UNIVERSIDADE FEDERAL DO PARANÁ

EDUARDO HENRIQUE FIDELES RIBEIRO
WIVIANE CAROLINE MANEIRA

SISTEMA DE CONTROLE PARA ESTERÇAMENTO VEÍCULAR VISANDO APLICAÇÃO NA INDÚSTRIA CANAVIEIRA

CURITIBA
2019

EDUARDO HENRIQUE FIDELES RIBEIRO
WIVIANE CAROLINE MANEIRA

SISTEMA DE CONTROLE PARA ESTERÇAMENTO VEÍCULAR VISANDO APLICAÇÃO NA INDÚSTRIA CANAVIEIRA

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Engenharia Elétrica da Universidade Federal do Paraná como parte das exigências para a obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Roman Kuiava

Coorientador: Prof. Dr. Henri Frederico Eberspacher

CURITIBA
2019

"O prazer no trabalho aperfeiçoa a obra".

Aristóteles

AGRADECIMENTOS

A Equipe agradece ao orientador Prof. Dr. Roman Kuiava, e ao coorientador Prof. Dr. Henri Frederico Eberspacher, pela oportunidade de realizar este trabalho, pelos conselhos e conhecimentos transmitidos, pelo compromisso e comprometimento em suas orientações ao longo do desenvolvimento deste projeto.

A Equipe agradece os amigos Leandro, Lin, Luciane, Matheus, Rafaela, e Renato, pela amizade, pelo companheirismo, pelo auxílio em diversas e intermináveis provas, e por tornarem a jornada mais enriquecedora e prazerosa.

Eu, Eduardo Henrique Fideles Ribeiro, agradeço à minha família, em especial aos meus pais, Benedito e Rosineide, por me apoiar em todos os momentos durante a jornada na universidade, incentivando a dar o meu melhor o tempo todo. Agradeço a minha namorada Rafaela, pelo carinho, por estar ao meu lado durante essa etapa da vida e também por todo o apoio e colaboração durante a elaboração desse projeto. Agradeço também aos colegas da empresa Lactec, por todo o apoio, dicas e sugestões durante o desenvolvimento do projeto, fazendo com que pudesse enfrentar o desafio do projeto e atingir os objetivos especificados.

Eu, Wiviane Caroline Maneira, agradeço à minha família, em especial aos meus pais, Wilmar e Sônia, por me apoiarem nessa trajetória e em tantas outras, por serem minha base e meu porto seguro. Agradeço ao meu namorado Bruno, pelo carinho, pelo incentivo, e por estar ao meu lado em todos os momentos. E por fim, agradeço aos colegas de trabalho da Volvo do Brasil, pela enorme contribuição em minha carreira profissional, e por estarem dispostos a colaborar na realização deste projeto.

RESUMO

RIBEIRO, Eduardo; MANEIRA, Wiviane. Sistema de Controle para Esterçamento Veícular Visando Aplicação na Indústria Canavieira. 2019. 113 f. Trabalho de Conclusão de Curso – , Universidade Federal do Paraná. CURITIBA, 2019.

O presente trabalho apresenta o desenvolvimento de um sistema de esterçamento veícular, baseado em uma rota pré-definida pelo usuário do veículo através de uma interface *human-machine*, visando aplicação na indústria canavieira. O sistema consite no controle angular de um motor elétrico sem escovas, em que se utiliza um motor elétrico trifásico de 24V DC. Para o acionamento do motor, foi utilizado um *driver* constituído por transistores MOSFETs em estrutura de ponte, onde cada transistor é controlado por sinais PWM, fazendo com que a tensão DC de entrada seja convertida para uma tensão de acionamento modulada para cada fase do motor. Para o controle do *driver* é utilizado um processador ARM-Cortex M4, que controla a velocidade e direção de rotação do motor. Neste projeto também é verificado a posição angular do motor, realizando leituras do sensor de posição através de barramento CAN de comunicação, o qual foi implementado através de comunicação SPI com o circuito integrado MCP2515. Este último converte os dados recebidos do padrão CAN para o padrão SPI. Para a definição do ângulo que o motor deve ser posicionado, é verificado através de um aplicativo *Android* a localização fornecida pelo GPS e a pré-gravação da rota desejada, podendo assim calcular a direção do movimento e verificar a posição angular que deve ser estabelicida pelo motor. Após finalizado os cálculos, os dados de ângulo do motor são transmitidos do aplicativo para o processador com o uso de comunicação *Bluetooth*. Tendo recebido os dados, o processador verifica com o sensor de posição angular, o ângulo atual do motor e compara com o ângulo desejado, para então acionar o *driver* e posicionar o motor na posição angular desejada. Para um melhor posicionamento angular do motor, foi projetado o controle PID do sistema, onde realiza o controle de velocidade do motor de acordo com a posição angular desejada do sistema. Os resultados obtidos do controlador do projeto se mostraram satisfatórios e de acordo com a aplicação proposta, assim como os códigos desenvolvidos para controle dos periféricos apresentaram o correto funcionamento.

Palavras-chave: motor elétrico sem escovas, barramento CAN, controle PID, controle de ângulo de esterçamento.

ABSTRACT

RIBEIRO, Eduardo; MANEIRA, Wiviane. Vehicle Steering Control System Aiming for Application in the Sugar Cane Industry. 2019. 113 f. Trabalho de Conclusão de Curso – , Universidade Federal do Paraná. CURITIBA, 2019.

The aim of this work is the development of a vehicle steering system, based on a predefined route by the vehicle user through a human-machine interface, aiming application in the sugarcane industry. The system consists of the angular control of a brushless electric motor, where a 24V DC three-phase electric motor is used. To drive the motor, a bridge structure mosfet driver was used, where each mosfet is controlled by PWM signals, causing the input DC voltage to be converted to a modulated drive voltage for each motor phase. To control the driver is used an ARM-Cortex M4 processor, which controls the speed and direction of motor rotation. In this project, is also verified the motor angular position, taking position sensor readings through CAN bus communication, which was implemented through SPI communication with the integrated circuit MCP2515. The integrated circuit MCP2515 converts the data received from the CAN standard to the SPI standard. To define the angle that the engine should be positioned, it's verified through an Android application the location provided by the GPS and the pre-recorded route desired, thus being able to calculate the movement direction and to verify the angular position that must be established by the motor. After calculations are completed, motor angle data is transmitted from the application to the processor using Bluetooth communication. Having received the data, the processor checks with the angular position sensor the current motor angle and compares it with the desired angle, then drives the driver and positions the motor at the desired angular position. For better motor angular positioning, the system PID control was designed, where it performs the motor speed control according to the desired angular position of the system. The results obtained from the project controller were satisfactory according to the proposed application, as well as the codes developed to control the peripherals presented the correct functioning.

Keywords: brushless electric motor, CAN bus, PID control, Steering angle control.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PID	Proporcional Integral Derivativo
GPS	*Global Positioning System*
RTK	*Real Time Kinematic*
IHM	Interface Homem-Máquina
MCU	*Motor Control Unit*
USB	*Universal Serial Bus*
HDMI	*High-Definition Multimedia Interface*
GPIO	*General Propose Input-Output*
LAN	*Local Area Network*
Wi-Fi	*Wireless Fidelity*
ADC	*Analog Digital Converter*
RAM	*Random Access Memory*
SRAM	*Static Random Access Memory*
UART	*Universal Asynchronous Receiver Transmitter*
IDE	*Integrated Development Environment*
PWM	*Pulse Width Modulation*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
SPI	*Serial Peripheral Interface*
OTG	*On-The-Go*
CAN	*Controller Area Network*
ECU	*Engine Control Unit*
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
CS	*Chip Select*
CLK	*SPI clock*

MISO	*Master Input Slave Output*
MOSI	*Inter-Intergrated Circuit*
MOSI	*Master Output Slave Input*
MISO	*Master Input Slave Output*
CI	*Circuito Integrado*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto

Veículos autônomos são denominados como veículos que apresentam sistemas que permitem uma maior assistência ao motorista ao executar a condução. Tais sistemas podem ter diversos níveis de automação, podendo ser apenas um sensor de estacionamento, até mesmo veículos que são capazes de seguir um trajeto sem a interferência humana (PISSARDINI, 2013).

Na figura 1, é possível observar os níveis de automação encontrados hoje em dia no mercado. Para o nível 0, não existem automação. Para o nível 1, são considerados os sistemas que tem função de auxiliar o motorista, como sensores de estacionamento ou sensores de ponto-cego presentes em espelhos retrovisores.

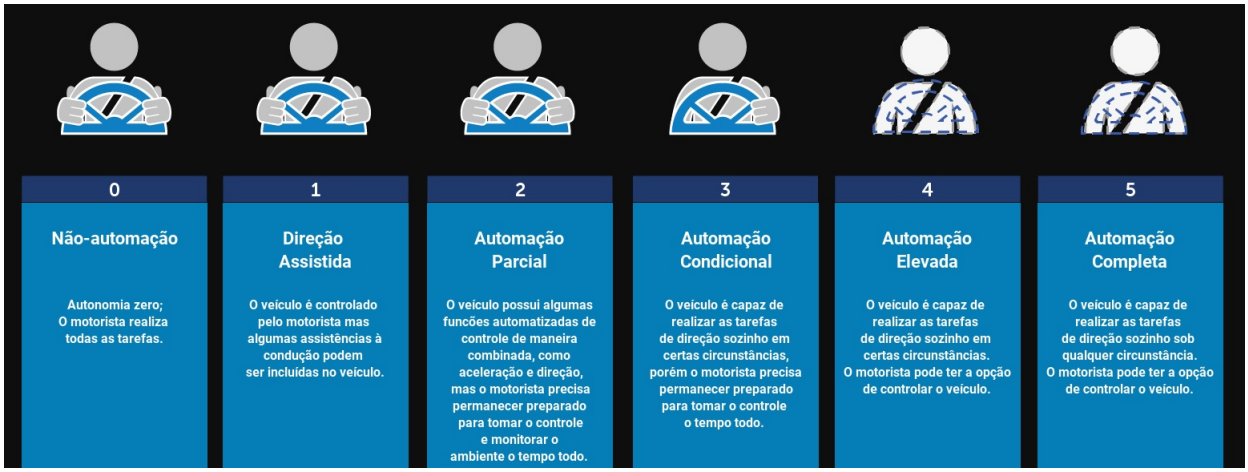

Figura 1 – Níveis de automação encontrados no mercado – Fonte: (PISSARDINI, 2013)

No nível 2 de automação, tem-se uma parcialidade em automação, ou seja, o veículo pode ter funções para automatizar alguns controles, como controle de velocidade ou controle de direção. Ao chegar ao nível 3, tem-se uma automação condicional, sendo que o veículo é capaz de realizar algumas tarefas sozinho, porém é necessária a presença do motorista para tomar o controle da situação caso haja alguma falha.

Por fim, no nível 5 de automação, tem-se a automação completa, ou seja, o veículo é capaz de realizar as tarefas determinadas em qualquer circunstância, sem a obrigatoriedade da presença do motorista. Deve-se lembrar que, ao considerar veículos com automação nível 0 até automação nível 4, o motorista é indispensável na utilização do veículo. 3

Devido à falta de uma infra-estrutura adequada nas cidades para circulação de veículos totalmente autônomos, ou seja, nível 5, as fabricantes procuram adicionar gradualmente tecnologias que colaboram com os motoristas, como, por exemplo, assistência na direção e estacionamento, administração de frenagem, alertas de proximidade com outros veículos e

adaptação de condução. Além disso, também é possível citar ajuste de velocidade conforme condições do trajeto, sistema de detecção de obstáculos e usuários nas vias.

Os veículos que possuem tecnologias as quais colaboram com os motoristas, podem ser utilizados em diversas aplicações, abrangendo veículos de passeio, veículos comerciais, veículos militares, veículos de construção, e também veículos agrícolas.

Exemplificando o transporte de cargas, pode-se citar VERA, veículo elétrico da empresa Volvo Trucks, o qual é controlado e monitorado através de um centro de controle e tem o potencial de tornar o transporte mais seguro, mais limpo e mais eficiente. Na figura 2, é possível observar o veículo autônomo. Tal veículo faz parte de uma solução integrada para transportar mercadorias de um centro de logística para um terminal portuário em Gotemburgo, Suécia (VOLVO, 2019).

Figura 2 – Veículo autônomo para transporte de cargas – Fonte: (VOLVO, 2019)

1.2 Problemática

Ao falar da indústria canavieira, para que a colheita mecanizada seja um sucesso, é necessário que haja a remoção de tocos e pedras do solo, bem como seu nivelamento. Também é necessário haver o planejamento de áreas de manobra, espaçamento e sentido de plantio (MAGRO, 2019).

Mesmo com a implementação de colheita mecanizada, um dos maiores problemas é o pisoteio de linhas de plantio durante o cultivo e colheita de cana-de-açúcar, uma vez que exige-se a precisão de centímetros ao andar pelas demarcações (MAGRO, 2019).

Na figura 3, tem-se um exemplo de linhas de plantio, considerando uma plantação que utiliza um espaçamento de 190 cm com sulcos duplos. Nesse exemplo, tem-se uma vista frontal do espeçamento dos sulcos juntamente com a largura dos pneus do veículo que acompanha a colheitadeira.

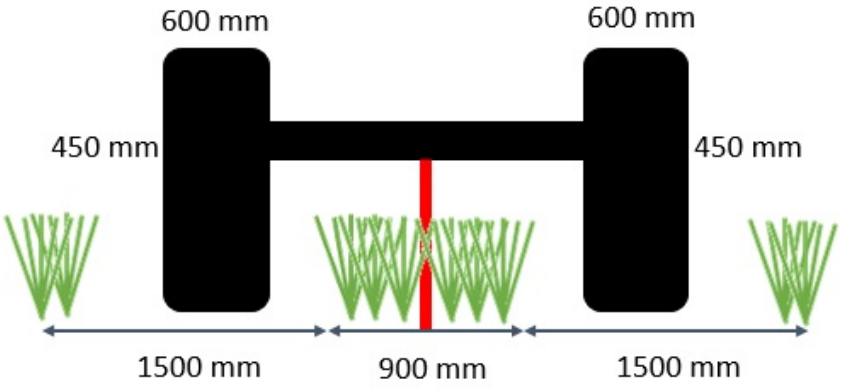

Figura 3 – Exemplo de espaçamento de linhas de plantio – Fonte: (MAGRO, 2019).

É possível observar ainda que a largura do pneu é de 600 mm. Como o espaçamento do sulco é de 1500 mm, o pneu deve ser esterçado no máximo 450 mm para cado lado do sulco, a fim de não haver o pisoteio da cana-de-açúcar.

Na figura 4, é possível observar a brotação danificada da cana-de-açucar na linha de plantio onde aconteceu o pisoteio.

Figura 4 – Brotação defeituosa onde houve o pisoteio da linha de plantio– Fonte: (MAGRO, 2019).

1.3 Objetivos Gerais

O objetivo geral deste projeto é desenvolver um sistema de controle de esterçamento, baseado em uma rota pré-definida pelo usuário do veículo através de uma interface *human-machine*, implementando tecnologias de automação e conectividade.

Este sistema deverá ser utilizado nas indústrias canavieiras, visando o não-pisoteamento das linhas de plantio de cana-de-açucar.

1.4 Objetivos Específicos

Tendo em vista o objetivo geral do projeto, foi elaborado os objetivos específicos estabelecidos para cada etapa de desenvolvimento do projeto.

- Desenvolver *hardware* de acionamento e controle do motor *brushless*;
- Desenvolver as rotinas de controle dos periféricos;
- Implementar o controlador PID;
- Desenvolver a interface *Android*;
- Implementar os protocolos de comunicação;
- Implementar a rotina de execução das rotas;
- Integralizar o sistema e executar testes.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentadas as teorias envolvidas no desenvolvimento deste projeto.

2.1 Identificação de Sistemas

A identificação de sistemas aborda a construção de modelos matemáticos da dinâmica de um sistema, de acordo com os dados, geralmente entrada e saída, obtidos de forma experimental(LJUNG, 1999).

Os modelos matemáticos resultantes são peças-chave para projetar um controlador. Para obter tais modelos, existem diversas técnicas, as quais constituem as modelagens. O modelo obtido é classificado de acordo com a modelagem utilizada, podendo ser do tipo caixa-preta, caixa branca ou caixa cinza (AGUIRE, 2007).

Na modelagem do tipo caixa-branca, a função matemática que descreve o comportamento dinâmico do sistema a ser modelado é pré-conhecida. O modelo é desenvolvido a partir dos princípios da Física e da Química que regem o sistema, a fim de empregar um modelo teórico que represente o sistema como um todo. Dessa forma, é considerado apenas as variáveis de entrada, e a física do sistema (AGUIRE, 2007).

Na modelagem do tipo caixa-preta, também conhecida como empírica, as variáveis que são utilizadas na identificação do sistema são apenas as entradas e saídas do sistema. Nesse caso, não é necessário ter conhecimento prévio sobre o sistema, pois essa modelagem utiliza relações de causa-efeito ao correlacionar os dados.

A modelagem do tipo caixa cinza é uma combinação das modelagens caixa-preta e caixa branca. Neste caso, para a identificação do sistema, é utilizado o conhecimento envolvendo os princípios físicos e químicos do sistema, mas também faz-se necessário as informações de entrada e saída disponíveis.

2.1.1 Etapas da Identificação de Sistemas

Para a correta identificação de um sistema, é necessário que seja feita a coleta e processamento de sinais de entrada e saída obtidos. Além disso, deve-se escolher um conjunto de modelos e selecionar o modelo que represente o sistema de maneira mais semelhante possível (LJUNG, 1999).

Na figura 5, tem-se ilustrado um fluxograma com as etapas primordiais para a identificação de sistemas. Basicamente, para identificação de um sistema, é necessário conduzir testes dinâmicos e coletar dados, e também escolher a representação matemática que será utilizada. A partir disso, é possível determinar a estrutura do modelo, estimar os parâmetros e validar o modelo.

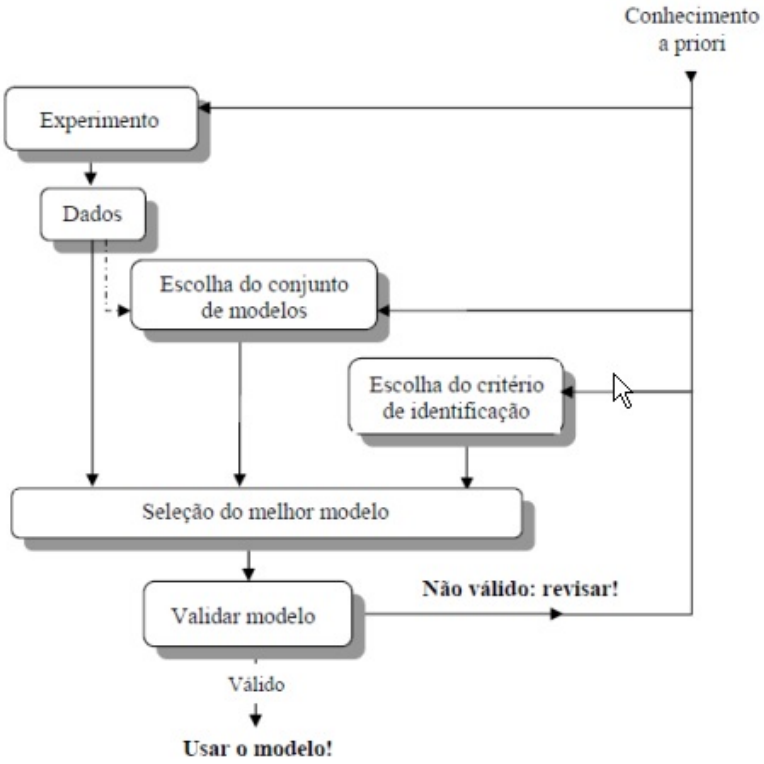

Figura 5 – Diagrama de blocos com as etapas de identificação de sistemas – Fonte: (ANTUNES, 2017)

2.1.1.1 Coleta de dados

Os dados que serão utilizados na identificação do sistema dinâmico é obtido através da análise da resposta do sistema a partir de um sinal responsável por excitar o sistema. Dessa forma, os dados obtidos garantem a informação desejada sobre o sistema que deseja-se modelar (AGUIRE, 2007).

Nesta etapa, é importante que os sinais de entrada, os quais são responsáveis pela excitação do sistema, e o tempo de amostragem, satisfaçam uma série de propriedades para assegurar que os dados obtidos sejam condizentes experimentalmente com o sistema. (RODRIGUES, 2000).

2.1.1.2 Escolha da representação do modelo

A partir de algumas características do sistema, como por exemplo, linearidade, a representação mais adequada para o modelo do sistema deve ser escolhida. Diante das possibilidades disponíveis, tem-se, para exemplo de representações, funções de transferência, equações diferenciais, modelos ARMAX, NARX, ARX, e NARMAX (AGUIRE, 2007).

A escolha de qual representação será utilizada deve ser condicionada a partir dos objetivos do modelo, das informações disponíveis sobre o sistema, e também das ferramentas que serão utilizadas para sua obtenção.

2.1.1.3 Escolha da estrutura do modelo

A próxima etapa trata-se em determinar a estrutura da representação. Essa estrutura será dimensionada de acordo com a representação, por exemplo, a partir do número de zeros e pólos considerando uma função de transferência, ou então número de termos considerando ARMAX (AGUIRE, 2007).

2.1.1.4 Estimação de parâmetros

Nesta etapa, é necessário a determinação de valores acordantes para os parâmetros do modelo. Utilizam-se como base, todas as informações obtidas nas etapas anteriores, adicionando-se a escolha de algoritmos a fim de estimar com mais precisão tais parâmetros. Para estimar os algoritmos que serão utilizados, é aplicado o método dos mínimos quadrados. Como resultado, obtém-se um modelo representativo para o sistema desejado.

2.1.1.5 Validação do modelo

Por fim, a partir da obtenção do modelo representativo para o sistema, passa-se à etapa de validação do modelo. Nesta etapa, tem-se validado a incorporação das características desejadas do sistema original.

O resultado obtido nesta etapa faz-se a partir de comparações entre a simulação do modelo obtido nas etapas anteriores e os dados reais do sistema. Se for de desejo estimar a qualidade do modelo obtido, deve-se utilizar diferentes grupos de dados para validação do modelo (AGUIRE, 2007).

2.2 Sistemas de controle

A ideia básica de controle é a manutenção de uma certa variável através de sua medida, ou seja, mede-se o valor atual da variável de interesse para que seja comparado com o valor desejado (*set-point*). A diferença entre esses valores (medido e desejado) é o desvio ou erro, e é utilizado para aplicar um sinal de correção ao sistema de modo a reduzir ou anular esse desvio (OLIVEIRA, 1999).

Em relação a natureza do controle, existem os auto-operados e os operados por alguma energia externa. Os controles auto-operados são aqueles em que toda energia necessária para seu funcionamento é obtida do próprio meio controlado. O controle de nível por boia é um exemplo de controle auto-operado. Já os operados por energia externa podem ser dos tipos pneumático, hidráulico ou elétrico/eletrônico (MATIAS, 2002).

Os sistemas de controle podem ser classificados em sistemas de malha aberta e sistemas de malha fechada. Os sistemas de malha aberta são aqueles em que a saída não possui influência sobre a entrada, ou seja, a saída não é medida e nem comparada com a entrada. Os sistemas de malha fechada são aqueles em que a saída possui influência sobre

a entrada, e portanto a saída é medida e comparada com a entrada (ARAÚJO, 2007). As figuras 6 e 7 mostram em diagramas de blocos os sistemas de malha aberta e malha fechada, respectivamente.

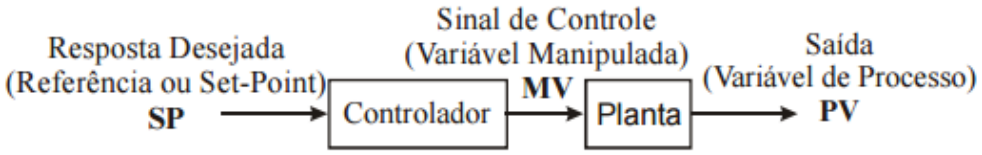

Figura 6 – Sistema de controle de malha aberta – Fonte: (ARAÚJO, 2007)

Os sistemas de controle podem, ainda, ser classificados em função da ação que exercem. Dessa forma, existem quatro tipos básicos de ações que podem ser exercidas: a primeira trata-se da ação de controle *ON-OFF*; a segunda trata-se do controle com ação proporcional (P); a terceira trata-se do controle com ação integral (I); e a quarta, trata-se do controle com ação derivativa (D). Também é possui obter uma ação mista, baseada nas combinações entre essas quatro ações (MATIAS, 2002).

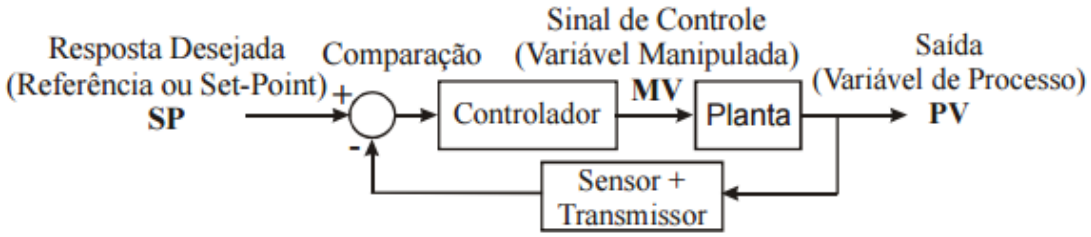

Figura 7 – Sistema de controle de malha fechada – Fonte: (ARAÚJO, 2007)

Os sistemas de controle podem ainda ser classificados em função da ação que exercem. Existem quatro tipos básicos de ações: ação de controle *ON-OFF*, controle com ação proporcional (P), controle com ação integral (I), controle com ação derivativa (D), além de combinações dessas ações (MATIAS, 2002).

