FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Plataforma de ensaio para controladores de veículos em platooning

Pedro Daniel de Oliveira Neves Gomes da Silva

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Engº Rui Esteves Araújo (PhD)

Co-orientador: Engº António Manuel Figueiredo Lopes (Msc)

30 de Outubro de 2020

Resumo

Esta dissertação documenta o desenvolvimento de uma plataforma para ensaio de algoritmos de controlo de veículos em pelotão. A plataforma tem como objetivo o estudo de condução em pelotão, especialmente controlo longitudinal, para tal foi dotada de sensores e atuadores relevantes para este controlo, tendo sido para isso levantado outras plataformas já desenvolvidas e as suas despectivas valências.

O microcontrolador empregue, ESP32-WROVER-B da Espressif, é um chip de baixo custo com um grande leque de periféricos, incluindo WLAN 2.4GHz e Bluetooth Low Energy (BLE), bem como dois núcleos de processamento capazes de 600 MIPS e com um SDK do fabricante assente num *kernel* freeRTOS. Existe também uma framework para a plataforma Arduino, suportada pelo fabricante, que torna o código mais legível e de fácil utilização.

Para a comunicação entre os veículos foi usado o protocolo ESP-NOW da Espressif de forma a diminuir o atraso desta mesma. O motor é alimentado pela drive DRV8828, uma drive de corrente constante da Texas Instruments, de forma a ser possível controlar o motor em modo de binário.

O software produzido encontra-se dividido num exemplo de controlo direto pelo transmissor R/C e em bibliotecas que lidam com o acesso ao hardware.

Abstract

This dissertation documents the development of a vehicle platooning controller test platform. The platform aims to study platooning control, specifically longitudinal control, for this it is equipped with relevant sensors and actuators for this control, having studied other platforms already developed and their respective assets.

The microcontroller used, Espressif's ESP32-WROVER-B, it's a low-cost chip with a large number of peripherals, including WLAN 2.4 GHz and Bluetooth Low Energy (BLE), as well as two processing cores capable of 600 MIPS and a manufacture's SDK based on a freeRTOS kernel. The is an Arduino's platform framework available, supported by the manufacture, that improves the code readability and use.

For communication between vehicles it was used Espressif's ESP-NOW protocol to allow a reduction of the delay. The motor is powered by the driver DRV8828, a Texas Instrument's constant current driver, to allow control of the motor's torque.

The software created is split in an example of R/C transmitter direct control and libraries to handle hardware access.

Agradecimentos

Mais do que por mero formalismo protocolar quero aqui expressar o meu agradecimento aos meus pais que me proporcionaram a vida e condições morais e materiais, bem como o apoio prestador pela minha irmã, para a minha formação humana e profissional e no meu percurso académico de uma forma geral aos muitos e ilustres professores que me transmitiram sólidos e consistentes conhecimentos das matérias que lecionavam.

Neste conjunto destaco, por merecido, o meu especial agradecimento aos excelentíssimos Engenheiros António Manuel Figueiredo Lopes e Rui Esteves Araújo pelo incansável apoio prestado no decorrer desta dissertação

Pedro Silva

"Your test equipment is lying to you
and it is your job to figure out how."

Charles Rush

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ACC	Adaptative Cruise Control
BLE	Bluetooth Low Energy
CACC	Colaborative Adapatative Cruise Control
FIFO	First In First Out
IBUS	FlySky Serial Protocol for receivers
IC	Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IOT	Internet of things
MIPS	Million of Instructions per Second
MPU	Micro Processing Unit
NTP	Network Time Protocol
PPR	Pulsos Por Revolução
QFN	Quad-Flat No-leads
R/C	Rádio Controlo
RTT	Round Trip Time
RPI3	Rapsberry Pi 3
SDK	Software Development Kit
UART	Universal asynchronous receiver/transmitter
USB	Universal Serial Bus
V2V	Vehicle-to-vehicle
WLAN	Wireless Local Area Network

Capítulo 1

Introdução

Com a disponibilidade das redes celulares 5G é expectável que num futuro próximo o conceito de platooning de veículos venha a ter mais cenários de aplicação, incluindo a mobilidade urbana. No passado, o conceito base desta técnica de circulação tem sido adotado em contexto de comboios de camiões de mercadorias, onde o primeiro se traduz como sendo o veículo líder, formando o resto dos veículos uma fila com proximidade reduzida entre eles.

O objetivo desta formação é reduzir simultaneamente o arrasto aerodinâmico nos veículos seguidores, bem como reduzir o congestionamento de transito [2], [3]. Em termos sintéticos, o principal objetivo desta dissertação é o projeto, desenvolvimento e teste de uma plataforma de ensaio de algoritmos de controlo de platooning, em que cada veículo, de um modo individual, observa o ambiente exterior e com base nas comunicações segue o veículo anterior de forma autónoma e estável.

1.1 Contexto

O conceito de condução em pelotão baseia-se no agrupamento próximo de veículos, deixando o mínimo espaço possível entre estes. Este conceito visa um aumento de eficiência no transporte rodoviário. Estes benefícios vêm da redução da quantidade energética necessária[4], diminuição do congestionamento das estradas [5], bem como a diminuição de acidentes, podendo 90% ser atribuídos a erro humano[6].

As técnicas de controlo de pelotão para veículos podem ser divididas em duas categorias ACC, Adaptative Cruise Control e CACC Colaborative Adapatative Cruise Control[7].

Na técnica ACC cada veículo é instrumentado e de forma automática consegue manter uma distância fixa ao veículo imediatamente à sua frente. Esta técnica já é implementada comercialmente em viaturas não como um controlador de pelotão, mas como um auxiliar ao condutor. Devido à quantidade de informação disponível ao controlador estar limitada ao horizonte do veículo, e ao atraso cumulativo na perceção de obstáculos, esta técnica exige uma distância de espaçamento

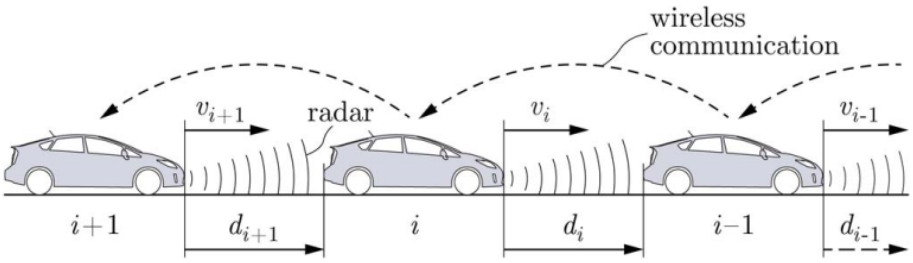

Figura 1.1: Veículos em platooning [1].

maior[8]. Por exemplo num pelotão de 3 veículos o primeiro veículo trava, o segundo vai demorar algum tempo a perceber esta ação e o terceiro só vai perceber depois do segundo responder e depois do seu próprio atraso de perceção.

A técnica CACC melhora esta última introduzindo comunicações entre os veículos. Assim todos os veículos no sistema recebem a informação de todos os elementos da cadeia ao mesmo tempo.

Um estudo realizado por VanderWerf, et al., simulou um troço de autoestrada de 16km com uma faixa e rampas de acesso a cada 1,6km concluiu que o número de veículos por faixa por hora duplica usando a técnica de CACC versus condução manual [9].

1.2 Objectivos

O objetivo principal do trabalho realizado consiste no desenvolvimento de uma plataforma capaz de testar algoritmos de controlo de veículos em pelotão simulando, no maior grau possível, um veículo real.

Com o objetivo de se produzir uma plataforma de custo reduzido e de fácil reprodução, esta deve cumprir os seguintes requisitos:

- Controlo do motor por binário - intervalos inferiores a 5%;
- Medição da distância ao veículo anterior - Gama de 1-2 m e resolução de 1 mm;
- Comunicação entre veículos - latência inferior a 10ms;
- Gravação de registos em cartão SD - Sem interferência na execução;
- Medição da aceleração do veículo - freqência de medicão < 20ms;
- Medição da velocidade do veículo;
- Autonomia de 30min;

1.3 Estrutura da dissertação

A presente dissertação está dividida em 6 capítulos, sendo o presente uma contextualização do interesse desta plataforma de desenvolvimento, no segundo capítulo apresentam-se soluções já existentes e as suas valências bem como um levantamento de plataformas para construção da solução encontrada, no terceiro capítulo apresenta-se os componentes da solução proposta bem como pormenores da sua utilização, no quarto capítulo apresenta-se a implementação de software usada na plataforma, no quinto capítulo apresenta-se a construção e implementação da plataforma e no sexto e último capítulo, discutem-se conclusões e trabalhos futuros.

Capítulo 2

Caracterização do problema

2.1 Definição do problema

O objetivo central da dissertação constitui o projeto e construção de veículos que possam operar em modo de pelotão. Por outras palavras, o problema a tratar consiste no desenvolvimento de uma plataforma de platooning para ensaio de algoritmos de controlo.

2.2 Sistematização de soluções existentes

O trabalho de preparação da dissertação consistiu numa primeira fase na pesquisa de soluções existentes no mercado.

A figura 2.1 sistematiza as plataformas existentes atendendo ao sistema de tração e direção.

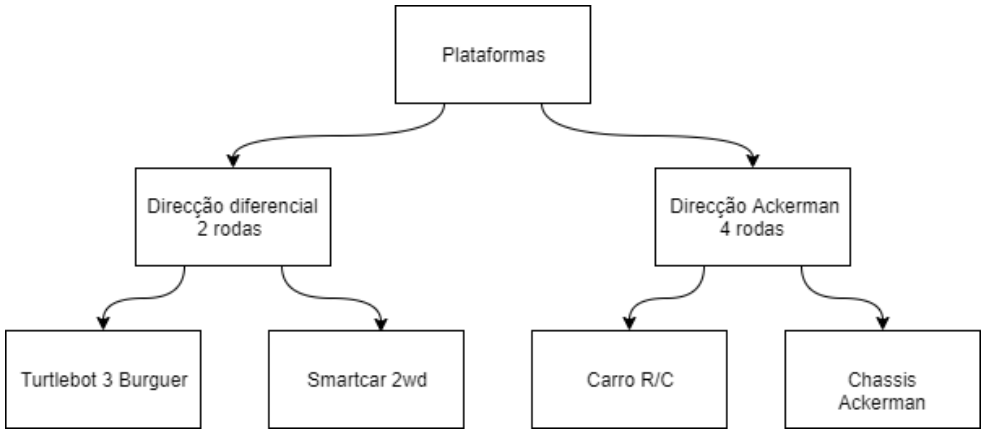

Figura 2.1: Grupos de plataformas consideradas.

Numa primeira fase foram estudadas soluções já desenvolvidas por outros investigadores, tendo-se chegado à conclusão que estas têm como âmbito o controlo autónomo do veículo, incluindo para isso lidar e câmara estéreo, bem com plataformas de processamento computacional

Figura 2.2: Plataformas existentes: a) MIT racecar; b) F1TENTH; c) HyphaRos Mini.

adequado ao tratamento de imagem. Correspondem a soluções caras, obrigando a processos administrativos demorados e que reduzem o espaço de aprendizagem do saber fazer.

Na figura 2.2 encontram-se as principais plataformas encontradas na literatura da especialidade. Nas sub-seções seguintes serão apresentadas de forma sintética cada um dos veículos.

2.2.1 MIT racecar

Plataforma desenvolvida pelo MIT [10] para ensaio de algoritmos de navegação autónoma. Equipada com lidar 2D e câmara estéreo, a necessidade de poder computacional é bastante elevada sendo para isso usado uma placa Nvidia Jetson TX2. Esta solução é bastante completa, mas para o problema proposto é demasiado cara, principalmente devido à necessidade de haver pelo menos 2 veículos.

2.2.2 F1TENTH

Plataforma desenvolvida dela Universidade de Pensilvânia [11] com o intuito de iniciar uma liga de competição em condução autónoma, uniformizando o hardware entre os concorrentes. Tal como a plataforma do MIT é uma solução muito completa, mas igualmente cara.

2.2.3 HyphaRos Mini

Plataforma desenvolvida pela empresa chinesa Virtuoso Robotics [12] como plataforma económica para a introdução ao middlware ROS e a soluções de condução autónoma.

2.3 Plataformas consideradas

Baseado no trabalho de Braescu [13] para os requisitos necessários foram selecionadas plataformas de locomoção que permitissem a realização de um veículo de baixo custo para o ensaio de algoritmos de controlo.

