2020):

Tabela 3.1 – Especificações técnicas do sensor capacitivo C18-8-DPC (elaborada pelo autor).

Descrição	**Valor**
Material do corpo	Latão niquelado
Diâmetro	18 mm
Distância de detecção	8 mm
Tensão de saída	10 V a 36 V (DC)
Sinal de saída	PNP
Tipos de portas de saída	NA + NF
Tensão de alimentação	10 V a 36 V (DC)
Corrente máxima de saída	200 mA
Consumo máximo de corrente	15 mA
Frequência máxima de chaveamento	50 Hz
Grau de proteção	IP66

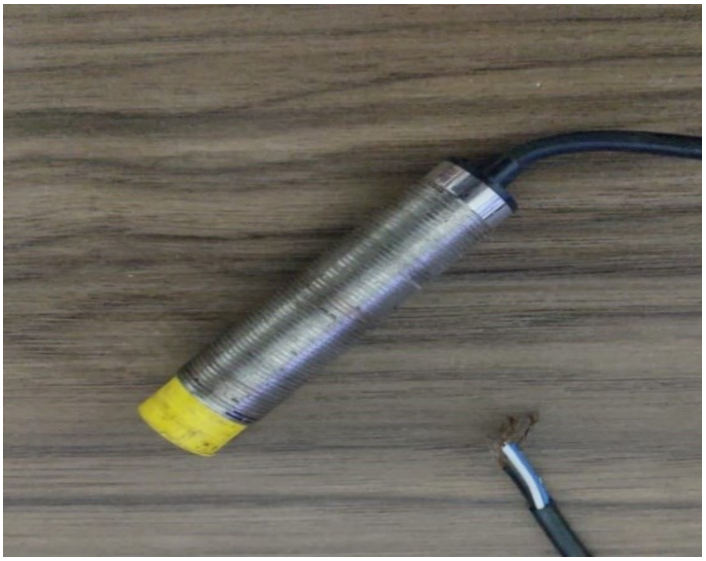


Figura 3.3 – Sensor capacitivo tubular C18-8-DPC (elaborada pelo autor).

O circuito eletrônico utilizado para leitura do sensor é apresentado na Figura 3.4. O sensor possui 3 fios, um deles é ligado ao terra (GND), outro em uma fonte de 12 volts e o último é o sinal. Quando o sensor não está acionado, o sinal fornecido é de 0 V. Já acionado, há uma tensão de 12 V na saída. Contudo, a tensão máxima nos canais de entrada do Arduino UNO é de 5 V. Para conectar o sensor ao Arduino, implementou-se o divisor resistivo mostrado na Figura 3.4. O código para leitura do sinal do sensor capacitivo encontra-se no Apêndice C.1.

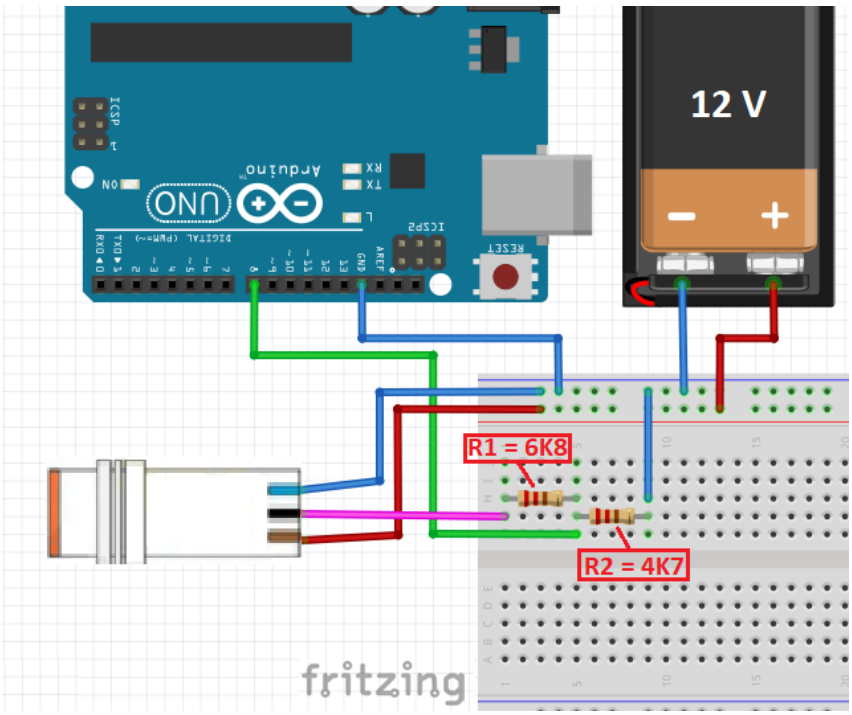


Figura 3.4 – Montagem do circuito para comunicação com o sensor capacitivo tubular C18-8-DPC (elaborada pelo autor).

Tipicamente os veículos comerciais utilizam uma bóia fixada dentro do tanque para a aferição do nível de combustível. Tal mecanismo não é permitido na competição. Como alternativa, adotaram-se dois sensores capacitivos, operando na configuração normalmente fechado, para identificar os níveis críticos (superior e inferior) de combustível. O Sensor 1 foi instalado na altura que representa 70% do volume do tanque e o Sensor 2 de 30%. Sendo assim fica estabelecido que se ambos os sensores estiverem ativos (nível lógico alto), o nível de combustível está acima ou igual a 70% do volume do tanque, cerca de 3,7 L. Caso os dois sensores estiverem em nível lógico baixo o combustível está abaixo de 30%. Com o Sensor 1 em nível logico baixo e o Sensor 2 nível alto, temos que o combustível está na faixa de 30% a 70%. Essa lógica de funcionamento está ilustrada na Figura 3.5.

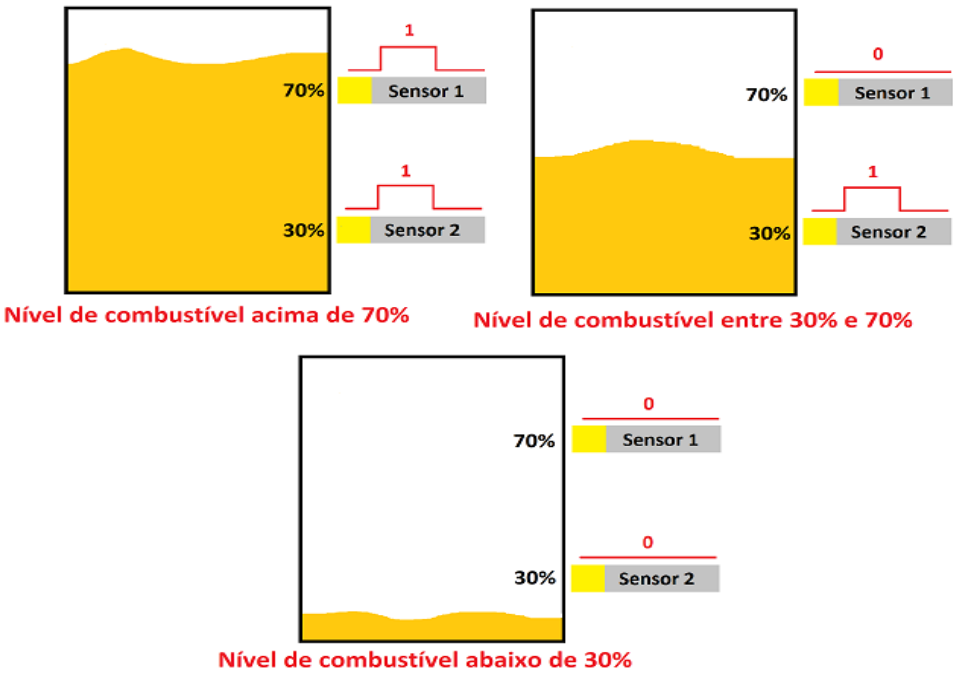


Figura 3.5 – Instalação dos sensores capacitivos no tanque de combustível (elaborada pelo autor).

3.2.2 Medição da velocidade do veículo por meio de sensor indutivo

O funcionamento desse tipo de sensor é análogo ao do sensor capacitivo descrito na Seção 3.2.1. Mais precisamente, um campo magnético é gerado na extremidade do sensor. Na presença de um objeto metálico, há uma variação do campo. Essa variação é detectada por um circuito eletrônico, que ativa a saída do sensor (NILSSON, 2015). Esse processo é ilustrado na Figura 3.6.

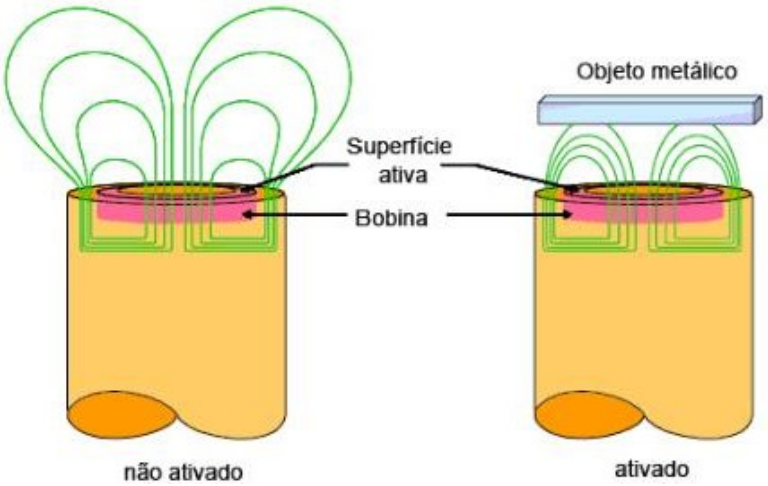


Figura 3.6 – Representação de funcionamento do sensor indutivo (Disponível em: https://www.docsity.com/pt/elementos-de-automacao-aula-08-sensores/4768233/. Acessado em: 22 Abril 2020).

O sensor indutivo escolhido é do tipo tubular I18-5-DPC-K12 (Figura 3.7). A seguir, Tabela 3.2, encontram-se as especificações deste sensor (METALTEX, 2020).

Tabela 3.2 – Especificações técnicas do sensor indutivo I18-5-DPC-K12 (elaborada pelo autor).

Descrição	**Valor**
Material do corpo	Metálico
Diâmetro	18 mm
Distância de detecção	5 mm
Tensão de saída	10 V a 36 V (DC)
Tipos de portas de saída	PNP
Lógica de saída	NA + NF
Tensão de alimentação	10 V a 36 V (DC)
Corrente máxima de saída	200 mA
Consumo máximo de corrente	15 mA
Frequência máxima de chaveamento	50 Hz
Grau de proteção	IP67

Figura 3.7 – Sensor indutivo tubular I18-5-DPC-K12 (elaborada pelo autor).

O sensor indutivo foi instalado na roda dianteira direita posicionado perpendicularmente ao disco de freio (Figura 3.8). Tal sensor, detecta a passagem de três parafusos igualmente espaçados que fixam o disco de freio.

Figura 3.8 – Instalação sensor indutivo tubular I18-5-DPC-K12 (elaborada pelo autor).

O cálculo da velocidade é realizado estabelecendo uma relação entre as grandezas lineares e angulares. A velocidade linear de um corpo em movimento circular uniforme é igual ao produto da velocidade angular pelo raio da trajetória descrita pelo corpo (HIBBELER, 2005), isto é,

$$V = \omega \, . \, r \tag{3}$$

em que V é a velocidade linear do corpo (m/s), ω é a velocidade angular (rad/s) e r é o raio da roda (m). Sendo assim, a equação da velocidade linear pode ser escrita como:

$$V = \frac{2\pi}{T} \, . \, r \tag{4}$$

A velocidade é calculada contando o tempo T entre os pulsos causados pela passagem dos parafusos do freio. Como há três pulsos por volta, calcula-se a velocidade fazendo-se

$$V = \frac{2\pi}{3T} \, . \, r \tag{5}$$

O circuito eletrônico utilizado para leitura do sensor indutivo e o código de implementação no Arduino são similares aos apresentados na Seção 3.2.1.