No caso deste projeto deseja-se aplicar um controle PID, então para melhor entendimento foi dado enfoque a este modelo de controle. Assim, o subtópico a seguir descrevem o funcionamento do controle PID.

2.2.1 Controle PID

Controladores PID são largamente utilizados nas indústrias, pois podem ser aplicados na maioria dos sistemas de controle. Quando o modelo matemático de um sistema não é conhecido, métodos analíticos não podem ser utilizados, e controladores PID são mais úteis. Embora um controlador PID não possa, em muitas situações, proporcionar um controle ótimo, ele realiza um controle satisfatório (OGATA, 2011).

A figura 8 mostra um diagrama de blocos de um sistema qualquer que implementa uma ação PID de controle. A variável do processo pv(t) é subtraída da entrada de referência

r(t) para criar o erro e(t). O erro é multiplicado pelos termos P, I e D, que estiverem ativados, e são somados para dar origem ao sinal u(t), que é enviado para a saída do controlador. O atuador característico do sistema irá agir conforme o sinal u(t) (SILVEIRA, 2016).

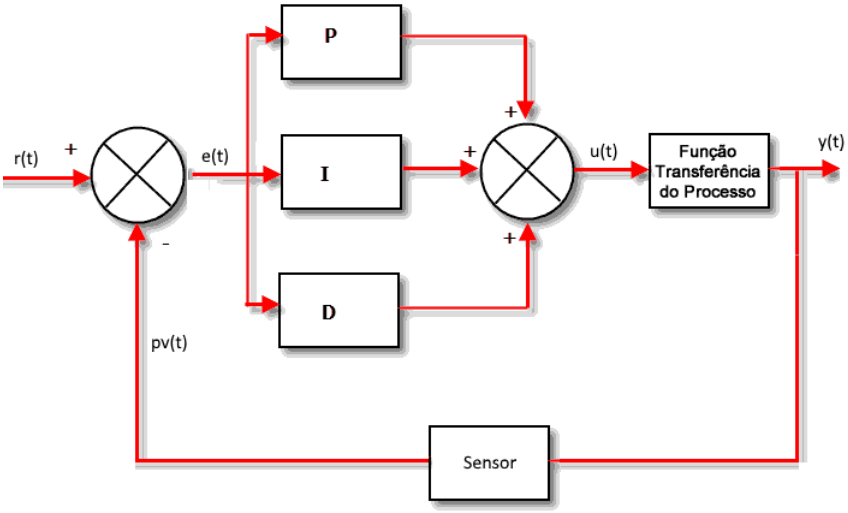

Figura 8 – Diagrama de blocos de controle de um processo – Fonte: (SILVEIRA, 2016)– modificado

Várias combinações dos termos P, I e D podem ser implementadas. A combinação mais utilizada na indústria é a ação proporcional PI. A ação P e a ação PID são as vezes implementadas, sendo que a ação PD é raramente utilizada, mas é útil para o controle de servomotores. A seguir, serão explicados cada uma dessas ações, bem como suas combinações (SILVEIRA, 2016).

2.2.1.1 Ação de controle P

Com apenas o controle proporcional implementado, o sinal de erro seria multiplicado apenas pelo ganho proporcional K_p. Dessa forma, o sinal de controle será proporcional ao módulo de e(t), multiplicado pelo ganho proporcional K_p, como mostrado na equação 1 (BAZANELLA; JR., 2000).

Quando o erro é zero, $u(t) = 0$. A medida que o erro cresce, u(t) aumenta até um valor máximo (100 %). O valor que provoca o erro máximo define a banda proporcional P_b. A figura 9 mostra o efeito da redução da banda proporcional. Quanto menor o valor de P_b, maior é a ação proporcional de controle (NOVUS, 2003).

$$u(t) = K_p e(t) \tag{1}$$

Na figura 9.A P_b é grande e o processo é estabilizado, porém em um valor muito abaixo do *setpoint* SP. Com a diminuição de P_b, o processo se estabiliza em um valor mais próximo de SP, como mostra a figura 9.B. Porém, se a banda proporcional for muito pequena o processo pode se tornar instável, como observado na figura 9.C. O ajuste da banda proporcional

é chamado de sintonia de controle e é ajustada definindo-se valores de K_p para o ganho proporcional P (NOVUS, 2003; SILVEIRA, 2016).

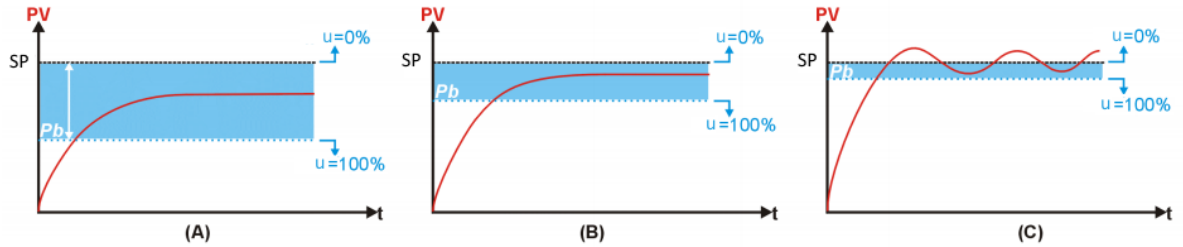

Figura 9 – Efeito da banda proporcional no comportamento da variável de saída – Fonte: (NOVUS, 2003)–modificado

A banda proporcional é relacionada com o ganho proporcional K_p da seguinte forma:

$$K_p = \frac{\text{Variação da saída}}{P_b}$$

O uso da ação proporcional P elimina as oscilações provocadas pelo controle *ON-OFF*. Porém, o erro de *offset* não pode ser corrigido por essa ação (OLIVEIRA, 1999).

2.2.1.2 Ação de controle PI

A ação integral isolada não é uma técnica de controle, sendo aplicada em conjunto com uma ação proporcional. Esse tipo de ação gera uma resposta u(t) proporcional à amplitude e duração do erro. A ação integral I elimina o erro de *offset* produzido pela ação proporcional P (NOVUS, 2003). A resposta da ação I pode ser expressa como:

$$\frac{ds}{dt} = K_I e(t)$$

onde $\frac{ds}{dt}$ é a taxa de variação da saída do controlador, K_I é o ganho integral e e(t) o erro. O inverso de K_I é chamado de tempo integral T_i e é usado para descrever a ação integral.

O tempo integral T_i ou *reset-time* corresponde ao tempo em que o ganho K_p é duplicado. O sinal u(t) para uma ação integral pode ser expresso como mostrado na equação 2 (OLIVEIRA, 1999).

$$u(t) = \frac{1}{T_i}\int_0^t e(t)dt \tag{2}$$

A ação proporcional integral pode ser descrita matematicamente como mostrado na equação 3. A ação PI corrige os erros instantâneos e elimina ao longo do tempo qualquer erro de *offset* (OLIVEIRA, 1999).

$$u(t) = K_p e(t) + \frac{1}{T_i}\int_0^t e(t)dt \tag{3}$$

Através da figura 10 é possível observar o efeito da ação PI. A figura 10.A mostra a ação proporcional desse processo, em que um erro de *offset* é criado. Na figura 10.B a ação I foi incluída em determinado tempo e o erro de *offset* em regime permanente foi sendo gradualmente eliminado. A ação proporcional é utilizada para eliminar o erro em regime permanente, porém se esse termo for muito atuante poderá levar o processo à instabilidade (NOVUS, 2003).

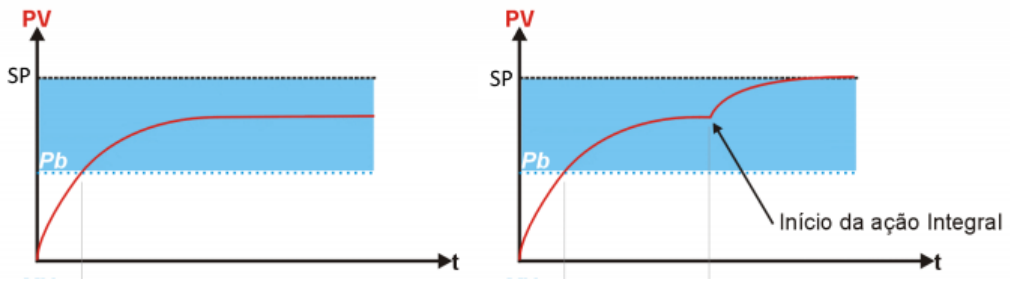

Figura 10 – Efeito da ação PI em um processo – Fonte: (NOVUS, 2003)–modificado

2.2.1.3 Ação de controle PD

A ação derivativa D produz uma resposta que é proporcional à velocidade de variação do erro e(t). Essa ação diminui a oscilação da variável de saída, mas só atua quando há variação de erro, caso o processo seja estável, seu efeito é nulo (NOVUS, 2003).

A ação derivativa melhora o desempenho do processo durante o período transitório, onde há variações do erro (NOVUS, 2003).

A figura 11 mostra o efeito do acréscimo da ação D em comparação a um processo que utiliza apenas ação proporcional.

Uma solução para evitar o *overshoot* mostrado na figura 11.A, que representa um controlador P, seria aumentar a banda proporcional, porém isso aumentaria o erro e(t) em regime permanente.

Outra solução seria implementar o termo derivativo D, que reduz ou elimina o *overshoot* reduzindo o sinal u(t) caso a saída esteja crescendo muito rápido.

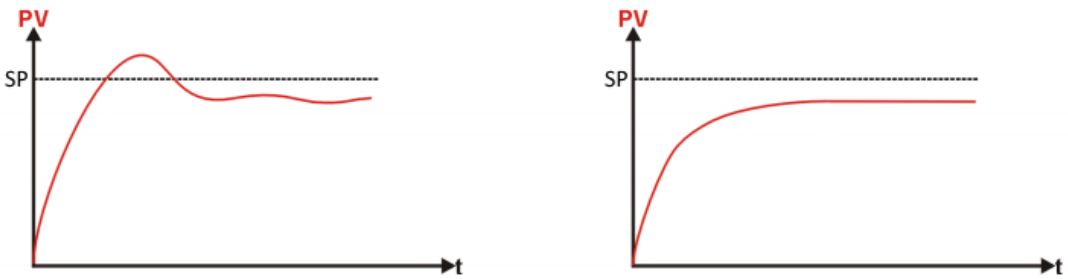

Figura 11 – Efeito da ação PD em um processo – Fonte: (NOVUS, 2003)–modificado

O efeito da aplicação do termo D na ação proporcional é mostrada na figura 11.B, onde é observado que o *overshoot* foi eliminado (NOVUS, 2003). A ação derivativa pode ser

representada matematicamente como mostrado na equação 4:

$$u(t) = T_d \frac{de(t)}{dt} + S_o \tag{4}$$

onde $\frac{de}{dt}$ representa a taxa de variação do erro e(t), S_o a saída para erro zero e T_d o tempo derivativo. O tempo derivativo ou ganho derivativo T_d é o tempo gasto pela ação derivativa para obter o mesma quantidade operacional da ação proporcional (OLIVEIRA, 1999).

Com isso, a aplicação da ação derivativa gera uma correção proporcional à velocidade do erro e não atua caso o erro seja constante. Além disso, quanto mais rápida é a variação do erro, maior será a correção (OLIVEIRA, 1999).

2.2.1.4 Ação de controle PID

A combinação das três ações de controle dá origem ao controle de ação PID. Esse tipo de controle é o mais sofisticado utilizado em sistemas de malha fechada. A ação proporcional elimina as oscilações, a ação integral diminui o erro de *offset*, enquanto a ação derivativa se antecipa previamente evitando que o erro se torne maior quando o sistema tem uma correção lenta comparada a velocidade do erro, que é o caso de controles de temperatura, por exemplo (OLIVEIRA, 1999). A ação PID pode ser expressa matematicamente como mostra a equação 5.

$$u(t) = K_p e(t) + \frac{1}{K_i} \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{5}$$

Com a união dessas três ações cria-se a dificuldade de ajustar o valor de cada um dos termos, o que é chamado de sintonia do PID. Quando o modelo matemático da planta pode ser obtido, várias técnicas podem ser utilizadas para determinar os coeficientes do controle PID. No entanto, quando a planta de um sistema não é conhecida ou é muito complexa é necessário recorrer a abordagens experimentais para determinar tais coeficientes (OGATA, 2011).

2.2.2 Controlador PID digital

A saída de um controlador PID foi apresentada na equação 5. Quando executada em tempo real, essa equação torna-se bastante complexa e exige muito processamento de um microcontrolador. Esse fato faz com que seja necessário obter aproximações nos termos integral e derivativo dessa equação e representá-la de forma discreta, afim de poder realizar a implementação em microcontroladores (MACIEL, 2012).

Equações representadas no tempo discreto necessitam de um período de amostragem. Esse período precisa ser suficiente para que todos os cálculos dependentes da variável amostrada possam ser realizados (OGATA, 2011 apud BROSLER, 2014).

A implementação de um controlador PID digital exige que os termos da equação 5 estejam disponíveis para o microcontrolador, ou seja, o valor do erro e os termos integral e derivativo.

Será utilizado a aproximação da derivada pelo método das diferenças finitas de Euler, e uma aproximação da integral pela integração trapezoidal, para que essa equação possa ser implementada no microcontrolador (PEREIRA, 2003 apud BROSLER, 2014).

2.2.2.1 Aproximação da integral pelo método da integração trapezoidal

Esse método consiste em dividir uma função contínua em intervalos de tempo igual ao período de amostragem T e considerar um variação linear dentro desse intervalo. Essa aproximação é mostrada na figura 12. A integral então pode ser aproximada pela equação 6.

$$\int_{t_0}^{t_1} \mathrm{e(t)dt} = \frac{\mathrm{e(kT\text{-}T)+e(kT)}}{2} T \tag{6}$$

Como a integral deve considerar todos os valores acumulados no tempo, a equação 7 representa o valor da integral de e(kT) (BROSLER, 2014).

$$\mathrm{u}_i\mathrm{(kT)} = \mathrm{u}_i\mathrm{(kT\text{-}T)} + \frac{\mathrm{e(kT\text{-}T)+e(kT)}}{2} T \tag{7}$$

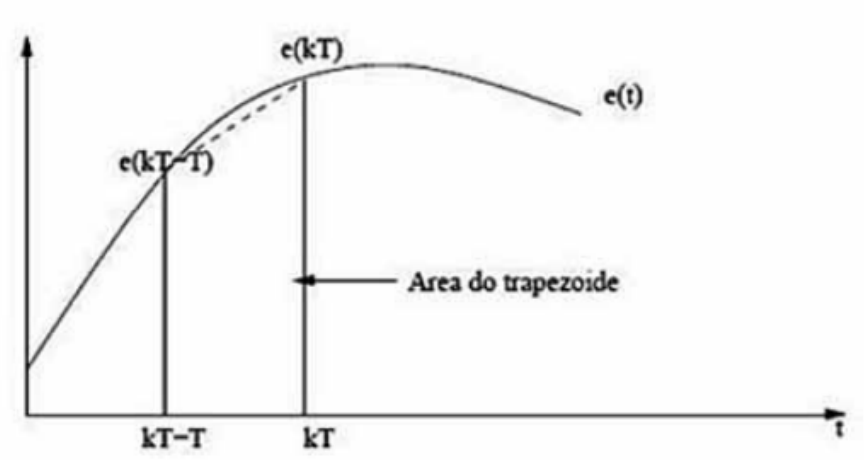

Figura 12 – Aproximação da integral – Fonte: (BROSLER, 2014)

2.2.2.2 Aproximação da derivada pelo método de Euler

A equação 8, pelo método de Euler, é uma aproximação para a derivada do erro e(t):

$$\frac{\mathrm{de(t)}}{\mathrm{dt}} = \frac{\mathrm{e(kT)\text{-}e(kT\text{-}T)}}{\mathrm{T}} \tag{8}$$

onde T é o período de amostragem, e(kT) é o valor do erro no instante atual e e(kT-T) é o valor do erro no instante anterior de amostragem. A figura 13 representa essa aproximação (BROSLER, 2014).

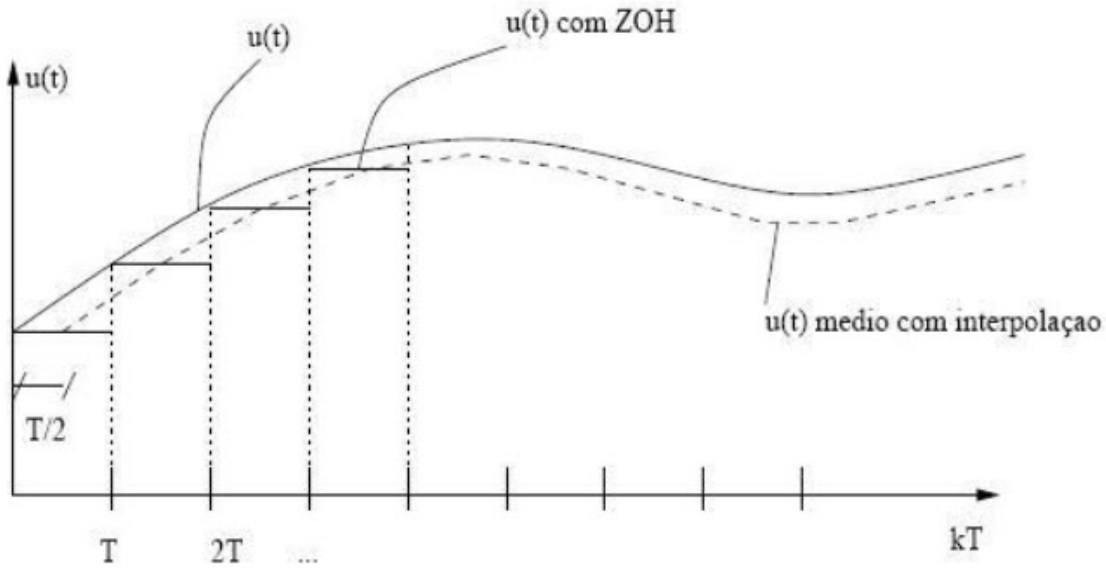

Figura 13 – Aproximação da derivada – Fonte: (MACIEL, 2012)

Com as equações 8 e 7 é possível escrever a equação algébrica aproximada para o controlador PID, que é mostrada na equação 9:

$$u(kT) = K_p e(kT) + u_i(kT\text{-}T) + K_i \frac{e(kT\text{-}T)+e(kT)}{2} T + K_d \frac{e(kT)\text{-}e(kT\text{-}T)}{T} \quad (9)$$

onde e(kT) é o erro atual, e(kT-T) é o erro do instante anterior, u_i é a ação integral do instante anterior e u(kT) é a saída atual que será aplicada no atuador do sistema.

2.3 Sintonização de Controladores

A sintonização de um controlador tem foco em determinar os parâmetros que serão utilizados em um controlador PID, os quais serão cruciais para um melhor controle (NOVUS, 2003). Neste subtópico, serão apresentados métodos para sintonia de controladores, de acordo com a resposta de um sistema, podendo ser em malha aberta, e em malha fechada.

2.3.1 Sintonia baseada na resposta em malha fechada

Um dos métodos mais conhecidos é o de Ziegler-Nichols, chamado de método do ganho limite, o qual coloca o sistema em oscilação harmônica. Este método baseia-se em determinar o período crítico (Pu) e o ganho crítico (Ku). O ganho crítico é definido como o máximo valor de ganho considerando um controlado com ação proporcional e sistema estável. E o período crítico, é denominado como o período registrado para obter o valor do ganho crítico (LJUNG, 1999).

Outro método que considera-se importante citar é o IMC (*Internal Model Control*, o qual foi proposto por Daniel Rivera, Manfred Morarl e Slgurd Skogestad em Danlel em 1986.

Tal método consiste em determinar o ajuste dos parâmetros PID a partir de uma função de transferência da planta, a qual é projetada a partir de um modelo interno do sistema (RIVERA D. E.; MORARI, 1986).

Ao considerar um sistema sem atraso de resposta, o método indica regras de ajuste para os parâmetros do controlador PID, Tais regras são especificadas de acordo com uma função de um parâmetro que é responsável por determinar a velocidade de resposta (RIVERA D. E.; MORARI, 1986).

Quando menor o valor de tal parâmetro, mais veloz e com mais desempenho é a resposta. Porém, nessas condições, a resposta também acaba apresentando sensibilidade à perurbações que possam ocorrer no processo (RIVERA D. E.; MORARI, 1986).

2.3.2 Sintonia baseada na resposta em malha aberta

Considerando o oposto da sintonia em malha fechada, para determinar os parâmetros de sintonia do controlador em malha aberta, necessita-se de um distúrbio, apenas, imposto ao sistema (LJUNG, 1999).

Para os ajustes em malha aberta, as técnicas utilizadas levam em consideração a curva de reação do sistema, ou seja, a reação do sistema de acordo com uma perturbação em degrau na saída do controlador, mais precisamente em sua variável manipulada (LJUNG, 1999).

Para obter uma sintonia mais precisa, os métodos que são utilizados comparam a curva de reação do sistema à resposta de um sistema simplificado, como por exemplo, um sistema de primeira ordem, sendo esta a forma mais comum de aproximação. (LJUNG, 1999).

2.4 Motor elétrico sem escovas

O motor elétrico do tipo sem escovas, como própio nome diz, é um motor sem escovas. A falta de escovas faz com que esse modelo de motor elétrico apresente a redução na geração de interferência eletromagnética para o sistema, o que no caso dos motores com escovas a interferência eletromagnética se apresenta em grande parte do espectro de frequência (LENZ, 2016).

Trata-se de um motor síncrono, que tem sua alimentação através de um inversor (também chamado de *driver*) de corrente contínua, geralmente de baixa tensão (LENZ, 2016).

A falta de escovas também apresenta a vantagem de que haja baixo atrito durante a operação do motor, reduzindo a produção de calor, aumentando a durabilidade. A diminuição do calor, faz com que melhore a transferência de potência do motor, aumentando também a eficiência elétrica, quando comparado a um motor com escovas (LENZ, 2016).

Aumentando a eficiência do motor, tem-se um melhora significativa na utilização de energia do sistema, fazendo com que aumente o tempo de vida útil da bateria utilizada como alimentação do sistema. A figura 14 apresenta um exemplo de motor sem escovas (LENZ, 2016).

A partir da figura 14, é possível observar que o motor sem escovas é constituído por três pares de enrolamentos, o rotor com ímã permanente e o estator. Tratando-se do rotor com ímã permanente e o estator, o funcionamento se baseia na energização dos pares de enrolamentos do motor, sendo gerado um campo magnético temporário que repele ou atrai os ímãs permanentes. A força produzida é convertida em rotação do eixo do rotor que faz com que o motor funcione.

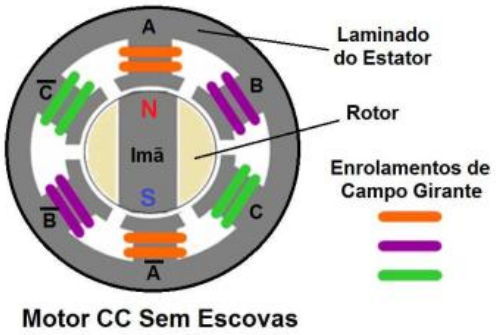

Figura 14 – Motor *brushless* – Fonte: (LENZ, 2016)

Para compreender como ocorre a energização dos pares de enrolamentos, considere os enrolamentos A, B e C com seus respectivos pares A', B' e C', apresentado na figura 14. Assim, a figura 15 monstra o instante do acionamento de cada enrolamento.

A partir da figura 15, pode-se observar que primeiramente ocorre a energização do enrolamento A e A', o que gera um polo norte em um dos enrolamentos e um polo sul no outro enrolamento, fazendo com que o ímã permanente, que é o rotor, seja atraído (ENGINEERING, 2014).

Quando o rotor atinge a posição de equilíbrio dos polos magnéticos, desliga-se a corrente dos enrolamentos A e A' comutando para os enrolamentos B e B', gerando os polos norte e sul para cada enrolamento novamente, fazendo com que o rotor seja atraído para a nova posição dos polos magnéticos (ENGINEERING, 2014).

Atingido o ponto de equilíbrio do rotor novamente, desliga-se a corrente dos enrolamentos B e B' comutando para os enrolamentos C e C', fazendo surgir novos polos e atraindo o rotor (ENGINEERING, 2014).

Após atingido o ponto de equilíbrio dos polos magnéticos, retorna para o acionamento dos enrolamentos A e A', reiniciando o ciclo de funcionamento do motor sem escovas (ENGINEERING, 2014).

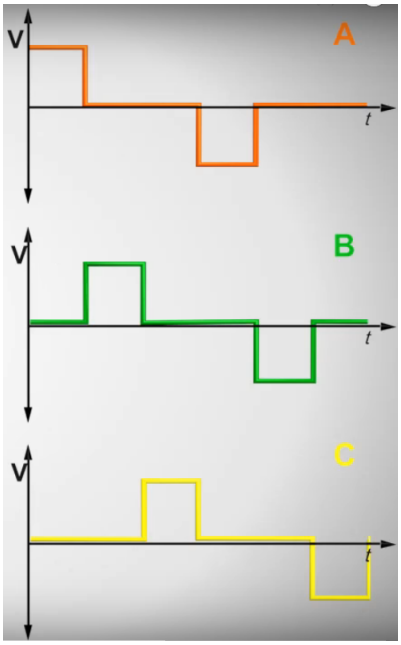

Figura 15 – Gráfico de energização dos enrolamentos do motor – Fonte: (ENGINEERING, 2014)

2.5 Barramento CAN

Criado em 1986 por Robert Bosch, o protocolo CAN é um protocolo de comunicações série, composto por um par trançado que interliga as ECUs presentes no barramento. Possui capacidade multi-mestre, ou seja, várias unidades de controle podem pedir acesso ao meio de transmissão simultaneamente (BOSCH, 2005).

Este protocolo comporta também o conceito de *multicast*, ou seja, uma mensagem pode ser transmitida à vários receptores de maneira simultânea. Na figura 16, pode ser observado um exemplo de uma rede CAN, conectando várias ECUs (BOSCH, 2005).