Na figura 2.3 apresentam-se um conjunto de plataformas de custo mais reduzido que foram consideradas no processo de seleção.

Em seguida discute-se de forma breve as principais características de cada um das plataformas.

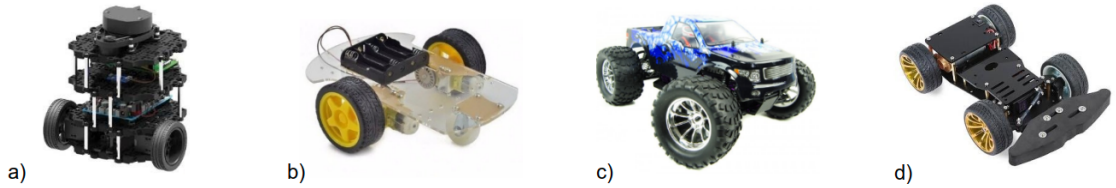

Figura 2.3: Plataformas consideradas: a) Turtle-bot 3 burguer; b) Smart-car 2wd; c) Carro R/C; d) Chassis Ackerman.

2.3.1 Turtlebot 3 burguer

Projeto criado em 2010 por Melonee Wise e Tully Foote [14] com o objetivo de produzir uma plataforma robótica de baixo custo com software open-source desenvolvimento de aplicações robóticas móveis. A propulsão é obtida com recurso a servos industriais Dynamixel XL430-W250. O kit inclui a plataforma móvel, bateria, Rpi 3, lidar e mcu OpenCR1.0.

2.3.2 Smartcar 2wd

Plataforma de baixo custo para introdução a sistemas robóticos, movida por motores DC com caixa de redução. Sistema muito redutor em termos de expansão futura e com alta limitação de performance de funcionamento

2.3.3 Carro R/C

Adaptação de um carro telecomandado para uma plataforma robótica, incluindo odometria e controlo em malha fechada da velocidade do veículo. Kit inclui transmissor e recetor 2 canais 2.4ghz, servo de direção, motor, esc e bateria. Sistema possível de modificar para os requisitos pretendidos, mas com dificuldade acrescida de acoplamento do encoder às rodas.

2.3.4 Chassis Ackerman

Kit inclui motor DC com caixa de redução e servo de direção. Maior flexibilidade podendo escolher os componentes que se requer para o sistema. Sistema de direção usado em veículos reais, tornando a plataforma mais interessante. O motor usado nesta solução já inclui encoder.

Tabela 2.1: Comparação das plataformas consideradas.

	Turtlebot3 burger	Smartcar 2wd	Carro rc 1/10	Chassis Ackerman
Preço base	579,00 €	13,00 €	126,00 €	63,25 €
Largura (mm)	178	150	300	185
Comprimento (mm)	138	210	400	210
Velocidade (m/s)	0,22	0,68	16,85	4,5
Bateria	Lipo 3S 1800mAh	Lipo 2s 2200mah	Nimh 7.2V 2000mAh	Lipo 3s 1300mah
Esc	x	DRV8828	RoboClaw 2x7A	DRV8828
Imu	acc + mag + gyro	BNO055	BNO055	BNO055
Sonar	US015	US015	US015	US015
Encoder	x	AMT102-V	AMT102-V	x
Cartão SD	x	leitor spi	leitor spi	leitor spi
Controlador	rpi3	esp32	esp32	esp32
Total	610,68 €	158,46 €	316,78 €	183,08 €

Na tabela 2.1 encontram-se sistematizadas as principais informações relativas a cada plataforma e uma estimativa inicial do custo.

A plataforma escolhida foi o kit "Chassis Ackerman", dado ter um custo impotável e permitir a construção com razoável robustez física do veículo.

Capítulo 3

Conceção da plataforma

3.1 Arquitetura do sistema

A plataforma a produzir foi assim assente em componentes selecionados por forma a verificarem os requisitos estabelecidos na secção 2.

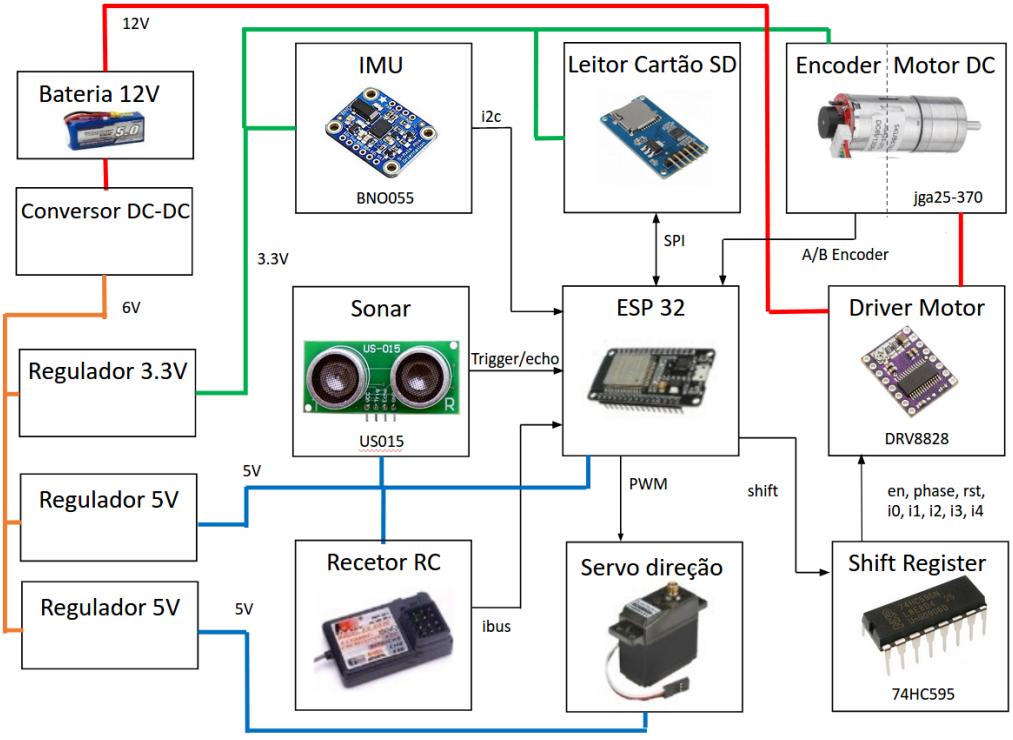

Figura 3.1: Arquitetura do sistema.

Na figura 3.1 está esquematizado o sistema desenvolvido, na secção esquerda as fontes de alimentação com a divisão dos componentes entre elas, e na direita as ligações dos sensores e atuadores ao microcontrolador.

Tabela 3.1: Autonomia da plataforma.

	standby	idle	max
Corrente (mA)	30	400	2500
Autonomia (h)	3,33	2,5	0,4

A alimentação da plataforma é assegurada por uma bateria de Polímero de Lítio de 3 células série LiPo 3S 1000mAh, cujo débito de corrente foi medido e a duração estimada na tabela 3.1.

3.2 Micro-controlador

O microcontrolador (uC) utilizado foi o ESP32 da Espressif, uma plataforma de baixo custo com capacidade de comunicação por WiFi e Bluetooth baseado no microprocessador de 32 bits Tensilica Xtensa LX6 [15, 16].

O fabricante produz variantes do SOC com 1 ou 2 cores e opção de memória flash embutida no SOC. Este uC pode ser comprado em QFN (quad-flat no-leads) 48 pinos ou está disponível em módulos surface-mount incluindo memória flash e todos os circuitos necessários para o funcionamento deste chip. Existem módulos produzidos pelo fabricante bem como por terceiros.

Dos módulos existentes foi selecionado o ESP32-WROVER-B com 4MB de flash e 8MB de PSRAM. Este microcontrolador é baseado no chip ESP32-D0WDQ6. Existem dois núcleos de CPU a 240 MHz capazes de 600 MIPS, 520kB de memória RAM e um coprocessador de baixo consumo para monitorizar os periféricos. O ESP32 integra periféricos para sensores capacitivos de toque, sensores efeito Hall, interface cartão SD, Ethernet, SPI alta velocidade, UART, I2S, I2C e CAN [17].

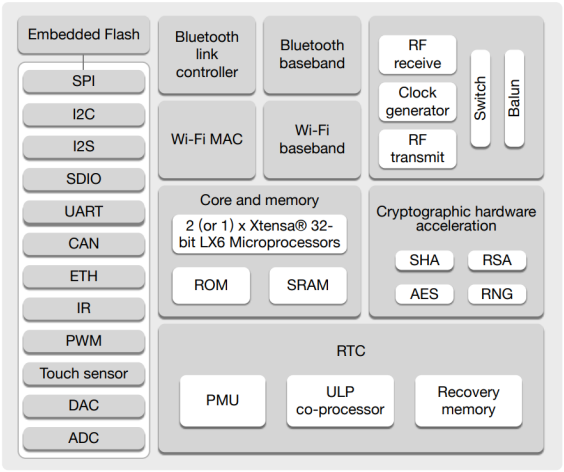

Figura 3.2: ESP32 - Diagrama de blocos.

De forma a simplificar a inclusão deste microcontrolador foi utilizada a placa de desenvolvimento ESP32-DEVKITC-VB que inclui para além do uC um regulador para 3,3V bem como um conversor USB-UART CP2102N.

3.2.1 Shift Register

Na figura 3.3 apresenta-se o circuito "Shift Register"necessário para fazer a expansão do número de saídas existentes no microcontrolador.

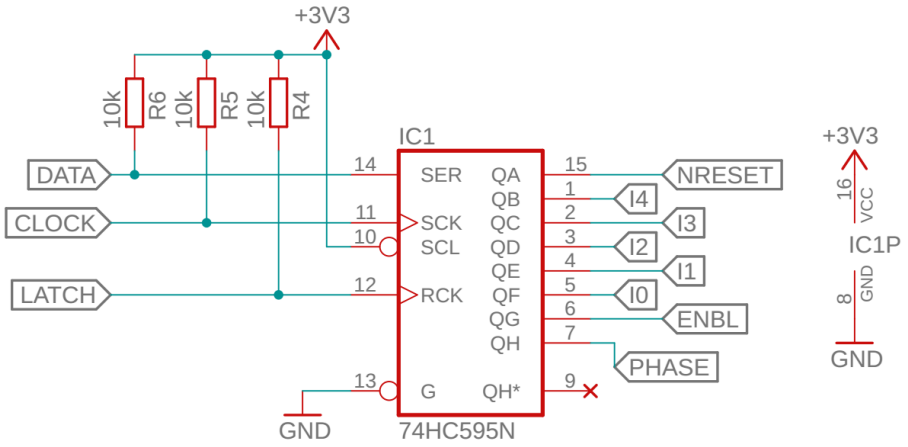

Figura 3.3: Esquemático do Shift Register.

O Shift register utilizado é o 74HC595 e faz a interface com a drive do motor. Este componente utiliza 3 pinos do microcontrolador (data, clock, latch) para controlar o estado dos 8 pinos de saída[18].

3.3 Motor

O motor incluído no kit é um jga25-370 um motor DC com caixa de redução estandardizado. Foi assim possível substituir o motor original do kit por um com encoder incorporado.

O motor selecionado foi o Pololu #4864 34:1 MP 12V w/encoder um motor DC de escovas do tamanho 25D com caixa de redução. Este motor tem uma velocidade de rotação nominal de 220 rpm sem carga e uma corrente de stall de 2,1A [19].

3.3.1 Encoder

O motor tem acoplado um encoder de 12 pulsos por rotação com dois canais desfasados de 90º. Estes sinais são lidados com recurso ao periférico do ESP32 o contador de pulsos PCNT com 8 unidades independentes de 2 canais cada uma.

O PCNT é capaz de aumentar ou reduzir um contador comandado por 2 sinais: pulso e direção. Usando no canal 0 o sinal A como pulso e o sinal B como direção obtemos a contagem em meia-quadratura do encoder. Ao inverter os sinais no canal 1, isto é o sinal B como pulso e o sinal A como direção a contagem é efetuada em quadratura completa tendo assim 48 PPR do motor.

O motor tem acoplado uma caixa de velocidade de redução de 34:1 e o acoplamento final do motor ao eixo traseiro introduz uma relação entre estes de 1:2 assim a relação total entre o motor e as rodas de tração é de 17:1.