3.2.3 Medição da temperatura da transmissão e do bloco do motor empregando sensores resistivos do tipo sonda

O funcionamento de alguns sensores de temperatura é baseado na variação da resistência elétrica de acordo com a mudança de temperatura. Mais precisamente, esse tipo de sensor é composto por um termístor cuja resistência elétrica é inversamente proporcional à temperatura. (DALLAS, 2001).

O sensor de temperatura escolhido é o DS18B20 (Figura 3.9). Esse sensor foi projetado para medições precisas em ambientes úmidos ou em recipientes com líquido. A temperatura é enviada em graus Celsius para o microcontrolador usando apenas um fio, por meio do protocolo de comunicação *One wire* ou *1-wire* (DALLAS, 2001).

Figura 3.9 – Sensor de temperatura tipo sonda DS18B20 (Disponível em: https://capsistema.com.br/index.php/2019/12/11/guia-do-sensor-de-temperatura-ds18b20-com-arduino/ Acessado em: 15 Dezembro 2020).

Na Tabela 3.3 são apresentadas as especificações do sensor (DALLAS, 2001).

Tabela 3.3 – Especificações técnicas do sensor de temperatura DS18B20 (elaborada pelo autor).

Descrição	**Valor**
Tensão de operação	3 V a 5,5 V (DC)
Faixa de medição	-55°C a 125°C
Precisão	±0,5°C
Resolução	9 ou 12 bits (configurável)
Período de atualização	menor que 750 ms
Encapsulamento	aço inoxidável
Dimensão do encapsulamento	6 mm x 50 mm
Comprimento do cabo	1 m

Foram instalados dois sensores de temperatura no veículo. Um dentro da carcaça da transmissão CVT, para medir a temperatura da correia de transmissão. Outro junto ao bloco do motor, para detectar um possível superaquecimento.

A montagem do circuito para leitura deste sensor é apresentada na Figura 3.10. A resistência de 4,7k Ω é um resistor de *pull-up* (NOGUEIRA, 2020) necessário no protocolo *One-wire.*

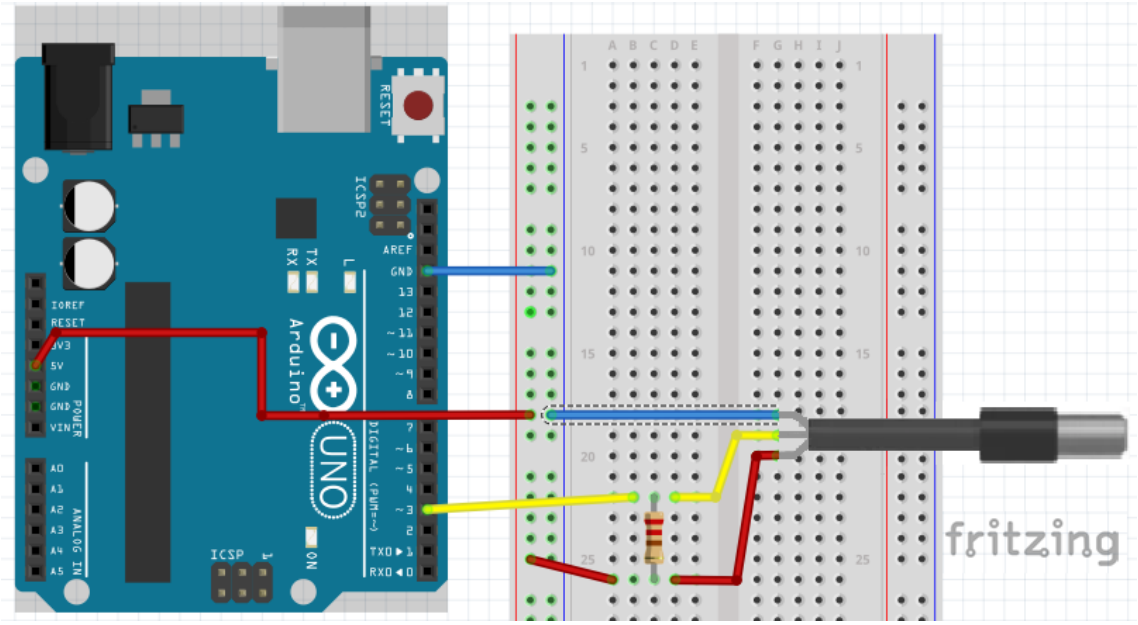


Figura 3.10 – Montagem do circuito eletrônico para leitura do sensor de temperatura DS18B20 (elaborada pelo autor).

O código utilizado no Arduino para leitura do sensor de temperatura encontra-se no Apêndice C.2.

3.2.4 Medição da inclinação do veículo utilizando uma IMU do tipo MPU-6050

A unidade de medição inercial IMU é o dispositivo empregado para determinar a orientação de um objeto. A IMU é composta por, pelo menos, um giroscópio e um acelerômetro. O primeiro fornece as medidas de velocidade angular, enquanto o último é empregado para determinar a aceleração linear (SAHAWNEH et al., 2008).

O sensor escolhido para medir os ângulos de rotação do veículo foi o MPU 6050 (Figura 3.11) que contém acelerômetro e giroscópio tipo MEMS e um sensor de temperatura em um único chip que utiliza o Arduino (INVENSENSE, 2013).

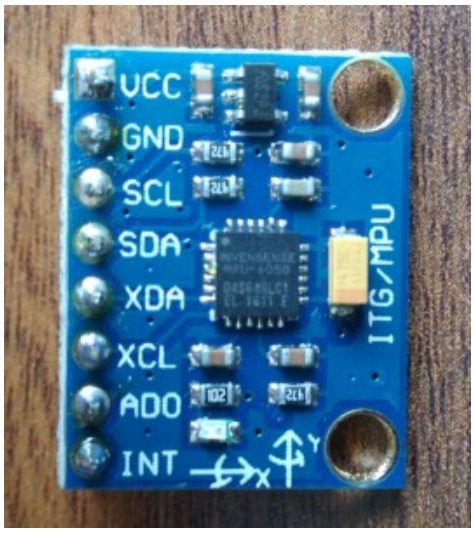


Figura 3.11 – Chip MPU-6050 (elaborada pelo autor).

A seguir são apresentadas as especificações do sensor (INVENSENSE, 2013).

Tabela 3.4 – Especificações técnicas do sensor MPU-6050 (elaborada pelo autor).

Descrição	**Valor**
Tensão de operação	3 V a 5 V (DC)
Conversor AD	16 *bits*
Comunicação	Protocolo padrão I2C
Faixa do giroscópio (configurável)	± 250, ± 500, ± 1000, ±2 000°/s
Faixa do acelerômetro (configurável)	± 2, ± 4, ± 8, ± 16 g
Dimensões	20 mm x 16 mm x 1 mm

A definição dos eixos *x*, *y* e *z* em relação ao corpo do veículo é mostrada na Figura 3.12.

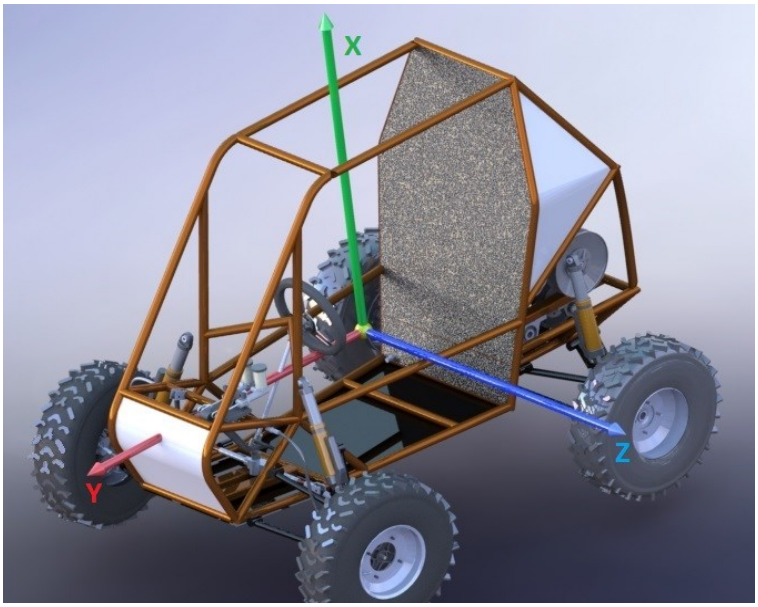


Figura 3.12 – Representação dos eixos *x, y, z* em relação ao corpo do veículo (elaborada pelo autor).

Os dados do acelerômetro e do giroscópio são utilizados em conjunto para obtenção das informações de inclinação do veículo. Desse modo, reduzem-se os efeitos de ruído de medida e obtêm-se informações mais precisas (MOURA, 2013).

Em particular, três bibliotecas foram utilizadas: MPU6050.h, Wire.h e I2Cdev.h na obtenção dos dados. Essas bibliotecas podem ser encontradas em *https://github.com/jrowberg/i2cdevlib*. Para calcular os ângulos de rotação dos eixos utilizou-se o Filtro Complementar devido à fácil implementação e ao baixo custo computacional (COLTON, 2007).

O filtro complementar resolve os problemas durante a aquisição dos valores, pois integra um filtro passa-baixas diminuindo o ruído do acelerômetro e um passa-altas para o giroscópio melhorando a acuracidade. A Figura 3.13 apresenta o diagrama esquemático do filtro implementado.

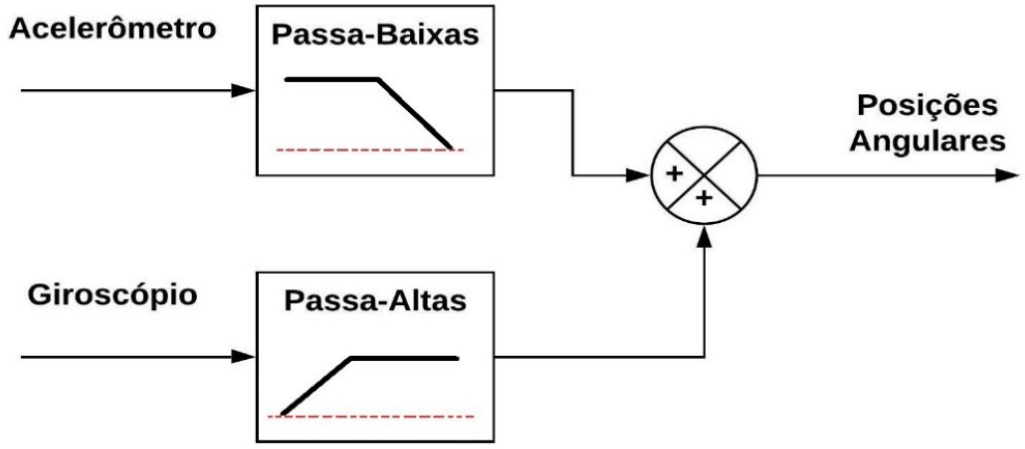


Figura 3.13 – Diagrama esquemático do filtro complementar (elaborada pelo autor).