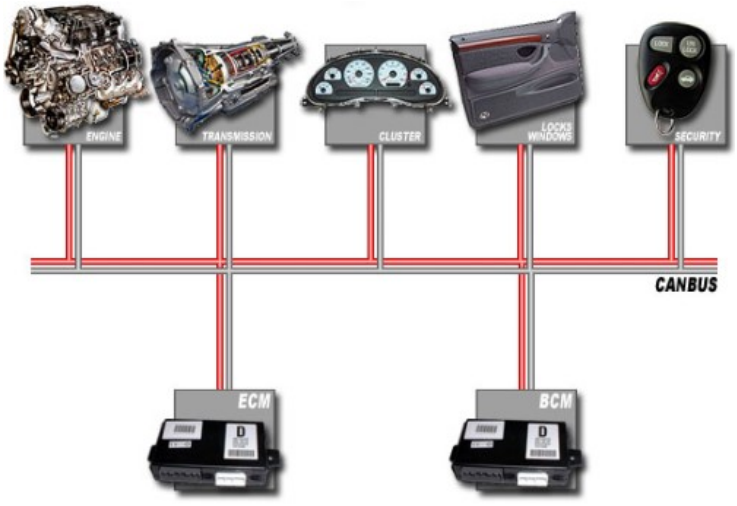

Figura 16 – Exemplo de uma rede CAN – Fonte: canbuskits.com

É possível adicionar novos nós (ECU's) em uma rede CAN sem necessitar alterar *software* ou *hardware* do sistema, desde que o novo nó não seja emissor, ou necessite de transmissão adicional de dados.

O CAN é uma rede CSMA/DC, ou seja, a transmissão de dados é atrasada pelos nós caso o barramento esteja ocupado. Assim que o barramento estiver livre, qualquer nó pode iniciar uma transmissão (BOSCH, 2005).

No caso de conflitos de acesso ao barramento, é feita uma comparação de acordo com bit e identificador da origem de comunicação. Todos os nós monitoram o barramento enquanto é transmitido o identificador. O nó que transmite os dados com o menor identificador, ganha o direito de transmitir os dados no barramento, enquanto os outros nós aguardam.

Exemplificando, no caso de desejar transmitir um bit de nível lógico '0', e haver um bit de nível lógico '1' querendo ser transmitido, o bit de nível lógico '0' tem prioridade na transmissão (BOSCH, 2005).

Após os dados com maior prioridade serem enviados, ao ficar livre, o barramento CAN faz a retransmissão das mensagens que ficaram "em espera"automaticamente, cujas mensagens passarão novamente pelo processo de priorização dos dados.

A respeito da taxa máxima de transmissão, é variável de acordo com o comprimento do barramento. Para comprimento de até 40 m, a taxa é de 1M bit/s. Para comprimento de até 1 Km, a taxa é de 50K bit/s. Para comprimento de até 500 m, a taxa é de 125K bit/s. Caso o comprimento do barramento seja superior à 1 Km, é recomendada a utilização de dispositivos repetidores de sinal (BOSCH, 2005).

2.5.1 *Frames*

Uma *frame* é a menor unidade de informação transferida, e existem diferentes possibilidades de tamanhos de *frames*, porém em todos eles existe um padrão de organização do início do *frame*. Na figura 17, é possível observar o formato de frame de dados utilizado com as versões 1.0, 1.1, 1.2 e 2.0A do barramento CAN, o qual possui identificador de 11 *bits*.

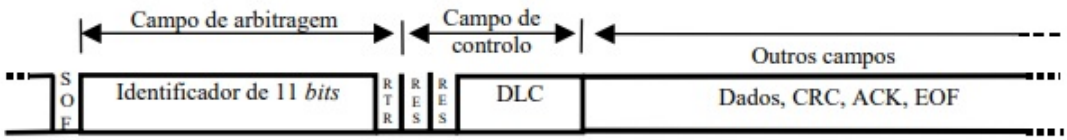

Figura 17 – Formato de frame de dados de acordo com as versões 1.0, 1.1, 1.2 e 2.0A – Fonte: (BOSCH, 2005)

Já na figura 18, é possível observar o formato de frame de dados de acordo com a versão 2.0B *standart*, o qual apresenta um identificador de 29 *bits*. Já na figura 19, é possível observar o formato de frame estendido, definido na versão 2.0 B, o qual apresenta uma diferença

em relação aos formatos citados anteriormente, sendo o número de *bits* do campo de arbitragem e valor dos *bits* de controle.

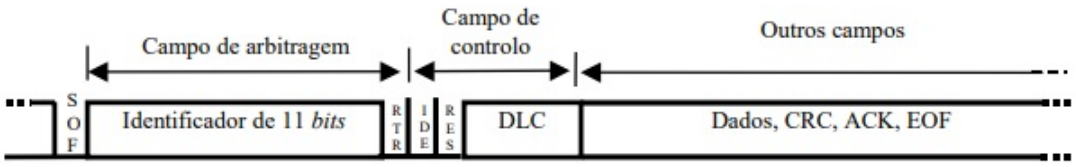

Figura 18 – Formato de frame de dados de acordo com a versão 2.0B *standart* – Fonte: (BOSCH, 2005)

Todas as mensagens enviadas começam com o *start of frame* (SOF), juntamente com os *bits* do identificador, que podem ser um ou três *bits* para controle. Esses *bits* do identificador servem para definir quando se tem um *frame* do tipo *standart* ou *extended*, e também para definir se a *frame* é de dados ou remota.

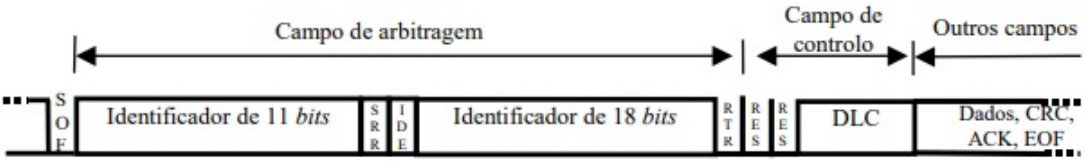

Figura 19 – Formato de frame de dados de acordo com a versão 2.0B *extended* – Fonte: (BOSCH, 2005)

Além de *frame* de dados e *frame* remota, tem-se ainda *frame* de erro, e *frame* de sobrecarga. A *frame* de dados apresenta os dados que serão enviados do emissor para o receptor. A *frame* remota é uma solicitação de dados, e é enviada quando um nó receptor deseja transmitir algum dado para o barramento.

Quando um erro é detectado no barramento, uma *frame* de erro é transmitida por qualquer nó. Por fim, utiliza-se a *frame* de sobrecarga com intuito de provocar um atraso adicional entre uma *frame* remota ou de dados, e a *frame* seguinte.

2.6 Comunicação Serial

Com o avanço dos microcontroladores integrados de acordo com aplicações específicas, a comunicação serial fez-se necessária uma vez que não é possível estender extensos barramentos de comunicação paralelos, a fim de evitar placas de circuito impresso custosas e grandes.

A interligação serial pode ser separada nas categorias síncrona e assíncrona. Nesta sessão, será abordado alguns dos métodos de comunicação mais conhecidos.

2.6.1 Protocolo SPI

A comunicação SPI (*Serial Peripheral Interface*) trata-se de um protocolo síncrono, o qual foi desenvolvido pela Motorola e possui quatro sinais. Estabelece comunicação entre

mestre e escravo, não permitindo endereçamento (UPENDER, 1994).

Tal comunicação é feita através de, no mínimo, quatro fios, onde três são destinados à conexões de *clock* e dados, e deve-se ter uma conexão para cada escravo que será conectado ao mestre.

É importante lembrar que um mestre pode ter diversos escravos conectados. Na figura 20, é possível observar as conexões citadas acima, num sistema contendo um mestre e três escravos (UPENDER, 1994).

Tratando-se de comunicação síncrona, um pino é destinado à temporização da comunicação, gerando o sinal de *clock*, o qual é denominado SCK (*Serial Clock*). Já para o tráfego de dados, tem-se 2 pinos disponíveis. Um deles, denominado MOSI (*Master Output/Slave Input*) é utilizado para a transferência de dados do mestre para o escravo.

Já o outro pino, denominado MISO (*Master Input/Slave Output*), é utilizada para envio de dados do escravo para o mestre. Por fim, cada pino denominado SS (*Slave Select*) é utilizado para a seleção do escravo, quando há mais de um no sistema.

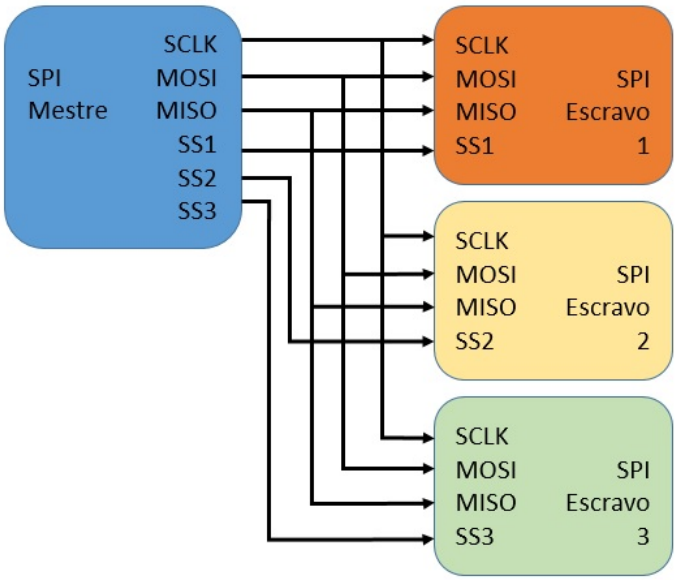

Figura 20 – Exemplo de comunicação SPI entre mestre e escravos – Fonte: Autoria própria

O grande diferencial está em poder atingir velocidades de comunicação elevadas, devido à pouca deformação do sinal. Além disso, a troca de dados sempre ocorre em ambas as direções, sendo uma comunicação *full-duplex*.

Na figura 21, pode ser observado a forma de onda da comunicação SPI. De acordo com a imagem, é possível observar que a configuração das bordas do *clock* dá-se através de CPOL (*Clock Polarity*), e também através da configuração da fase do *clock*, chamada de CPHA (*Clock Phase*).

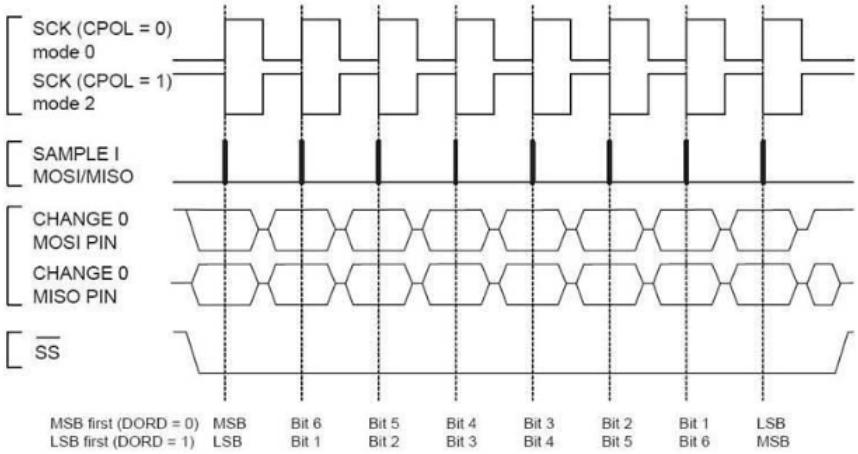

Figura 21 – Exemplo da forma de onda da comunicação SPI entre mestre e escravo – Fonte: (UPENDER, 1994)

2.6.2 Protocolo I2C

A comunicação I2C (*Inter-Intergrated Circuit*) é um protocolo utilizado na conexão de periféricos de baixa velocidade. Trata-se de um barramento serial multi-metre, o qual foi desenvolvido pela Philips.

Este protocolo utiliza duas conexões bidirecionais para dreno aberto, SDA (*Serial Data*) e SCL (*Serial Clock*). Além disso, são especificados ainda dois sinais para comunicação, sendo um deles com o sinal de *clock* obtido a partir do mestre, e o outro sinal de dados, sendo bidirecional (NXP, 2014).

Na figura 22, é possiível observar as formas de onda quando ocorre transferências de dados. Após a condição *START (S)*, um endereço escravo é enviado. Esse endereço tem sete bits de comprimento, seguido por um oitavo bit, que é um bit de direção de dados (W), onde um 'zero' indica uma transmissão (*WRITE*); um 'one' indica uma solicitação de dados (*READ*) (NXP, 2014).

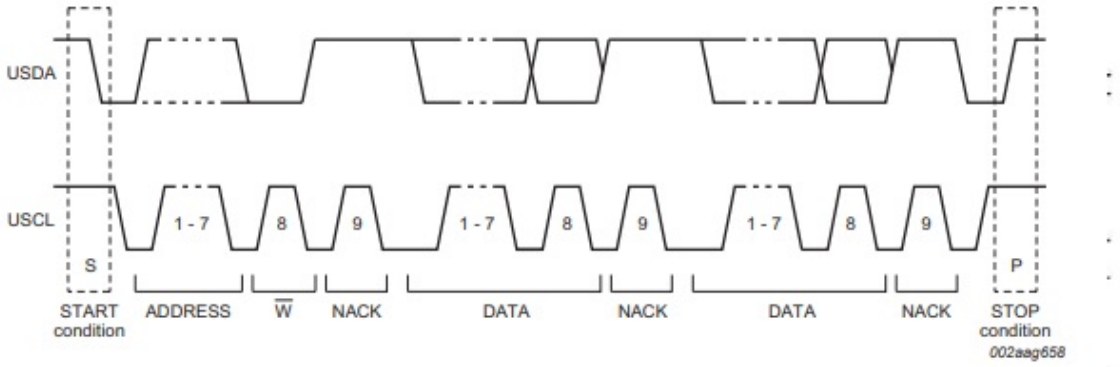

Figura 22 – Exemplo da forma de onda da comunicação SPI entre mestre e escravo – Fonte: (NXP, 2014)

Uma transferência de dados é sempre encerrada por uma condição de PARADA (P) gerada pelo mestre. No entanto, se um mestre ainda deseja se comunicar no barramento, pode gerar uma condição *START* repetida e endereçar outro escravo sem primeiro gerar uma condição *STOP* (NXP, 2014).

2.6.3 Comunicação UART

A comunicação UART (*Universal Asynchronous Receiver Transmitter*) é um protocolo assíncrono que transmite dados de um microprocessador para computador utilizando apenas conexções de transmissão e recepção (TX/RX). Sendo um sistema de comunicação *full-duplex*, sua utilização se dá principlamente entre periféricos (INSTRUMENTS, 2010).

Na figura 23, tem-se um exemplo de sua forma de onda. Quando parado, o pino de saída está no estado lógico 1. Para cada transmissão de dados, começa-se com um bit *START*, que é sempre zero (0).

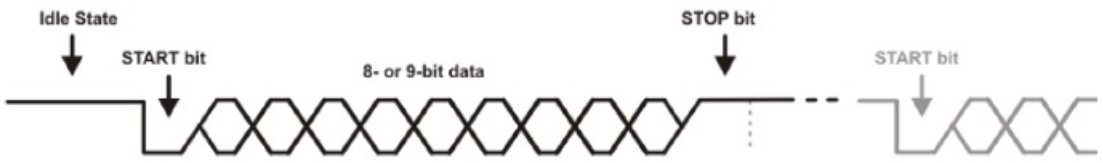

Figura 23 – Exemplo da forma de onda da comunicação UART – Fonte: (INSTRUMENTS, 2010)

Em cada pacote de dados, tem-se 8 ou 9 bits de tamanho, onde o bit menos significativo é sempre o primeiro a ser transferido. Em cada transmissão de dados, termina-se com um bit de *STOP*, que tem sempre estado lógico 1.

2.7 Comunicação *Bluetooth*

A comunicação *bluetooth* é um padrão de comunicação, sendo sem fio e de curto alcance. Apresenta também baixo custo e baixo consumo de energia, utilizando tecnologia de rádio. Como exemplos de seu uso, pode-se citar os dispositivos celulares e computadores.

Ao falar da arquitetura *bluetooth*, pode-se citar a presença de um transceiver, sendo o *hardware* e uma pilha de protocolos, sendo o *software*.. Tal arquitetura dispõe de funcionalidades básicas as quais tornam possível a conexão de dispositivos e a troca de uma variedade de tipos de dados entre estes dispositivos (MCDERMOTT-WELLS, 2004).

A freqüência para utilização opera entre entre 2.4 GHz e 2.485 GHz (considerando faixa de rádio não licenciada). Já a taxa de transmissão pode chegar à 1 Mbps ou, a 2 ou 3 Mbps, considerando o mecanismo *Enhanced Data Rate*.

Na figura 24, pode ser observada a pilha de protocolos, onde destaca-se o grupo de aplicação, grupo de protocolos de *middleware* e grupos de protocolos de transporte.

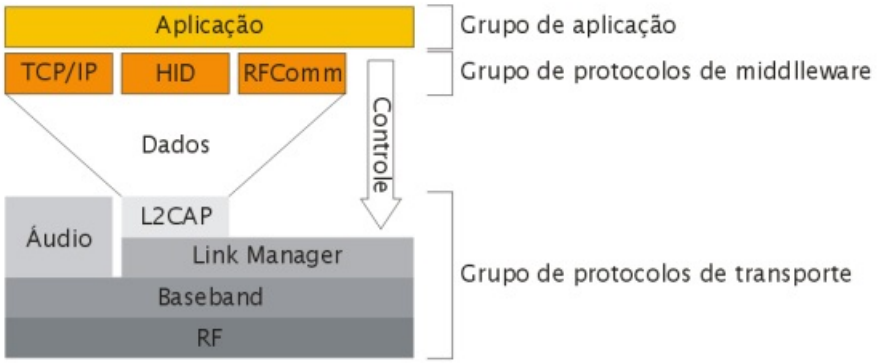

Figura 24 – Pilha de protocolos *Bluetooth* – Fonte: (MCDERMOTT-WELLS, 2004)

3 MATERIAIS E MÉTODOS

3.1 Materiais Utilizados

3.1.1 Motor elétrico sem escovas

Para o desenvolvimento do projeto, foi utilizado um motor elétrico acoplado com a caixa de direção, onde a partir de um sistema de controle foi possível automatizar o funcionamento do motor.

O motor escolhido foi tipo sem escovas (*brushless*), devido a facilidade de aplicar uma MCU (*Motor Control Unit*) para este modelo de motor elétrico (LENZ, 2016). Na figura 25, tem-se o motor que foi utilizado nesse projeto, o qual é trifásico, com tensão de alimentação igual a 24 V DC.

Possui sensor de posição *hall* analógico absoluto, que varia de 0 a 5 V DC, e codificador de comutação baseado em sensor *hall*, com 5 V DC, e saídas de coletor aberto.

Figura 25 – Motor elétrico sem escovas que será utilizado neste projeto – Fonte: Autoria própria

Seu acionamento é do tipo trapezoidal, utilizando comutação hall de seis etapas. Este motor apresenta uma velocidade sem carga de 183 rpm, torque de parada de, no mínimo, 30 Nm, torque contínuo de 5 Nm, e torque de pico de 10 Nm.

Além disso, este motor possui 2 conectores integrados, sendo um do tipo *Delphi Metri-Pack*, sendo utilizado para sinais, e um conector do tipo *Amphenol*, sendo utilizado para alimentar as fases. Na figura 26, é possível observar tais conectores do motor elétrico.

Como o motor elétrico é montado acima da caixa de direção, caso venha a falhar, não interferirá no funcionamento do esterçamento do sistema de direção. Logo, a relação de transmissão do sistema será a mesma relação de transmissão da caixa de direção.

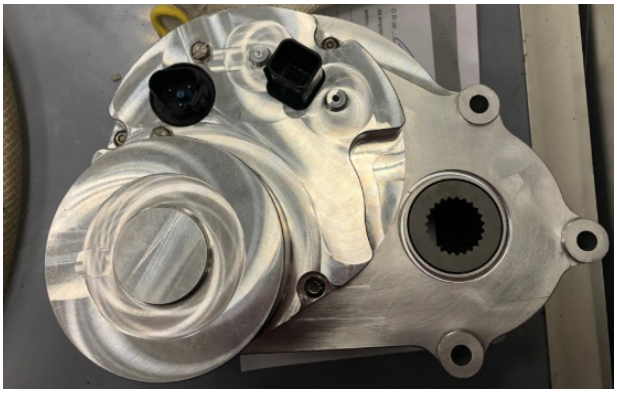

Figura 26 – Conectores elétricos do motor elétrico sem escovas que será utilizado neste projeto – Fonte: Autoria própria

3.1.2 Driver do Motor Elétrico

Para *driver* do motor elétrico, optou-se pelo modelo JYQD V7, do fabricante JUYI. Na figura 27, tem-se a figura da placa do *driver* escolhido.

Figura 27 – Placa do *driver* com os componentes soldados – Fonte: Autoria Própria

O *driver* escolhido apresenta a potência de saída de 500W, com entrada DC de 12 a 36 volts e corrente de saída máxima de 15 A. Na figura 28, tem-se o diagrama de conexões do *driver*. Trata-se de uma placa que não possui dissipador de calor. Caso a potência do motor seja menor de 60W, não há necessidade de adicionar dissipador de calor, apenas é recomendado garantir a ventilação e isolamento do equipamento. Já no caso em que o motor elétrico apresente potência acima de 60W, é recomendado adicionar dissipador de calor.

Ao lado direito da figura 28, está apresentado os conectores de entrada do *driver*, sendo um conector para o retorno dos sensores Hall e o outro conector os sinais de controle do

driver. No caso do conector do sensor *hall*, tem-se o pino de alimentação de 5V e GND para o circuito do sensor, e também os pinos dos sinais enviados pelo sensor *hall*.

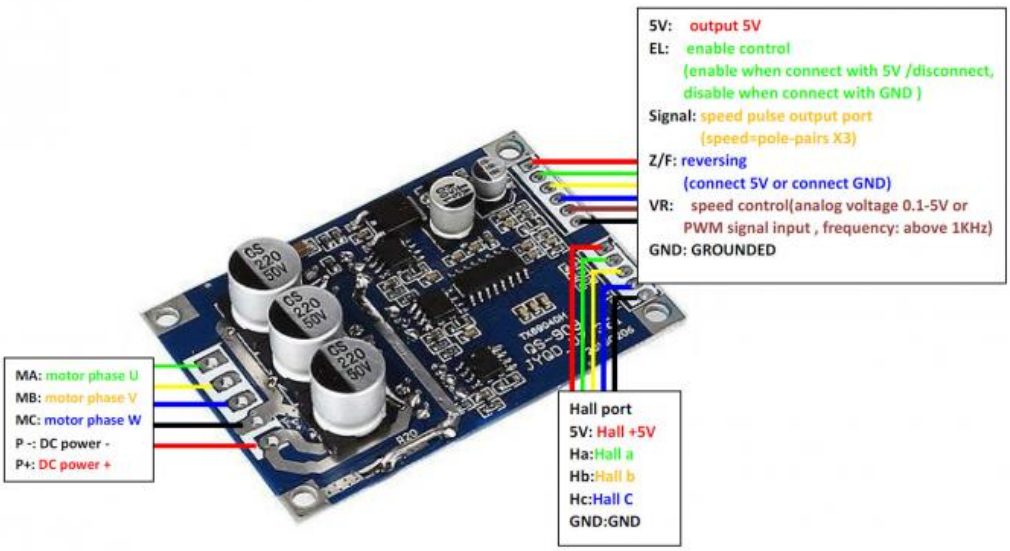

Figura 28 – Diagrama de conexões da placa – Fonte: Juyi Tech

Para o conector de controle, a porta de saída de 5V da placa proíbe o uso de dispositivo externo, pois não possui potência suficiente para alimentar outros circuitos. A função dessa porta é para ajustes de nível de tensão do controle da placa, para definição da rotação do motor, através de potenciômetros e chaves.

A porta definida como "VR" é a porta utilizada para o controle de velocidade. Possui uma regulagem de velocidade linear de tensão analógica que pode variar entre 0,1V e 5V. Apresenta uma resistência de entrada de 20K Ω, e também frequência *PWM* que varia entre 1KHz e 20KHz.

Já a porta nomeada como "Z/F" é a porta de controle de direção de rotação do motor elétrico. Para configurar a direção de avanço, basta conectar o nível alto "5V", e para configurar a direção reversa, basta conectar o nível 0V ou conectar ao GND.

A porta "EL" tem a função de habilitar o controle automático do motor pela placa do *driver*. Já a porta "*Signal*" tem como função o retorno de um sinal quadrado com a frequência de rotação do motor, medidas através dos sinais recebidos pelo sensor *hall*.

Agora, considerando as portas em destaque no lado esquerdo da figura 28, tem-se as portas de alimentação. Podemos observar as portas "MA", "MB", e "MC", as quais são as portas responsáveis por enviar os sinais da fase A, fase B e face C do motor, respectivamente.

3.1.3 Tiva C Series TM4C123G

A placa Tiva C TM4C123G é uma placa de baixo custo no mercado, que apresenta um microcontrolador Tiva TM4C123GH6PMI ARM Cortex-M4F, 40 pinos de expansão disponíveis em conector *headers* e circuito de depuração integrado. Além disso, apresenta também um

botão de reset, 2 conectores USB Micro-B, e chave para seleção de fonte de alimentação. Na figura 29, tem-se a ilustração da placa (INSTRUMENTS, 2013).

Figura 29 – Tiva C Series TM4C123G - Fonte: Texas Instruments

A placa pode ser alimentada de 4,75 V à 5,25 V, podendo ser via conector USB Micro-B da interface de *debugger* (ICDI) conectado à um computador ou fonte externa; ou até mesmo via conector USB Micro-B device conectado ao PC ou fonte externa. Na figura 30, é possível observar a disposição das funcionalidades da placa (INSTRUMENTS, 2013).