O diâmetro das rodas é de 64 mm pelo que a relação de distância percorrida para pulso do encoder pode ser calculada pela equação 3.3

$$r_{encoder-roda} = 34:1 \times 1:2 = 17:1 \tag{3.1}$$

$$P_{roda} = 2 \times \pi \times \frac{d}{2} = 201,06mm \tag{3.2}$$

$$mm/pulso = \frac{P_{roda}}{PPR \times r_{encoder-roda}} = 0,246 \tag{3.3}$$

3.4 Recetor R/C

A escolha do receptor r/c recaiu num compatível com o protocolo FLYSKY AFHDS 2A e com saída ibus. Dos vários modelos disponíveis o A8S é o mais compacto.

3.5 Driver Motor

O driver do motor selecionada foi a DRV8828 da Texas Instruments uma driver de controlo de corrente. A corrente máxima admissível é 3A e a tensão máxima 45V.

Na figura 3.4 apresenta-se o circuito necessário para a implementação deste ic.

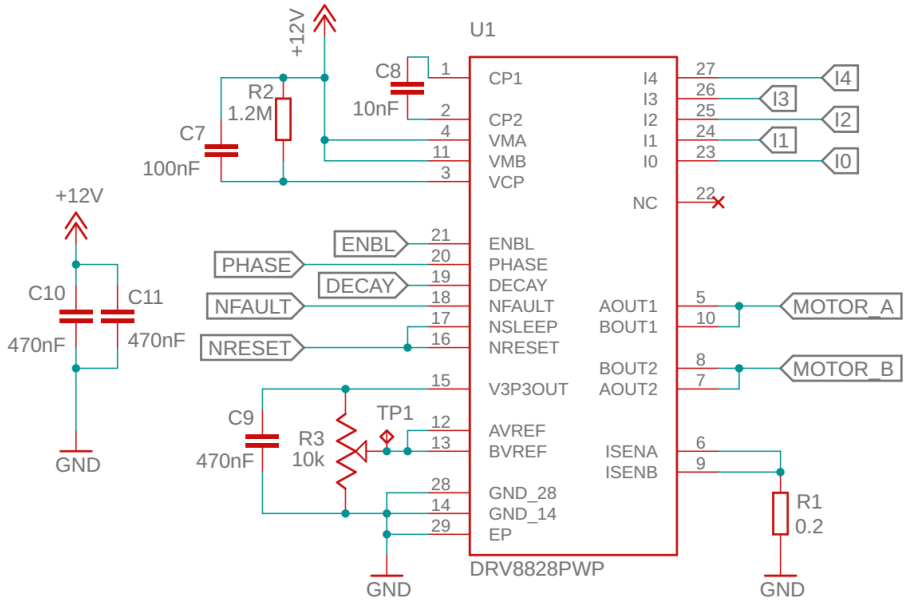

Figura 3.4: Esquemático do driver motor DRV8828.

O driver usa uma tensão de referência VREF, bem como um *prescaler* de 5 bits (I4, I3, I2, I1 e I0) para determinar a corrente a aplicar ao motor [20, 21].

A corrente que a driver controla é obtida com a equação 3.4

$$ITRIP = \frac{VREF}{5 \times RSENSE} \tag{3.4}$$

3.5.1 Linearização do motor

Saturando VREF o driver não consegue garantir a linear correspondência do comando com a corrente no motor, pois este com a tensão usada de 12V consome no máximo 1,8A.

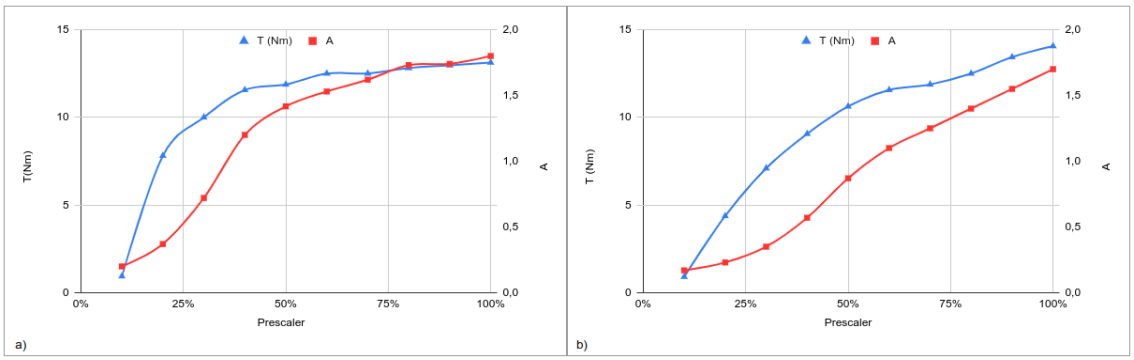

Figura 3.5: Gráfico corrente/binário — a) VREF máximo. b) VREF 1,8V.

Na figura 3.5 compara-se o binário de arranque e a corrente consumida pelo motor a diferentes percentagens de *prescaler* aplicadas. Para medir este binário o veículo foi amarrado a uma massa através de uma polia transferindo a força de tração deste para levantar a massa. O peso da massa foi medido com uma balança e a redução deste é resultado da força de aceleração aplicada ao veículo.

3.6 Fonte de alimentação

Da bateria de lítio com tensão nominal de 11.1 V é necessário produzir 3.3V e 5V para alimentar todos os componentes do veículo. De forma a reduzir as interferências no funcionamento do microcontrolador a alimentação do servo foi separada.

Na figura 3.6 é apresentado o circuito implementado para a conversão da tensão da bateria nos rails de alimentação necessários para o veículo.

Foi escolhido o regulador linear LD1117 da STMicroelectronics pela universalidade da sua utilização, reduzido número de componentes externos necessários, boa estabilidade e proteção contra sobrecarga e excesso de temperatura [22].

Para evitar a dissipação de potências elevadas nos reguladores lineares foi utilizado um módulo conversor DC-DC de tipologia abaixador de 12V para 7V baseado no MP2307.

O diodo D1, um SS34 da Vishay, protege o circuito no caso de inversão da polaridade da bateria sendo a corrente média limite de 3A e os 40V de tensão máxima admissível superiores aos 12V da bateria bem como aos 2A medidos de corrente consumida [23].

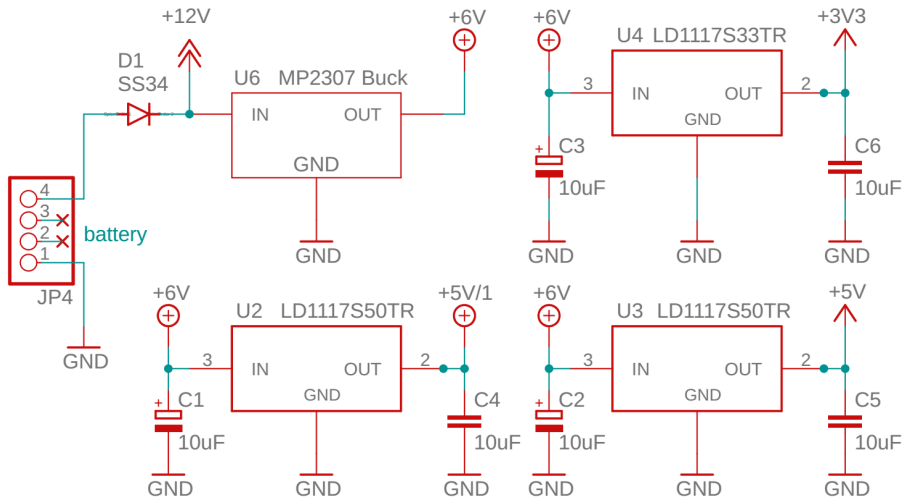

Figura 3.6: Esquemático das fontes de alimentação.

3.7 Sonar

Sensor ultra-sónico funciona num principio semelhante ao sonar. Emite um som de alta frequência e mede o tempo até ouvir o sinal refletido. Este sinal que se propaga à velocidade do som no ar demora tanto mais tempo a ser refletido quanto mais longe o obstáculo se encontrar.

O sensor escolhido foi o US015 com resolução de 1mm e capacidade de medição desde 2cm até 4m.

Este sensor inicia uma medição com um pulso no pino *TRIGGER* e devolve o tempo de voo como o tempo ON de um pulso no pino *ECHO*.

3.8 Leitor cartões SD

O leitor de cartões é um simples adaptador físico para conectar um cartão SD a uma porta SPI do microcontolador.

O cartão SD permite guardar todos os dados considerados relevantes ao longo dos ensaios a realizar para posteriormente serem analisados e processados.

Capítulo 4

Implementação do software

O código foi desenvolvido na framework arduino para o Esp32[24]. Esta framework simplifica o desenvolvimento e a leitura do código, bem como permite o uso de bibliotecas *opensource*.

Para a compilação e upload do código foi utilizado o IDE Platformio sobre o Visual Studio Code. No desenvolvimento do software foi utilizado um repositório de controlo de versões Git com hospedagem no GitLab: `https://gitlab.com/pgs21/imu-rc-l298n`

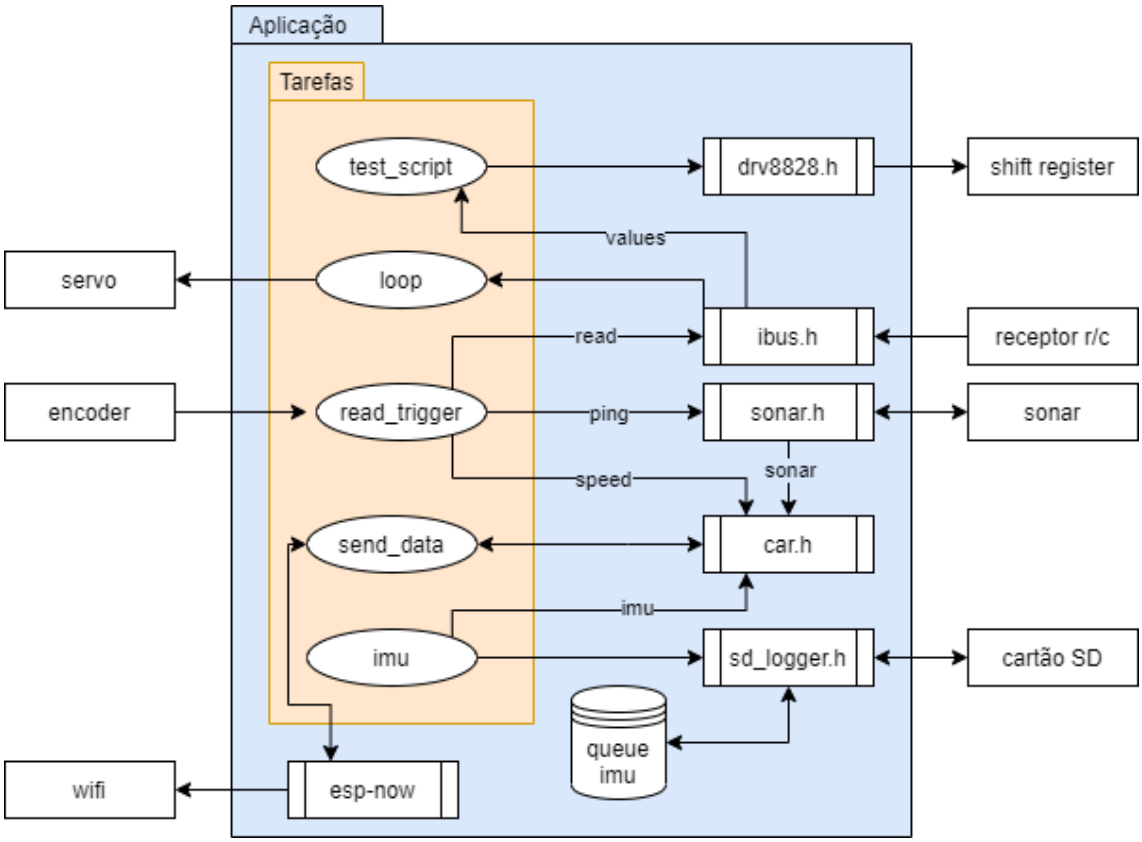

Figura 4.1: Arquitetura do software.

Na figura 4.1 é apresentada a arquitetura do software e a divisão do código gerado em bibliotecas que fazem interface com o hardware do veículo e em tarefas que implementam a lógica do

veículo.

4.1 freeRTOS

O SDK do Esp32 baseia-se num kernel freeRTOS com um escalonador a correr tarefas de maneira periódica. Este escalonador utiliza uma estratégia *round-robin* preemptiva para garantir uma execução real-time das tarefas.