Os ângulos de inclinação do veículo são obtidos fazendo-se

$$\theta_k = [\alpha(\theta_{k-1} + \Theta\ .\ \delta t)] + [(1-\alpha)\ \Phi] \qquad (6)$$

sendo θ o ângulo filtrado, Θ obtida com o giroscópio após o filtro passa alta, Φ é o ângulo de inclinação calculado por meio do acelerômetro e após o filtro passa baixa, α [0, 1] é o coeficiente do filtro, δt é o período de amostragem e k o índice de tempo discreto. O coeficiente α define um valor de confiança entre os dados de acelerômetro e giroscópio, neste trabalho adotou-se α = 0,98. O código utilizado no Arduino para leitura do sinal do sensor encontra-se no Apêndice C.3.

Essas informações são utilizadas pela equipe para estimar as inclinações máximas de subida e descida suportadas pelo veículo. Tais limitantes estão associadas à potência do motor e à capacidade de frenagem, respectivamente.

O MPU-6050 possui 8 pinos (Figura 3.14), sendo terra (GND), alimentação (VCC) e seis para comunicação. Desses, foram utilizados somente os pinos SCL (*Serial Clock*) e SDA (*Serial Data*) para comunicação via I2C. A montagem do circuito para a utilização deste sensor é apresentada na Figura 3.15.

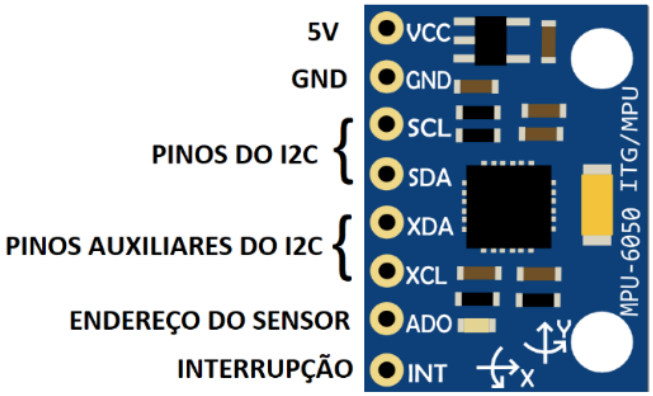


Figura 3.14 – Pinos do sensor MPU-6050 (elaborada pelo autor).

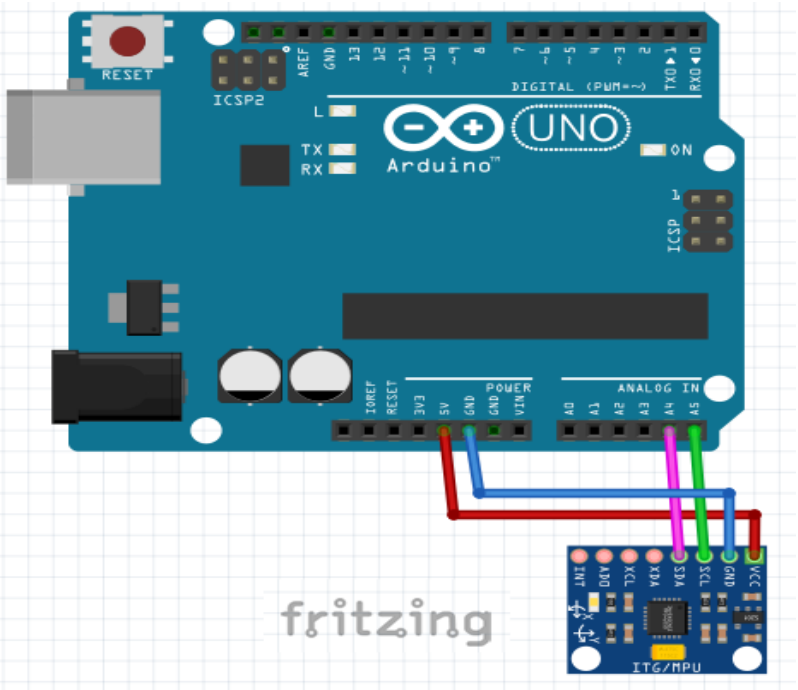


Figura 3.15 – Montagem do circuito eletrônico do sensor MPU-6050 (elaborada pelo autor).

3.2.5 Montagem do dispositivo de comunicação LoRa Ra-02

Como definido anteriormente, o módulo LoRa Ra-02 foi utilizado para realizar o envio das informações do veículo para o *box*. A Figura 3.16 mostra a montagem do circuito do módulo LoRa com o Arduino.

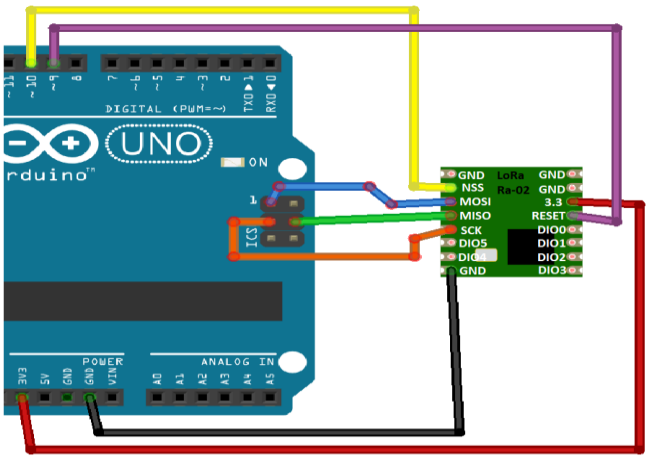


Figura 3.16 – Montagem do circuito eletrônico do módulo LoRa Ra-02 (elaborada pelo autor).

A forma de configuração do LoRa e como a comunicação é realizada estão descritas no Apêndice B.

3.3 Unidade de Controle Eletrônico

Foi desenvolvida uma placa de circuito impresso, utilizando uma placa de fenolite cobreada, que será a Unidade de Controle Eletrônico (ECU) do veículo (Figura 3.17). A ECU é empregada para mostrar informações no painel e enviar dados para o *box*.

Para o processamento dos dados foi utilizado um microcontrolador ATmega 328P que compõe as placas Arduino UNO (ATMEL, 2015). Foi utilizado 12 entradas do microcontrolador, com um tempo de varredura dos sinais de 500 ms. O diagrama esquemático e o PCB (*Printed Circuit Board*) da ECU encontram-se no Apêndice D e Apêndice E respectivamente.

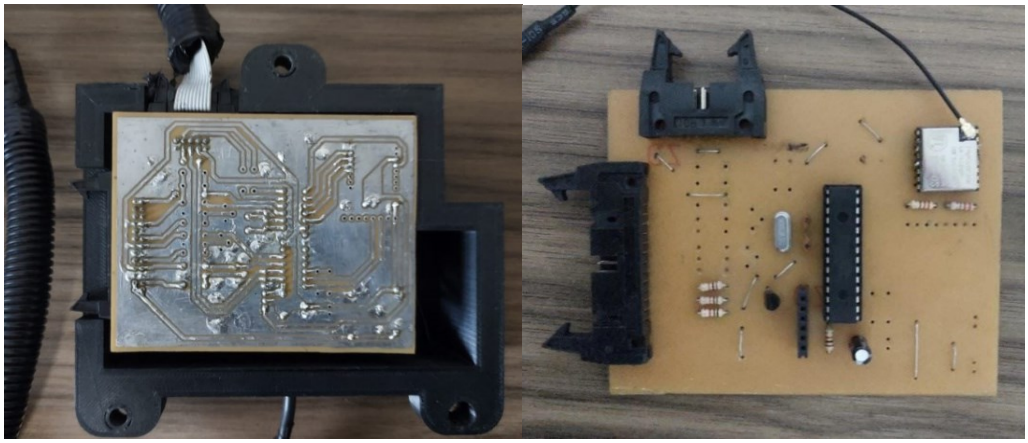
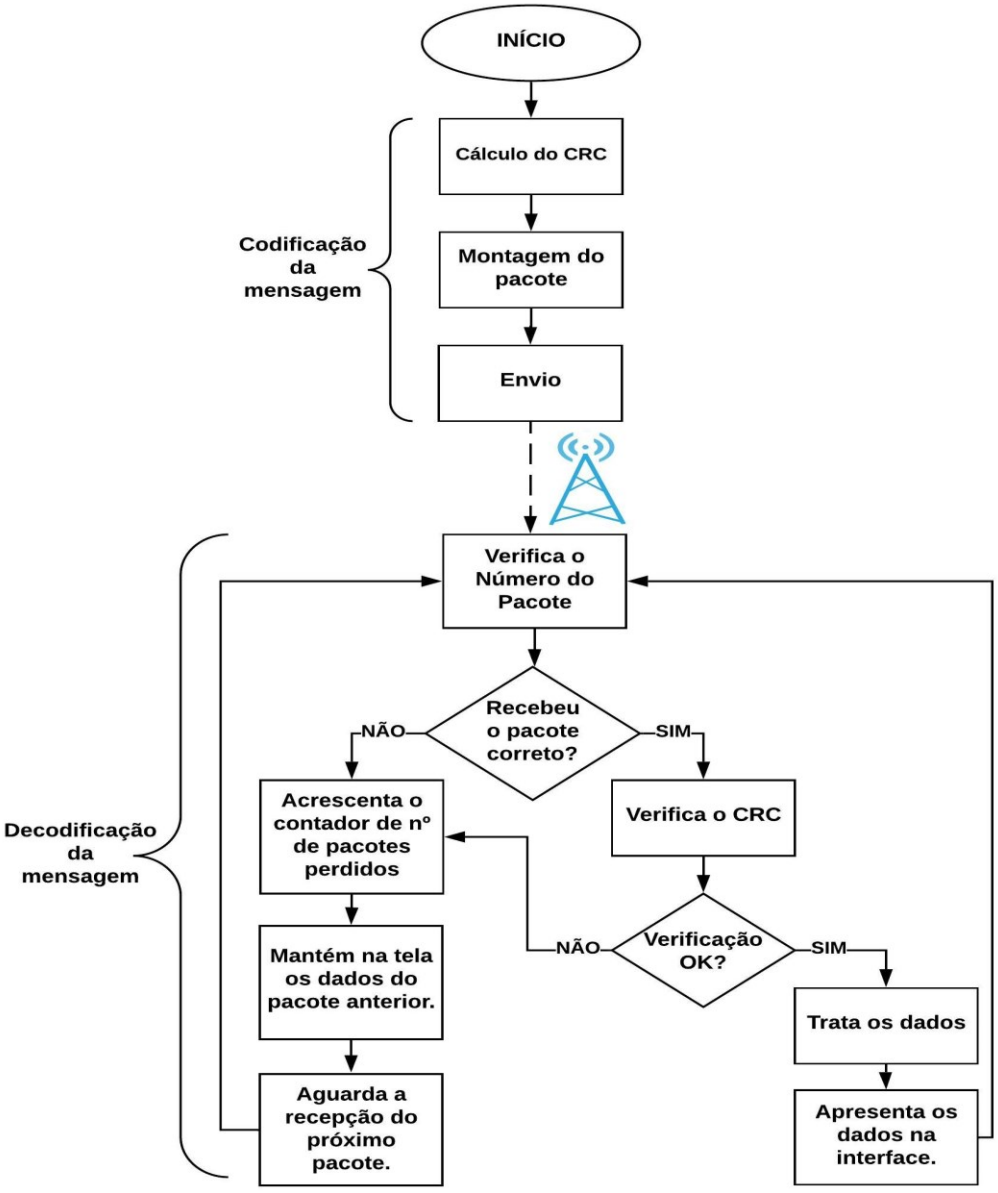


Figura 3.17 – Foto da placa ECU projetada para a telemetria (elaborada pelo autor).

3.4 Comunicação LoRa

A comunicação entre veículo e *box* é realizada seguindo protocolo apresentado no Fluxograma 3.1. Com o cálculo do CRC, o remetente monta o pacote e o envia. O destinatário recebe e checa o número do pacote para verificar se é o esperado. Em caso positivo, valida-se o CRC. Caso contrário um contador é acrescido e a interface mantém os dados do último pacote na tela aguardando o próximo pacote válido. Esse processo também ocorre caso haja incompatibilidade na validação do CRC. No sucesso de ambas validações, os dados são tratados e apresentados ao usuário.