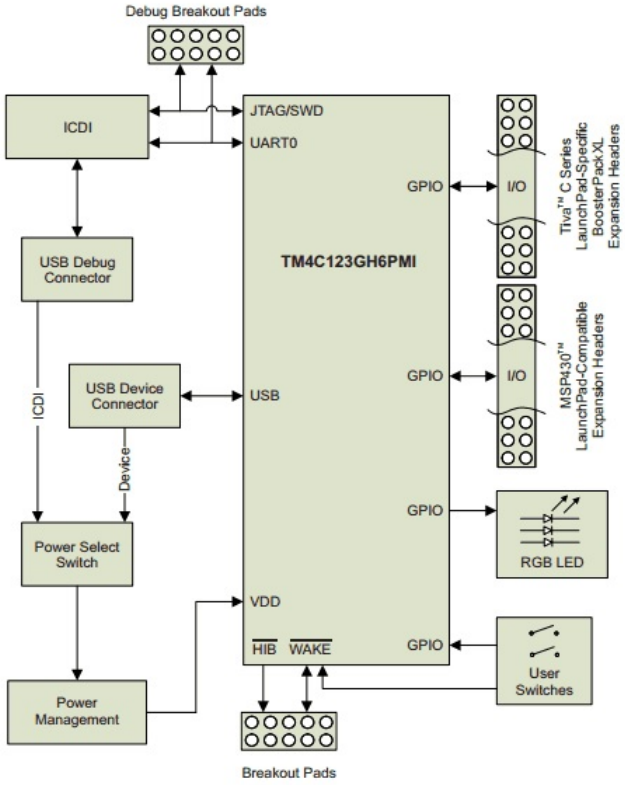

Figura 30 – Diagrama de blocos da placa - Fonte: Texas Instruments

O microcontrolador TM4C123GH6PMI é o grande diferencial desta placa, sendo um ARM Cortex-M4F, da família de núcleos de processador RISC de 32-bit, podendo ser observado na figura 31.

Figura 31 – Microcontrolador TM4C123GH6PMI - Fonte: Texas Instruments

Este microcontrolador possui um núcleo de 32 bit, memória *flash* de 256 KB, memória SRAM de 32 KB, e memória EEPROM de 2KB. Além disso, o microcontrolador apresenta 2 módulos CAN 2.0 e USB 2.0 OTG. Além disso, a placa utiliza o CI TPS73633DRB como regulador de tensão, o qual mantém a tensão de entrada estabilizada em 3,3V (INSTRUMENTS, 2013).

3.1.4 DAC MCP4725

O conversor digital-analógico utilizado neste projeto é o DAC MCP4725, possuindo 12 bits de resolução, os quais são controlados através de uma interface de comunicação I2C. O conversor pode ser observado na figura 32 (MICROCHIP, 2007).

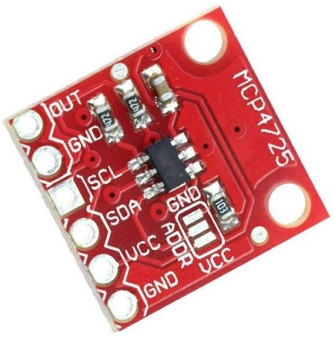

Figura 32 – Conversor analógico-digital DAC MCP4725 - Fonte: Microchip

Utiliza o CI MCP4725, e apresenta tensão de operação entre 2,7V e 5,5V. Para utilização do barramento I2C, os sinais são enviados através dos pinos *SCL* e *SDA*, e no pino *OUT* é obtido o valor analógico correspondente (MICROCHIP, 2007).

3.1.5 CAN Bus MCP2515

O CAN Bus MCP2515 é um módulo eletrônico que permite a integração de um microcontrolador à rede CAN, e pode ser observado na figura 33. Este módulo possui tensão de alimentação de 5V, corrente de operação de 5mA, realizando a comunicação através de barramento SPI com o micontrolador e enviando/recebendo os dados com o padrão CAN (MICROCHIP, 2002).

Figura 33 – Módulo CAN Bus MCP2515 - Fonte: Microchip

O CI MCP2515 utiliza o padrão CAN 2.0B podendo ser configurado para a comunicação no padrão *standard* e *extended* especificados no protocolo CAN. Apresenta três *buffers* de envio de mensagens e dois *buffers* de recepção dos dados, onde estes *buffers* podem ser configurados com o uso de duas máscaras e seis filtros de aceitação dos dados enviados e/ou recebidos, com a intenção de controlar as mensagens indesejadas no sistema diminuindo a sobrecarga do microcontrolador (MICROCHIP, 2002).

3.1.6 HC05

O módulo HC-05 é um módulo que transmite informações via *Bluetooth*, as quais provêm de uma comunicação serial. Apresenta tensão de alimentação de 3,3V, corrente de 35mA quando pareado e de 8mA quando conectado. Este módulo pode ser observado na figura 34 (MICROCHIP, 2010).

Figura 34 – Módulo *Bluetooth* HC05 - Fonte: Itead Studio

Este módulo apresenta dois pinos destinados à alimentação do sistema, dois para programação em modo *master*, e dois para transmissão e recepção de dados via comunicação serial. Um dos diferenciais deste módulo, é a possibilidade de alternância entre os modos escravo e mestre, sendo possível o recebimento e também a transmissão de dados (MICROCHIP, 2010).

Para assegurar uma transmissão de dados consistente, possui cobertura de sinal de até 10 metros, potência de transmissão de até 4 dBm, suportando uma taxa máxima de transmissão serial de 1382400 bps (MICROCHIP, 2010).

3.2 Recursos Computacionais

3.2.1 MATLAB

Para algumas etapas do projeto se tem a necessidade da implementação de simulações do sistema, para que seja possível a realização de previsões do funcionamento do sistema, facilitando a implementação do sistema físico.

Com isso, para as simulações foi escolhido o *software* Matlab, onde é uma plataforma de projeto para o modelagens de engenharia e matemáticas. Com o Matlab pode-se realizar análises matemáticas, desenvolver algoritmos e desenvolver modelos e aplicações para os sistemas.

Para se ter uma visualização da interface do matlab, tem-se a figura 35 onde apresenta a tela inicial do *software*.

O uso do Matlab é feito por diversas aréas de estudo, tanto nas áreas cientificas como em aplicações no setor industrial. Pode ser realizado projetos com uma grande variedade de aplicações, como aprendizado de máquina, processamento de sinal, processamento de imagem e vídeo, sistemas de controle, análises financeiras, análises biológicas, sistemas elétricos e eletrônicos, dentre outros.

No caso desse projeto, necessita-se do Matlab para simulações de sistemas mecânicos e eletrônicos, tendo ênfase na construção de sistemas com o uso de motores elétricos.

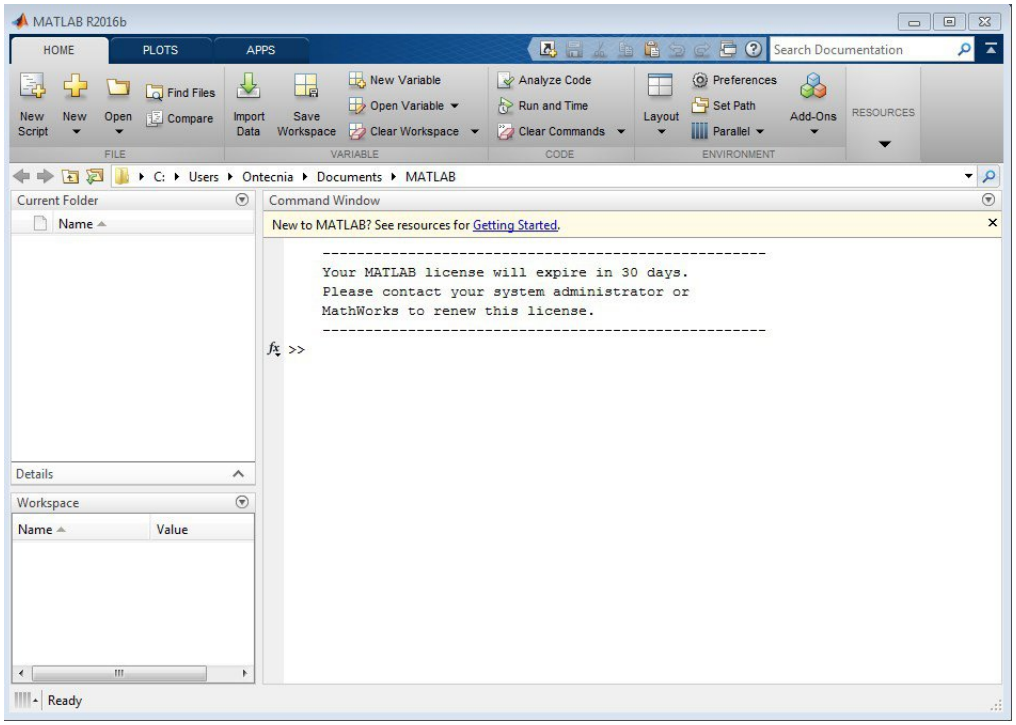

Figura 35 – Janela inicial do Matlab – Fonte: (MATLAB, 2018)

3.2.2 Android Studio

Para o desenvolvimento do aplicativo Android, tem-se a ferramenta Android Studio disponibilizada pela Google, onde é um ambiente de desenvolvimento (IDE) baseado no sistema IntelliJ IDEA. A figura 36 apresenta a tela inicial do software (DEVELOPERS, 2018).

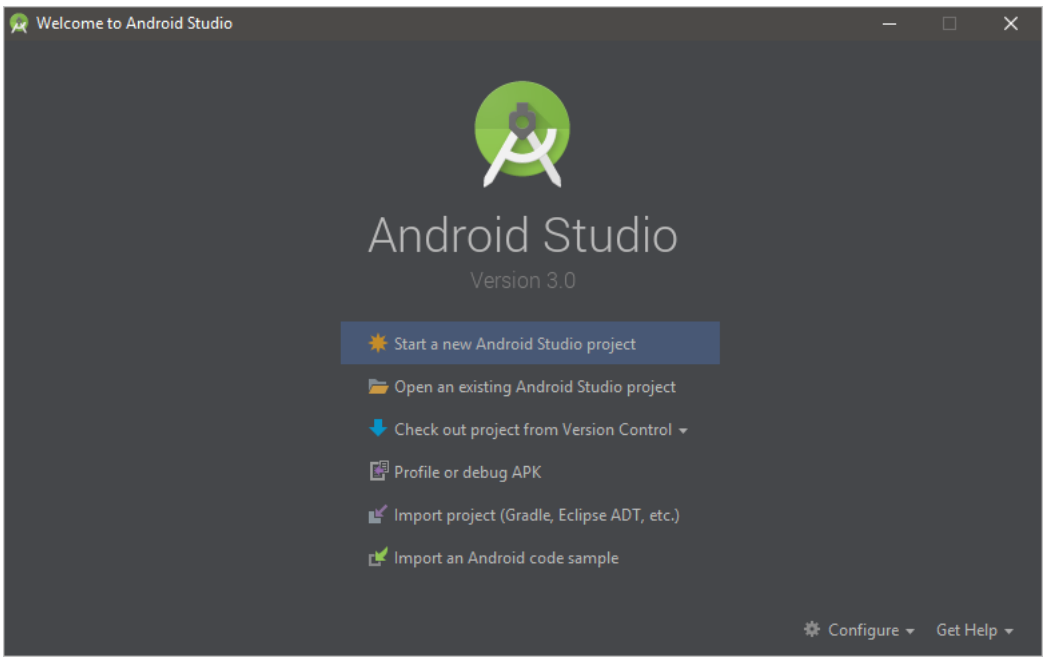

Figura 36 – Tela inicial do Android Studio – Fonte: (DALTO, 2017)

O Android Studio apresenta um sistema para edição dos códigos, compilação e depuração dos projetos desenvolvidos na ferramenta. Além destas funcionalidades, o Android Studio contém mais alguns recursos que auxiliam no desenvolvimento dos projetos, onde os tópicos a seguir apresentam estes recursos (DEVELOPERS, 2018).

- Sistema Grandle de compilação;

- Sistema de emulação, podendo aplicar simuladores dos dispositivos onde se deseja utilizar o aplicativo, como pode ser observado na figura 37;

- Uma IDE integralizada, podendo desenvolver para qualquer tipo de dispositivo Android;

- Sistema *Instant Run*, podendo realizar alterações nas aplicação sem a necessidade de recompilar todo o projeto;

- Integralização com o sistema GitHub, podendo exportar e importar códigos diretamente da IDE;

- Sistema de verificação de código suspeito, verificando problemas com desempenho, usabilidade e compatibilidade do projeto;

- Apresenta a integralização co outra plataformas de desenvolvimento, por exemplo o Eclipse.

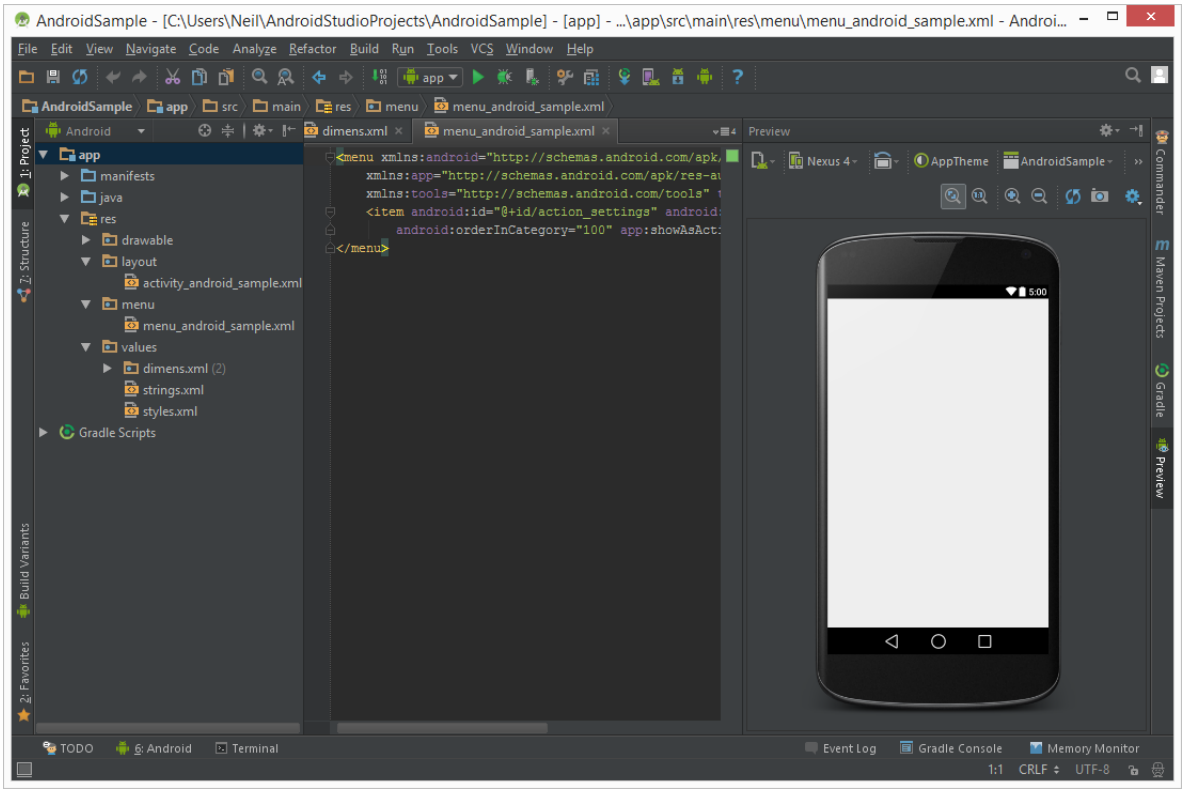

Figura 37 – Android Studio com o simulador – Fonte: (CUSTOMERSYS, 2015)

A interface de desenvolvimento do Android Studio é interessante, pois apresenta várias opções de temas, assim como a possibilidade de reconfigurar as teclas de atalho do sistema. Dessa forma, o usuário possui liberdade para configurar da maneira que mais o agrada (CARVALHO, 2013).

O Android Studio também conta com um sistema de auto-complete, onde é executado de forma automática, sem a necessidade de acionar algum atalho para utilizar, o que torna mais rápido o desenvolvimento de um projeto (CARVALHO, 2013).

Com a funcionalidade de integração com sistemas de controle de versão contida no Android Studio, é possível controlar as alterações nos códigos do projeto sem a necessidade de sair da interface de desenvolvimento (CARVALHO, 2013).

Dessa forma, pode-se realizar os comandos do sistema de controle de versão encolhido diretamente na IDE. Os sistemas de controle de versão suportados pelo Android Studio são o Mercuirial, Git e Subversion (CARVALHO, 2013).

A tela de criação do layout do Android Studio é um dos principais motivos para os usuários terem a escolhido, pois é um sistema bem dinâmico com a possibilidade de arrastar e soltar os objetos que se deseja aplicar na aplicação em desenvolvimento.

Além das várias possibilidades de temas que se pode aplicar para a sua aplicação, entre outras funcionalidades. A figura 38 apresenta um exemplo de tela de layout do Android Studio (CARVALHO, 2013).

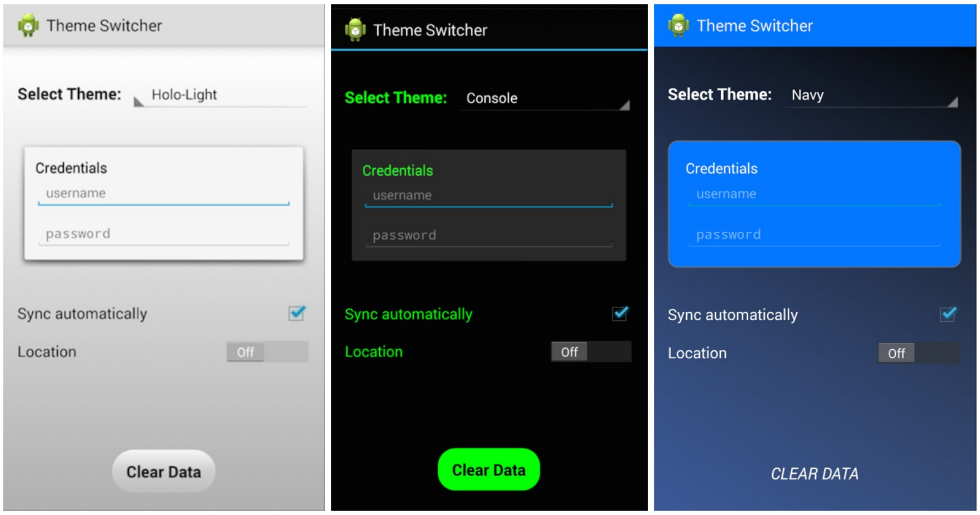

Figura 38 – Possíveis temas para as aplicações no Andriod Studio – Fonte: (CODEPATH, 2016)

3.2.3 Code Composer Studio

Exclusivo para processadores embarcados da Texas Instruments (TI), trata-se de ambiente de desenvolvimento integrado (IDE) para desenvolver aplicativos. A interface de trabalho pode ser observada na figura 39.

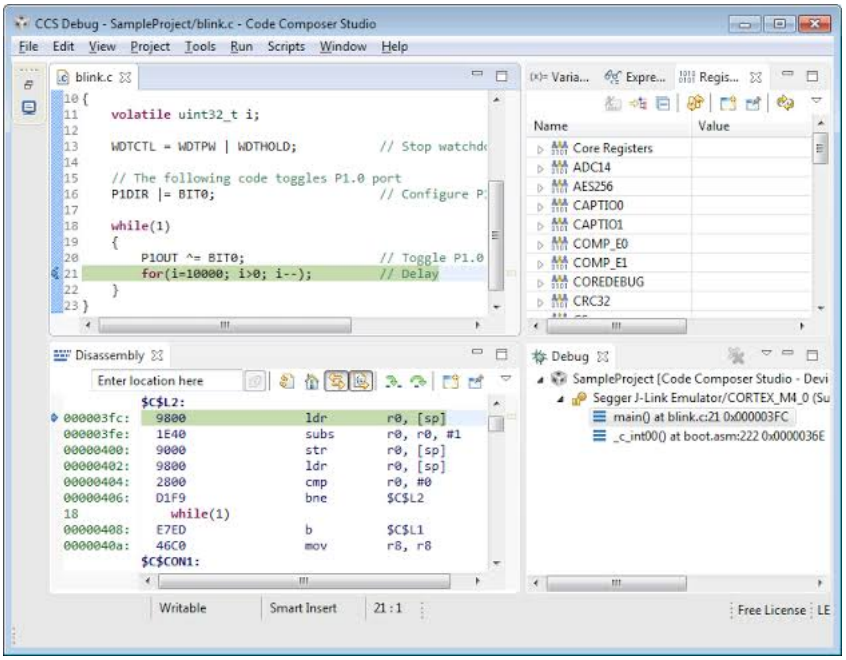

Figura 39 – Interface do *Code Composer Studio* – Fonte: Texas Instruments

A ferramenta code composer studio, tem como função o desenvolvimento dos códigos, depuração dos códigos e gravação nos processadores do fabricante Texas Instruments. No caso do projeto, foi utilizado as bibliotecas do *TivaWare*, onde auxiliam no desenvolvimento de funcionalidades utilizando o processador TM4C123G com arquitura Arm-Cortex M4.

3.3 Métodos

3.3.1 Bearing

Em navegação, *bearing* é o ângulo horizontal entre a direção entre dois objetos, ou então podendo ser entre um objeto e o norte verdadeiro. O *bearing* é medido em graus e calculado no sentido horário a partir do norte verdadeiro (KEAY, 1995).

Aplicando *bearing* em aplicações com GPS, quando o destino pretendido é inserido em um celular, o recurso GPS indica a posição em relação ao seu destino. Com essas informações, ele pode calcular a direção que deve-se tomar para avançar em direção ao seu destino. O cálculo ocorre no grau mais próximo e normalmente é a rota mais direta do ponto A ao ponto B (KEAY, 1995).

O *bearing* é calculado como um ângulo medido em graus no sentido horário do norte verdadeiro. O vértice do ângulo representa sua localização atual, enquanto os dois raios apontam para o norte e em direção ao destino da meta, respectivamente. Pode-se calcular manualmente o rumo entre dois pontos usando um mapa, uma bússola e um transferidor. No entanto, caso haja informações de latitudes e longitudes exatas dos pontos em questão, poderá ser usado a equação 10.

$$\beta = atan^2(X, Y) \tag{10}$$

Para obter os valores de X e Y da equação 10, deve-se usar a equação 11 e 12, respectivamente. Em tais equações, tem-se que L representa a longitude, representa a latitude, e é o *bearing*.

$$X = cos(b).sen(L) \tag{11}$$

$$Y = cos(a).sin(b) - sin(a).cos(b).cos(L) \tag{12}$$

.

4 DESENVOLVIMENTO

4.1 Desenvolvimento do controlador PID

Para o controle do motor *brushless* utilizado no projeto, foi necessário a implementação de um controlador PID, para que fosse possível alcançar a precisão no posicionamento angular do motor.

Primeiramente foram realizados testes de acionamento do motor, onde foi aplicado um degrau unitário de entrada e verificado o comportamento do motor. Assim, foi possível verificar o comportamento do motor no formato apresentado na figura 40.

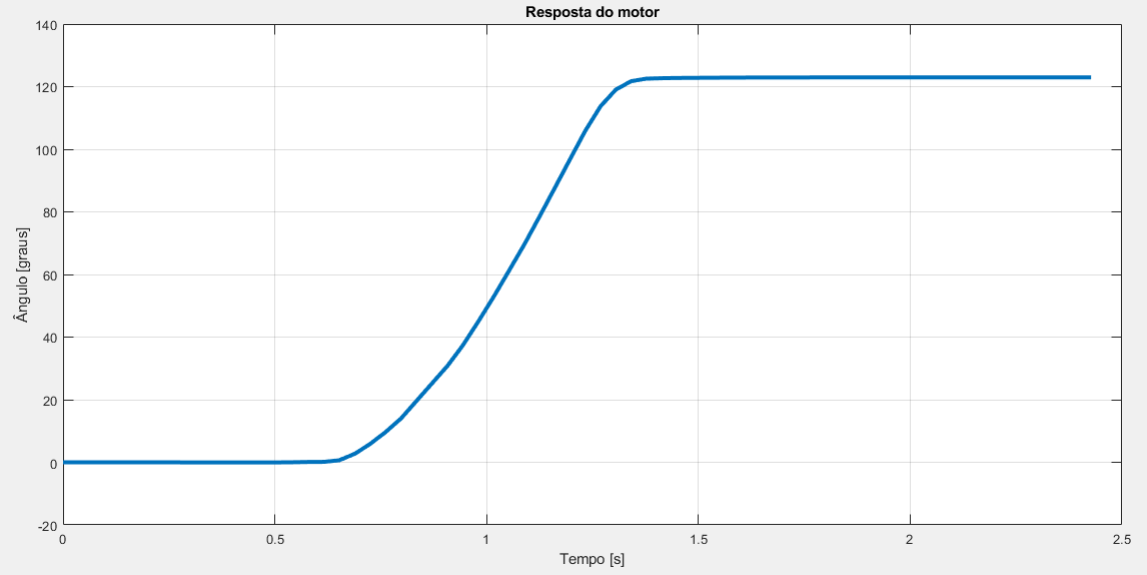

Figura 40 – Resposta ao degrau do motor *brushless* - Fonte: Autoria própria.

A partir da curva apresentada na figura 40, pode-se observar que apresentou-se dificuldade na verificação do transitório do motor, tendo em vista que a referência do sistema estava se movendo junto com a dinâmica do sistema. Então foi necessário encontrar a referência dinâmica da curva, para então poder analisar de maneira estática o comportamento do sistema. Assim, foi utilizado a equação 13 para a alteração de referência do sistema.

$$P = P - \omega * t \tag{13}$$

Sendo a variável P a posição angular do motor medida em radianos, ω o valor da velocidade angular do motor em $\frac{rad}{s}$ e t o tempo em segundos.