O kernel freeRTOS é responsável por lidar com execução multitarefa, mudança de contexto, acesso a recursos partilhados, mecanismos de sincronização e gestão de tempo.

As tarefas implementadas estão descritas na tabela 4.1.

Tarefa	Descrição	Prioridade	Período (ms)
IMU	Recolha de dados da IMU	10	5
test_script	Lógica de controlo do veículo	1	10
read_trigger	Trigger periódico para inicio de funções	3	20
send_data	Envio de informação entre os veículos	4	10
loop	Controlo servo	2	20

Tabela 4.1: Tarefas do sistema.

4.2 Tarefas

O código criado neste trabalho corresponde a um exemplo de utilização desta plataforma, sendo assim implementado o controlo do veículo líder diretamente pelo comando remoto, sendo estes comandos retransmitidos para o veículo seguidor. As figuras 4.2, 4.4, 4.5, 4.3 e 4.6 representam a lógica executada ciclicamente pelo microcontrolador nas tarefas IMU, test_script. read_trigger, send_data e loop respectivamente.

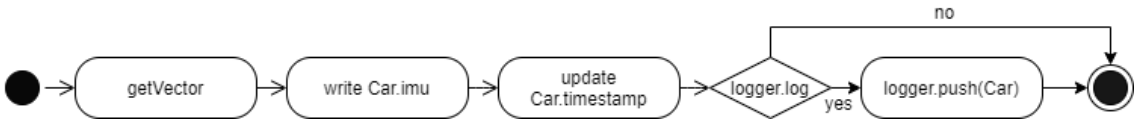

Figura 4.2: Tarefa IMU.

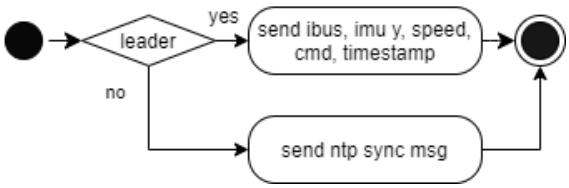

Figura 4.3: Tarefa send_data.

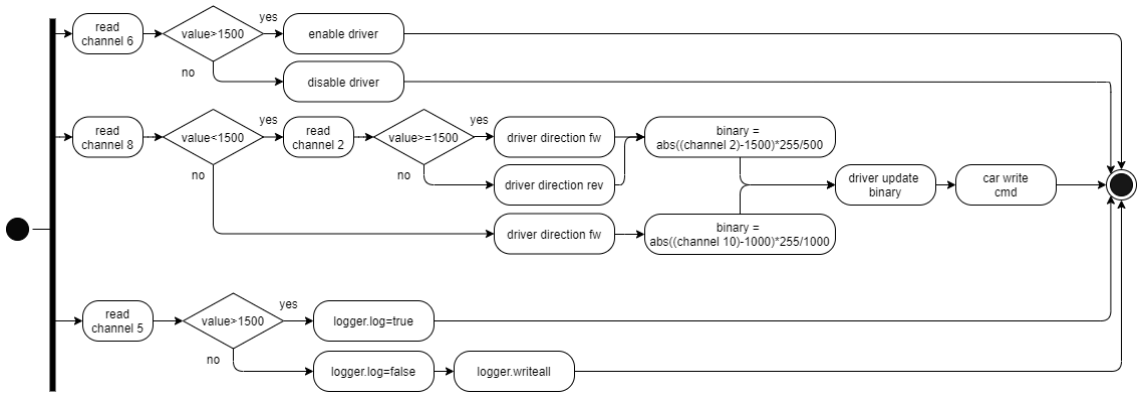

Figura 4.4: Tarefa test_script.

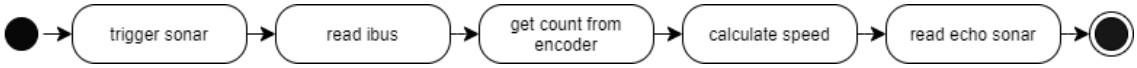

Figura 4.5: Tarefa read_trigger.

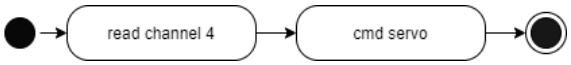

Figura 4.6: Tarefa loop.

4.3 Comunicação

A comunicação foi implementada utilizando o protocolo ESP NOW da Espressif um protocolo *connectionless* assente no IEEE 802.11n mas sem necessidade de router ou servidor DHCP usando antes endereços MAC [25].

Este protocolo foi desenhado para aplicações iot usando poucos recursos do mirocontrolador e permitindo reduzir o Round Trip Time (RTT) da comunicação.

4.4 Bibliotecas

Para implementar a interface com o hardware da plataforma foi desenvolvido um conjunto de classes que abstraem o utilizador dos pormenores.

4.4.1 car.h

```
class CAR {
public:
   CAR(); // novo objeto
   ~CAR(); // destrutor
   imuVector imu; // vetor imu
   float speed; // velocidade
   int sonar; // distancia ao obstáculo
   int cmd; // comando enviado para o driver 0-255
   long offset; // offset temporal para o carro anterior
```

```
10    int delay; // atraso na comunicação com o carro anterior
11    unsigned long timestamp; // timestamp das leituras
12    void newImu(imu::Vector<3> newimu); // novos valores da imu
13 };
```

Código 4.1: Classe car.

Classe criada para guardar as variáveis de interesse do veículo.

4.4.2 sd_logger.h

```
1  class sd_logger{
2  public:
3     sd_logger(byte CS, int queueSize); // inicia cartão sd e fila
4     ~sd_logger(); // destrutor
5     void push(CAR data); // envia dados para a fila
6     void clear(); // limpa a fila
7     void writeAll(); // escreve a fila no cartão
8     boolean cardError = true; // flag erro a abrir cartão
9     boolean log = false; // flag habilitar registo de dados
10 private:
11    File file; // ficheiro
12    String fileName; // nome do ficheiro
13    QueueHandle_t queueIMU; // fila de dados
14    int _queueSize = 1000; // tamanho da fila
15    void appendFile( const char * message); // adicionar a ficheiro já aberto
16    void writeFile( const char * path,
17                    const char * message); // criar ficheiro
18    void openFile(const char * timestamp); // abrir ficheiro
19    void closeFile(void); // fechar ficheiro
20 };
```

Código 4.2: Classe sd_logger.

Classe criada para guardar uma fila FIFO com os valores a escrever no cartão de memória. O método *push* coloca na fila a estrutura passada como argumento e o método *write_all* escreve a fila completa para o cartão de memória. Isto foi feito para reduzir o impacto causado pela abertura e fecho do ficheiro na execução do código.

O método *clear* reinicializa a fila *queueIMU*

4.4.3 ibus.h

```
1  class IBUS {
2  public:
3     IBUS (byte rxPin, byte txPin); // iniciar classe e porta série
4     IBUS (void); // iniciar classe
5     ~IBUS(); // destrutor
6     void read(); // ler buffer porta série 2
```

```
7     void addValues( int ch[14]); // modificar vetor de dados completo
8     int value(byte channel); // ler valor de canal
9  private:
10    int rcValue[14]={0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // valor dos canais
11    unsigned long lastreceive = 0; // tempo de últimos dados recebidos
12 };
```

Código 4.3: Classe ibus.

Classe responsável pela leitura, interpretação e verificação dos dados recebidos pelo protocolo ibus. Esta classe pode ser instanciada com os pinos atribuídos à porta série 2, iniciando esta para receber as comunicações do recetor R/C, ou só como um objeto para guardar os valores recebidos do líder.

Esta classe disponibiliza o método *channel* para facilitar o acesso aos dados dos canais corrigindo o *offset* do índice.

O método *read* verifica a existência de novos valores no buffer de receção da porta série e caso estes passem o teste de CRC atualiza estes valores no vetor privado, bem como implementa um *wathcdog* que no caso de falha de receção de quaisquer valores na porta série por mais de 1 segundo coloca os valores a zero.

O método *addvalues* permite substituir o vetor de valores privado *rcValue* por um novo vetor.

4.4.4 drv8828.h

```
1  class DRV8828 {
2  public:
3     DRV8828 (byte clkPin, byte latchPin, byte dataPin); //iniciar classe
4     ~DRV8828(); // destrutor
5     void enable(boolean mode); // modificar bit de reset e enable
6     void direction(boolean mode); // modificar bit de phase
7     void binary(byte torque); // calcular e modificar 5 bits de binário
8     int cmd(); // comando a enviar ao driver
9     void shiftsend(); // enviar comando para driver
10 private:
11    byte _latchPin, _clkPin, _dataPin; // pinos do shift register
12    byte _shiftBuffer; // byte a enviar
13    int _binary = 0; // último binário calculado
14    boolean enabled = false; // estado do bit enable
15    void shiftOutSlow(uint8_t dataPin, uint8_t clockPin,
16          uint8_t bitOrder, byte val); // shiftar dados para ic
17    SemaphoreHandle_t xSemaphoreShift = NULL; // semáforo acesso ao ic
18 };
```

Código 4.4: Classe drv8828.

4.4.5 sonar.h

```
class US015SONAR {
public:
   US015SONAR(byte tiggerPin, byte echoPin); // iniciar sonar
   ~US015SONAR(); // destrutor
   long value(); // retorna a última distância medida
   void ping(); // envia ping para o sensor
private:
   static US015SONAR *instance; // apontador para a instância da classe
   void IRAM_ATTR classISRHandler(); // calcula o tempo positivo do pulso echo
   static void IRAM_ATTR classInterruptHandler(); // método para lidar com
       interrupt
   int us015_mode=0; // flag estado da contagem do pulso, 0 = contagem por iniciar
   long us015_time=0; // tempo do início do pulso
   long us015_width = 0; // distância ao obstáculo
   byte _triggerPin, _echoPin; // pinos do sensor
};
```

Código 4.5: Classe drv8828.

Sabendo o tempo de voo do sinal ultrassónico e assumindo a velocidade de propagação do som no ar ao nível do mar como constante, a distância pode ser calculada com a equação 4.1

$$d = \frac{t_{pulso}}{2} \times v_{som} \tag{4.1}$$

Capítulo 5

Construção e implementação da plataforma

5.1 Modelo 3D

De forma a integrar o sistema a desenvolver no chassis do kit adquirido este foi transposto para um modelo 3D com as medidas críticas. Este modelo está presente na figura 5.1.

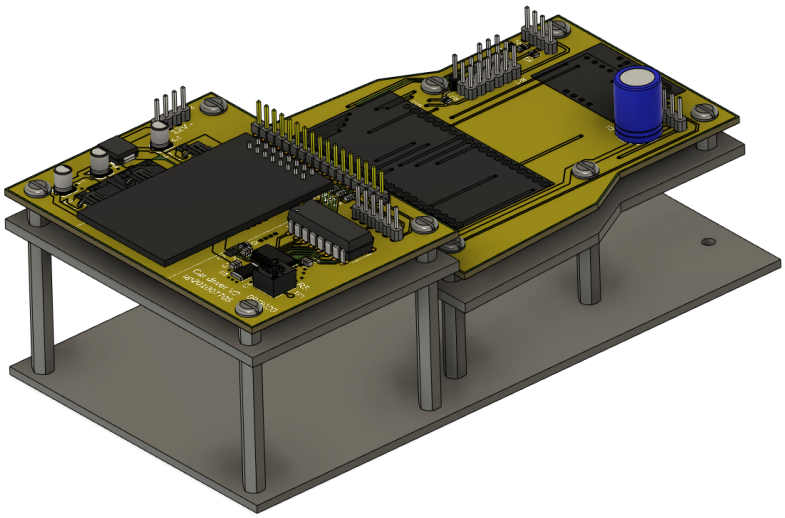

Figura 5.1: Modelo 3D.

Neste modelo foi possível analisar que a distância entre as placas erra ideal para fazer a interligação com pinos de header 1x17 2,54

5.2 Protótipo

Para ensaio do driver do motor foi necessário produzir uma placa de breakout para o ic.

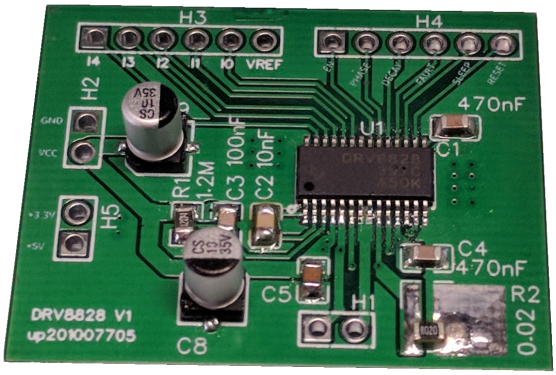

Figura 5.2: Breakout driver.