Fluxograma 3.1 – Fluxograma do protocolo de comunicação implementado (elaborada pelo autor).

Cada pacote é composto por uma concatenação ordenada das informações dos sensores. A ordem é mostrada na Tabela 3.5. Em particular, as seguintes informações são incluídas no pacote: Número do pacote, velocidade do veículo, temperatura da CVT, temperatura do motor, distância percorrida, nível de combustível, estado do botão de emergência, rotações em torno dos eixos longitudinal e transversal, nível de bateria, e por último o código CRC.

Tabela 3.5 – Exemplo de pacote enviado e definição do significado de cada posição (elaborada pelo autor).

Posição	Exemplo de pacote em Hexadecimal	Significado
0	4	Número do Pacote
1	8	
2	2	Velocidade
3	D	
4	2	Temperatura da CVT
5	D	
6	9	
7	5	Temperatura do motor
8	4	
9	2	
10	9	Distância percorrida
11	E	
12	4	
13	3	Nível de combustível e estado de emergência
14	3	Rotação em torno do eixo Z
15	2	
16	5	Rotação em torno do eixo Y
17	5	
18	7	Nível da bateria
19	D	
20	5	Código CRC
21	8	
22	7	
23	C	

Cabe salientar que os dados são armazenados em um formato hexadecimal. Então, tem-se um pacote de 96 bits ou 12 bytes. Como resultado,

tem-se um pacote de 12 bytes que são representados na forma hexadecimal. É importante ressaltar que a equipe adota um padrão de envio de dados onde todos os dados enviados estão no formato hexadecimal.

Também é importante comentar que o *software* e o *hardware* envolvidos na medição de distância percorrida, estado de emergência e nível de bateria foram desenvolvidos previamente pela Equipe Cerrado. Por isso, não foram descritos neste trabalho.

A informação contida na posição 13 do pacote conta somente com 4 *bits*, sendo que cada *bit* representa uma informação (Tabela 3.6). Como dito anteriormente, as informações são enviadas em hexadecimal portanto o valor 3 representa 0011 em binário. A primeira e segunda posições indicam o nível de combustível (1 – cheio, 0 – vazio) e a terceira o botão de emergência (1 – acionado, 0 – desabilitado). O *bit* restante pode ser utilizado futuramente pela equipe para acompanhamento de outras informações.

Tabela 3.6 – Exemplo da décima terceira posição do pacote que representa o nível de combustível e o botão de emergência (elaborada pelo autor).

Posição	***Informação em Hexadecimal***	***bit***	**Significado**
13	***3***	**1**	Vago
		1	Botão de emergência
		0	Nível de combustível 30%
		0	Nível de combustível 70%

CAPÍTULO IV

Interface

Com toda a instrumentação e o protocolo de comunicação concluídos, procedeu-se para o desenvolvimento de uma interface para apresentação dos dados para a equipe no *box*. Tal interface é detalhada na sequência.

4.1 Tela principal da interface

A Figura 4.1 apresenta a tela principal da interface. O *software* Labview foi utilizado no desenvolvimento dessa tela. O *layout* foi proposto pela equipe de modo a facilitar o acompanhamento do desempenho do veículo. Pode-se verificar a distância percorrida, o tempo de prova, a inclinação do veículo, o estado do botão de emergência, o nível de combustível (cheio ou vazio), a velocidade, as temperaturas de motor e CVT e outras informações.

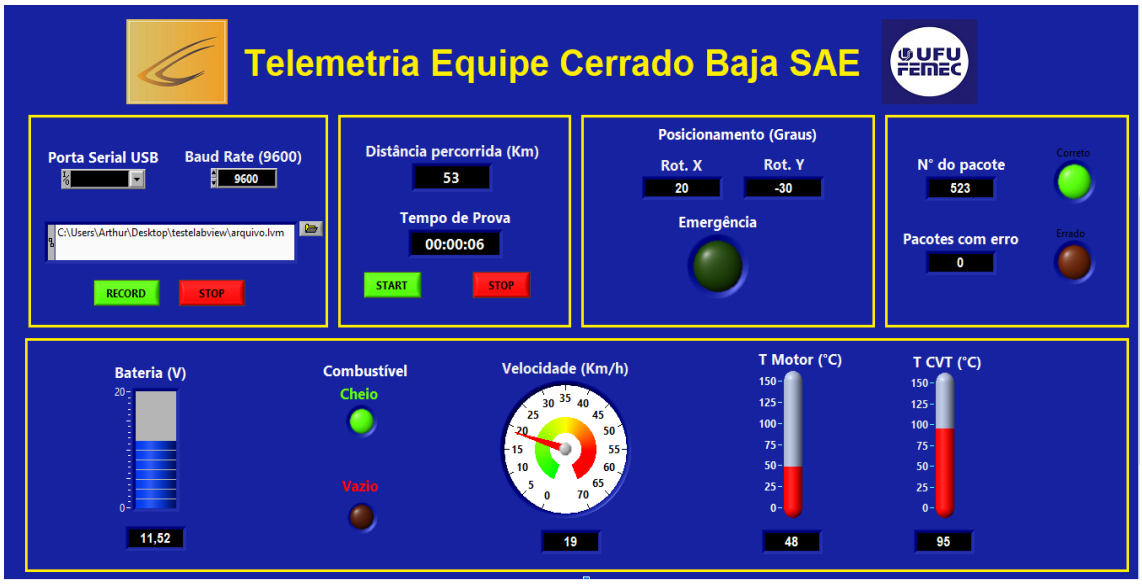


Figura 4.1 – *Layout* da tela principal desenvolvida para a telemetria (elaborada pelo autor).

A interface foi programada para receber os dados através de uma porta USB e realizar toda a decodificação dos dados. Para isso, foi necessário instalar o *drive*r de instrumento NI-VISA da *National Instruments,* que implementa o padrão VISA (*Virtual Instrument Software Architecture*) de entrada-saída. O VISA é um padrão para a configuração, programação e resolução de problemas de sistemas de instrumentação.

O código de implementação para a leitura da porta serial é apresentado na Figura 4.2. Nesta parte são feitas algumas configurações para o funcionamento da serial. O *Baud Rate* de comunicação e a porta USB podem ser escolhidos através dos seus respectivos campos na interface (vide Figura 4.1).

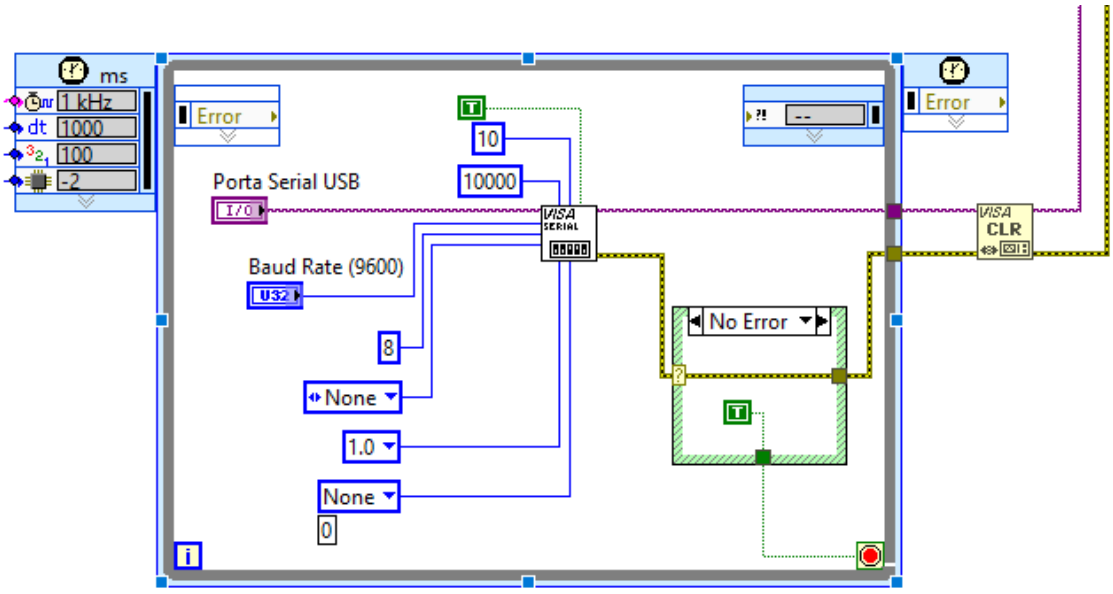


Figura 4.2 – Trecho do programa da interface da telemetria no qual os dados são lidos através de uma porta USB serial (elaborada pelo autor).

Logo após o recebimento do pacote via USB acontece toda a decodificação dos dados armazenado o pacote em uma variável do tipo *string,* a qual é utilizada para a validação dos dados através do cálculo do CRC. Caso a validação falhe, o processo é interrompido e um contador responsável por contabilizar o número de pacotes com falha é incrementado. Senão, o programa prossegue. Então, o pacote é decodificado, de acordo com a posição de cada informação na *string* e, em seguida, os valores são convertidos para a base decimal. Os dados tratados são exibidos na interface. Esse procedimento é realizado pelo código da Figura 4.3.

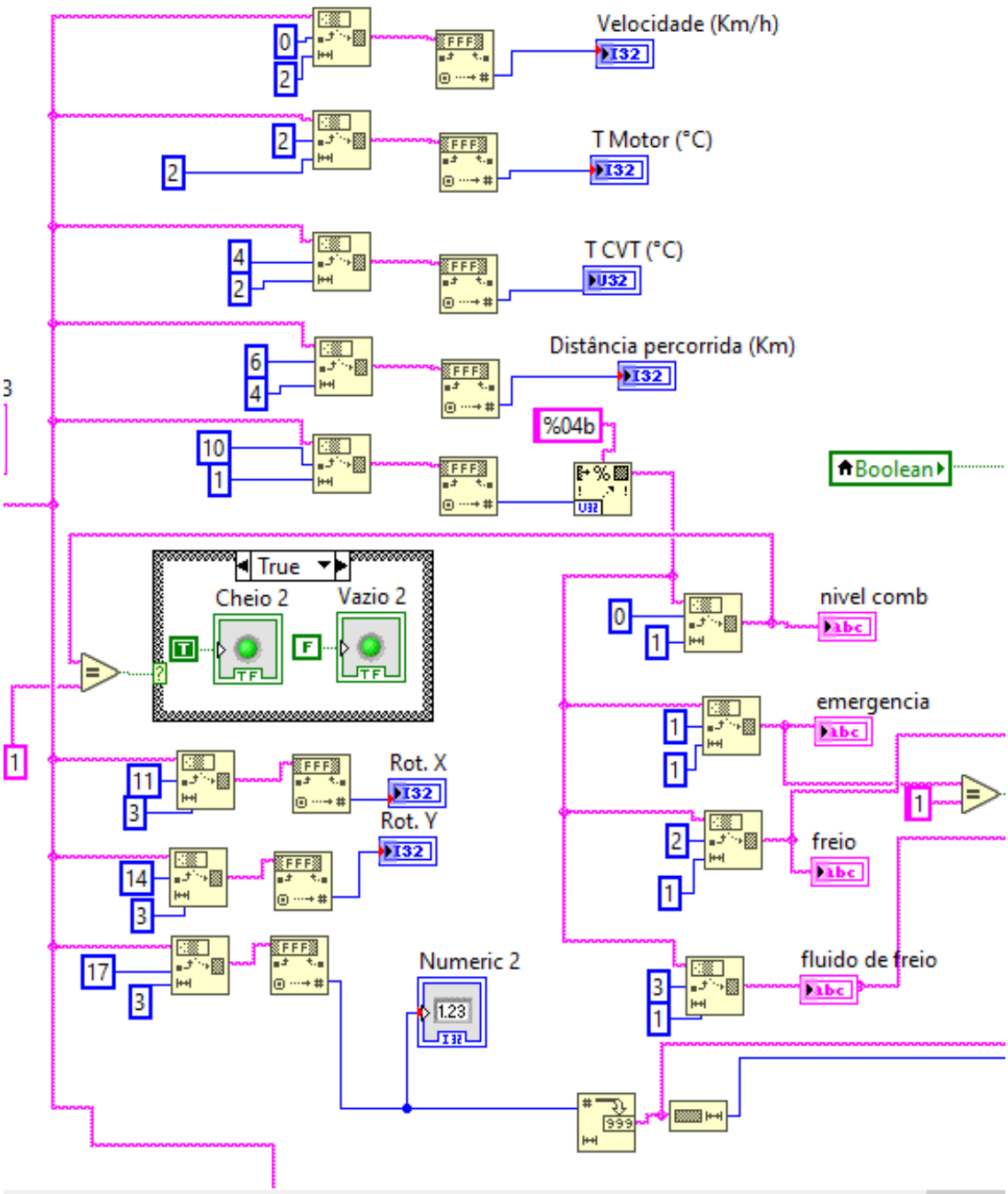


Figura 4.3 – Trecho do programa da interface da telemetria no qual os dados são convertidos e apresentados na tela principal (elaborada pelo autor).