No entanto, apenas realizando a mudança de referência do sistema ainda não era possível identificar o transitório do motor. Então foi realizado a redução do número de pontos analisados, considerando apenas o transitório superior da curva apresenta na figura 40. A figura resultante da redução do número de pontos pode ser observada na figura 41.

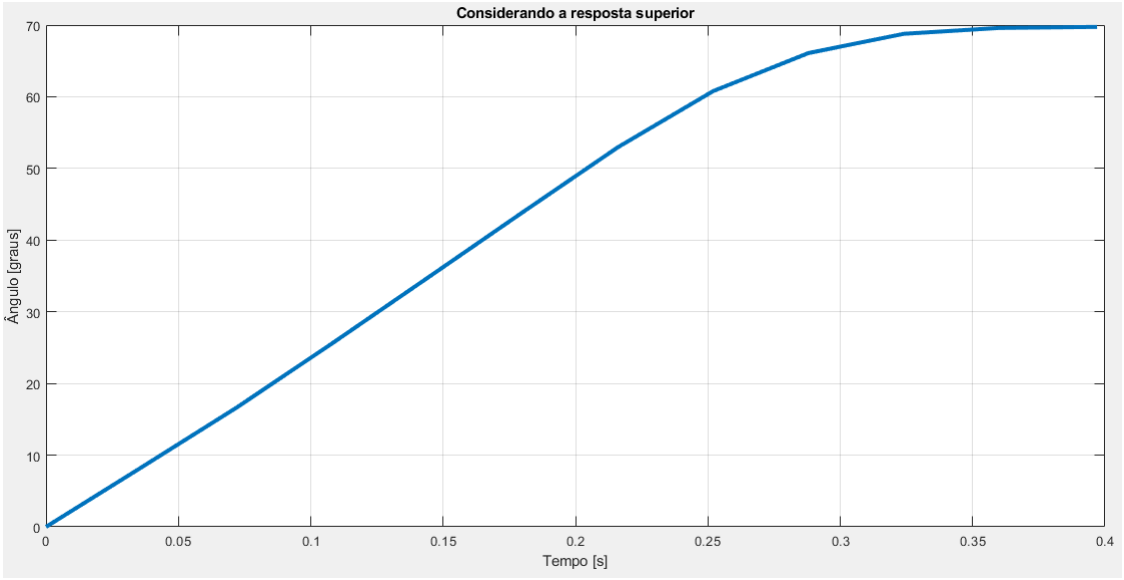

Figura 41 – Considerando a parte superior da resposta ao degrau - Fonte: Autoria própria.

Então tendo a redução do número de pontos analisados e aplicando a mudança de referência, foi possível chegar na curva apresenta na figura 42.

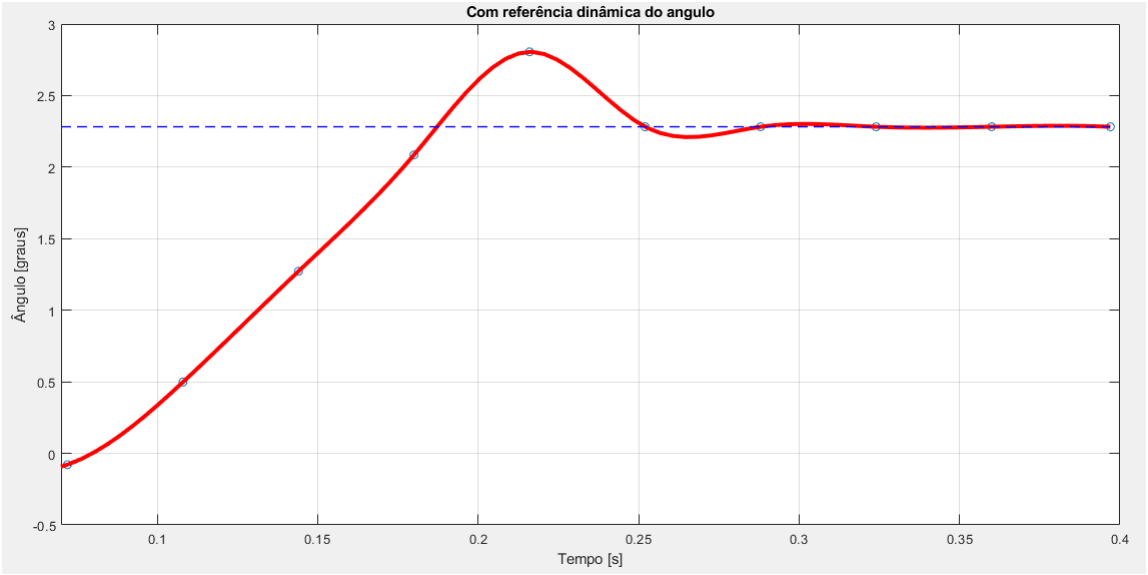

Figura 42 – Transitório do motor *brushless* - Fonte: Autoria própria.

A partir da curva apresentada na figura 42, é possível observar o comportamento transitório característico de um sistema de segunda ordem. Então a equação de transferência que descreve o comportamento de um sistema de segunda ordem pode ser observada na equação 14.

$$G(s) = \frac{K}{\tau.s + 2.\zeta.\tau.s + 1} \tag{14}$$

A partir da equação 14, pode-se observar o parâmetro K sendo o ganho, τ sendo a constante de tempo, ζ a constante de amortecimento e s a constante de Laplace.

Então para o cálculo do ganho K, basta analisar a curva apresenta na figura 42 e verificar identificar o valor angular em que o sistema se estabiliza após a passagem do transitório, onde chegou-se ao valor de 2,2822 para o ganho K.

Para o cálculo do parâmetro ζ, tem-se a equação 15 apresenta a seguir.

$$\zeta = \sqrt{\frac{ln(OS)^2}{\pi^2 + ln(OS)^2}} \tag{15}$$

Sendo o parâmetro OS da equação 15 o *overshoot* do sistema, definido pela equação 16 apresentada a seguir.

$$OS = \frac{Pico - K}{K} \tag{16}$$

Sendo o $Pico$ o valor máximo atingido pela curva durante o transitório do motor, onde analisando a curva da figura 42 chegou-se ao valor de 2,8057. Assim, calculando o valor de OS e substituindo na equação 15, tem-se o valor de OS e ζ.

$$OS = 0{,}2294$$

$$\zeta = 0{,}4243$$

Com o valor de ζ definido, é possível identificar o tipo de amortecimento que o motor está submetido. Sendo os possíveis tipos de amortecimento de sistemas apresentado na tabela 1.

Tabela 1 – Tipos de amortecimento de sistemas - Fonte: Autoria própria.

Constante	Amortecimento
0 $<\zeta<$1	Sob amortecido
$\zeta = 1$	Criticamente amortecido
$\zeta >$1	Super amortecido

Com a tabela 1, pode-se observar que o motor está submetido ao amortecimento sub amortecido, fato que também é possível observar pelo comportamento aparente da curva apresentado na figura 42.

Então sabendo que o comportamento do amortecimento do motor é sob amortecido, é possível calcular o parâmetro τ a partir de equação 17 apresentada a seguir.

$$\tau = \frac{t_p.\sqrt{1-\zeta^2}}{\pi} \tag{17}$$

Sendo t_p o instante de tempo em que ocorre o pico do transitório do motor, onde chegou-se ao valor de 0,2166 segundos. Assim, calculando o valor de τ tem-se o resultado apresentado a seguir.

$$\tau = 5{,}2313.10^{-2}$$

Com todos os parâmetros calculados, é possível substituir na equação e encontrar a função de transferência do motor. Assim, tem-se a equação de transferência do motor apresentada na equação 18.

$$G(s) = \frac{2{,}2822}{(2{,}7366.10^{-3}.s^2) + (0{,}4439).s + 1} \tag{18}$$

Com a função de transferência do motor é possível verificar iniciar a modelagem dos parâmetros P, I e D do controlador com auxílio do *software* MATLAB.

Para a validação da função de transferência, primeiramente foi montado o modelo da equação e simulado com o mesmo degrau de entrada. O modelo montado no Simulink do matlab pode ser observado na figura 43.

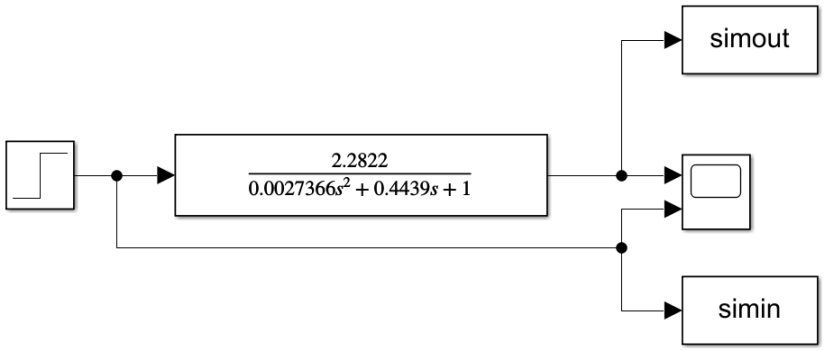

Figura 43 – Diagrama de blocos do modelo Simulink do matlab - Fonte: Autoria própria.

A partir da simulação do diagrama apresentado na figura 43, foi gerado o gráfico apresentado na figura 44.

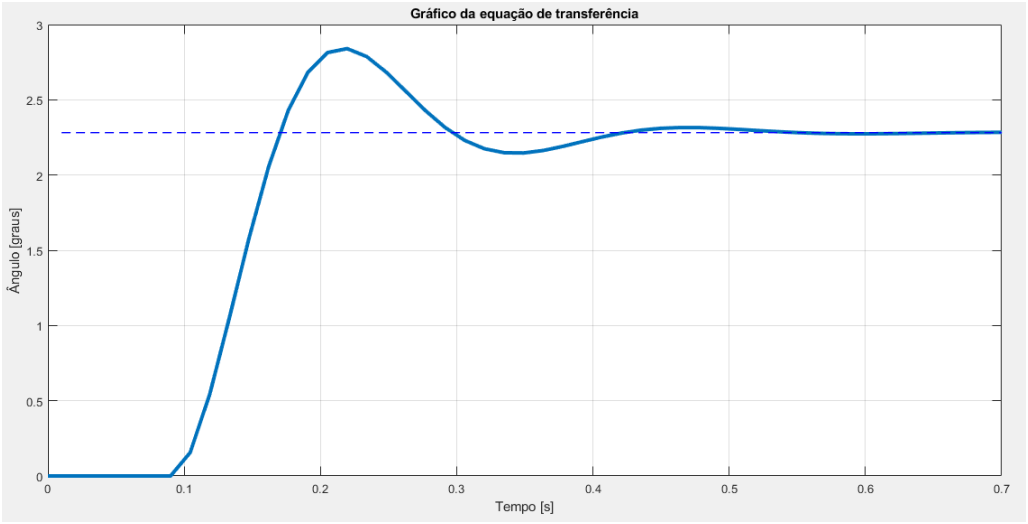

Figura 44 – Resposta do modelo encontrado para o motor - Fonte: Autoria própria.

A partir da curva apresentada na figura 44, pode-se observar que o modelo proposto

se comporta com a resposta aproximada a apresentada pelo motor, onde pode ser observada na figura 42.

Como próximo passo do projeto do controlador PID, foi montado o diagrama Simulink com o bloco de controle PID e realizado a simulação do sistema. A figura 45 apresenta o diagrama projetado no Simulink.

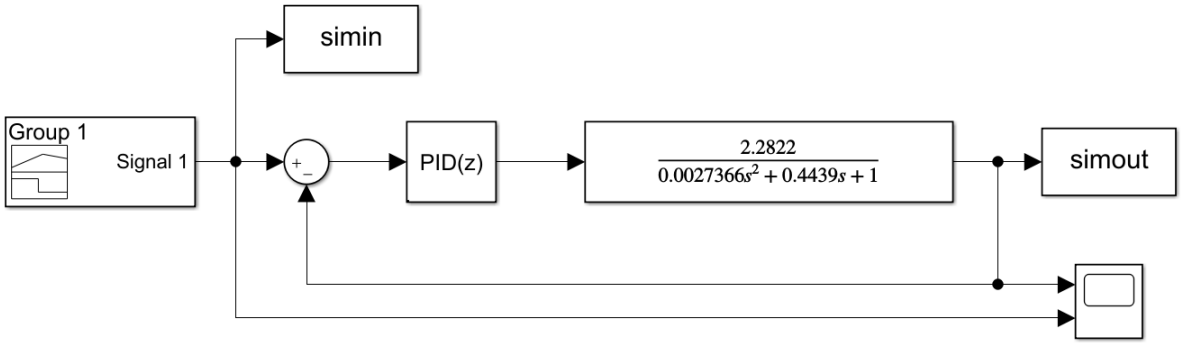

Figura 45 – Diagrama Simulink para o projeto dos parâmetro P,I e D - Fonte: Autoria própria.

Com o diagrama apresentado na figura 45, foi utilizado a ferramenta de PID tuner presente no bloco de PID do matlab, onde reliza os cálculos do PID de acordo com os métodos de aproximação de controle PID digital, decritos no tópico 2.2.2. Assim, podendo chegar aos parâmetros P,I e D apresentados a seguir.

$$P = 0.7456 \qquad I = 2.0184 \qquad D = 0.0256$$

Então substituindo os parâmetros P, I e D no bloco PID do diagrama apresentado na figura 45, encontrou-se a curava de reposta do sistema apresentada na figura 46.

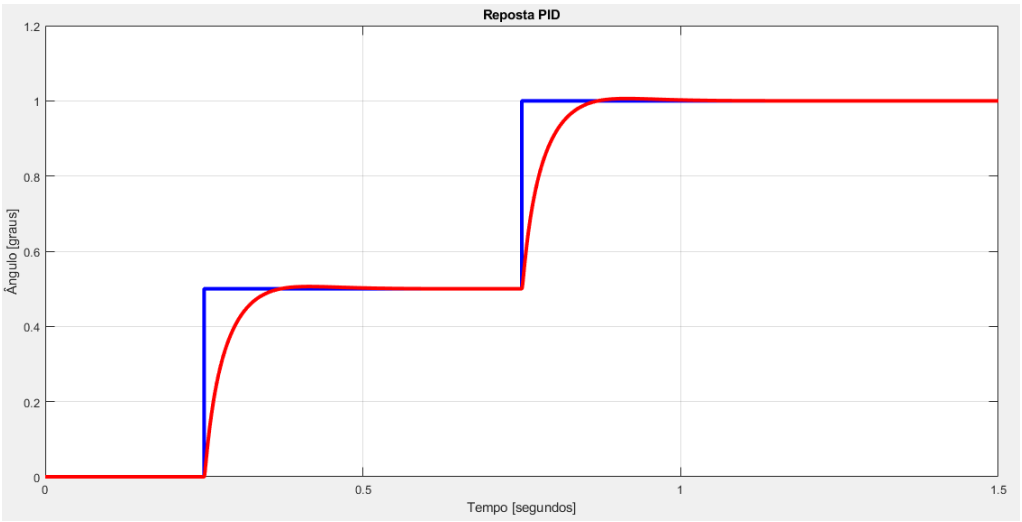

Figura 46 – Resposta simulado com o controle PID - Fonte: Autoria própria.

Sendo a curva em azul o degrau de entrada aplicado e a curva em vermelho a resposta do motor com atuação do controle PID projetado.

Então realizando a sobreposição das curvas apresentadas nas figuras 44 e 46, chegou-se a curva apresentada na figura 47.

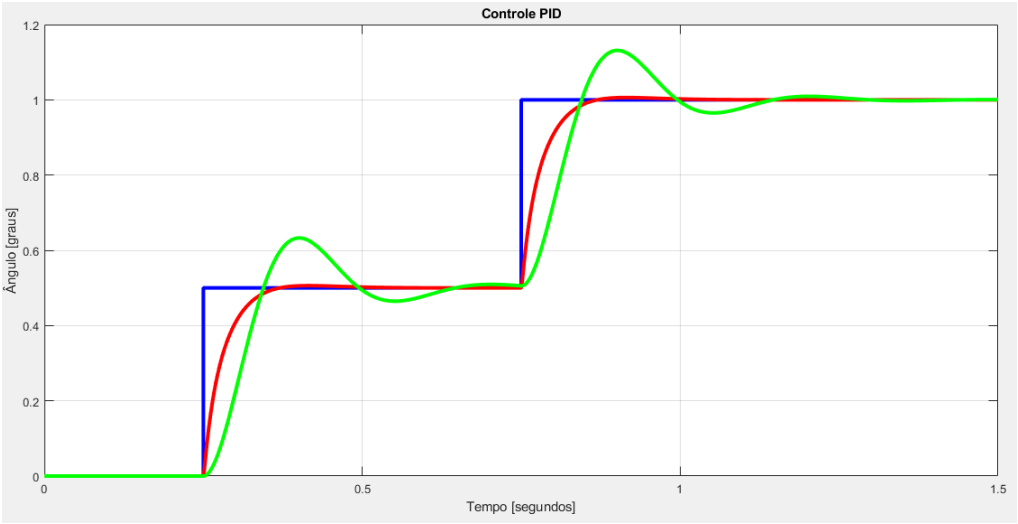

Figura 47 – Resposta com e sem controle PID - Fonte: Autoria própria.

Sendo a curva em azul a aplicação do degrau de entrada, a curva em vermelho é a resposta do motor com controle PID e a curva em verde a resposta do motor sem o controle PID.

4.2 Desenvolvimento de *Hardware*

Para o desenvolvimento do *Hardware* do projeto, foi inicialmente verificado os requisitos do projeto, onde seria necessário o acionamento de um motor *brushless* para o controle da direção do veículo. No entanto, conforme a movimentação do motor, seria necessário verificar a posição angular do volante para saber a curva que o veículo traçaria. Também fez-se necessário a comunicação com uma antena de GPS para saber a posição em que o veículo se encontra no trajeto desejado.

Tendo em vista os materiais utilizados no projeto, tem-se o diagrama de blocos apresentado na figura 48, onde é demonstrado como foi construído o *hardware* do projeto.

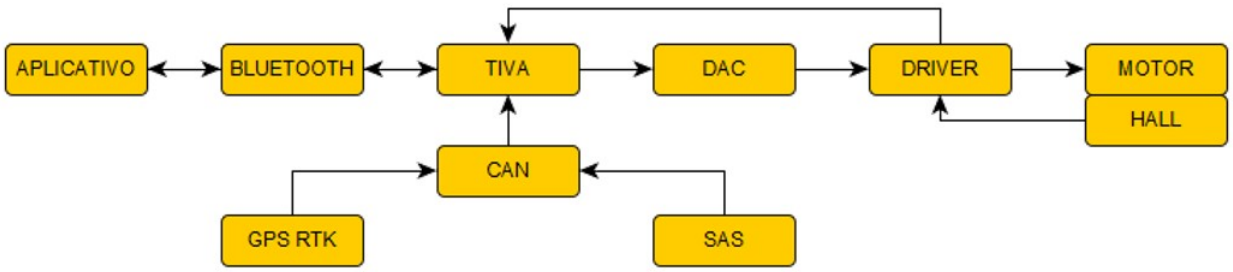

Figura 48 – Diagrama de blocos do *hardware* do projeto – Fonte: Autoria própria.

Com o diagrama de blocos apresentado na figura 48, pode-se observar que para o processamento e controle do sistema, foi especificado o microcontrolador Tiva, onde o toda a lógica de controle e atuação do sistema será programada.

Para o acionamento do motor por parte do microcontrolador, foi ligado os pinos de controle do *driver* do motor, apresentado no tópico 3.1.2, com os pinos de controle programados no microcontrolador, onde para o sentido de rotação do motor foi estabelecido apenas o comportamento de GPIO do microcontrolador.

Já para o controle de velocidade, foi utilizado a DAC MCP4725 como forma de controle do nível de tensão aplicado ao pino de entrada do *driver*, onde a comunicação foi realizada através de barramento I2C. Dessa forma, a DAC MCP4725 recebe o valor de tensão que deve ser gerado no pino de saída e repassa para o *driver* do motor, sendo então acionado o motor na velocidade correspondente.

Como sensor de ângulo do motor, foi utilizado uma placa externa que realiza as leituras de ângulo e os valores medidos são repassados por barramento CAN, onde foi utilizado a placa MCP2515 para realizar as leituras do dados transmitidos e repassados os dados para o microcontrolador através de barramento SPI.

Também através do barramento CAN são recebidos os dados de GPS, os quais são repassados ao aplicativo desenvolvido para a operação do sistema utilizando comunicação *bluetooth*. A aplicação desenvolvida realiza os cálculos com os dados de GPS recebidos e retorna os valores de ângulo para o microcontrolador poder atuar sobre o motor. Maiores detalhes do funcionamento da aplicação e o algoritmo de controle será discutida nos tópicos a seguir.

Para a alimentação dos circuitos discutidos no diagrama de blocos da figura 48, foi utilizado o circuito do regulador abaixador LM2596, que tem como função receber na entrada a tensão de 24V e reduzir para 5V, tensão que é direcionada para alimentação dos circuitos utilizados.

Uma vez definida a comunicação entre os periféricos utilizados, foi elaborada a placa de circuito impresso para unificar todos os *hardwar5es* utilizados no funcionamento do sistema. Tal placa projetada pode ser observada na figura 49.

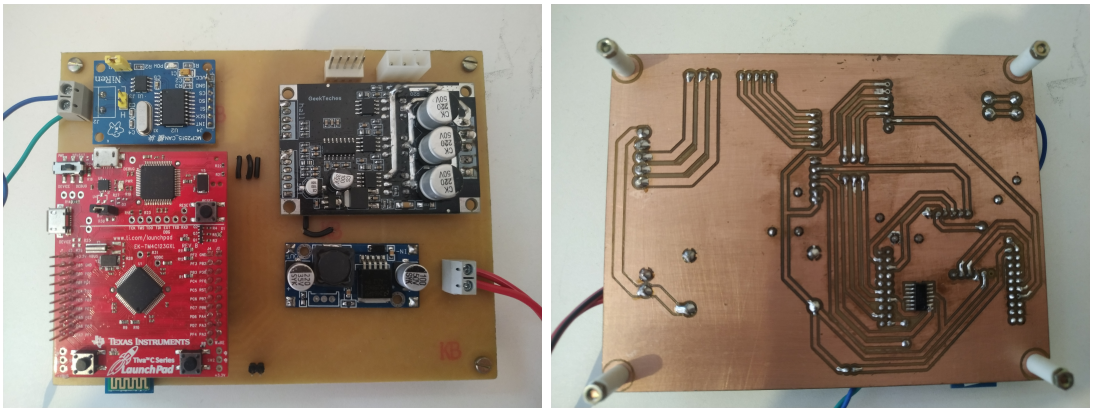

Figura 49 – Placa de circuito impresso do projeto - Fonte: Autoria própria.

A partir da figura 49 é possível observar a presença de quatro conectores, sendo o conector de 24V de alimentação dos circuitos, o conector de saída de cada fase do motor *brushless*, conector do sensor *hall* do motor e o conector do barramento CAN utilizado para a leitura do sensor de posição. A figura 50 apresenta demarcado cada conector discutido.

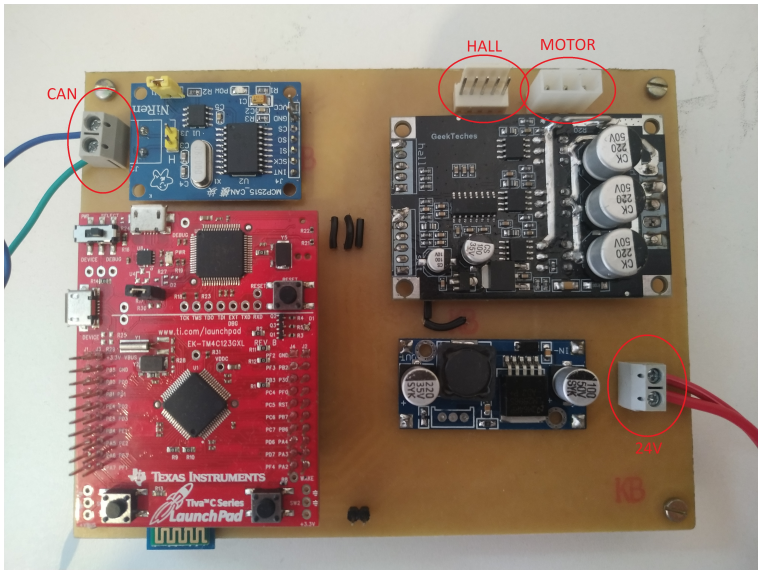

Figura 50 – Identificação dos conectores da placa de circuito impresso - Fonte: Autoria própria.

A partir da figura 49, é possível observar também os circuitos discutidos anteriormente que foram utilizados, como o *driver* do motor *brushless*, o circuito de alimentação LM2596, o circuito do CAN MCP2515 e o microcontrolador TIVA. O circuito da DAC MCP4725 e do *bluetooth* HC-05 estão montados na parte inferior do TIVA, a figura 51 apresenta a disposição das placa do DAC e *bluetooth*.

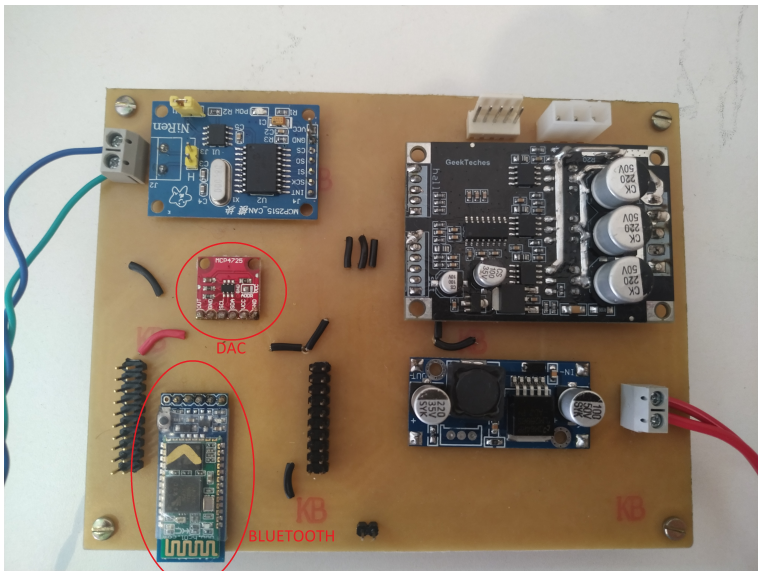

Figura 51 – Posicionamento da DAC e *Bluetooth* - Fonte: Autoria própria.