Esta placa foi utilizada para ensaiar todo o sistema, bem como para desenvolvimento do software juntamente com os restantes componentes ligados em veraboard.

Figura 5.3: Protótipo.

5.3 PCBs

As placas de circuito impresso foram desenhadas no software Fusion 360 recorrendo ao pacote Eagle.

As pcbs foram criadas para caber nos apoios já existentes do kit e com a interligação entre placas sem recorrer a cabos. Estas pcbs foram exportadas como *gerbers* e enviadas a um fabricante que as produziu.

Capítulo 6

Conclusão

A plataforma foi desenhada para possibilitar o desenvolvimento de algoritmos de controlo longitudinal de platooning. A divisão do sistema em duas pcbs permite uma fácil adaptação a futuras necessidades de sensorização ou atuação.

Para determinar o atraso e offset na comunicação foi implementado o protocolo NTP tendo sido registado um valor médio de 1ms de round trip time com um atraso máximo de 2ms, sendo este valor muito reduzido para uma comunicação assente no protocolo 802.11n.

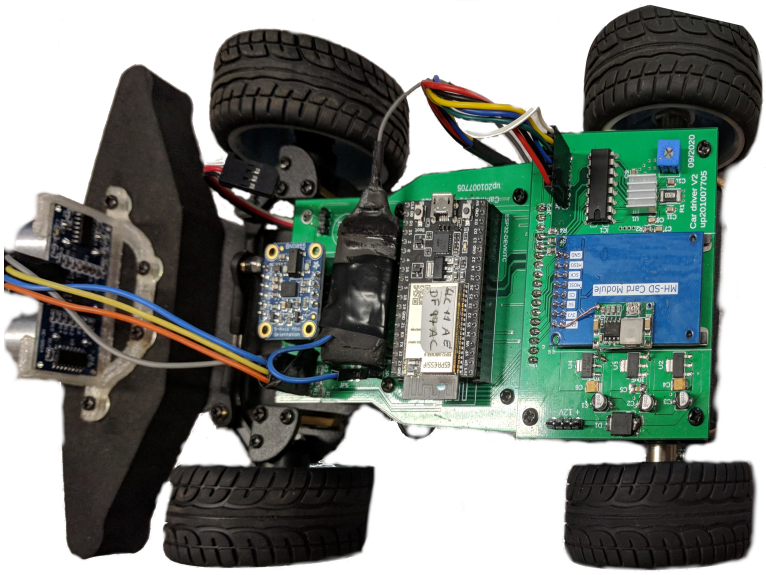

Figura 6.1: Plataforma final.

6.1 Verificação de requisitos

No capítulo 2 foram levantados os requisitos que a plataforma teria de cumprir. Pode-se verificar na figura 6.2 o cumprimento destes mesmos.

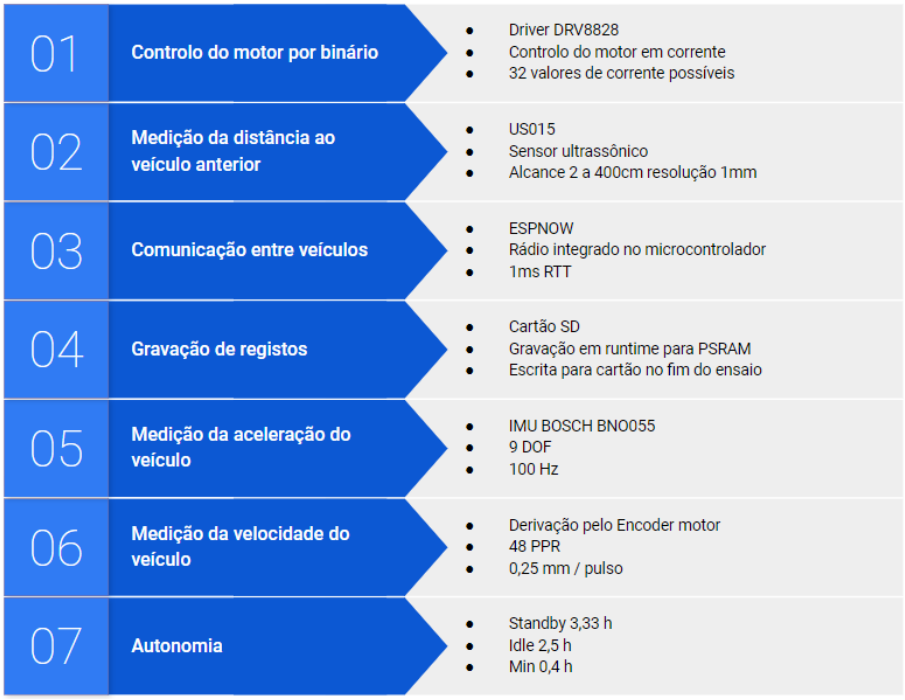

Figura 6.2: Verificação de requisitos

6.2 Trabalho futuro

Trabalhos futuros nesta plataforma podem prever a inclusão de sistemas para permitir o controlo lateral dos veículos, bem como o desenvolvimento de uma base-station para controlo e monitorização dos veículos.

Referências

[1] Jeroen Ploeg, Dipan P. Shukla, Nathan van de Wouw, e Henk Nijmeijer. Controller synthesis for string stability of vehicle platoons. *IEEE Transactions on Intelligent Transportation Systems*, 15:854–865, 4 2014. URL: http://ieeexplore.ieee.org/document/6683051/, doi:10.1109/TITS.2013.2291493.

[2] Carl Bergenhem, Henrik Pettersson, Erik Coelingh, Cristofer Englund, Steven Shladover, e Sadayuki Tsugawa. Overview of platooning systems. *19th Intelligent Transport Systems World Congress, ITS 2012*, 1(June 2014), 2012.

[3] Mark Michaelian e Fred Browand. Field experiments demonstrate fuel savings for close-following. *University of California at Berkeley*, 2000.

[4] Patrick Hong, Bogdan Marcu, Fred Browand, e Aaron Tucker. Drag forces experienced by two, full-scale vehicles at close spacing. Em *SAE Technical Papers*, 1998. URL: https://escholarship.org/uc/item/50q2p3sn, doi:10.4271/980396.

[5] JAMES B. MICHAEL, DATTA N. GODBOLE, JOHN LYGEROS, e RAJA SENGUPTA. Capacity Analysis of Traffic Flow Over a Single-Lane Automated Highway System. *ITS Journal - Intelligent Transportation Systems Journal*, 4(1-2):49–80, jan 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.766http://www.tandfonline.com/doi/abs/10.1080/10248079808903736, doi:10.1080/10248079808903736.

[6] Datta N. Godbole, Farokh H. Eskafi, e Pravin P. Varaiya. Automated Highway Systems. *IFAC Proceedings Volumes*, 29(1):5506–5511, jun 1996. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474667017585584, doi:10.1016/S1474-6670(17)58558-4.

[7] Jeroen Ploeg. *Analysis and design of controllers for cooperative and automated driving*. Tese de doutoramento, Eindhoven: Technische Universiteit Eindhoven, 2014. URL: https://doi.org/10.6100/IR772224, doi:10.6100/IR772224.

[8] San Gen Hu, Hui Ying Wen, Lunhui Xu, e Hui Fu. Stability of platoon of adaptive cruise control vehicles with time delay. *Transportation Letters*, 11(9):506–515, oct 2019. URL: https://www.tandfonline.com/doi/full/10.1080/19427867.2017.1407488, doi:10.1080/19427867.2017.1407488.

[9] L. C. Davis. Effect of adaptive cruise control systems on traffic flow. *Physical Review E*, 69(6):274–290, jun 2004. URL: https://link.aps.org/doi/10.1103/PhysRevE.69.066110, doi:10.1103/PhysRevE.69.066110.

[10] RACECAR. URL: https://mit-racecar.github.io/.

[11] F1tenth. URL: http://f1tenth.org/index.html.

[12] GitHub - Hypha-ROS/hypharos_minicar: 1/20 MiniCar: An ackermann based rover for MPC and Pure-Pursuit controller. URL: https://github.com/Hypha-ROS/hypharos{_}minicar.

[13] Florin Catalin Braescu. Basic control algorithms for vehicle platooning prototype model car. Em *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*, páginas 180–185. IEEE, oct 2017. URL: http://ieeexplore.ieee.org/document/8107031/, doi:10.1109/ICSTCC.2017.8107031.

[14] TurtleBot. URL: https://www.turtlebot.com/.

[15] Espressif Systems. *ESP32-WROVER-B Datasheet*, 2 2020. Version 1.4.

[16] Espressif Systems. *ESP32 Technical Reference Manual*, 11 2019. Version 4.1.

[17] Espressif Systems. *ESP32 Series Datasheet*, 4 2020. Version 3.4.

[18] Semiconductor Components Industries, LLC. *8-Bit Serial-Input/Serial or Parallel-Output Shift Register with Latched 3-State Outputs*, 3 2007. Rev. 1.

[19] Pololu 25d 34:1 mp 12v w/encoder. URL: https://www.pololu.com/product/4864/specs.

[20] Texas Instruments. *DRV8828 H-Bridge Motor Controller IC*, 11 2015. Rev. G.

[21] Texas Instruments. *Best Practices for Board Layout of Motor Drivers*, 1 2019. Rev. A.

[22] STMicroelectronics. *Adjustable and fixed low drop positive voltage regulator*, 2 2020. Rev. 37.

[23] Vishay General Semiconductor. *Surface-Mount Schottky Barrier Rectifier*, 4 2020. Rev. 23-Apr-2020.

[24] Espressif Systems. *ESP-IDF Programming Guide*, 3 2020. Version 4.1.

[25] Espressif Systems. *ESP-NOW User Guide*, 7 2016. Version 1.0.