Por fim, os dados podem ser armazenados em um arquivo de texto para uma posterior análise e também para geração dos relatórios que serão apresentados posteriormente. A gravação dos dados inicia-se com o *click* do botão *RECORD* que mudará para o status RECORDING sinalizando que os dados estão sendo gravados e finaliza com o *click* do botão *STOP*. A Figura 4.4

mostra o código de implementação para gravação dos dados. Um exemplo de arquivo de texto gerado encontra-se no Apêndice F.

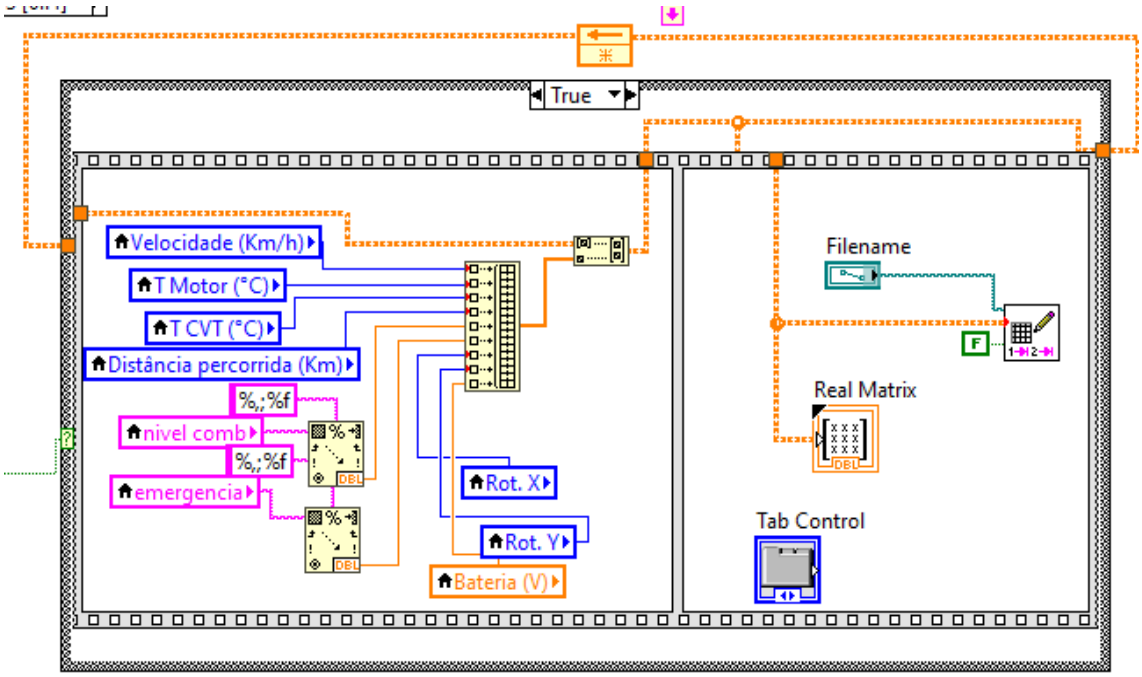


Figura 4.4 – Trecho do programa da interface da telemetria para gravação dos dados recebidos em um arquivo de texto (elaborada pelo autor).

4.2 Tela de relatórios

A Figura 4.5 apresenta a tela de relatórios da interface. Nesta tela em particular, apresentam-se os comportamentos de temperatura de motor e CVT, nível de bateria, inclinação do veículo e velocidade em formato gráfico, permitindo análises do comportamento do veículo durante um certo intervalo de tempo. A interface também conta com um campo para apresentar o número de vezes que o botão de emergência foi acionado.

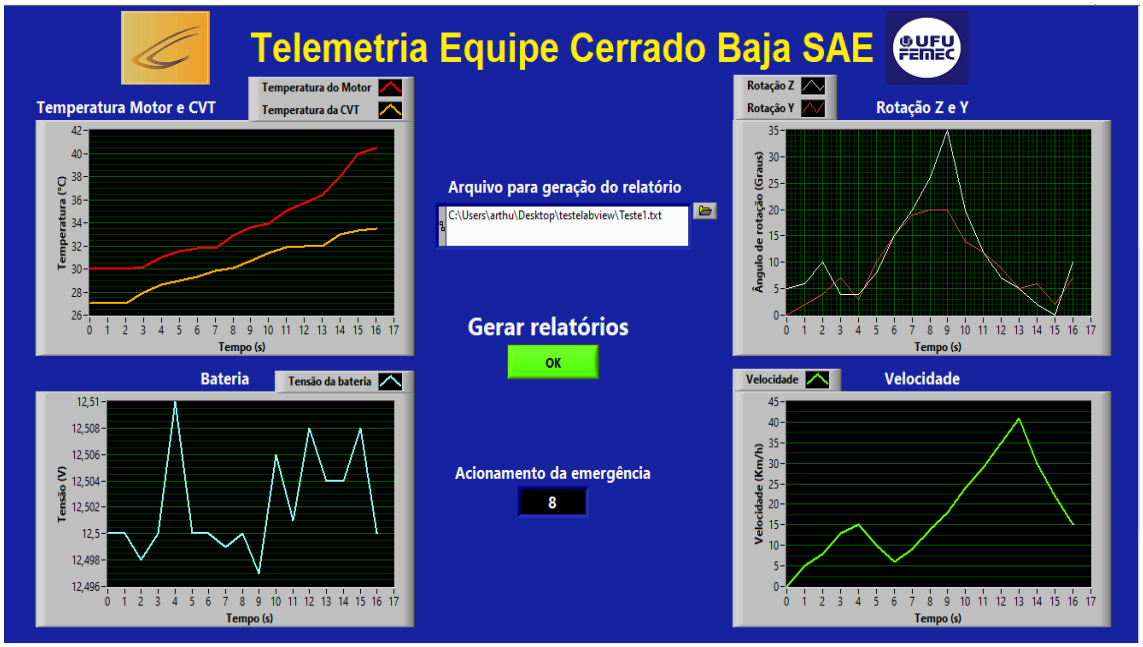


Figura 4.5 – *Layout* da tela de relatórios (elaborada pelo autor).

Para gerar os gráficos basta selecionar o caminho do arquivo de texto salvo e clicar no botão OK. Então, os gráficos serão gerados na tela. Os códigos para leitura do arquivo de texto e geração dos gráficos são apresentados nas Figuras 4.6 e 4.7.

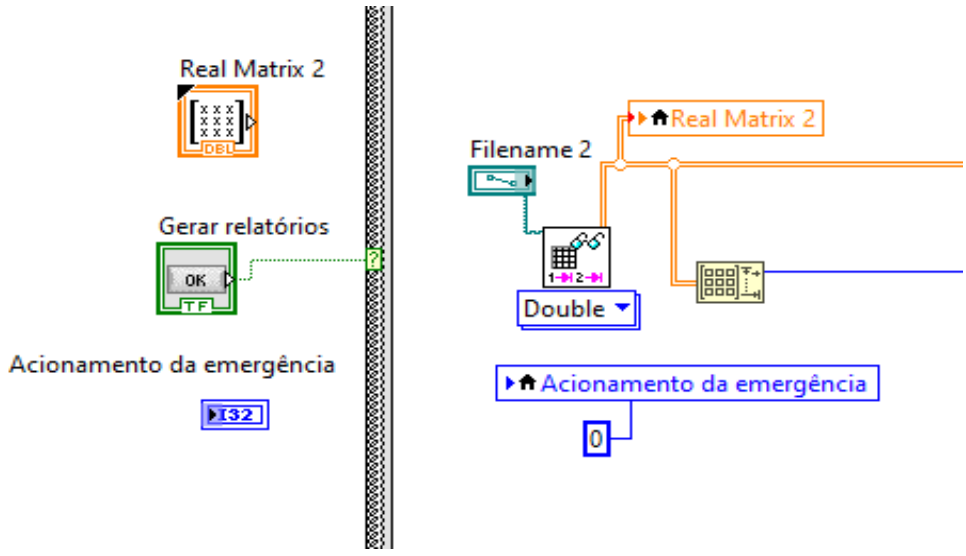


Figura 4.6 – Trecho do programa da interface da telemetria no qual os dados são lidos de um arquivo de texto e armazenados em uma matriz (elaborada pelo autor).

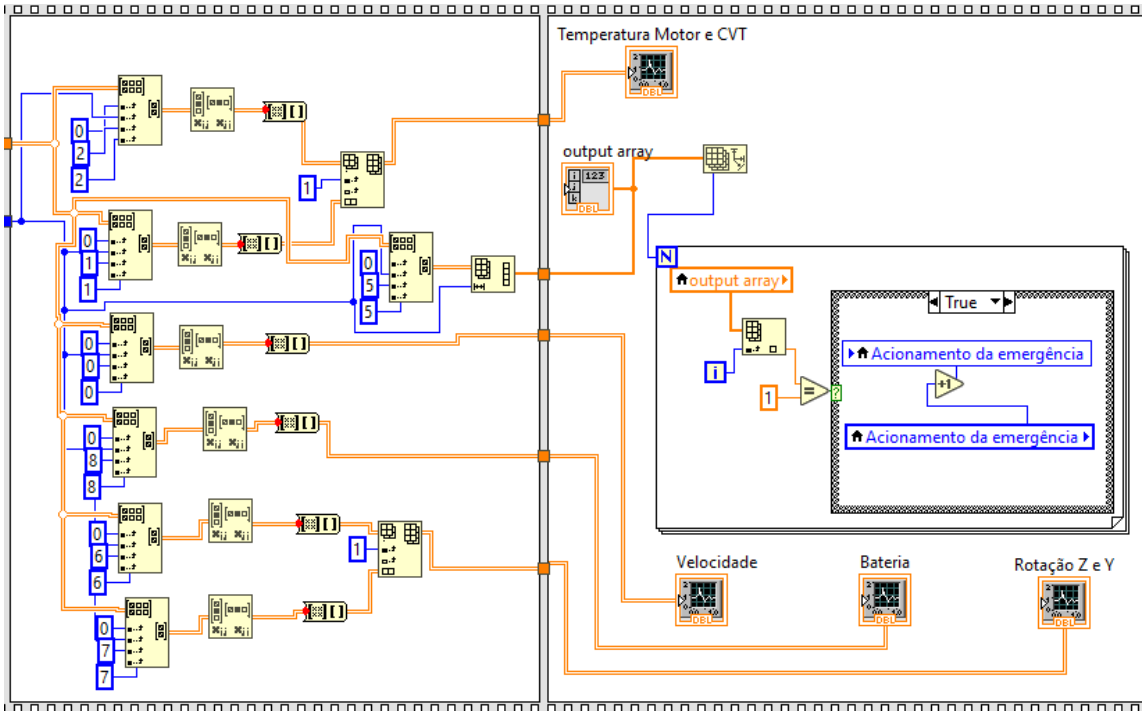


Figura 4.7 – Trecho do programa da interface da telemetria no qual os dados são extraídos da matriz e plotados nos gráficos (elaborada pelo autor).