Tendo a placa eletrônica finalizada, foi projetado o protótipo mecânico do projeto, onde

foi acoplado o volante, motor e o sensor de posicionamento angular, formando o barramento de direção do veículo. Após foi acoplado os conectores do placa eletrônica com os conectores do motor. Tendo o protótipo mecânico finalizado e pronto para a realização dos testes. A figura 52 apresenta o protótipo sistema.

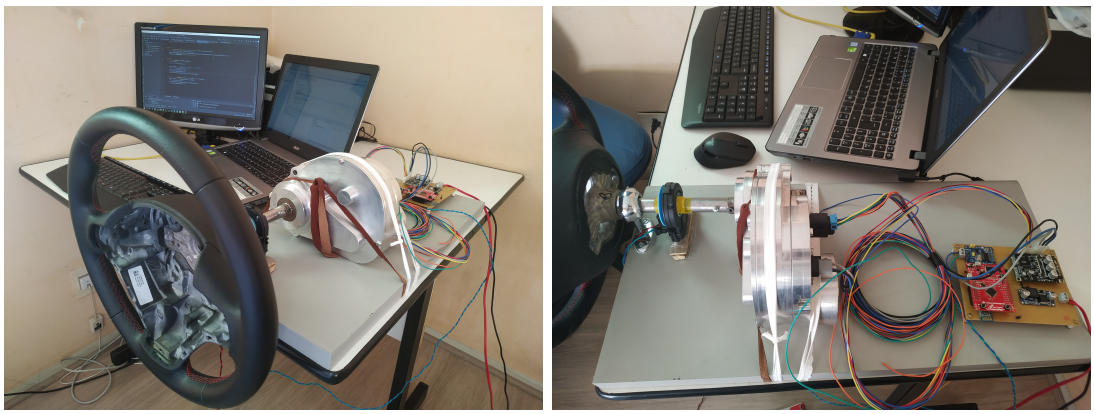

Figura 52 – Protótipo completo do projeto - Fonte: Autoria própria.

4.3 Desenvolvimento de *Firmware*

Com relação ao *firmware* desenvolvido, será discutido a estruturação do código por completo e a forma em que os módulo interagem, assim a explicação das funcionalidades de cada módulo. Os códigos completos podem ser observados no anexo A.

Para o desenvolvimento do *firmware*, foi estabelecido uma estrutura em módulos, a fim de obter um melhor entendimento dos códigos e facilitar a implementação do mesmo. A ideia da estruturação do *firmware* pode ser observada na figura 53.

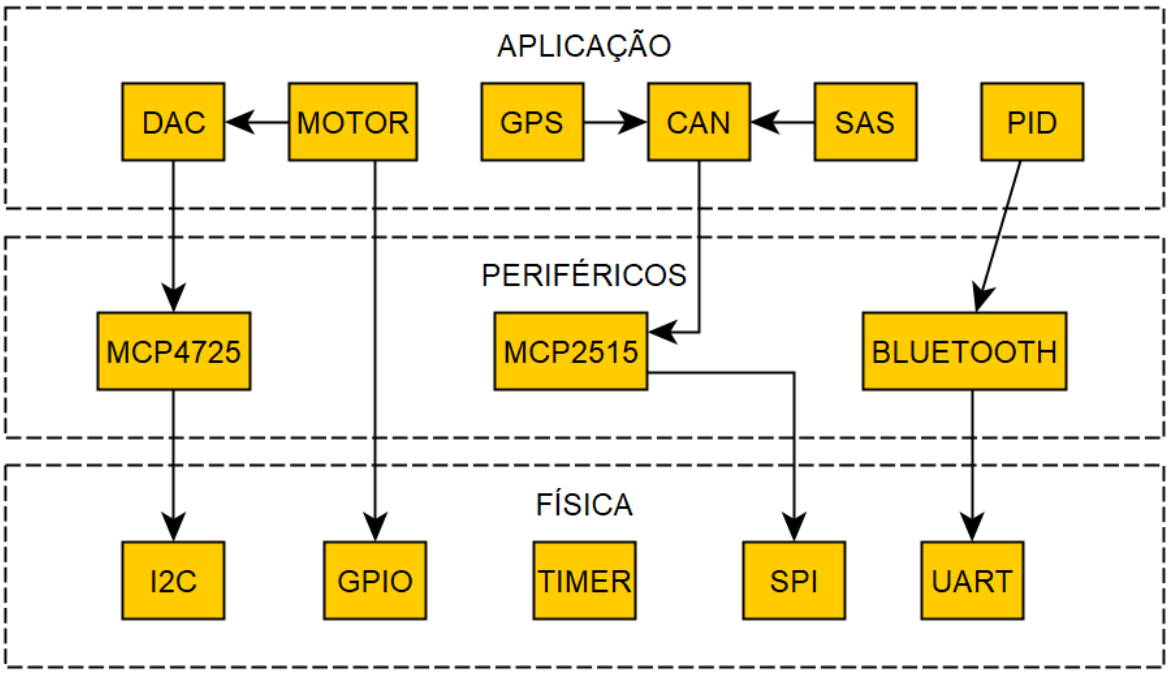

Figura 53 – Estrutura do *firmware* - Fonte: Autoria própria.

A partir da figura 53, é possível observar que o código foi estruturado em camadas, seguindo a sequência camada física, periféricos e aplicação.

Na camada física, foi estabelecido os periféricos de *hardware* do microcontrolador, como as configurações dos barramentos de comunicação I2C, SPI e UART, configuração das portas de IO e a configuração do *TIMER* para a temporização das rotinas.

A camada dos periféricos faz uso dos módulos de código apresentados na camada física, onde as dependências de cada módulo do periférico com os módulos da camada física pode ser observado na figura 53.

A camada de aplicação fica com os módulo lógicos do sistema, onde realizam os cálculos necessários para o funcionamento do sistema. Quando necessário o acionamento de algum periférico de hardware, a aplicação passa as ações para a camada de periférico que por sua vez aciona o módulo da fisíca para efetivamente executar a ação desejada.

Tendo em vista os módulo da camada física, o barramento I2C foi configurado com os pinos PB2 e PB3 do microcontrolador como os pinos SCL e SDA do barramento, respectivamente. Após, é inicializado o barramento I2C na frequência de 80MHz de operação do microcontrolador. O código 4.1 apresenta os comandos de configuração da I2C.

```
1  //*****************************************************************************
2  // Configuration of I2C0 pins to communicate with external modules
3  //*****************************************************************************
4  void I2CConfiguration(void)
5  {
6      SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
7      SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
8      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
9
10     GPIOPinConfigure(GPIO_PB2_I2C0SCL);
11     GPIOPinConfigure(GPIO_PB3_I2C0SDA);
12
13     GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
14     GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
15
16     I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);
17     HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
18 }
```

Código 4.1 – Código de configuração I2C

Tendo o barramento I2C configurado basta a utilização dos comandos de envio e recepção dos dados, podendo ser observado nos códigos 4.2 e 4.3.

```
1  //*****************************************************************************
2  // Function to Send an I2C command to the specified slave
3  //*****************************************************************************
4  void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...)
5  {
6      I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
7
8      va_list vargs;
9
10     va_start(vargs, num_of_args);
```

```
11      I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
12
13      if(num_of_args == 1)
14      {
15          I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
16          while(I2CMasterBusy(I2C0_BASE));
17          va_end(vargs);
18      }
19      else
20      {
21          I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
22          while(I2CMasterBusy(I2C0_BASE));
23
24          uint8_t i=0;
25          for(i = 1; i < (num_of_args - 1); i++)
26          {
27              I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
28              I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
29              while(I2CMasterBusy(I2C0_BASE));
30          }
31
32          I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
33          I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
34          while(I2CMasterBusy(I2C0_BASE));
35          va_end(vargs);
36      }
37  }
```

Código 4.2 – Código de envio de mensagem I2C

```
1  //*****************************************************************************
2  // Function to read specified register on slave device
3  //*****************************************************************************
4  uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
5  {
6      I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
7      I2CMasterDataPut(I2C0_BASE, reg);
8      I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
9      while(I2CMasterBusy(I2C0_BASE));
10     I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);
11     I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
12     while(I2CMasterBusy(I2C0_BASE));
13     return I2CMasterDataGet(I2C0_BASE);
14 }
```

Código 4.3 – Código de recepção de mensagem I2C

No caso do barramento SPI, foi configurado os pinos PA3, PB4, PB6 e PB7 como CS, CLK, MISO, MOSI, respectivamente, de acordo com o padrão do barramento SPI. Utilizando o barramento de I2C do microcontrolador, configurado em modo master com a frequência de 4MHz de operação e 8 bits por envio de mensagem. O código 4.4 apresenta a configuração da SPI.

```
1  //*****************************************************************************
2  // SPI master configuration pins
3  //*****************************************************************************
```

```
4  void SPIMasterConfiguration(void)
5  {
6      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
7      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3);
8      GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_PIN_3);
9
10     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
11     GPIOPinConfigure(GPIO_PB4_SSI2CLK);
12     GPIOPinConfigure(GPIO_PB6_SSI2RX);
13     GPIOPinConfigure(GPIO_PB7_SSI2TX);
14     GPIOPinTypeSSI(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_6 | GPIO_PIN_7);
15
16     SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);
17     SSIClockSourceGet(SSI2_BASE);
18     SSIConfigSetExpClk(SSI2_BASE, SysCtlClockGet(),
19                        SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER,
20                        4000000, 8);
21     SSIEnable(SSI2_BASE);
22     SPIFifoClear();
23 }
```

Código 4.4 – Código de configuração SPI

Tendo a configuração da SPI realizada, basta executar os comandos de chip select e os comandos de escrita e leitura dos dados. Os códigos 4.5, 4.6 e 4.7 apresenta os códigos para o chip select, escrita e leitura, respectivamente.

```
1  //*****************************************************************************
2  // SPI function to define the state of CS pin
3  //*****************************************************************************
4  void SPICsPin(uint8_t csCommand)
5  {
6      GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, csCommand);
7  }
```

Código 4.5 – Código de chip select da SPI

```
1  //*****************************************************************************
2  // SPI write function
3  //*****************************************************************************
4  void SPIByteWrite(uint8_t byte)
5  {
6      SSIDataPutNonBlocking(SSI2_BASE, byte);
7  }
```

Código 4.6 – Código de escrita da SPI

```
1  //*****************************************************************************
2  // SPI read function
3  //*****************************************************************************
4  void SPIByteRead(uint32_t *byte)
5  {
6      SSIDataGetNonBlocking(SSI2_BASE, byte);
7  }
```

Código 4.7 – Código de leitura da SPI

Para o caso da utilização da UART é semelhante ao executado para a configuração da I2C e SPI, onde inicialmente realiza a configuração do periférico para então executar os comandos de escrita e leitura dos dados do barramento. No caso do projeto foi configurado o periférico de UART5 do microcontrolador, onde utiliza os pinos PE4 e PE5 sendo RX e TX, respectivamente. Os códigos 4.8, 4.9 e 4.10 demonstram a configuração da UART e os comandos de envio e recepção dos dados, respectivamente.

```
//*****************************************************************************
// Configuration of UART5 pins
//*****************************************************************************
void UART5Configuration(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
    GPIOPinConfigure(GPIO_PE4_U5RX);
    GPIOPinConfigure(GPIO_PE5_U5TX);
    GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
    UARTConfigSetExpClk(UART5_BASE, SysCtlClockGet(), 9600,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
    UARTEnable(UART5_BASE);
}
```

Código 4.8 – Código de configuração da UART

```
//*****************************************************************************
// Function to write char with UART5
//*****************************************************************************
void UART5Write(uint8_t c)
{
    UARTCharPutNonBlocking(UART5_BASE, c);
}
```

Código 4.9 – Código de escrita da UART

```
//*****************************************************************************
// Function to read char with UART5
//*****************************************************************************
void UART5Read(uint8_t *c)
{
    *c = UARTCharGetNonBlocking(UART5_BASE);
    SysCtlDelay(25000);
}
```

Código 4.10 – Código de leitura da UART

No caso do módulo de GPIO foi estabelecido para a configuração das portas de IO utilizadas no projeto, sendo as funções de acionamento do sentido de rotação do motor e as funções para o cálculo da velocidade de rotação do motor a partir da interrupção de borda programada. Os códigos 4.11 e 4.12 apresentam os comandos definidos para os pinos de controle do motor e a configuração dos pinos de interrupção.

```
//*****************************************************************************
// GPIO PC7 configuration to set the direction of motor rotation
//*****************************************************************************
```

```
4  void GPIOConfiguration(void)
5  {
6      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
7      GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);
8      GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x0);
9  }
10
11 //*****************************************************************************
12 // GPIO function to set motor right direction
13 //*****************************************************************************
14 void GPIOSetMotorRight(void)
15 {
16     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
17 }
18
19 //*****************************************************************************
20 // GPIO function to set motor left direction
21 //*****************************************************************************
22 void GPIOSetMotorLeft(void)
23 {
24     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
25 }
```

Código 4.11 – Código de configuração e estados dos pino das GPIO para o motor

```
1  //*****************************************************************************
2  // GPIO PF4 configuration to measure the velocity of motor
3  //*****************************************************************************
4  void GPIOIntConfiguration(void)
5  {
6      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
7      GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);
8      GPIOIntDisable(GPIO_PORTF_BASE, GPIO_PIN_4);
9      GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_RISING_EDGE);
10     GPIOIntRegister(GPIO_PORTF_BASE, GPIOInterruptHandler);
11     GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_4);
12     GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4);
13 }
```

Código 4.12 – Código do pino de interrupção de borda para calculo da velocidade do motor

No caso do módulo TIMER, foi utilizado para temporizar a execução dos códigos, separando as ações do controle, acionamentos e comunicação de acordo com a necessidade do sistema e de forma a executar cada ação no momento em que se tenha utilizadade, reduzindo o disperdicio de tempo de execução do microcontrolador e otimizando a execução do código. O código 4.13 apresenta os comandos utilizados para a configuração do TIMER.

```
1  //*****************************************************************************
2  // Timer interrupt configuration to measure the velocity of motor
3  //*****************************************************************************
4  void TimerIntConfiguration(void)
5  {
6      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
7      TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC_UP);
8      TimerLoadSet(TIMER0_BASE, TIMER_A, TIMER_MAX_VALUE);
9      TimerIntDisable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
```

```
10      TimerIntRegister(TIMER0_BASE, TIMER_A, TimerInterruptHandler);
11      TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
12      TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
13      TimerEnable(TIMER0_BASE, TIMER_A);
14  }
```

Código 4.13 – Código de configuração do TIMER

Tendo em vista a camada dos periféricos, foi estabelecido o módulo MCP4725 onde realiza a configuração do barramento I2C que será utilizado para a comunicação com o CI MCP4725, também implementa a função de escrita dos dados realizando a preparação do dado desejado no padrão de recepção estabelecido pelo fabricante do CI MCP4725, para então transmitir o dado para o módulo I2C enviar o dado ao barramento. Os códigos 4.14 apresenta o comando de configuração e envio do módulo MCP4725.

```
1  //*****************************************************************************
2  // Function to configure MCP4725 DAC
3  //*****************************************************************************
4  void MCP4725Configuration(void)
5  {
6      I2CConfiguration();
7  }
8
9  //*****************************************************************************
10 // Function to send DAC data to MCP4725 module using I2C0 bus
11 //*****************************************************************************
12 void MCP4725Write(uint16_t dacValue)
13 {
14     uint8_t high, low;
15     high = (dacValue >> 4) & 0x00FF;
16     low = ((dacValue & 0x000F) << 4);
17     I2CSend(MCP4725_ADDRESS,3,MCP4725_WRITE_REGISTER,high,low);
18 }
```

Código 4.14 – Código de configuração e envio de dados do MCP4725

Considerando o módulo MCP2515, foi estabelecido a função de configuração do módulo onde realiza a configuração do módulo de SPI que é utilizado como interface de comunicação com o CI MCP2515. Já os comandos de escrita e leitura dos dados, foi construido de acordo com as funcionalidades disponibilizadas pelo fabricante do CI MCP2515, sendo a funcionalidade de escrita e leitura dos registradores, a possibilidade de escrita e leitura de um bit específico dos registradores, as funcionalidades de leitura do status de operação do CI MCP2515 para facilitar o controle na utilização do circuito integrado e os comandos de escrita e leitura dos dados no buffer de envio e recepção dos dados do CI MCP2515. O código 4.15 apresenta os comandos utilizados para o controle do periférico MCP2515.

```
1  //*****************************************************************************
2  // MCP2515 Spi Configuration
3  //*****************************************************************************
4  void MCP2515SpiConfiguration(void)
5  {
6      SPIMasterConfiguration();
7  }
```

```
8
9  //*****************************************************************************
10 // MCP2515 transfer data
11 //*****************************************************************************
12 uint8_t MCP2515TransferData(uint8_t byte)
13 {
14     uint32_t ret;
15     SPIByteWrite(byte);
16     while(true == SSIBusy(SSI2_BASE));
17     SPIByteRead(&ret);
18     return ret;
19 }
20
21 //*****************************************************************************
22 // MCP2515 write register
23 //*****************************************************************************
24 void MCP2515WriteRegister(uint8_t address, uint8_t data)
25 {
26     SPICsPin(MCP2515_CS_ACT);
27     MCP2515TransferData(MCP2515_WRITE_CMD);
28     MCP2515TransferData(address);
29     MCP2515TransferData(data);
30     SPICsPin(MCP2515_CS_DACT);
31 }
32
33 //*****************************************************************************
34 // MCP2515 read register
35 //*****************************************************************************
36 void MCP2515ReadRegister(uint8_t address, uint8_t *data)
37 {
38     SPICsPin(MCP2515_CS_ACT);
39     MCP2515TransferData(MCP2515_READ_CMD);
40     MCP2515TransferData(address);
41     *data = MCP2515TransferData(MCP2515_DUMMY_DATA);
42     SPICsPin(MCP2515_CS_DACT);
43 }
44
45 //*****************************************************************************
46 // MCP2515 clear register bits
47 //*****************************************************************************
48 void MCP2515ClearBits(uint8_t address, uint8_t bits)
49 {
50     SPICsPin(MCP2515_CS_ACT);
51     MCP2515TransferData(MCP2515_BIT_MODIFY_CMD);
52     MCP2515TransferData(address);
53     MCP2515TransferData(bits);
54     MCP2515TransferData(MCP2515_BIT_CLEAR);
55     SPICsPin(MCP2515_CS_DACT);
56 }
57
58 //*****************************************************************************
59 // MCP2515 set register bits
60 //*****************************************************************************
61 void MCP2515SetBits(uint8_t address, uint8_t bits, uint8_t data)
62 {
63     SPICsPin(MCP2515_CS_ACT);
64     MCP2515TransferData(MCP2515_BIT_MODIFY_CMD);
```

```
65      MCP2515TransferData(address);
66      MCP2515TransferData(bits);
67      MCP2515TransferData(data);
68      SPICsPin(MCP2515_CS_DACT);
69  }
70
71  //*****************************************************************************
72  // MCP2515 reset
73  //*****************************************************************************
74  void MCP2515Reset(void)
75  {
76      uint8_t status = 0;
77
78      SPICsPin(MCP2515_CS_ACT);
79      MCP2515TransferData(MCP2515_RESET_CMD);
80      SPICsPin(MCP2515_CS_DACT);
81
82      while(MCP2515_MODE_CFG != status)
83      {
84          MCP2515ReadRegister(MCP2515_CANSTAT, &status);
85      }
86  }
87
88  //*****************************************************************************
89  // MCP2515 request to send message
90  //*****************************************************************************
91  void MCP2515RequestSend(uint8_t txBuffID)
92  {
93      SPICsPin(MCP2515_CS_ACT);
94      MCP2515TransferData(MCP2515_RTS_CMD | (1U << txBuffID));
95      SPICsPin(MCP2515_CS_DACT);
96  }
97
98  //*****************************************************************************
99  // MCP2515 read status operation
100 //*****************************************************************************
101 void MCP2515ReadStatus(uint8_t *status)
102 {
103     SPICsPin(MCP2515_CS_ACT);
104     MCP2515TransferData(MCP2515_READ_STATUS_CMD);
105     *status = MCP2515TransferData(MCP2515_DUMMY_DATA);
106     SPICsPin(MCP2515_CS_DACT);
107 }
108
109 //*****************************************************************************
110 // MCP2515 write data
111 //*****************************************************************************
112 void MCP2515Write(MCP2515Msg_t *mcpMsg)
113 {
114     uint8_t i, tbufdata[4];
115     tbufdata[0] = (mcpMsg->msgID >> 3);
116     tbufdata[1] = ((mcpMsg->msgID & 0x07) << 5);
117     tbufdata[2] = 0;
118     tbufdata[3] = 0;
119
120     SPICsPin(MCP2515_CS_ACT);
121     MCP2515TransferData(MCP2515_WRITE_TXB_CMD);
```

```
122     for(i=0;i<4;i++) MCP2515TransferData(tbufdata[i]);
123     MCP2515TransferData(mcpMsg->msgLen);
124     for(i=0;i<8;i++) MCP2515TransferData(mcpMsg->msgData[i]);
125     SPICsPin(MCP2515_CS_DACT);
126 
127     MCP2515TransferData(0x81);
128 }
129 
130 //*****************************************************************************
131 // MCP2515 read data
132 //*****************************************************************************
133 void MCP2515Read(MCP2515Msg_t *mcpMsg)
134 {
135     uint8_t i, status, tbufdata[4], len;
136     MCP2515ReadStatus(&status);
137 
138     if(status == 3)
139     {
140         SPICsPin(MCP2515_CS_ACT);
141         MCP2515TransferData(MCP2515_Read_RXB_CMD);
142         for(i=0;i<4;i++) tbufdata[i] =
143         MCP2515TransferData(MCP2515_DUMMY_DATA);
144         mcpMsg->msgID = (tbufdata[0] << 3) + (tbufdata[1] >> 5);
145         len = MCP2515TransferData(MCP2515_DUMMY_DATA);
146         mcpMsg->msgLen = len & 0x0F;
147         for(i=0;i<8;i++) mcpMsg->msgData[i] =
148         MCP2515TransferData(MCP2515_DUMMY_DATA);
149         SPICsPin(MCP2515_CS_DACT);
150     }
151 }
```

Código 4.15 – Código com os comandos de controle do periférico do MCP2515

Para o módulo do BLUETOOTH, semelhante ao realizado no módulo de MCP4725 e do MCP2515, inicialmente é estabelecido o comando de configuração onde realiza aciona o comando de configuração do módulo de UART para utilizar o barramento serial que será utilizado para a comunicação com o periférico HC-05. Após são definidos os comandos de escrita e leitura dos dados recebidos pelo periférico HC-05. O código **??** apresenta os comando estabelecidos para a utilização do módulo do BLUETOOTH.

```
1  //*****************************************************************************
2  // Bluetooth configuration
3  //*****************************************************************************
4  void BluetoothConfig(void)
5  {
6      UART5Configuration();
7  }
8  
9  //*****************************************************************************
10 // Function to check Bluetooth available data
11 //*****************************************************************************
12 uint8_t BluetoothAvailableData(void)
13 {
14     uint8_t ret;
15     ret = UART5AvailableData();
```

```
16      return ret;
17  }
18
19  //*****************************************************************************
20  // Function to send bluetooth data
21  //*****************************************************************************
22  void BluetoothSend(Bluetooth_t BT)
23  {
24      uint8_t data[9], i;
25
26      switch(BT.type)
27      {
28          case 'G':
29              data[0] = BT.type;
30              data[1] = ((BT.latitude & 0xFF000000) >> 24);
31              data[2] = ((BT.latitude & 0x00FF0000) >> 16);
32              data[3] = ((BT.latitude & 0x0000FF00) >> 8);
33              data[4] = ((BT.latitude & 0x000000FF) >> 0);
34              data[5] = ((BT.longitude & 0xFF000000) >> 24);
35              data[6] = ((BT.longitude & 0x00FF0000) >> 16);
36              data[7] = ((BT.longitude & 0x0000FF00) >> 8);
37              data[8] = ((BT.longitude & 0x000000FF) >> 0);
38              for(i=0;i<9;i++){
39                  UART5Write(data[i]);
40              }
41              break;
42          case 'A':
43              data[0] = BT.type;
44              data[1] = ((BT.angle & 0xFF00) >> 8);
45              data[2] = ((BT.angle & 0x00FF) >> 0);
46              for(i=0;i<3;i++){
47                  UART5Write(data[i]);
48              }
49              break;
50      }
51  }
52
53  //*****************************************************************************
54  // Function to receive bluetooth data
55  //*****************************************************************************
56  void BluetoothRecv(Bluetooth_t *BT)
57  {
58      uint8_t data[10],i;
59
60      if(UART5AvailableData())
61      {
62          UART5Read(&data[0]);
63          switch(data[0])
64          {
65              case 'G':
66                  for(i=0;i<8;i++){
67                      UART5Read(&data[i+1]);
68                  }
69                  BT->latitude = (data[1]<<24) + (data[2]<<16)
70                               + (data[3]<<8) + data[4];
71                  BT->longitude = (data[5]<<24) + (data[6]<<16)
72                                + (data[7]<<8) + data[8];
```

```
73                  break;
74              case 'A':
75                  for(i=0;i<2;i++){
76                      UART5Read(&data[i+1]);
77                  }
78                  BT->angle = (data[1]<<8) + data[2];
79                  break;
80          }
81      }
82  }
```

Código 4.16 – Código com os comandos de controle do periférico do BLUETOOTH

No caso do módulo BLUETOOTH, foi implementado o protocolo de comunicação para a comunicação do microcontrolador com o aplicativo do projeto, onde a estrutura do protocolo pode ser observada na tabela 2.