Anexo A

PCB

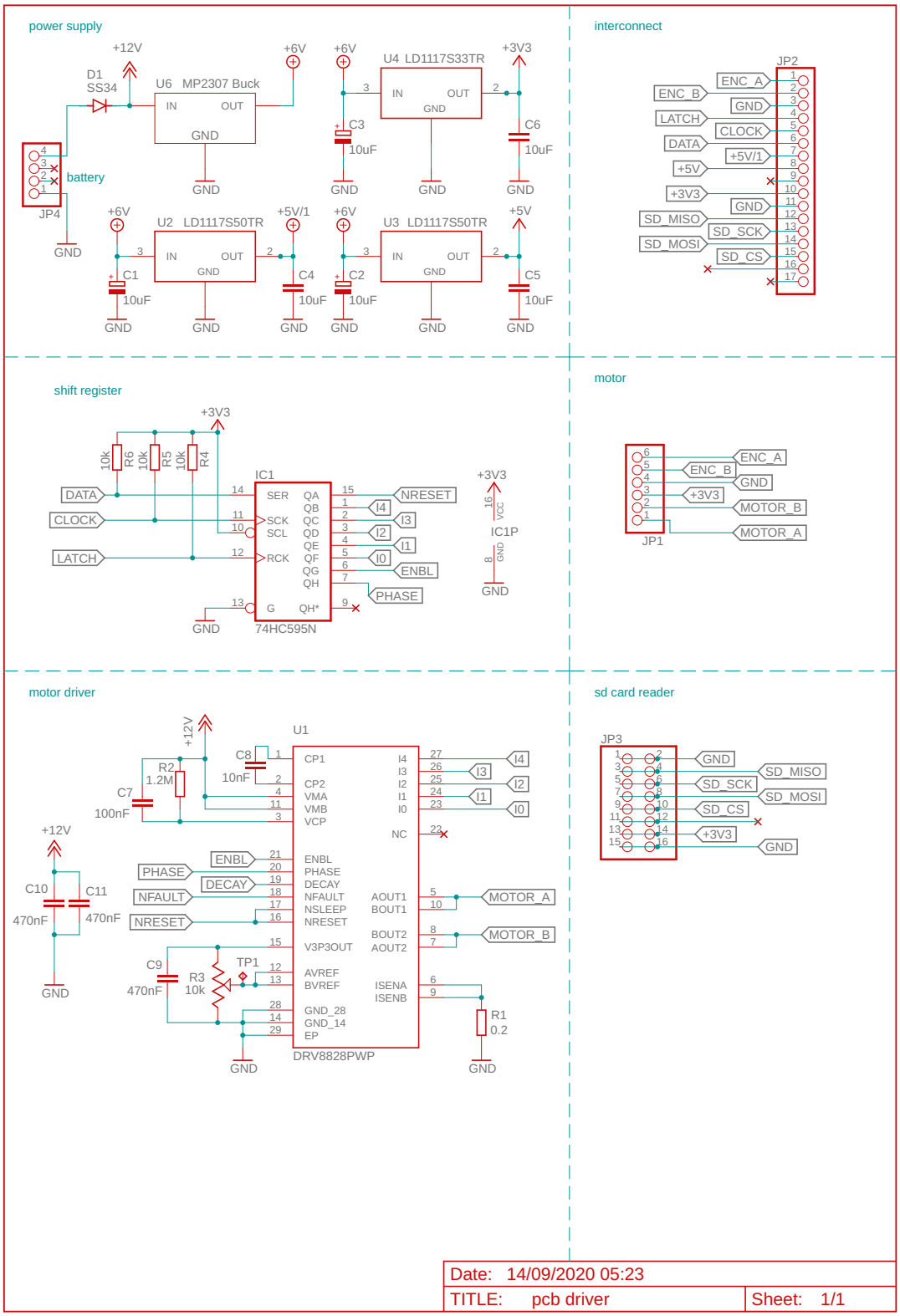

Figura A.1: Esquemático pcb driver

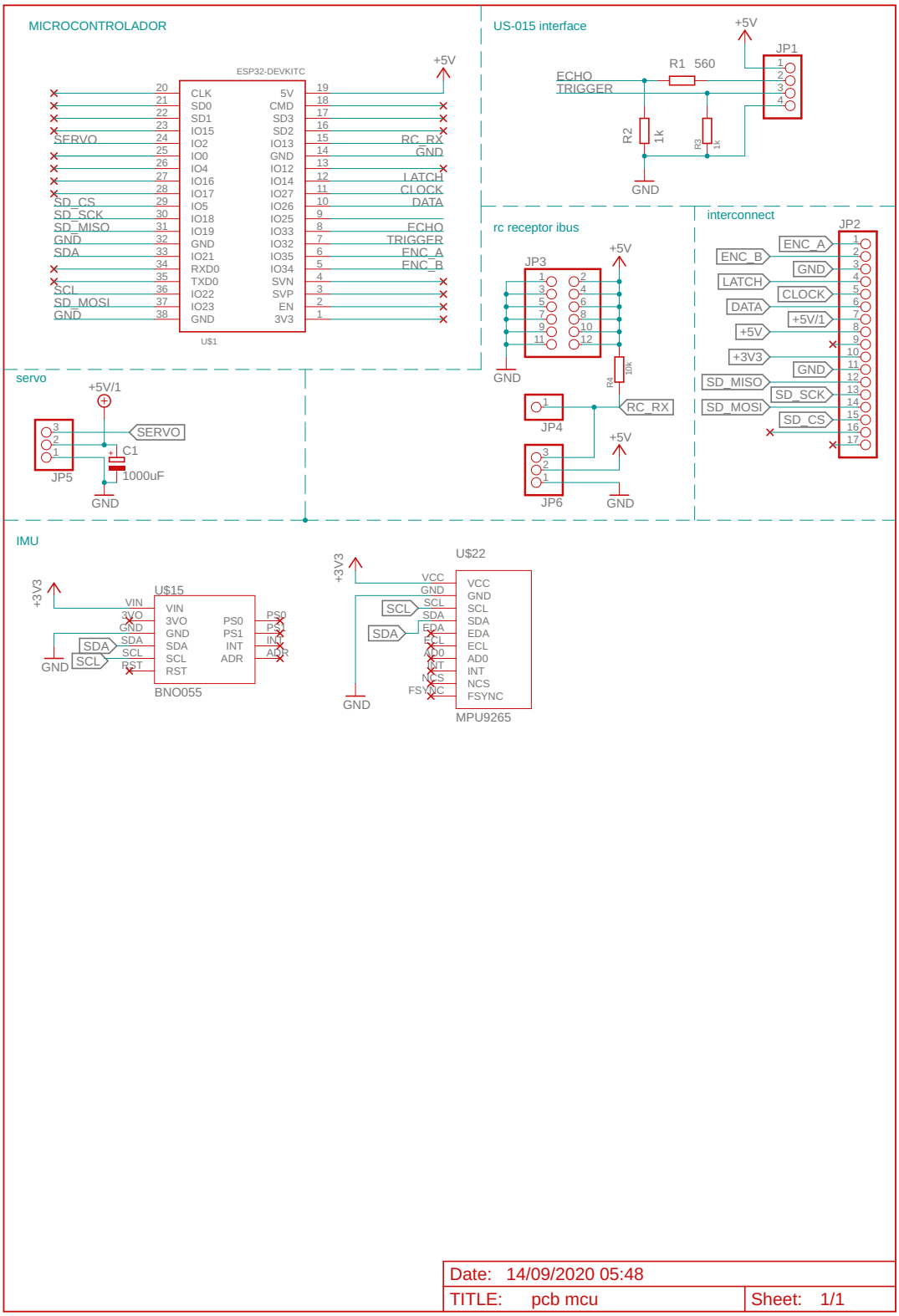

Figura A.2: Esquemático pcb mcu

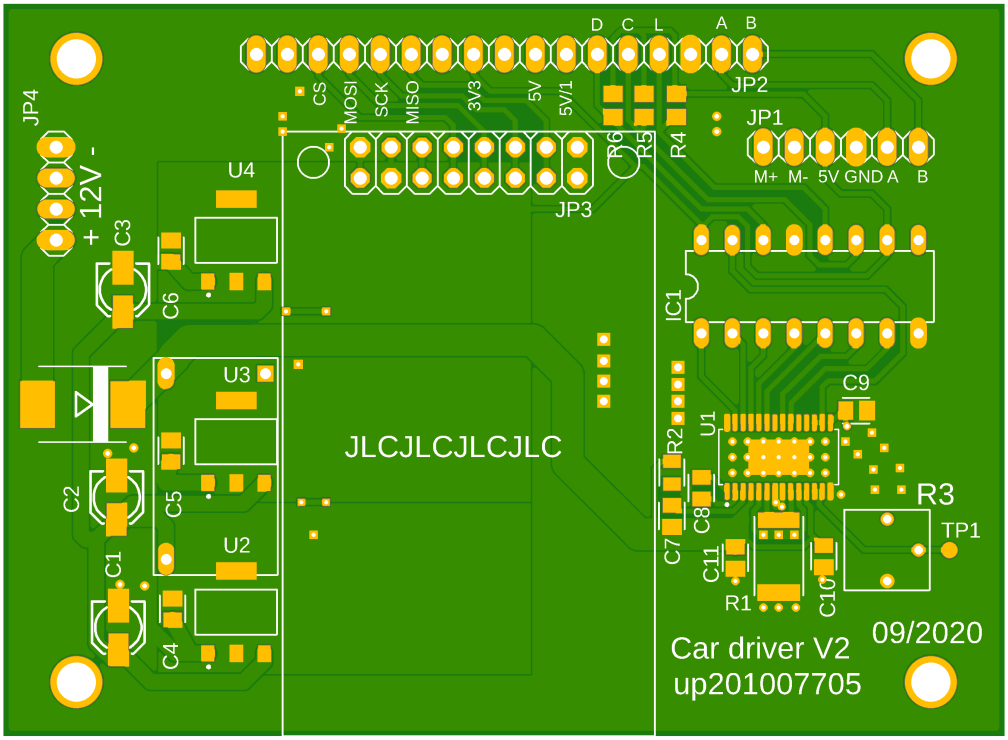

Figura A.3: Layout pcb driver

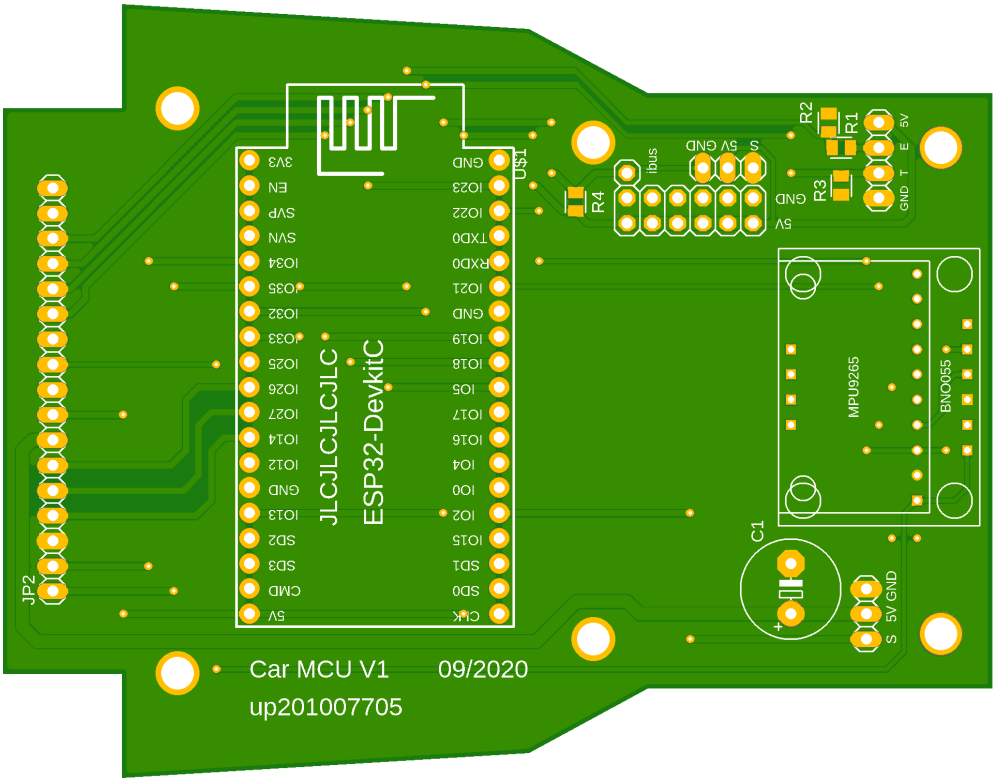

Figura A.4: Layout pcb MCU

Anexo B

Source

B.1 main.cpp

```
#include <Arduino.h>
#include "config.h"

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

#include <WiFi.h> // WiFi control for ESP32
#include <esp_now.h>

#include <ESP32Encoder.h>
#include <drv8828.h>
#include <ibus.h>
#include <car.h>
#include <sonar.h>
#include <sd_logger.h>
#include "esp32-hal-ledc.h"

ESP32Encoder encoder;
DRV8828 driver(clockPIN, latchPIN, dataPIN);

#if POSITION == 0
IBUS ibus(rxRcPIN,txRcPIN);
#elif POSITION == 1
IBUS ibus;
#endif

US015SONAR sonar(US015TRIGGER, US015ECHO);
sd_logger logger(SD_CS, 1000);

CAR car[2];
```

```
33
34 delay_msg incomingReadings, outgoingMsg;
35
36 TaskHandle_t imuTask, sdwriterTask, readTask, sendTask;
37
38 SemaphoreHandle_t xSemaphoreSerial = NULL , xSemaphoreCarData = NULL;
39
40 // Check I2C device address and correct line below (by default address is 0x29 or
       0x28)
41 // id, address
42 Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28);
43
44 //Task for reading IMU values
45 void IMU(void* parameters)
46 {
47   TickType_t xLastWakeTime;
48   const TickType_t xFrequency = pdMS_TO_TICKS( 5);
49   // Initialise the xLastWakeTime variable with the current time.
50   xLastWakeTime = xTaskGetTickCount();
51   for ( ;; )
52   {
53     vTaskDelayUntil( &xLastWakeTime, xFrequency );
54     imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
55     car[POSITION].newImu(euler);
56     car[POSITION].timestamp = esp_timer_get_time()/1000;
57     if(logger.log)
58       logger.push(car[POSITION]);
59
60 #ifdef MEMDEBUG
61     xSemaphoreTake(xSemaphoreSerial, 0);
62     Serial.print("TASK: ");
63     Serial.print(pcTaskGetTaskName(NULL));
64     Serial.print(" min free stack: ");
65     Serial.println(uxTaskGetStackHighWaterMark(NULL));
66     xSemaphoreGive(xSemaphoreSerial);
67 #endif
68
69     xLastWakeTime = xTaskGetTickCount();
70   }
71   vTaskDelete( NULL );
72 }
73
74 //---------Task with logic of car
75 // Ch2 joystick trottle | Ch10 VRA throttle
76 // Ch8 select between joystick and VRA
77 // Ch6 enable motor
78 // Ch5 log data
79 void test_script(void* parameters)
80 {
81   TickType_t xLastWakeTime;
```

```
82  const TickType_t xFrequency = pdMS_TO_TICKS( 10 );
83  // Initialise the xLastWakeTime variable with the current time.
84  xLastWakeTime = xTaskGetTickCount();
85
86  for(;;){
87    vTaskDelayUntil( &xLastWakeTime, xFrequency );
88
89    // enable motor driver with channel 6 > 1500ms
90    if(ibus.value(6)>1500){
91      driver.enable(true);
92    }
93    else{
94      driver.enable(false);
95    }
96
97    // chose between joystick and vra
98    int binary =0;
99    if(ibus.value(8)<1500){
100
101     //select direction of motor
102     if(ibus.value(2)>=1500){
103       driver.direction(false);
104     }
105     else{
106       driver.direction(true);
107     }
108     // calculate absolute throttle position from 0 to 255
109     binary = abs(ibus.value(2)-1500)*255/500;
110   }
111   else{
112
113     //in vra mode assume forward commands
114     driver.direction(false);
115     binary = abs(ibus.value(10)-1000)*255/1000;
116   }
117   driver.binary(binary);
118   car[POSITION].cmd=driver.cmd();
119
120   if(ibus.value(5)>= 1500 && logger.log==false){
121     logger.log = true;
122     Serial.println("start logging");
123   }
124   else if(ibus.value(5)< 1500 && logger.log==true){
125     Serial.println("write file");
126     logger.log = false;
127     logger.writeAll();
128     Serial.println("file written");
129   }
130
```

```
131     driver.shiftsend(); // send updates to driver
132
133 #ifdef MEMDEBUG
134     xSemaphoreTake(xSemaphoreSerial, 0);
135     Serial.print("TASK: ");
136     Serial.print(pcTaskGetTaskName(NULL));
137     Serial.print(" min free stack: ");
138     Serial.println(uxTaskGetStackHighWaterMark(NULL));
139     xSemaphoreGive(xSemaphoreSerial);
140 #endif
141
142     xLastWakeTime = xTaskGetTickCount();
143   }
144
145   vTaskDelete( NULL );
146 }
147
148 //Task for reading Sonar, Ibus, encoder
149 void read_trigger(void* parameters)
150 {
151   TickType_t xLastWakeTime;
152   const TickType_t xFrequency = pdMS_TO_TICKS( 20 );
153   // Initialise the xLastWakeTime variable with the current time.
154   xLastWakeTime = xTaskGetTickCount();
155   long lastcount = 0, lastUpdatemicros = 0;
156   for ( ;; )
157   {
158     vTaskDelayUntil( &xLastWakeTime, xFrequency );
159     sonar.ping();
160 #if POSITION == 0
161     ibus.read();
162 #endif
163     float delta = (micros() - lastUpdatemicros)/1000;
164     long count = encoder.getCount();
165     car[POSITION].speed = (count - lastcount)/delta;
166     car[POSITION].sonar = sonar.value();
167     lastcount = count;
168     lastUpdatemicros = micros();
169
170 #ifdef MEMDEBUG
171     xSemaphoreTake(xSemaphoreSerial, 0);
172     Serial.print("TASK: ");
173     Serial.print(pcTaskGetTaskName(NULL));
174     Serial.print(" min free stack: ");
175     Serial.println(uxTaskGetStackHighWaterMark(NULL));
176     xSemaphoreGive(xSemaphoreSerial);
177 #endif
178
179     xLastWakeTime = xTaskGetTickCount();
```

```
180   }
181   vTaskDelete( NULL );
182 }
183
184 void send_data(void* parameters){
185   TickType_t xLastWakeTime;
186
187 #if POSITION == 0
188   const TickType_t xFrequency = pdMS_TO_TICKS( 10 );
189 #elif POSITION == 1
190   const TickType_t xFrequency = pdMS_TO_TICKS( 1000 );
191 #endif
192   // Initialise the xLastWakeTime variable with the current time.
193   xLastWakeTime = xTaskGetTickCount();
194   for ( ;; )
195   {
196     vTaskDelayUntil( &xLastWakeTime, xFrequency );
197 #if POSITION == 0
198     dataSent data2Send;
199     data2Send.ibus2= ibus.value(2);
200     data2Send.ibus4= ibus.value(4);
201     data2Send.ibus5= ibus.value(5);
202     data2Send.ibus6= ibus.value(6);
203     data2Send.ibus8= ibus.value(8);
204     data2Send.ibus10= ibus.value(10);
205     data2Send.y = (float) car[0].imu.y;
206     data2Send.timestamp = car[0].timestamp;
207     data2Send.speed = car[0].speed;
208     data2Send.cmd = car[0].cmd;
209     esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &data2Send,
          sizeof(dataSent));
210
211 #elif POSITION == 1
212     outgoingMsg.hop = 1;
213     outgoingMsg.a = millis();
214     // Send message via ESP-NOW
215  // Serial.println("sent delay request");
216     esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &outgoingMsg,
          sizeof(outgoingMsg));