4.3 Resultados

Com a finalidade de exemplificar o funcionamento do sistema desenvolvido, foi realizado um teste de 10 minutos com a telemetria em funcionamento no veículo. A Figura 4.8 apresenta o comportamento de algumas informações ao longo desse teste. Pode-se observar o bom funcionamento do sistema de telemetria permitindo avaliar o estado e o desempenho do veículo ao longo do teste.

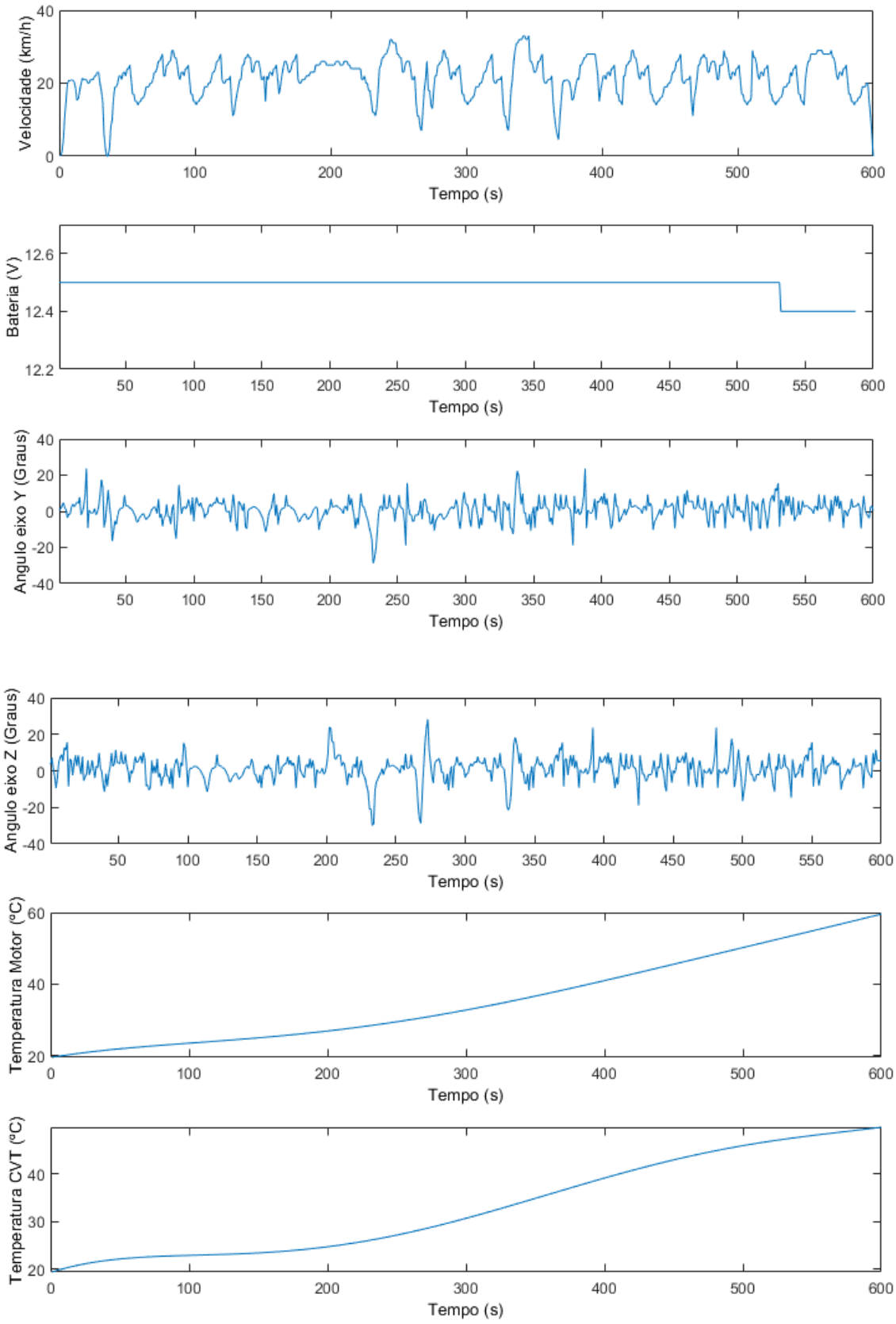


Figura 4.8 – Evolução temporal de velocidade, temperatura, inclinação, nível de bateria no teste realizado (elaborada pelo autor).

CAPÍTULO V

CONCLUSÃO

Este trabalho envolveu a implementação de telemetria no veículo da Equipe Cerrado Baja SAE. Para isso, primeiramente dois módulos de comunicação foram comparados por meio de testes de desempenho avaliando como requisitos o custo e o alcance máximo. Com efeito, escolheu-se a tecnologia LoRa para realização da telemetria, pois, apenas com esse módulo, foi possível cobrir a distância entre o *Box* e o ponto mais extremo da pista.

Então, projetou-se e construiu-se uma ECU, bem como circuitos eletrônicos e trechos de código para leitura de diferentes sensores. Em particular, mediram-se nível de combustível, velocidade, temperatura do motor e transmissão CVT, e rotação em torno dos eixos longitudinal e lateral do veículo. Mais ainda, um protocolo de comunicação foi criado e implementado. Por fim, desenvolveu-se uma interface para apresentação e o acompanhamento dos dados.

Como resultado do presente trabalho, a equipe pode detectar falhas ao longo das provas e acompanhar o desempenho do veículo por meio da interface ou de relatórios gerados automaticamente.

Trabalhos futuros implementar alguma técnica de criptografia na transmissão de dados entre o box e veículo. Isso impediria que informações confidenciais sobre o desempenho do veículo fossem acessadas por outras equipes.

REFERÊNCIAS

AI-THINKER, DataSheet: **Ra-02 LoRa Product Specification V1**. Shenzhen Ai-Thinker Technology Co., 2017. Disponível em: https://docs.ai-thinker.com/. Acesso em: 24 fev. 2021

ATMEL, DataSheet: **8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash**. Atmel pp.1-293, 2015. Disponível em: https://www.alldatasheet.com/. Acesso em: 24 fev. 2021.

AUGUSTIN, Aloÿs; YI, Jiazi; CLAUSEN, Thomas; TOWNSLEY, William. A Study of LoRa: long range & low power networks for the internet of things. **Sensors**, [S.L.], v. 16, n. 9, p. 1466, 9 set. 2016. MDPI AG. http://dx.doi.org/10.3390/s16091466.

COLTON, S. The Balance Filter. A Simple Solution for Integrating Accelerometer and Gyroscope Measurements for a Balancing Platform. **Submitted as Chief Delphi White Paper**. Jun. 2007. Disponível em: https://docplayer.net/57007441-The-balance-filter-a-simple-solution-for-integrating-accelerometer-and-gyroscope-measurements-for-a-balancing-platform.html. Acesso em: 24 fev. 2021.

DALLAS, DataSheet: **DS18B20 Programmable Resolution 1-Wire Digital Thermometer**. Dallas Semiconductor, pp. 1-27, 2001. Disponível em: https://www.alldatasheet.com/. Acesso em: 24 fev. 2021.

DIAS, Jullierme Emiliano Alves. **Eletrônica, Instrumentação e Telemetria do Veículo UFVBAJA**. 2010. 50 f. Monografia (Conclusão do Curso de Licenciatura) - Curso de Engenharia Elétrica, Departamento De Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas, Universidade Federal de Viçosa, Viçosa, 2010.

DIGI, International. DataSheet: **XBee/XBee-PRO Zigbee RF Modules**, User Guide, 2018. Disponível em: https://www.digi.com/resources/documentation/. Acesso em: 24 fev. 2021.

HIBBELER, Russell C. **Dinâmica: mecânica para engenharia**. 10. ed. São Paulo: Prentice Hall, 2005. 2 v. Tradutor técnico Mário Alberto Tenan.

INVENSENSE, DataSheet: **MPU-6000 and MPU-6050 Product Specification Revision 3.4.** InvenSense Inc, pp. 1-52, 2013. Disponível em: https://www.alldatasheet.com/. Acesso em: 24 fev. 2021.

KUROSE, James F; ROSS, Keith W. **Redes de computadores e a internet:** uma abordagem top-down. 3. ed. São Paulo: Pearson, 2006.

LORAWAN. **LoRaWAN 1.1 Specification**, 2020. Disponível em: https://lora-alliance.org/resource-hub/lorawanr-specification-v11. Acesso em: 22 abr. 2020.

MAX STREAM, DataSheet: **XBee™ Series 2 OEM RF Modules**. Digi International Inc, pp. 4-6, 2007. Disponível em: https://www.datasheetarchive.com/. Acesso em: 24 fev. 2021.

METALTEX, DataSheet: **Sensores capacitivos cilíndricos, linha C**. Metalflex, pp. 1, 2020. Disponível em: https://www.metaltex.com.br/. Acesso em: 24 fev. 2021.

MOREIRA, Adriano J. C. **Alguns aspectos que condicionam o desempenho dos sistemas de transmissão**. 1999. Disponível em: http://www3.dsi.uminho.pt/adriano/Teaching/Comum/FactDegrad.html. Acesso em: 24 fev. 2021.

MOURA, Rafael Santos. **Desenvolvimento de um sistema de orientação espacial inercial**. 2013. 150 f. TCC (Graduação) - Curso de Engenharia Elétrica Com Ênfase em Eletrônica, Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2013.

NILSSON, James W; RIEDEL, Susan A. **Circuitos elétricos**. 10. ed. São Paulo: Pearson Education do Brasil, 2015.

NOGUEIRA, Danilo. **Conhecendo resistor pull up-down**: funcionamento resistores pull-up e pull-down. Funcionamento resistores pull-up e pull-down. 2018. Disponível em: https://autocorerobotica.blog.br/resistores-pull-up-e-pull-down-como-funcionam/. Acesso em: 24 fev. 2021.

NUNES, Tomaz Filgueira. **Telemetria de um veículo Baja SAE através de rede CAN**. 2016. 48 f. TCC (Graduação) - Curso de Engenharia Mecatrônica, Universidade Federal do Rio Grande do Norte, Natal, 2016.

PETERSON, Larry L; DAVIE, Bruce s. **Redes de computadores**: uma abordagem de sistemas. Uma Abordagem de Sistemas. 1. Ed. Rio de Janeiro: Elsevier, 2004.

QUEIRÓS, João Miguel Ribeiro. **Sistema de Sensorização e Telemetria de um VEC (Veículo Eléctrico de Competição)**. 2011. 117 f. Dissertação (Mestrado) - Curso de Mestrado Integrado em Engenharia Electrotécnica e de Computadores Major Automação, Faculdade de Engenharia da Universidade do Porto, Cidade do Porto, 2011.