Tabela 2 – Protocolo de comunicação do microcontrolador com o aplicativo.

Protocolo	
Comando	Dados
G	Latitude, Longitude
A	Ângulo

A partir da tabela 2, pode-se observar a presença de dois comandos, o comando de GPS e ângulo. Quando recebido o comando "A" referente ao ângulo, o código realiza o tratamento do número de bytes para a construção dos dados de ângulo, caso recebido o comando "G" referente ao GPS, o código realiza o tratamento dos dados relacionados aos bytes de latitude e longitude recebidos da antena de GPS. A forma que é realizada o tratamento dos dados pode ser observada no código 4.16

Tendo em vista a camada da aplicação, tem-se os módulos de DAC e MOTOR, onde trabalham em conjunto para o acionamento do motor. Assim, inicialmente no módulo DAC é acionado o comando de configuração que realiza a configuração do módulo do MCP4725, descrito anteriormente. Também é estabelecido o comando de escrita que será encaminhado ao módulo com o valor desejado por parte da lógica estabelecida no sistema. O código 4.17 apresenta a estrutura do código estabelecida para o módulo DAC.

```
1  //*****************************************************************************
2  // Function to configure DAC
3  //*****************************************************************************
4  void DACConfiguration(void)
5  {
6      MCP4725Configuration();
7  }
8
9  //*****************************************************************************
10 // Function to send DAC data
11 //*****************************************************************************
12 void DACWrite(uint16_t dacValue)
13 {
```

```
14      MCP4725Write(dacValue);
15  }
```

Código 4.17 – Código com os comandos de controle do módulo DAC

No caso do módulo MOTOR, é onde foi utilizado a estrutura de escrita do módulo DAC e os comandos de direção de rotação do módulo GPIO, verificando e estabelecendo o dado, fechando toda a estrutura de código da verificação do valor desejado de tensão do usuário até a escrita no CI MCP4725 para realizar o acionamento do motor. O código 4.18 apresenta os comandos do módulo MOTOR.

```
1  //*****************************************************************************
2  // Update motor parameters
3  //*****************************************************************************
4  #include "SteeringControl/motor.h"
5
6  //*****************************************************************************
7  // Update motor parameters
8  //*****************************************************************************
9  void UpdateMotor(motor_t *motor)
10 {
11     if(motor->direction == 0)
12     {
13         GPIOSetMotorRight();
14         DACWrite(motor->voltage);
15     }
16     else if(motor->direction == 1)
17     {
18         GPIOSetMotorLeft();
19         DACWrite(motor->voltage);
20     }
21 }
```

Código 4.18 – Código com os comandos de controle do módulo MOTOR

Considerando o módulo CAN, primeiramente foi construído o comando de configuração de acordo com as instruções do fabricante do CI MCP2515, utilizando os comandos implementados no módulo MCP2515. O código 4.19 apresenta os comandos de configuração do módulo CAN.

```
1  //*****************************************************************************
2  // Configuration of MCP2515 CAN module
3  //*****************************************************************************
4  void CanConfiguration(void)
5  {
6      MCP2515SpiConfiguration();
7
8      SysCtlDelay(800);
9
10     MCP2515Reset();
11
12     MCP2515SetBits(MCP2515_CANINTF, 0x40, 0x00);
13     MCP2515SetBits(MCP2515_CANCTRL, 0xE0, MCP2515_MODE_CFG);
14
15     SysCtlDelay(800);
```

```
16
17      MCP2515WriteRegister(MCP2515_CNF1, MCP2515_8MHz_250kBPS_CFG1);
18      MCP2515WriteRegister(MCP2515_CNF2, MCP2515_8MHz_250kBPS_CFG2);
19      MCP2515WriteRegister(MCP2515_CNF3, MCP2515_8MHz_250kBPS_CFG3);
20
21      SysCtlDelay(800);
22
23      uint8_t i = 0;
24      for(i=0;i<14;i++){
25          MCP2515WriteRegister(MCP2515_TXB0CTRL + i, 0x00);
26          MCP2515WriteRegister(MCP2515_TXB1CTRL + i, 0x00);
27          MCP2515WriteRegister(MCP2515_TXB2CTRL + i, 0x00);
28      }
29
30      MCP2515WriteRegister(MCP2515_RXB0CTRL, 0x00);
31      MCP2515WriteRegister(MCP2515_RXB1CTRL, 0x00);
32
33      MCP2515WriteRegister(MCP2515_CANINTE, 0x01 | 0x02);
34
35      MCP2515SetBits(MCP2515_RXB0CTRL, (0x60 | (1<<2)), (0x60 | (1<<2)));
36      MCP2515SetBits(MCP2515_RXB1CTRL, 0x60, 0x60);
37
38      MCP2515SetBits(MCP2515_CANINTF, 0x40, 0x00);
39      MCP2515SetBits(MCP2515_CANCTRL, 0xE0, MCP2515_MODE_NORMAL);
40
41      SysCtlDelay(800);
42  }
```

Código 4.19 – Código com o comando de configuração do móculo CAN

A partir do código 4.19, pode-se observar que inicialmente é acionado o comando de configuração do barramento SPI utilizado pelo MCP2515. Após são realizados os comandos de escrita nos registradores do CI MCP2515, onde a sequência em que os comandos são executados, assim como os valores que foram estabelecidos para cada registrador foram definidos de acordo com a folha de dados do fabricante do CI MCP2515.

Para melhor compreender o comando de configuração do MCP2515, primeiramente é realizado o comando de *reset* do MCP2515, pois caso se tenha alguma configuração errada salva previamente tem-se a garantia de que vai ser retorna ao padrão inicial de funcionamento do chip. Após é realizado o comando que desabilita a interface de interrupção de pinos do chip, pois no hardware do projeto não foi utilizado as interrupções de pino do chip, apenas acesso via *software* através do barramento I2C. Para então executar o comando que coloca o MCP2515 em modo de configuração.

Continuando a configuração do MCP2515, foi executado os comandos de configuração do *baud rate* do barramento CAN, onde foi configurado com 250kbps, pois a antena GPS e o sensor de posicionamento se comunicam com esse *baud rate*. Após é realizado a limpeza dos registradores de transmissão e recepção, para que não se mantenha possíveis dados anteriores armazenados.

O próximo passo foi a execução do comando de configuração dos *buffer* de recepção, onde foi configurado as mascaras e filtros de recepção para que a mensagem receba o tratamento

adequado antes de ser armazenada. E por fim é configurado o chip modo de operação normal, para iniciar a operação de acordo com os parâmetros configurados.

Tendo o barramento CAN devidamente configurado, basta executar os comandos de leitura e escrita do módulo CAN. Os códigos 4.20 e 4.21 apresentam a implementação dos comandos de leitura e escrita do módulo CAN, respectivamente.

```
1  //*************************************************************************
2  // Receive CAN data from MCP2515 module
3  //*************************************************************************
4  void CanRead(CANMsg_t *canMsg)
5  {
6      MCP2515Msg_t mcpMsg;
7      uint8_t buf[8];
8
9      mcpMsg.msgID = 0;
10     mcpMsg.msgIDMask = 0;
11     mcpMsg.flags = 0;
12     mcpMsg.msgLen = 0;
13     mcpMsg.msgData = buf;
14
15     MCP2515Read(&mcpMsg);
16
17     canMsg->msgID = mcpMsg.msgID;
18     canMsg->msgIDMask = mcpMsg.msgIDMask;
19     canMsg->flags = mcpMsg.flags;
20     canMsg->msgLen = mcpMsg.msgLen;
21     canMsg->msgData = mcpMsg.msgData;
22 }
```

Código 4.20 – Código com o comando de leitura do móculo CAN

```
1  //*************************************************************************
2  // Send CAN data using MCP2515 module
3  //*************************************************************************
4  void CanWrite(CANMsg_t *canMsg)
5  {
6      MCP2515Msg_t mcpMsg;
7
8      mcpMsg.msgID = canMsg->msgID;
9      mcpMsg.msgIDMask = canMsg->msgIDMask;
10     mcpMsg.flags = canMsg->flags;
11     mcpMsg.msgLen = canMsg->msgLen;
12     mcpMsg.msgData = canMsg->msgData;
13
14     MCP2515Write(&mcpMsg);
15 }
```

Código 4.21 – Código com o comando de escrita do módulo CAN

Fazendo uso dos códigos implementados no módulo CAN, tem-se os módulo do GPS e módulo do sensor de posicionamento angular, onde no caso do módulo do sensor de posicionamento angular é implementado o comando de leitura, onde utiliza o comando de leitura do módulo CAN, realizando também a separação dos dados recebidos de acordo com a estrutura de dados do sensor de posicionamento angular.

Para o módulo GPS a implementação do código é semelhante ao módulo do sensor de posicionamento angular, onde utiliza o comando de leitura do módulo CAN e realiza a separação dos dados de acordo com a estrutura dos dados do módulo GPS. O código 4.22 apresenta a estrutura do comando de leitura do módulo GPS.

```
//*****************************************************************************
// GPS read function
//*****************************************************************************
void GPSRead(gps_t *gps)
{
    CANMsg_t can;

    CanRead(&can);

    if(can.msgID == GPS_ID && can.msgLen == GPS_LEN)
    {
        gps->latitude = ((can.msgData[0] & 0xFF000000) >> 24) +
                        ((can.msgData[1] & 0x00FF0000) >> 16) +
                        ((can.msgData[2] & 0x0000FF00) >> 8) +
                        ((can.msgData[3] & 0x000000FF) >> 0);

        gps->longitude = ((can.msgData[4] & 0xFF000000) >> 24) +
                         ((can.msgData[5] & 0x00FF0000) >> 16) +
                         ((can.msgData[6] & 0x0000FF00) >> 8) +
                         ((can.msgData[7] & 0x000000FF) >> 0);
    }
}
```

Código 4.22 – Código com o comando de leitura do móculo GPS

Para o código do módulo PID, foram inicialmente calculado os parâmetro P,I e D. Sendo atribuído os parâmetros calculados ao código do PID. O código construído para o módulo PID pode ser observado no código 4.23.

```
//*****************************************************************************
// PID calculation function
//*****************************************************************************
void PIDCalc(pid_t *pid, int16_t setpoint, int16_t actual_position)
{
    if((setpoint != actual_position)
        && (actual_position < 721)
        && (actual_position > -721))
    {
        // Calculate error
        pid->error = setpoint - actual_position;

        // Calculate integral
        pid->integral = pid->integral + pid->error;

        // Calculate derivative
        pid->derivative = (pid->error - pid->last_error);

        // Calculate output
        pid->output = Kp*pid->error + Ki*pid->integral + Kd*pid->derivative;

        // Check output limits
```

```
23          if(pid->output > MAX_ANGLE) {
24              pid->output = MAX_ANGLE;
25          } else if(pid->output < MIN_ANGLE) {
26              pid->output = MIN_ANGLE;
27          }
28
29          // Clear variables
30          pid->last_error = pid->error;
31          pid->error = 0;
32      }
33      else
34      {
35          pid->error = 0;
36          pid->last_error = 0;
37          pid->derivative = 0;
38          pid->integral = 0;
39          pid->output = 0;
40          pid->motorVoltage = 0;
41          pid->motorDirection = 0;
42      }
43  }
```

Código 4.23 – Código com o comando de calculo de PID do móculo PID

A partir do código 4.23, pode-se observar que inicialmente é calculado o erro de ângulo presente no sistema, onde consiste na diferença do ângulo desejado com o ângulo atual da direção do veículo. Para então utilizar esse erro para os cálculos dos componentes integral e derivativo. Por fim, é utilizado as componentes integrais e derivativas, em conjunto com o parâmetros P, I e D, para calcular o valor de saída do controlador PID que deve ser aplicado ao ângulo da direção do veículo.

4.4 Desenvolvimento de *Software*

Para a definição da rota a ser percorrida pelo veículo, foi desenvolvido no projeto um aplicativo *Android*, onde tem a função da verificação e controle por parte da rota que o veículo deve percorrer, onde deve transmitir os valores de ângulo calculados para o controle PID, de forma que o veículo se mantenha em sua rota previamente estabelecida.

Para que esta funcionalidade seja possível, foi utilizado a comunicação *bluetooth* entre o aplicativo e o microcontrolador. Sendo recebido por parte do aplicativo o posicionamento GPS do veículo e retornado ao microcontrolador o ângulo em que deve ser estabelecido na direção do veículo. A comunicação entre o *bluetooth* e o microcontrolador segue o protocolo discutido anteriormente.

Pensando na estrutura do código do aplicativo, foi definido a estrutura do código de acordo com o diagrama apresentado na figura 54.

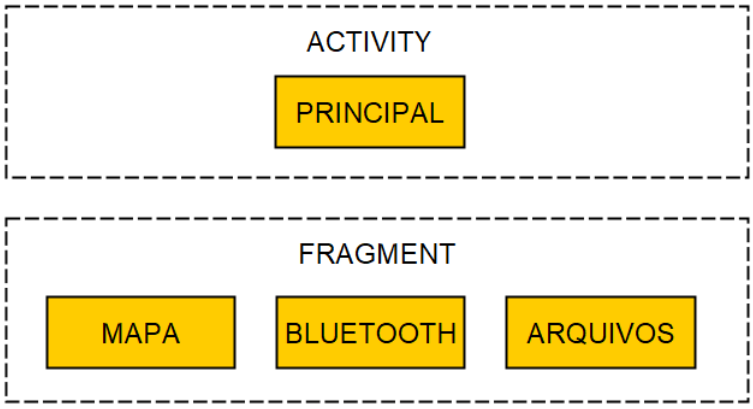

Figura 54 – Estrutura esquematizada do código do aplicativo - Fonte: Autoria própria.

A partir do diagrama da figura 54 é possível observar quatro módulos de operação, onde existe a presença de apenas uma *activity* e o restante dos módulos foram implementados como *fragments* da *activity* principal. Sendo implementado dessa maneira devido a maior facilidade de comunicação entre os módulos e a possibilidade de manter os *fragments* operando em *background*.

Para melhor entendimento e visualização do aplicativo, será discutido cada módulo separadamente. Assim, considerando a operação inicial do aplicativo, tem-se a visualização do módulo do MAPA, tem-se a figura 55 que apresenta a tela do aplicativo com o módulo do MAPA.

Figura 55 – Tela de apresentação do MAPA no aplicativo - Fonte: Autoria própria.

A partir da figura 55, pode-se observar que o módulo do MAPA consiste na apresentação do posicionamento GPS recebido, onde sempre está sincronizado com a posição atual do veículo. Mostrando a movimentação do veículo de acordo que o veículo segue a rota previamente gravada.

Para a transição entre funcionalidades do aplicativo, foi implementado o menu lateral onde disponibiliza o acesso aos outros módulos do aplicativo. A figura 56 apresenta a estrutura do menu construído.

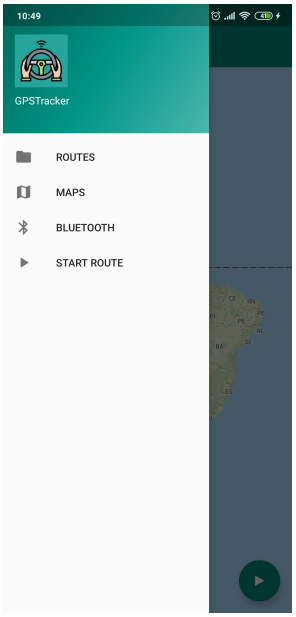

Figura 56 – Interface do menu do aplicativo - Fonte: Autoria própria.

Para o menu de rotas, foi implementado a lista de arquivos de rotas gravadas pelo sistema, onde é possível selecionar a rota previamente gravada desejada para o funcionamento do sistema. A figura 57 apresenta a interface das rotas.

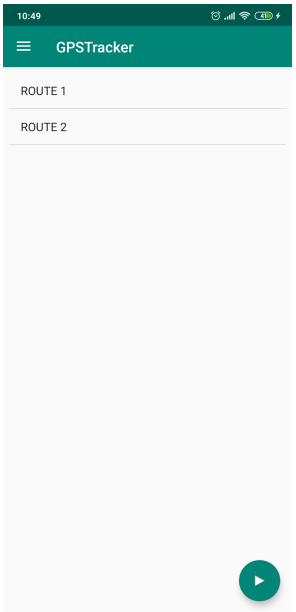

Figura 57 – Interface para seleção das rotas - Fonte: Autoria própria.

Para o menu do *bluetooth*, foi implementado todas as funcionalidades de conexão *bluetooth*, sendo o botão de inicialização do periférico, após as funcionalidades de tornar o

dispositivo visível para outros aparelhos, assim como a funcionalidade de descobrir os outros dispositivos *bluetooth* no alcance para realizar a conexão.

Tendo escolhido o dispositivo para a conexão, basta clicar para que a rotina de pareamento seja executada e utilizar o botão de inicio da conexão para efetivamente conectar ao dispositivo e iniciar a transmissão dos dados. A figura 58 apresenta a interface *bluetooth* desenvolvida.

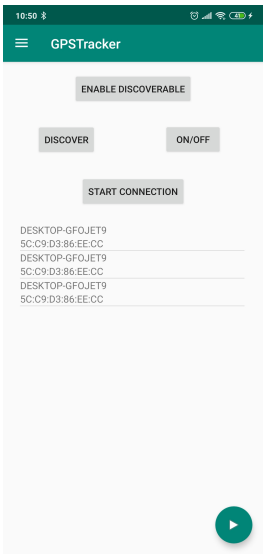

Figura 58 – Interface do bluetooth do aplicativo - Fonte: Autoria própria.

Para inicializar o sistema de gravação de rotas, basta clicar ao botão flutuante que aparece em todas as telas do aplicativo, para facilitar a ativação do sistema de gravação de rotas independente da tela em que esteja em operação. Quando acionado o botão de gravação de rotas, é interface do MAPA vai abrir automaticamente e rotina de gravação de rotas vai armazenar as posições GPS percorridas pelo veículo.

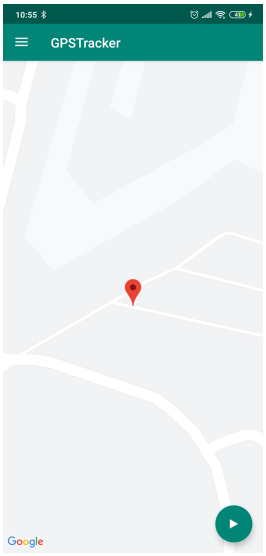

Figura 59 – Interface do módulo MAPA em execução - Fonte: Autoria própria.

Com a figura 55, é possível observar que o posicionamento GPS indicado se encontra muito distante, considerando a dimensão de um veículo. No entanto, essa é apenas a tela de inicialização do aplicativo, pois quando iniciado a operação de gravação ou execução de rotas, o módulo do MAPA aproxima a visualização para melhorar a identificação da movimentação do veículo dentro da rota gravada. A figura 59 apresenta o módulo do MAPA em execução.

Portanto essas são as principais funcionalidades do aplicativo, sendo que o código fonte de cada funcionalidade e de cada módulo discutido pode ser observado nos anexos, mais especificamente no anexo B.

5 RESULTADOS

Com relação ao levantamento da função de transferência do motor, a dificuldade encontrada foi em encontrar um equipamento com precisão o suficiente para conseguir capturar o transitório do motor *brushless* utilizado no projeto, pois o transitório ocorre rapidamente com uma baixa intensidade, como pode ser observado na figura 42.

Para o projeto do controlador PID, a dificuldade encontrada no desenvolvimento foi e identificar o modelo do motor e aplicar as corretas relações de cálculo para o levantamento dos parâmetros P, I e D. No entanto, identificou-se um modelo de segunda ordem para o motor, montou-se a equação 18 e chegou-se aos parâmetros do controlador PID.

No caso da sintonia do controlador PID, devido ao auxilio do software matemático MATLAB, a dificuldade foi em utilizar de forma correta a ferramenta para atingir o resultado esperado. Assim, chegou-se a curva da figura 46 e para melhorar a visualização do modelo, tem-se a curva da figura 47 que demonstra o teste com a resposta ao degrau do motor com e sem a atuação do controlador PID.

Após realizar a avaliação da curva do motor com o controle PID aplicado, encontrou-se o resultado apresentado na figura 60.

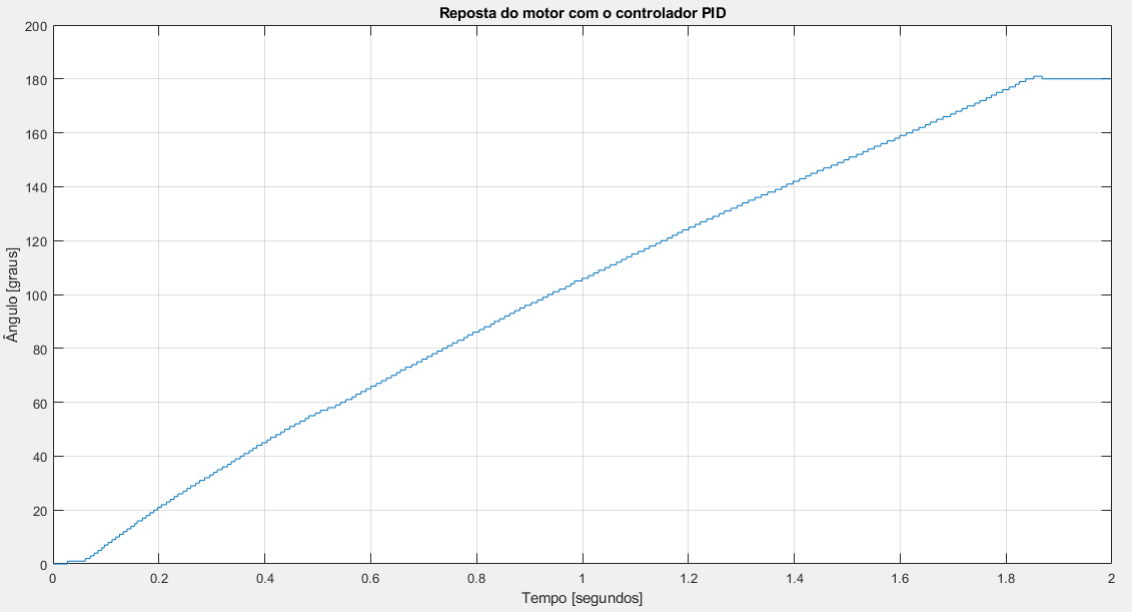

Figura 60 – Resposta do motor ao degrau entrada - Fonte: Autoria própria.

A partir da curva da figura 60, comparando com a curva do motor sem a aplicação do controle, apresentada na figura 40, pode-se observar que a resposta do motor controlado apresentou lentidão para atingir o valor final do degrau. No entanto, tornou-se um acionamento mais linearizado, o que facilitou o posicionamento do motor na posição especificada, ou seja, controlando o transitório do motor.

Outro ponto importante apresentada na figura 60, é que a curva foi gerada com o uso do sensor de posicionado angular utilizado no projeto. No caso da curva apresentada na figura 40, foi adquirido com o uso de um *data logger*. Devido a esses fatos, a curva da figura 60 apresenta uma menor resolução no sinal amostrado, porém não sendo prejudicial para observar o formato da curva e verificar o funcionamento do controlador.

Com relação a montagem do *hardware* do projeto, as dificuldades encontradas foram com relação a encontrar o melhor posicionamentos das placas eletrônicas utilizadas no projeto, de forma que não ocorresse problemas de interferência de operação de uma placa com a outra. No entanto, o resultado final da montagem se mostrou satisfatório para o projeto, tendo em vista o correto funcionamento de todos os módulos que foram montados na placa eletrônica.

Outro ponto de dificuldade no projeto do hardware, foi na correta definição da conectorização dos periféricos, verificando os valores de corrente necessário para cada módulo e colocando os conectores adequados. De forma a manter a operação correta, porém com fácil manipulação da placa eletrônica, tornando prático o seu uso.

Com relação aos testes do *firmaware* do projeto, a dificuldade encontrada durante a elaboração do *firmware* do projeto, se deu em definir a melhor estrutura organizacional do código, para facilitar o entendimento do código sem prejudicar o desempenho do *firmware*. Porém a estrutura definida, podendo ser observada na figura 53, se apresentou de acordo com a especificação do projeto, pois atingiu a forma de operação esperada para o correto funcionamento do sistema.

Com relação aos testes do aplicativo, a dificuldade encontrada no desenvolvimento do aplicativo, foi com relação a gestão das interfaces, onde inicialmente os módulo foram programados com *activits*, o que apresentou problemas para manter a conexão *bluetooth* após estabelecida, pois as *activits* são destruídas quando não estão ativas.

No entanto, a modificação implementação para o uso de *fragments* ao invés de *activits* se apresentou adequada para o aplicativo, devido a funcionalidade de manter o *fragment* operando em *back ground*, o que para a conexão *bluetooth* é adequado, pois caso seja comutado a apresentação da tela do aplicativo, a conexão se mantém estável. Podendo navegar por todas as interfaces do aplicativo sem problemas com a conexão.

6 CONCLUSÃO E TRABALHOS FUTUROS

Com relação ao levantamento da função de transferência do motor, apresentada na equação 18, pode-se observar que o modelo obtido se aproxima da curva real obtida pelo motor, comprovando o método para o levantamento pela modelagem de um sistema de segunda ordem.

Em relação aos parâmetros PID encontrados, pode-se observar na curva apresentada na figura 47, que os parâmetros projetados foram suficientes para melhorar o transitório do motor *brushless* utilizado no projeto.

Com relação aos testes com o *firmware* do projeto, o código se mostrou estável e conseguiu-se realizar os acessos aos periféricos, realizando também as leituras e os acionamentos necessários para o funcionamento do sistema.