217 #endif
218
219 #ifdef MEMDEBUG
220     xSemaphoreTake(xSemaphoreSerial, 0);
221     Serial.print("TASK: ");
222     Serial.print(pcTaskGetTaskName(NULL));
223     Serial.print(" min free stack: ");
224     Serial.println(uxTaskGetStackHighWaterMark(NULL));
225     xSemaphoreGive(xSemaphoreSerial);
226 #endif
```

```
227
228     xLastWakeTime = xTaskGetTickCount();
229   }
230   vTaskDelete( NULL );
231 }
232
233 void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
234   // len 20 for delay msg 28 for data
235   if(len == 28){
236     dataSent data2Receive;
237     if(xSemaphoreTake(xSemaphoreCarData, 0)==pdTRUE){
238       memcpy(&data2Receive, incomingData, sizeof(dataSent));
239       car[0].imu.y = data2Receive.y;
240       car[0].timestamp = data2Receive.timestamp + car[0].offset;
241       car[0].speed = data2Receive.speed;
242       car[0].cmd = data2Receive.cmd;
243       int ch[14] = {0,data2Receive.ibus2,0,data2Receive.ibus4,data2Receive.ibus5,
              data2Receive.ibus6,0,data2Receive.ibus8,0,data2Receive.ibus10,0,0,0,0};
244       ibus.addValues(ch);
245     }
246     xSemaphoreGive(xSemaphoreCarData);
247   }
248   else if(len == 20){
249     // delay msgs
250     memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
251     unsigned long timeReceive = millis();
252     if(incomingReadings.hop == 2){
253       // 2nd hop of the delay msg ( we are in the client )
254       incomingReadings.b = timeReceive;
255       car[0].delay=incomingReadings.b - incomingReadings.a - incomingReadings.y +
              incomingReadings.x;
256       car[0].offset = incomingReadings.x - incomingReadings.a - car[0].delay/2;
257
258 #ifdef DEBUG
259       Serial.print("delay: ");
260       Serial.print(car[0].delay);
261       Serial.print(", offset: ");
262       Serial.println(car[0].offset);
263 #endif
264
265     }
266     else{
267       // 1st hop of the delay msg ( we are in the server )
268       incomingReadings.hop = 2;
269       incomingReadings.x = timeReceive;
270       incomingReadings.y = millis();
271       esp_now_send(broadcastAddress, (uint8_t *) &incomingReadings,
              sizeof(incomingReadings));
272     }
```

```
273   }
274 }
275
276 void setup(void)
277 {
278   driver.enable(false);
279    ledcSetup(1, 50, 16); // channel 1, 50 Hz, 16-bit width
280    ledcAttachPin(2, 1); // GPIO 2 assigned to channel 1
281    ledcWrite(1,4900);
282
283   Serial.begin(115200);
284   Serial.println("Car V1"); Serial.println("");
285   Serial.print("MAC: "); Serial.println(WiFi.macAddress());
286   Serial.print("POSITION: "); Serial.println(POSITION);
287
288 #ifndef IGNOREERROR
289   if(logger.cardError==true){
290     while(true){
291       Serial.println("No sd card detected");
292       delay(1000);
293     }
294   }
295 #endif
296
297   xSemaphoreSerial = xSemaphoreCreateMutex();
298   xSemaphoreCarData = xSemaphoreCreateMutex();
299   //------------------------ IMU ------------------------
300   /* Initialise the sensor */
301   if(!bno.begin())
302   {
303     /* There was a problem detecting the BNO055 ... check your connections */
304     Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
305 #ifndef IGNOREERROR
306     while(1);
307 #endif
308   }
309
310   delay(1000);
311   /* Use external crystal for better accuracy */
312   bno.setExtCrystalUse(true);
313
314   encoder.attachFullQuad(EncAPin, EncBPin);
315
316   // Init ESP-NOW Wifi protocol
317   WiFi.mode(WIFI_STA);
318
319   if (esp_now_init() != ESP_OK) {
320     Serial.println("Error initializing ESP-NOW");
321     while(1);
```

```
322 }
323
324 // Register peer
325 esp_now_peer_info_t peerInfo;
326 memcpy(peerInfo.peer_addr, broadcastAddress, 6);
327 peerInfo.channel = 0;
328 peerInfo.encrypt = false;
329
330 // Add peer
331 if (esp_now_add_peer(&peerInfo) != ESP_OK){
332   Serial.println("Failed to add peer");
333   while(1);
334 }
335 // Register for a callback function that will be called when data is received
336 esp_now_register_recv_cb(OnDataRecv);
337
338 //----------------------- TASKS ----------------------
339
340 xTaskCreate(
341               IMU, /* Task function. */
342               "imu", /* String with name of task. */
343               2000, /* Stack size in words. */
344               NULL, /* Parameter passed as input of the task */
345               10, /* Priority of the task. */
346               &imuTask); /* Task handle. */
347
348 xTaskCreate(
349               read_trigger, /* Task function. */
350               "read_trigger", /* String with name of task. */
351               900, /* Stack size in words. */
352               NULL, /* Parameter passed as input of the task */
353               3, /* Priority of the task. */
354               &readTask); /* Task handle. */
355
356 xTaskCreate(
357               test_script, /* Task function. */
358               "script", /* String with name of task. */
359               15000, /* Stack size in words. */
360               NULL, /* Parameter passed as input of the task */
361               1, /* Priority of the task. */
362               NULL); /* Task handle. */
363
364 xTaskCreate(
365               send_data, /* Task function. */
366               "send_data", /* String with name of task. */
367               15000, /* Stack size in words. */
368               NULL, /* Parameter passed as input of the task */
369               1, /* Priority of the task. */
370               &sendTask); /* Task handle. */
```

```
371 }
372
373 void loop(void)
374 {
375   ledcWrite(1,ibus.value(4)*2.6+1300);
376   vTaskDelay(20/portTICK_PERIOD_MS);
377 }
```

B.2 config.h

```
1  #ifndef CONFIG
2  #define CONFIG
3
4  #define clockPIN 27
5  #define latchPIN 14
6  #define dataPIN 26
7
8  #define EncAPin 34
9  #define EncBPin 35
10
11 #define rxRcPIN 13
12 #define txRcPIN 4
13
14 // SD card cs
15 #define SD_CS 5
16
17 #define US015TRIGGER 32
18 #define US015ECHO 33
19
20 #define POSITION 0 // Position of the car 0 = leader
21
22 #if POSITION == 1
23 uint8_t broadcastAddress[] = {0x4C, 0x11, 0xAE, 0x71, 0x17, 0x68}; // MAC Address
       of 1st car
24 #elif POSITION == 0
25 uint8_t broadcastAddress[] = {0x4C, 0x11, 0xAE, 0xDF, 0x95, 0x4C}; // MAC Address
       of 2ndcar
26 #endif
27
28 // Disable sd card and imu checks
29 //#define IGNOREERROR
30
31 //#define DEBUG
32 //#define MEMDEBUG
33
34 struct dataSent
35 {
36   float y; //4
37   unsigned long timestamp; //4
```

```
38   int16_t ibus2, ibus4, ibus5, ibus6, ibus10, ibus8;
39   float speed; //4
40   char cmd; //2
41 };
42
43 struct delay_msg {
44   unsigned long a;
45   unsigned long b;
46   unsigned long x;
47   unsigned long y;
48   char hop;
49 };
50
51
52 #endif
```