SAHAWNEH, L.; JARRAH, M.A. Development and calibration of low cost MEMS IMU for UAV applications. In: **5th International Symposium on Mechatronics and Its Applications** (ISMA), 2008, Amman, Jordan: IEEE, 2008.

TANENBAUM, Andrew s. **Redes de computadores**. 1. Ed. Rio de Janeiro: Elsevier, 2003.

TEIXEIRA, Fernando César Rodrigues; OLIVEIRA, Maria Celia de; HELLENO, André Luís. Telemetria Automotiva via Internet Móvel. **Revista Ciência e Tecnologia**, [S.l.], v. 16, n. 28/29, fev. 2014. ISSN 2236-6733. Disponível em: http://www.revista.unisal.br/sj/index.php/123/article/view/264. Acesso em: 24 fev. 2021.

APÊNDICE A – IMPLEMENTAÇÃO DO CRC

A implementação do CRC foi realizada na plataforma IDE Arduino. Tal implementação é mostrada abaixo. Considere a variável *Data* como a mensagem a ser transmitida contendo informações de diferentes sensores como mostra a Figura A.1.

```
String Gerador = "11000000000000101";

Data = Vel + Temp_1 + Temp_2 + Dist + ComEmer + Rot_X + Rot_Y + Bate;

Data.concat(calcularResto(Data, Gerador));

Data = binToHex(Data);

// send packet
LoRa.beginPacket();
LoRa.print(Data);
LoRa.endPacket();

delay(1000);
```

Figura A.1 – Implementação do código para o cálculo do CRC no remetente (elaborada pelo autor).

Definiu-se o seguinte polinômio gerador: $G_{(x)} = x^{16} + x^{15} + x^2 + x^0$. Com a mensagem montada, deve-se agora calcular o CRC, para isso, concatena-se na variável *Data* o retorno da função *calcularResto().* A função *calcularResto()* recebe como parâmetros os dados a serem enviados e o gerador, realizando assim o cálculo do CRC como mostra a Figura A.2. No comando *for* calcula-se o grau do polinômio gerador e acrescenta-se esta quantidade de bits nulos ao final dos dados. O código dentro do comando *while* é responsável por realizar a divisão entre os dados e o gerador e assim retornar o valor do CRC.

```
String calcularResto (String Data, String Gerador) {
  String aux;

  for (int j=0; j<(Gerador.length()-1); j++)
 {
  Data = Data + "0";
 }

  while (Data.length() >= Gerador.length()) {
    if (Data.charAt(0) == '1') {
      for (int i = 0; i < Gerador.length(); i++) {
        if (Data.charAt(i) == Gerador.charAt(i)) {
          aux = Data.substring(i+1);
          Data =Data.substring(0, i) + "0" + aux;
        } else {
          aux = Data.substring(i+1);
          Data = Data.substring(0, i)+ "1" + aux;
        }
      }
    } else {
      Data.remove(0, 1);
    }
  }

  return Data;
}
```

Figura A.2 – Implementação da função calcularResto() (elaborada pelo autor).

Com o código CRC concatenado ao final da mensagem, converte-se em um número hexadecimal e o envio dos dados é realizado a cada 1 segundo.

No receptor implementa-se um código que verificará a integridade e um contador que será acrescentado quando uma mensagem apresentar erro.

```
int packetSize = LoRa.parsePacket();
if (packetSize) {
  // read packet
  while (LoRa.available()) {
    Data =(String)LoRa.read();
  }
  Data = hexToBin(Data);
  Resto = calcularResto(Data, Gerador);
  RestoInt =  binToDec (Resto);

if (RestoInt != 0){
  contador ++;
  }
```

Figura A.3 - Implementação do código para o cálculo do CRC no destinatário (elaborada pelo autor).

O código apresentado na Figura A.3 fica avaliando o canal e ao receber uma mensagem a coloca na variável Data. A informação recebida se encontra no formato Hexadecimal, portanto a convertemos em um sequência binaria com a função *hexToBin()*.

Em seguida chama-se a função *calcularResto()* para efetuar a divisão da mensagem recebida pelo gerador e retornar o resto desta divisão. Esta função é a mesma utilizada no código do remetente apresentado na Figura A.2, eliminando somente a parte da concatenação de bits nulos ao final da mensagem.

Agora a variável *Resto* contém uma sequência de bits que são convertidos em um número decimal e seu valor armazenado na variável *RestoInt,* que posteriormente é utilizada na condição do comando *if.* Caso a variável RestoInt for diferente do número inteiro zero, significa que a mensagem está corrompida e, portanto, o contador é acrescido de 1, caso contrário a informação está correta e seguirá para posteriores análises.

APÊNDICE B – CONFIGURAÇÃO DO MÓDULO LORA

Na programação do LoRa foi utilizada a biblioteca LoRa.h, que fornece funcionabilidades específicas auxiliando o desenvolvimento. Em particular, tal biblioteca é empregada para configurar os parâmetros de operação do módulo LoRa, enviar e receber o pacote e verificar o número de bytes disponíveis para leitura.

Para o funcionamento do LoRa, inicialmente devem ser escolhidas a potência (em dB) e a frequência (em Hz) do sinal de transmissão. Essa configuração é realizada por meio do código da Figura B.1. Neste trabalho, adotaram-se uma potência de 20 dB e uma frequência de 433 MHz para obtenção de um bom alcance de transmissão mesmo na presença de obstáculos como arvores, montanhas e veículos.

```
if (!LoRa.begin(915E6)) {
  Serial.println("Starting LoRa failed!");
  while (1);
}
LoRa.setTxPower(20);
LoRa.setFrequency(433E6);
```

Figura B.1 – Trecho do código de configuração do módulo LoRa (elaborada pelo autor).

Terminada a configuração dos módulos, pode-se iniciar a transmissão de dados. Para enviar o pacote "*data*", pode-se utilizar os comandos *LoRa.beginPacket(), LoRa.print(data) e LoRa.endPacket(),* como ilustrado na Figura B.2.

```
LoRa.beginPacket();
LoRa.print(data);
LoRa.endPacket();
```

Figura B.2 – Trecho do código para envio de um pacote de dados (elaborada pelo autor).

No receptor, verifica-se a existência de dados a serem recebidos (*LoRa.available()*). Caso existam, tais dados são lidos e armazenados em uma variável. Esse procedimento é mostrado na Figura B.3.

```
// read packet
while (LoRa.available()) {
  Data =(char)LoRa.read();
  Serial.print(Data);
}
```

Figura B.3 – Trecho do código onde ocorre o recebimento do pacote de dados (elaborada pelo autor).

Os códigos completos do transmissor e do receptor estão disponíveis no Apêndice C.4 e C.5 respectivamente.

APÊNDICE C – Códigos para leitura dos sensores

C.1 - Código de implementação do sensor capacitivo e indutivo

```
const int pinoSinal = 8; //PINO DIGITAL UTILIZADO PELO SENSOR
int leitura=HIGH; // sensor é ativado quando recebe sinal igual a 0(LOW)
void setup(){
  pinMode(pinoSinal, INPUT); //DEFINE O PINO COMO ENTRADA
}
void loop(){

  leitura = digitalRead(pinoSinal); // REALIZA A LEITURA DO PINO 8

  Serial.println(leitura); //ESCREVE AS INFORMAÇÕES NA SERIAL

}
```

C.2 - Código de implementação do sensor de temperatura

```
#include <OneWire.h>
#include <DallasTemperature.h>

OneWire pino(3);
DallasTemperature barramento(&pino);
DeviceAddress sensor;

void setup(void)
{
  Serial.begin(9600);
  barramento.begin();
  barramento.getAddress(sensor, 0);
}

void loop()
{
  barramento.requestTemperatures();
  float temperatura = barramento.getTempC(sensor);
}
```

C.3 - Código de implementação do MPU-6050

```
#include <MPU6050.h>
#include <Wire.h>
#include <I2Cdev.h>

MPU6050 accelgyro;

int16_t ax, ay, az;
int16_t gx, gy, gz;

#define G_GAIN 0.00875
#define AA 0.98

float acelx, acely, acelz, rate_gyr_x, rate_gyr_y, rate_gyr_z, gyroXangle, gyroYangle, gyroZangle;
float AccXangle, AccYangle, AccZangle, CFangleX, CFangleY, CFangleZ;
float const_calib = 16071.82;
float const_gravid = 9.81;

unsigned long pT;

void setup() {
Wire.begin(); // Inicia barramento I2C

// initializa conexão serial
Serial.begin(9600);
Serial.println("Initializing I2C devices...");

// verifica conexão com sensores
Serial.println("Testing device connections...");
Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection failed");

unsigned long pT = 0; // contador para determinar tempo de inicialização
}

void loop() {
unsigned long cT = micros(); // contar tempo de loop

accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); // obtem valores brutos dos sensores

unsigned long dT = cT - pT;
pT = cT;

acelx = ax * const_gravid / const_calib;
acely = ay * const_gravid / const_calib;
acelz = az * const_gravid / const_calib;

// Converte valor do acelerometro com base nos 3 eixos
AccXangle = (atan2(ax, sqrt(pow(ay,2) + pow(az,2)))*180) / 3.14;
AccYangle = (atan2(ay, sqrt(pow(ax,2) + pow(az,2)))*180) / 3.14;
AccZangle = (atan2(az, sqrt(pow(ax,2) + pow(ay,2)))*180) / 3.14;

// Converte valor do giro em graus por seg
// multiplicando uma contante relacionada à taxa de amostragem do sensor
// nesse caso, a taxa é +-250g -> 0.00875
rate_gyr_x = gx*G_GAIN;
rate_gyr_y = gy*G_GAIN;
rate_gyr_z = gz*G_GAIN;
```

```
// Calcula a distância percorrida por integração simples
// com base no tempo de loop (dT = cT – pT)
gyroXangle+=rate_gyr_x*dT;
gyroYangle+=rate_gyr_y*dT;
gyroZangle+=rate_gyr_z*dT;

// Fusão dos dados: giro + accel
// Métodos: filtro complementar
// Atribui peso de 0.98 ao valor do giro e 0.02 ao acelerometro

CFangleX=AA*(CFangleX+rate_gyr_x*(dT/1000000)) +(1 – AA) * AccXangle;
CFangleY=AA*(CFangleY+rate_gyr_y*(dT/1000000)) +(1 – AA) * AccYangle;
CFangleZ=AA*(CFangleZ+rate_gyr_z*(dT/1000000)) +(1 – AA) * AccZangle;
}
```

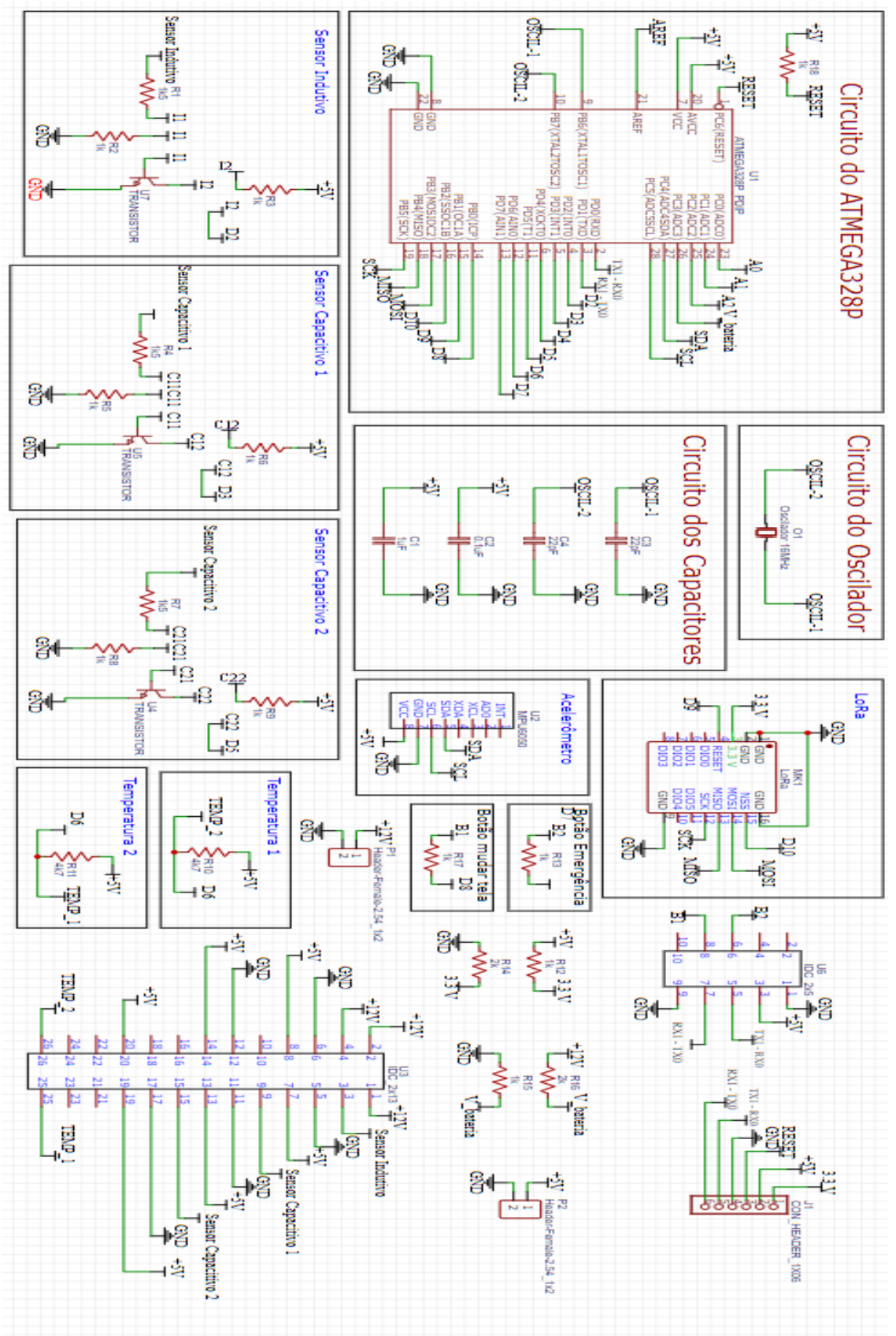
C.4 - Código de implementação do remetente (LoRa)

```
#include <SPI.h>
#include <LoRa.h>
int vel = 25;
int temp_1 = 50;
int temp_2 = 90;
int dist = 20110;
int emg = 3;
int rx = 2584;
int ry = 602;
int bate = 1050;
char CR;
String CHAVE = "11000000000000101";
String TEM_1, TEM_2, BAT, VE, DIS, EMGE, AT, crc, ROX, ROY, CRC  = "";

void setup () {
 Serial.begin(9600);
 Serial.println("LoRa Sender");
  if (!LoRa.begin(915E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  LoRa.setTxPower(20);
  LoRa.setFrequency(433E6);
}

void loop () {

  VE =  String(vel, BIN);
  TEM_1 =  String(temp_1, BIN);
  TEM_2 =  String(temp_2, BIN);
  DIS = String(dist, BIN);
  EMGE = String(emg, BIN);
  ROX = String(rx, BIN);
  ROY = String(ry, BIN);
  BAT = String(bate, BIN);

  VE = completar(VE, 8);
  TEM_1 = completar(TEM_1, 8);
  TEM_2 = completar(TEM_2, 8);
  DIS = completar(DIS,16);
  EMGE = completar(EMGE,4);
  ROX = completar(ROX,12);
  ROY = completar(ROY,12);
  BAT = completar(BAT,12);

  VE = binToHex(VE);
  TEM_1 = binToHex(TEM_1);
  TEM_2 = binToHex(TEM_2);
  DIS = binToHex(DIS);
  EMGE = binToHex(EMGE);
  ROX = binToHex(ROX);
  ROY = binToHex(ROY);
  BAT = binToHex(BAT);

  AT = VE + TEM_1 + TEM_2 + DIS + EMGE + ROX + ROY + BAT;
  AT.concat(calcularResto(AT, CHAVE));
  AT = binToHex(AT);

  // send packet
  LoRa.beginPacket();
  LoRa.print(AT);
  LoRa.endPacket();
  delay(1000);
}
```

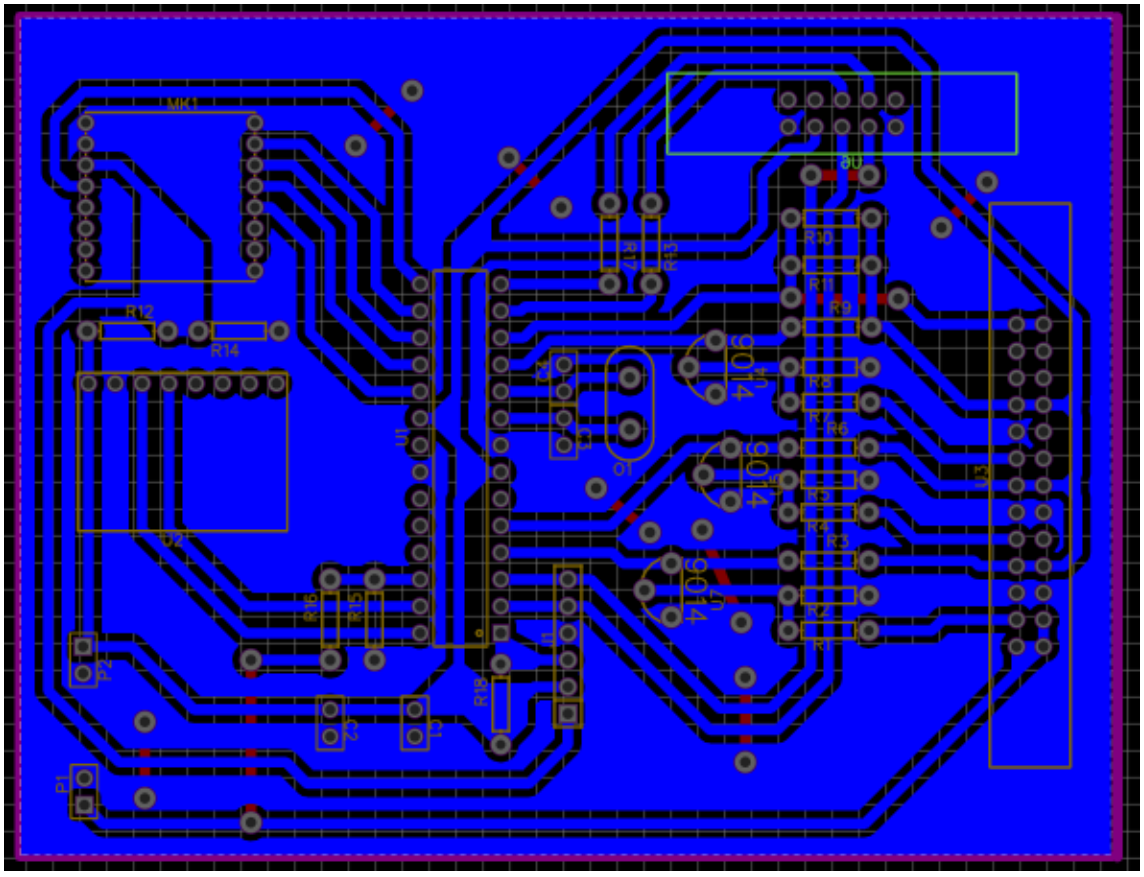
C.5 – Código de implementação do destinatário (LoRa)

```
#include <SPI.h>
#include <LoRa.h>
char Data;

void setup()
{
 Serial.begin(9600);
  while (!Serial);

  Serial.println("LoRa Sender");

  if (!LoRa.begin(915E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  LoRa.setTxPower(20);
  LoRa.setFrequency(433E6);
}

void loop()
{
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    // received a packet
    Serial.print("Received packet '");

    // read packet
    while (LoRa.available()) {
      Data =(char)LoRa.read();
      Serial.print(Data);
    }
  }
}
```

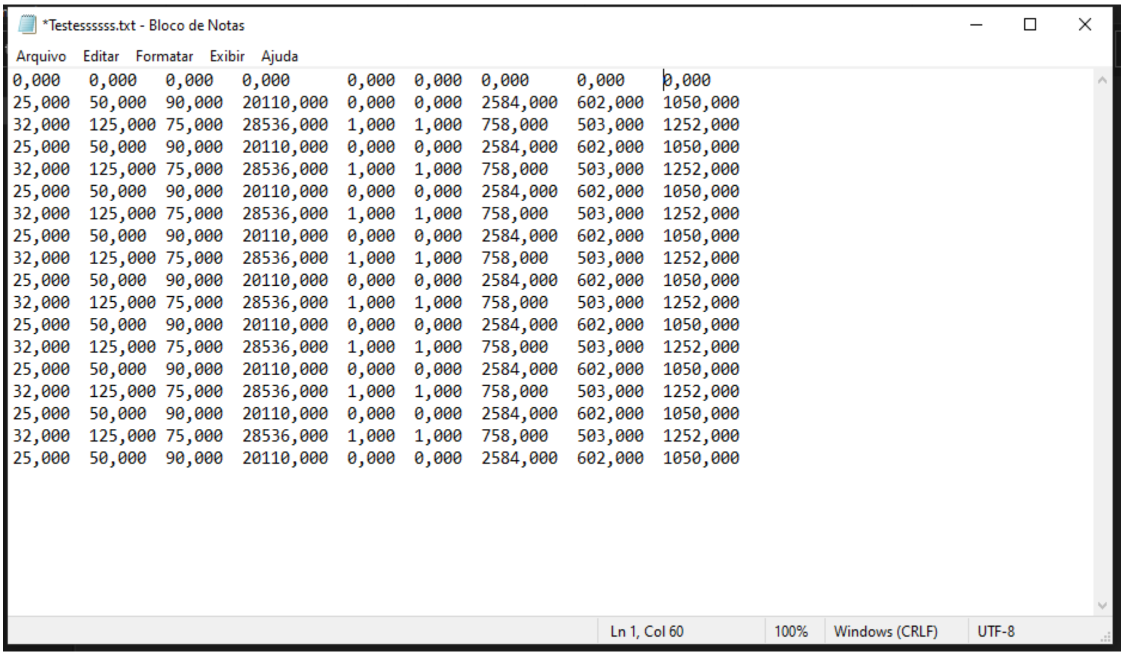
APÊNDICE D - Esquemático do circuito da placa ECU



APÊNDICE E – PCB do circuito da placa ECU



APÊNDICE F – Exemplo de um arquivo de texto gerado



0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000
32,000	125,000	75,000	28536,000	1,000	1,000	758,000	503,000	1252,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000
32,000	125,000	75,000	28536,000	1,000	1,000	758,000	503,000	1252,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000
32,000	125,000	75,000	28536,000	1,000	1,000	758,000	503,000	1252,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000
32,000	125,000	75,000	28536,000	1,000	1,000	758,000	503,000	1252,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000
32,000	125,000	75,000	28536,000	1,000	1,000	758,000	503,000	1252,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000
32,000	125,000	75,000	28536,000	1,000	1,000	758,000	503,000	1252,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000
32,000	125,000	75,000	28536,000	1,000	1,000	758,000	503,000	1252,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000
32,000	125,000	75,000	28536,000	1,000	1,000	758,000	503,000	1252,000
25,000	50,000	90,000	20110,000	0,000	0,000	2584,000	602,000	1050,000