Anexo C

Bibliotecas

C.1 sonar

C.1.1 sonar.h

```
#ifndef SONAR_h
#define SONAR_h
#if (ARDUINO >= 100)
 #include <Arduino.h>
#else
 #include <WProgram.h>
#endif
class US015SONAR {
public:
  US015SONAR(byte tiggerPin, byte echoPin);
  ~US015SONAR();
  long value();
  void ping();
private:
  static US015SONAR *instance;
  void IRAM_ATTR classISRHandler();
  static void IRAM_ATTR classInterruptHandler();
  int us015_mode=0;
  long us015_time=0;
  long us015_width = 0;
  byte _triggerPin, _echoPin;
};
#endif
```

C.1.2 sonar.cpp

```
#include "sonar.h"

US015SONAR* US015SONAR::instance = NULL;
```

```
4
5  US015SONAR::US015SONAR(byte triggerPin, byte echoPin) {
6    instance = this;
7    _triggerPin = triggerPin;
8    _echoPin = echoPin;
9    pinMode(_echoPin, INPUT);
10   attachInterrupt(digitalPinToInterrupt(_echoPin), classInterruptHandler, CHANGE);
11   pinMode(_triggerPin, OUTPUT);
12 }
13
14 US015SONAR::~US015SONAR() {
15 }
16
17 void US015SONAR::ping(){
18    digitalWrite(_triggerPin, HIGH);
19    delayMicroseconds(10);
20    digitalWrite(_triggerPin, LOW);
21 }
22
23 long US015SONAR::value(){
24   return us015_width;
25 }
26
27 void US015SONAR::classISRHandler() {
28   //Serial.println(us015_mode);
29   if(us015_mode == 0){
30     us015_time = esp_timer_get_time();
31     us015_mode = 1;
32   }
33   else if(us015_mode == 1){
34     us015_width = ((esp_timer_get_time()-us015_time)/ 2) * 0.34;
35     us015_mode = 0;
36   }
37 }
38
39 void US015SONAR::classInterruptHandler() {
40    instance->classISRHandler();
41 }
```

C.2 sd_logger

C.2.1 sd_logger.h

```
#ifndef _SDlogger_h
#define _SDlogger_h
#include <Arduino.h>
#include <FS.h>
#include <SD.h>
#include <car.h>

struct carData {
  CAR car;
  long timestamp;
};

class sd_logger{
private:
  File file;
  String fileName;
  QueueHandle_t queueIMU;
  int _queueSize = 1000;
  void appendFile( const char * message);
  void writeFile(const char * path, const char * message);
  void openFile(const char * timestamp);
  void closeFile(void);

public:
  sd_logger(byte CS, int queueSize);
  ~sd_logger();
  void push(CAR data);
  void clear();
  void writeAll();
  boolean cardError = true;
  boolean log = false;
};




#endif
```

C.2.2 sd_logger.cpp

```
#include "sd_logger.h"


sd_logger::sd_logger(byte CS, int queueSize)
```

```
{
  _queueSize = queueSize;
  if(!SD.begin(CS)) {
    cardError = true;
    //return;
  }
  else{
    cardError = false;
    queueIMU = xQueueCreate( _queueSize, sizeof( CAR ) );
  }
}

sd_logger::~sd_logger()
{
  SD.end();
}

void sd_logger::appendFile( const char * message) {
  if(!file) {
    return;
  }
  file.print(message);
}

void sd_logger::writeFile(const char * path, const char * message) {
  file = SD.open(path, FILE_WRITE);
  if(!file) {
    return;
  }
  file.print(message);
  file.close();
}

void sd_logger::openFile(const char * path){
  int i = 1;
  fileName = path;
  while (SD.open(fileName.c_str()))
  {
    // File exists try another name
    fileName = path + String(i);
    i++;
  }
  writeFile(fileName.c_str(), "mS, sonar, speed, cmd, x, y, z \r\n");
  file = SD.open(fileName.c_str(), FILE_APPEND);
}

void sd_logger::closeFile(void){
  file.close();
}
```

```
54
55 void sd_logger::push(CAR data){
56   if(uxQueueSpacesAvailable(queueIMU)>0){
57     xQueueSend(queueIMU, &data, 0);
58   }
59   else{
60     Serial.println("queue full");
61   }
62 }
63
64 void sd_logger::writeAll(){
65   openFile("/test");
66   while(uxQueueMessagesWaiting(queueIMU)>0){
67     CAR element;
68     xQueueReceive(queueIMU, &element, 0);
69     String msg = "\"" +\
70                 String(element.timestamp) + "\", \"" + \
71                 String(element.sonar) + "\", \"" + \
72                 String(element.speed) + "\", \"" + \
73                 String(element.cmd) + "\", \"" + \
74                 String(element.imu.x) + "\", \"" + \
75                 String(element.imu.y) + "\", \"" + \
76                 String(element.imu.z) + "\"\r\n";
77     appendFile(msg.c_str());
78   }
79   closeFile();
80 }
81
82 void sd_logger::clear(){
83   xQueueReset(queueIMU);
84 }
```

C.3 ibus

C.3.1 ibus.h

```
#ifndef IBUS_h
#define IBUS_h

#if (ARDUINO >= 100)
 #include <Arduino.h>
#else
 #include <WProgram.h>
#endif

#include <car.h>

class IBUS {
public:
  IBUS (byte rxPin, byte txPin);
  IBUS (void);
  void read();
  void addValues( int ch[14]);
  int value(byte channel);
  ~IBUS();

private:
  int rcValue[14] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  unsigned long lastreceive = 0;
};

#endif
```

C.3.2 ibus.cpp

```
#include "ibus.h"

IBUS::IBUS(byte rxPin, byte txPin) {
  Serial1.begin(115200,SERIAL_8N1, rxPin, txPin, false, 2);
}

IBUS::IBUS(void) {
}

IBUS::~IBUS() {
  Serial1.end();
}

void IBUS::read(){
  byte data[31] = {0};
```

```
16   int rcValuetmp[14] = {0};
17   if(Serial1.available()){
18     lastreceive = millis();
19     if(Serial1.peek() == 0x55){
20       if(Serial1.available()>= 31){
21         Serial1.readBytes(data,31);
22
23         rcValuetmp[ 0] = (data[ 2] << 8) + data[ 1];
24         rcValuetmp[ 1] = (data[ 4] << 8) + data[ 3];
25         rcValuetmp[ 2] = (data[ 6] << 8) + data[ 5];
26         rcValuetmp[ 3] = (data[ 8] << 8) + data[ 7];
27         rcValuetmp[ 4] = (data[10] << 8) + data[ 9];
28         rcValuetmp[ 5] = (data[12] << 8) + data[11];
29         rcValuetmp[ 6] = (data[14] << 8) + data[13];
30         rcValuetmp[ 7] = (data[16] << 8) + data[15];
31         rcValuetmp[ 8] = (data[18] << 8) + data[17];
32         rcValuetmp[ 9] = (data[20] << 8) + data[19];
33         rcValuetmp[10] = (data[22] << 8) + data[21];
34         rcValuetmp[11] = (data[24] << 8) + data[23];
35         rcValuetmp[12] = (data[26] << 8) + data[25];
36         rcValuetmp[13] = (data[28] << 8) + data[27];
37
38         int checksum = (data[30] << 8) + data[29];
39         int myChecksum = 0;
40         for( int c=0; c<14; c++)
41           myChecksum += rcValuetmp[c];
42
43         if( myChecksum == checksum ){
44           memcpy(rcValue, rcValuetmp, sizeof(rcValue[0])*14);
45         }
46
47       }
48     }
49     else Serial1.flush();
50   }
51   else if( millis()-lastreceive>=1000){
52     static const int rcValueClear[14] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0};
53     memcpy(rcValue, rcValueClear, sizeof(rcValue[0])*14);
54   }
55 }
56
57 int IBUS::value(byte channel){
58   return rcValue[channel-1];
59 }
60
61 void IBUS::addValues(int ch[14]){
62   memcpy(rcValue, ch, sizeof(rcValue[0])*14);
63 }
```

C.4 drv8828

C.4.1 drv8828.h

```
#ifndef DRV8828_h
#define DRV8828_h

#if (ARDUINO >= 100)
 #include <Arduino.h>
#else
 #include <WProgram.h>
#endif

class DRV8828 {
public:
  DRV8828 (byte clkPin, byte latchPin, byte dataPin);
  void enable(boolean mode);
  void direction(boolean mode);
  void binary(byte torque);
  int cmd();
  void shiftsend();
  ~DRV8828();

private:
  byte _latchPin, _clkPin, _dataPin;
  byte _shiftBuffer;
  int _binary = 0;
  boolean enabled = false;
  void shiftOutSlow(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, byte val);
  SemaphoreHandle_t xSemaphoreShift = NULL;
};

#endif
```

C.4.2 drv8828.cpp

```
#include "drv8828.h"

DRV8828::DRV8828(byte clkPin, byte latchPin, byte dataPin) {
  _clkPin = clkPin;
  _latchPin = latchPin;
  _dataPin = dataPin;
  pinMode(_latchPin, OUTPUT);
  pinMode(_clkPin, OUTPUT);
  pinMode(_dataPin, OUTPUT);
  _shiftBuffer = 0b00000000;
  xSemaphoreShift = xSemaphoreCreateMutex();
}
```

```
13
14 DRV8828::~DRV8828() {
15 }
16
17 void DRV8828::enable(boolean mode){
18  enabled = mode;
19  bitWrite(_shiftBuffer, 7, mode); // nReset
20  bitWrite(_shiftBuffer, 1, mode); // enable
21
22 }
23 void DRV8828::direction(boolean mode){
24  bitWrite(_shiftBuffer, 0, mode); // phase
25 }
26 void DRV8828::binary(byte torque){
27  _binary = torque;
28  torque = torque * 31/255;
29  if (torque> 31)
30    torque = 31;
31  if (torque < 0)
32    torque = 0;
33  for(int i = 0; i < 5; i++){
34    bitWrite(_shiftBuffer, i+2, bitRead(torque,i));
35  }
36 }
37
38 void DRV8828::shiftsend(){
39  if( xSemaphoreTake(xSemaphoreShift, 10/portTICK_PERIOD_MS) == pdTRUE){
40    digitalWrite(_latchPin, LOW);
41    delayMicroseconds(1);
42    shiftOutSlow(_dataPin, _clkPin, LSBFIRST, _shiftBuffer);
43    delayMicroseconds(1);
44    digitalWrite(_latchPin, HIGH);
45  }
46  xSemaphoreGive(xSemaphoreShift);
47 }
48
49 int DRV8828::cmd(){
50  return enabled*_binary;
51 }
52
53 void DRV8828::shiftOutSlow(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder,
      byte val)
54 {
55  int i;
56
57  for (i = 0; i < 8; i++) {
58    if (bitOrder == LSBFIRST)
59      digitalWrite(dataPin, !!(val & (1 << i)));
60    else
```

```
61      digitalWrite(dataPin, !!(val & (1 << (7 - i))));
62    delayMicroseconds(1);
63    digitalWrite(clockPin, HIGH);
64    delayMicroseconds(1);
65    digitalWrite(clockPin, LOW);
66  }
67 }
```

C.5 car

C.5.1 car.h

```
#ifndef CAR_h
#define CAR_h

#if (ARDUINO >= 100)
 #include <Arduino.h>
#else
 #include <WProgram.h>
#endif

#include <utility/imumaths.h>

struct imuVector
{
  double x;
  double y;
  double z;
};

class CAR {
public:
  CAR();
  ~CAR();
  imuVector imu;
  float speed;
  int sonar;
  int cmd;
  long offset;
  int delay;
  unsigned long timestamp;
  void newImu(imu::Vector<3> newimu);
private:
};

#endif
```

C.5.2 car.cpp

```
#include "car.h"

CAR::CAR() {
  imu.x = 0;
  imu.y = 0;
  imu.z = 0;
}
```

```
8
9  CAR::~CAR() {
10 }
11
12 void CAR::newImu(imu::Vector<3> newimu){
13   imu.x = (imu.x + newimu.x())/2;
14   imu.y = (imu.y + newimu.y())/2;
15   imu.z = (imu.z + newimu.z())/2;
16 }
17
18 int CAR::ibus(char channel){
19   return _ibus[channel -1];
20 }
```