Com relação aos resultados obtidos no teste do aplicativo, foram realizados os testes de comunicação *bluetooth*, no qual se apresentou estável para a utilização no projeto. Conseguindo receber os dados GPS, calcular e transmitir os valores de ângulo para o entrada do controlador PID programado no microcontrolador.

Tendo em vista o funcionamento geral do projeto, foi possível observar que os resultados alcançados foram satisfatórios, porém o sistema não se encontra pronto para a utilização como produto, pode ser considerado uma prova de conceito na utilização de motores do tipo *brushless* para controle de direção veicular.

Para se atingir aplicação como produto é necessário aumentar o nível de confiabilidade de todas as partes do projeto, ou seja, inserindo sistemas de proteção de *hardware* e *software*, o que no caso desse projeto se tem um nível de proteção baixo, devido ao fato de ser um prótipo inicial do sistema.

Outro ponto a ser melhorado no projeto, é com relação ao aplicativo desenvolvido no projeto, onde pode ser investido no desenvolvimento de um *design* próprio para as interfaces, tendo em vista que o aplicativo foi desenvolvido com as bibliotecas de *design* padrão do sistema Android.

Referências

AGUIRE, L. A. **Introdução à identificação de sistemas: técnicas lineares e não-lineares aplicadas a sistemas reais**. [S.l.]: UFMG, 2007. Citado 3 vezes nas páginas 18, 19 e 20.

ANTUNES, F. **Uma Nova Abordagem para Representações e Identificações de Classes de Sistemas Dinâmicos Não Lineares. Tese (Mestrado em Engenharia Elétrica)**. [S.l.], 2017. Disponível em: <http://saturno.unifei.edu.br/bim/0031985.pdf>. Acesso em: 08/10/2019. Citado 2 vezes nas páginas e 19.

ARAÚJO, F. M. U. de. **Sistemas de Controle**. Natal-RN, 2007. Disponível em: <http://www.dca.ufrn.br/~meneghet/FTP/Controle/scv20071.pdf>. Acesso em: 01/09/2018. Citado 2 vezes nas páginas e 21.

BAZANELLA, A. S.; JR., J. M. G. da S. **Ajuste de controladores PID**. [S.l.], 2000. Disponível em: <http://www.ece.ufrgs.br/~jmgomes/pid/Apostila/apostila/apostila.html>. Acesso em: 03/09/2018. Citado na página 22.

BOSCH, R. **Manual de Tecnologia Automotiva. Tradução de Helga Madjderey, Gunter W. Prokesch, Euryale de Jesus Zerbini, Sueli Pfeferman. 25 ed. São Paulo**. [S.l.]: Edgar Blücher, 2005. Citado 4 vezes nas páginas , 30, 31 e 32.

BROSLER, R. O. **Controlador PID utilizando microcontrolador PIC**. [S.l.], 2014. Disponível em: <https://repositorio.unesp.br/bitstream/handle/11449/123013/000806060.pdf?sequence=1>. Acesso em: 07/09/2018. Citado 3 vezes nas páginas , 25 e 26.

CARVALHO, S. **Android Studio: vantagens e desvantagens com relação ao Eclipse**. [S.l.], 2013. Disponível em: <https://imasters.com.br/android/android-studio-vantagens-e-desvantagens-com-relacao-ao-eclipse>. Acesso em: 10/09/2018. Citado na página 45.

CODEPATH. **Developing Custom Themes**. [S.l.], 2016. Disponível em: <https://guides.codepath.com/android/developing-custom-themes>. Acesso em: 10/09/2018. Citado 2 vezes nas páginas e 46.

CUSTOMERSYS. **Conhecendo o Android Studio**. [S.l.], 2015. Disponível em: <https://customersys.blogspot.com/2015/10/conhecendo-o-android-studio.html>. Acesso em: 10/09/2018. Citado 2 vezes nas páginas e 45.

DALTO, F. L. **Como criar uma splashscreen no Android Studio**. [S.l.], 2017. Disponível em: <https://medium.com/@ldfranco/criar-uma-splash-screen-no-android-cc2bf495ed7a>. Acesso em: 10/09/2018. Citado 2 vezes nas páginas e 44.

DEVELOPERS. **Conheça o Android Studio**. [S.l.], 2018. Disponível em: <https://developer.android.com/studio/intro/?hl=pt-br>. Acesso em: 10/09/2018. Citado na página 44.

ENGINEERING, L. **Brushless DC Motor, How it works**. [S.l.], 2014. Disponível em: <https://www.youtube.com/watch?v=bCEiOnuODac>. Acesso em: 07/09/2018. Citado 3 vezes nas páginas , 29 e 30.

INSTRUMENTS, T. **Universal Asynchronous Receiver/Transmitter (UART)**. [S.l.], 2010. Disponível em: <http://www.ti.com/lit/ug/sprugp1/sprugp1.pdf>. Acesso em: 01/10/2019. Citado 2 vezes nas páginas e 35.

INSTRUMENTS, T. **Tiva^{TM} C Series TM4C123G LaunchPad Evaluation Board - User's Guide**. [S.l.], 2013. Disponível em: <http://www.ti.com/lit/ug/spmu296/spmu296.pdf>. Acesso em: 01/10/2019. Citado 2 vezes nas páginas 40 e 41.

KEAY, W. **Land Navigation: Routefinding with Map Compass**. [S.l.]: UK: Clifford Press Ltd, 1995. Citado na página 47.

LENZ, A. L. **Como se Constituem e Operam os Motores BLDC (Motores CC Sem Escovas)**. [S.l.], 2016. Disponível em: <http://www.ebah.com.br/content/ABAAAgxbgAD/como-se-constituem-operam-os-motores-bldc-motores-cc-sem-escovas>. Acesso em: 08/09/2018. Citado 4 vezes nas páginas , 28, 29 e 37.

LJUNG, L. **System identification, Theory for theuser**. [S.l.]: New Jersey: Prentice Hall, 1999. Citado 3 vezes nas páginas 18, 27 e 28.

MACIEL, M. **Controle PID com aproximação Digital para utilização no PIC**. [S.l.], 2012. Disponível em: <http://microcontrolado.com/controle-pid-no-pic/>. Acesso em: 07/09/2018. Citado 3 vezes nas páginas , 25 e 27.

MAGRO, J. A. **Colheita Mecanizada da Cana Crua**. [S.l.], 2019. Disponível em: <http://www.colheitamecanizadacanacrua.com.br>. Acesso em: 01/10/2019. Citado 3 vezes nas páginas , 15 e 16.

MATIAS, J. **Teoria de Controle PID**. [S.l.], 2002. Disponível em: <http://coral.ufsm.br/beltrame/arquivos/disciplinas/medio_automacao_industrial/Artigo_Teoria_controle_PID.pdf>. Acesso em: 01/09/2018. Citado 2 vezes nas páginas 20 e 21.

MCDERMOTT-WELLS, P. **Bluetooth Overview**. [S.l.]: IEEE, 2004. Citado 3 vezes nas páginas , 35 e 36.

MICROCHIP. **MCP2515 - Datasheet**. [S.l.], 2002. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf>. Citado 2 vezes nas páginas 41 e 42.

MICROCHIP. **MCP4725 - Datasheet**. [S.l.], 2007. Disponível em: <https://www.sparkfun.com/datasheets/BreakoutBoards/MCP4725.pdf>. Acesso em: 01/10/2019. Citado na página 41.

MICROCHIP. **HC 05 - Datasheet**. [S.l.], 2010. Disponível em: <http://www.electronicaestudio.com/docs/istd016A.pdf>. Citado na página 42.

NOVUS, P. eletrônicos. **Controle PID básico**. [S.l.], 2003. Disponível em: <http://www.novus.com.br/downloads/Arquivos/artigopidbasico.pdf>. Acesso em: 03/09/2018. Citado 5 vezes nas páginas , 22, 23, 24 e 27.

NXP. **I2C-bus specification and user manual**. [S.l.], 2014. Disponível em: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>. Acesso em: 01/10/2019. Citado 3 vezes nas páginas , 34 e 35.

OGATA, K. **Modern Control Engineering**. 5a edição. ed. [S.l.]: AEEIZH, 2011. Citado 2 vezes nas páginas 21 e 25.

OLIVEIRA, A. L. de L. **Fundamentos de Controle de Processo**. Vitória-ES, 1999. Disponível em: <http://www.netsoft.inf.br/aulas/7_EAC_Sistemas_Realimentados/Ebook_CPM_Instrumentacao.pdf>. Acesso em: 02/09/2018. Citado 3 vezes nas páginas 20, 23 e 25.

PEREIRA, F. **Microcontroladores PIC: programação em C**. [S.l.], 2003. Acesso em: 07/09/2018. Citado na página 26.

PISSARDINI, R. S. . **Veículos Autônomos: Conceitos, Histórico e Estado-da-Arte». Anais do XXVII Congresso de Pesquisa e Ensino em Transportes**. [S.l.]: Anais do XXVII Congresso de Pesquisa e Ensino em Transportes, 2013. Citado 2 vezes nas páginas e 14.

RIVERA D. E.; MORARI, M. S. S. **Internal Model Control - PID Controller Design**. [S.l.]: Ind. Eng. Chem. Process., 1986. Citado na página 28.

RODRIGUES, G. G. **Identificação de Sistemas Dinâmicos Não-Lineares Utilizando Modelos NARMAX Polinomiais Aplicação a Sistemas Reais. Dissertação (Mestrado em Engenharia Elétrica)**. [S.l.], 2000. Disponível em: <http://www.bibliotecadigital.ufmg.br/dspace/bitstream/handle/1843/BUDB8D3GC3/giovani_guimaraes_rodrigues.pdf?sequence=1>. Acesso em: 01/10/2019. Citado na página 19.

SILVEIRA, C. B. **O Controle PID de Forma Simples e Descomplicada**. [S.l.], 2016. Disponível em: <https://www.citisystems.com.br/controle-pid/>. Acesso em: 03/09/2018. Citado 3 vezes nas páginas , 22 e 23.

UPENDER, B. P. **Communication Protocols for Embedded Systems**. [S.l.]: Embedded Systems Programming, 1994. Citado 3 vezes nas páginas , 33 e 34.

VOLVO. **VERA**. [S.l.], 2019. Disponível em: <https://www.volvotrucks.com/en-en/about-us/automation/vera.html>. Acesso em: 01/10/2019. Citado 2 vezes nas páginas e 15.

Anexos

ANEXO A – Firmware

A.1 Código Bluetooth

```
/*****************************************************************************//**
* @file bluetooth.c
* @date 8 de ago de 2019
* @author Eduardo Henrique Fideles Ribeiro
* @author Wiviane Caroline Maneira
* @brief Bluetooth source code
*******************************************************************************/
//*****************************************************************************
// Include
//*****************************************************************************
#include "SteeringControl/bluetooth.h"

//*****************************************************************************
// Bluetooth configuration
//*****************************************************************************
void BluetoothConfig(void)
{
    UART5Configuration();
}

//*****************************************************************************
// Function to check Bluetooth available data
//*****************************************************************************
uint8_t BluetoothAvailableData(void)
{
    uint8_t ret;
    ret = UART5AvailableData();
    return ret;
}

//*****************************************************************************
// Function to send bluetooth data
//*****************************************************************************
void BluetoothSend(Bluetooth_t BT)
{
    uint8_t data[9], i;

    switch(BT.type)
    {
        case 'G':
            data[0] = BT.type;
            data[1] = ((BT.latitude & 0xFF000000) >> 24);
            data[2] = ((BT.latitude & 0x00FF0000) >> 16);
            data[3] = ((BT.latitude & 0x0000FF00) >> 8);
            data[4] = ((BT.latitude & 0x000000FF) >> 0);
            data[5] = ((BT.longitude & 0xFF000000) >> 24);
            data[6] = ((BT.longitude & 0x00FF0000) >> 16);
            data[7] = ((BT.longitude & 0x0000FF00) >> 8);
            data[8] = ((BT.longitude & 0x000000FF) >> 0);
            for(i=0;i<9;i++){
```

```
51                  UART5Write(data[i]);
52              }
53              break;
54          case 'A':
55              data[0] = BT.type;
56              data[1] = ((BT.angle & 0xFF00) >> 8);
57              data[2] = ((BT.angle & 0x00FF) >> 0);
58              for(i=0;i<3;i++){
59                  UART5Write(data[i]);
60              }
61              break;
62      }
63  }
64
65  //*********************************************************************
66  // Function to receive bluetooth data
67  //*********************************************************************
68  void BluetoothRecv(Bluetooth_t *BT)
69  {
70      uint8_t data[10],i;
71
72      if(UART5AvailableData())
73      {
74          UART5Read(&data[0]);
75          switch(data[0])
76          {
77              case 'G':
78                  for(i=0;i<8;i++){
79                      UART5Read(&data[i+1]);
80                  }
81                  BT->latitude = (data[1]<<24) + (data[2]<<16)
82                               + (data[3]<<8) + data[4];
83                  BT->longitude = (data[5]<<24) + (data[6]<<16)
84                                + (data[7]<<8) + data[8];
85                  break;
86              case 'A':
87                  for(i=0;i<2;i++){
88                      UART5Read(&data[i+1]);
89                  }
90                  BT->angle = (data[1]<<8) + data[2];
91                  break;
92          }
93      }
94  }
```

A.2 Código CAN

```
1  /*********************************************************************//**
2  * @file can.c
3  * @date 15 de set de 2019
4  * @author Eduardo Henrique Fideles Ribeiro
5  * @author Wiviane Caroline Maneira
6  * @brief CAN source file
7  **********************************************************************/
8  //*********************************************************************
9  // Includes
10 //*********************************************************************
```

```
11 #include "SteeringControl/can.h"
12
13 //*****************************************************************************
14 // Configuration of MCP2515 CAN module
15 //*****************************************************************************
16 void CanConfiguration(void)
17 {
18     MCP2515SpiConfiguration();
19
20     SysCtlDelay(800);
21
22     MCP2515Reset();
23
24     MCP2515SetBits(MCP2515_CANINTF, 0x40, 0x00);
25     MCP2515SetBits(MCP2515_CANCTRL, 0xE0, MCP2515_MODE_CFG);
26
27     SysCtlDelay(800);
28
29     MCP2515WriteRegister(MCP2515_CNF1, MCP2515_8MHz_250kBPS_CFG1);
30     MCP2515WriteRegister(MCP2515_CNF2, MCP2515_8MHz_250kBPS_CFG2);
31     MCP2515WriteRegister(MCP2515_CNF3, MCP2515_8MHz_250kBPS_CFG3);
32
33     SysCtlDelay(800);
34
35     uint8_t i = 0;
36     for(i=0;i<14;i++){
37         MCP2515WriteRegister(MCP2515_TXB0CTRL + i, 0x00);
38         MCP2515WriteRegister(MCP2515_TXB1CTRL + i, 0x00);
39         MCP2515WriteRegister(MCP2515_TXB2CTRL + i, 0x00);
40     }
41
42     MCP2515WriteRegister(MCP2515_RXB0CTRL, 0x00);
43     MCP2515WriteRegister(MCP2515_RXB1CTRL, 0x00);
44
45     MCP2515WriteRegister(MCP2515_CANINTE, 0x01 | 0x02);
46
47     MCP2515SetBits(MCP2515_RXB0CTRL, (0x60 | (1<<2)), (0x60 | (1<<2)));
48     MCP2515SetBits(MCP2515_RXB1CTRL, 0x60, 0x60);
49
50     MCP2515SetBits(MCP2515_CANINTF, 0x40, 0x00);
51     MCP2515SetBits(MCP2515_CANCTRL, 0xE0, MCP2515_MODE_NORMAL);
52
53     SysCtlDelay(800);
54 }
55
56 //*****************************************************************************
57 // Send CAN data using MCP2515 module
58 //*****************************************************************************
59 void CanWrite(CANMsg_t *canMsg)
60 {
61     MCP2515Msg_t mcpMsg;
62
63     mcpMsg.msgID = canMsg->msgID;
64     mcpMsg.msgIDMask = canMsg->msgIDMask;
65     mcpMsg.flags = canMsg->flags;
66     mcpMsg.msgLen = canMsg->msgLen;
67     mcpMsg.msgData = canMsg->msgData;
```

```
68
69      MCP2515Write(&mcpMsg);
70  }
71
72  //***************************************************************************
73  // Receive CAN data from MCP2515 module
74  //***************************************************************************
75  void CanRead(CANMsg_t *canMsg)
76  {
77      MCP2515Msg_t mcpMsg;
78      uint8_t buf[8];
79
80      mcpMsg.msgID = 0;
81      mcpMsg.msgIDMask = 0;
82      mcpMsg.flags = 0;
83      mcpMsg.msgLen = 0;
84      mcpMsg.msgData = buf;
85
86      MCP2515Read(&mcpMsg);
87
88      canMsg->msgID = mcpMsg.msgID;
89      canMsg->msgIDMask = mcpMsg.msgIDMask;
90      canMsg->flags = mcpMsg.flags;
91      canMsg->msgLen = mcpMsg.msgLen;
92      canMsg->msgData = mcpMsg.msgData;
93  }
```

A.3 Código DAC

```
1   /*****************************************************************************//**
2   * @file dac.c
3   * @date 20 de set de 2019
4   * @author Eduardo Henrique Fideles Ribeiro
5   * @author Wiviane Caroline Maneira
6   * @brief DAC source file
7   *****************************************************************************/
8   //***************************************************************************
9   // Includes
10  //***************************************************************************
11  #include "SteeringControl/dac.h"
12
13  //***************************************************************************
14  // Function to configure DAC
15  //***************************************************************************
16  void DACConfiguration(void)
17  {
18      MCP4725Configuration();
19  }
20
21  //***************************************************************************
22  // Function to send DAC data
23  //***************************************************************************
24  void DACWrite(uint16_t dacValue)
25  {
26      MCP4725Write(dacValue);
27  }
```

A.4 Código GPIO

```
/*****************************************************************************//**
* @file gpio.c
* @date 8 de ago de 2019
* @author Eduardo Henrique Fideles Ribeiro
* @author Wiviane Caroline Maneira
* @brief GPIO source code
*****************************************************************************/
//*****************************************************************************
// Includes
//*****************************************************************************
#include "SteeringControl/gpio.h"

//*****************************************************************************
// GPIO PC7 configuration to set the direction of motor rotation
//*****************************************************************************
void GPIOConfiguration(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x0);
}

//*****************************************************************************
// GPIO function to set motor right direction
//*****************************************************************************
void GPIOSetMotorRight(void)
{
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
}

//*****************************************************************************
// GPIO function to set motor left direction
//*****************************************************************************
void GPIOSetMotorLeft(void)
{
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
}

//*****************************************************************************
// GPIO PF4 configuration to measure the velocity of motor
//*****************************************************************************
void GPIOIntConfiguration(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOIntDisable(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_RISING_EDGE);
    GPIOIntRegister(GPIO_PORTF_BASE, GPIOInterruptHandler);
    GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4);
}
```

A.5 Código GPS

```
/*****************************************************************************//**
* @file gps.c
* @date 21 de nov de 2019
* @author Eduardo Henrique Fideles Ribeiro
* @author Wiviane Caroline Maneira
* @brief  GPS source file
*******************************************************************************/
//*****************************************************************************
// Include
//*****************************************************************************
#include "SteeringControl/gps.h"

//*****************************************************************************
// GPS read function
//*****************************************************************************
void GPSRead(gps_t *gps)
{
    CANMsg_t can;

    CanRead(&can);

    if(can.msgID == GPS_ID && can.msgLen == GPS_LEN)
    {
        gps->latitude = ((can.msgData[0] & 0xFF000000) >> 24) +
                        ((can.msgData[1] & 0x00FF0000) >> 16) +
                        ((can.msgData[2] & 0x0000FF00) >> 8) +
                        ((can.msgData[3] & 0x000000FF) >> 0);

        gps->longitude = ((can.msgData[4] & 0xFF000000) >> 24) +
                         ((can.msgData[5] & 0x00FF0000) >> 16) +
                         ((can.msgData[6] & 0x0000FF00) >> 8) +
                         ((can.msgData[7] & 0x000000FF) >> 0);
    }
}
```

A.6 Código I2C

```
/*****************************************************************************//**
* @file i2c.c
* @date 8 de ago de 2019
* @author Eduardo Henrique Fideles Ribeiro
* @author Wiviane Caroline Maneira
* @brief I2C source code
*******************************************************************************/
//*****************************************************************************
// Include
//*****************************************************************************
#include "SteeringControl/i2c.h"

//*****************************************************************************
// Configuration of I2C0 pins to communicate with external modules
//*****************************************************************************
void I2CConfiguration(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
```

```
21
22      GPIOPinConfigure(GPIO_PB2_I2C0SCL);
23      GPIOPinConfigure(GPIO_PB3_I2C0SDA);
24
25      GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
26      GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
27
28      I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);
29      HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
30  }
31
32  //*****************************************************************************
33  // Function to Send an I2C command to the specified slave
34  //*****************************************************************************
35  void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...)
36  {
37      I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
38
39      va_list vargs;
40
41      va_start(vargs, num_of_args);
42      I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
43
44      if(num_of_args == 1)
45      {
46          I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
47          while(I2CMasterBusy(I2C0_BASE));
48          va_end(vargs);
49      }
50      else
51      {
52          I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
53          while(I2CMasterBusy(I2C0_BASE));
54
55          uint8_t i=0;
56          for(i = 1; i < (num_of_args - 1); i++)
57          {
58              I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
59              I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);
60              while(I2CMasterBusy(I2C0_BASE));
61          }
62
63          I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
64          I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
65          while(I2CMasterBusy(I2C0_BASE));
66          va_end(vargs);
67      }
68  }
69
70  //*****************************************************************************
71  // Function to read specified register on slave device
72  //*****************************************************************************
73  uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
74  {
75      I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
76      I2CMasterDataPut(I2C0_BASE, reg);
77      I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
```

```
78      while(I2CMasterBusy(I2C0_BASE));
79      I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);
80      I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
81      while(I2CMasterBusy(I2C0_BASE));
82      return I2CMasterDataGet(I2C0_BASE);
83  }
```

A.7 Código MCP2515

```
1   /*****************************************************************************//**
2   * @file mcp2515.c
3   * @date 31 de ago de 2019
4   * @author Eduardo Henrique Fideles Ribeiro
5   * @author Wiviane Caroline Maneira
6   * @brief MCP2515 source file
7   *****************************************************************************/
8   //*****************************************************************************
9   // Includes
10  //*****************************************************************************
11  #include "SteeringControl/mcp2515.h"
12
13  //*****************************************************************************
14  // MCP2515 Spi Configuration
15  //*****************************************************************************
16  void MCP2515SpiConfiguration(void)
17  {
18      SPIMasterConfiguration();
19  }
20
21  //*****************************************************************************
22  // MCP2515 transfer data
23  //*****************************************************************************
24  uint8_t MCP2515TransferData(uint8_t byte)
25  {
26      uint32_t ret;
27      SPIByteWrite(byte);
28      while(true == SSIBusy(SSI2_BASE));
29      SPIByteRead(&ret);
30      return ret;
31  }
32
33  //*****************************************************************************
34  // MCP2515 write register
35  //*****************************************************************************
36  void MCP2515WriteRegister(uint8_t address, uint8_t data)
37  {
38      SPICsPin(MCP2515_CS_ACT);
39      MCP2515TransferData(MCP2515_WRITE_CMD);
40      MCP2515TransferData(address);
41      MCP2515TransferData(data);
42      SPICsPin(MCP2515_CS_DACT);
43  }
44
45  //*****************************************************************************
46  // MCP2515 read register
47  //*****************************************************************************
48  void MCP2515ReadRegister(uint8_t address, uint8_t *data)
```

```
49 {
50     SPICsPin(MCP2515_CS_ACT);
51     MCP2515TransferData(MCP2515_READ_CMD);
52     MCP2515TransferData(address);
53     *data = MCP2515TransferData(MCP2515_DUMMY_DATA);
54     SPICsPin(MCP2515_CS_DACT);
55 }
56
57 //*****************************************************************************
58 // MCP2515 clear register bits
59 //*****************************************************************************
60 void MCP2515ClearBits(uint8_t address, uint8_t bits)
61 {
62     SPICsPin(MCP2515_CS_ACT);
63     MCP2515TransferData(MCP2515_BIT_MODIFY_CMD);
64     MCP2515TransferData(address);
65     MCP2515TransferData(bits);
66     MCP2515TransferData(MCP2515_BIT_CLEAR);
67     SPICsPin(MCP2515_CS_DACT);
68 }
69
70 //*****************************************************************************
71 // MCP2515 set register bits
72 //*****************************************************************************
73 void MCP2515SetBits(uint8_t address, uint8_t bits, uint8_t data)
74 {
75     SPICsPin(MCP2515_CS_ACT);
76     MCP2515TransferData(MCP2515_BIT_MODIFY_CMD);
77     MCP2515TransferData(address);
78     MCP2515TransferData(bits);
79     MCP2515TransferData(data);
80     SPICsPin(MCP2515_CS_DACT);
81 }
82
83 //*****************************************************************************
84 // MCP2515 reset
85 //*****************************************************************************
86 void MCP2515Reset(void)
87 {
88     uint8_t status = 0;
89
90     SPICsPin(MCP2515_CS_ACT);
91     MCP2515TransferData(MCP2515_RESET_CMD);
92     SPICsPin(MCP2515_CS_DACT);
93
94     while(MCP2515_MODE_CFG != status)
95     {
96         MCP2515ReadRegister(MCP2515_CANSTAT, &status);
97     }
98 }
99
100 //*****************************************************************************
101 // MCP2515 request to send message
102 //*****************************************************************************
103 void MCP2515RequestSend(uint8_t txBuffID)
104 {
105     SPICsPin(MCP2515_CS_ACT);
```

```
106     MCP2515TransferData(MCP2515_RTS_CMD | (1U << txBuffID));
107     SPICsPin(MCP2515_CS_DACT);
108 }
109 
110 //*****************************************************************************
111 // MCP2515 read status operation
112 //*****************************************************************************
113 void MCP2515ReadStatus(uint8_t *status)
114 {
115     SPICsPin(MCP2515_CS_ACT);
116     MCP2515TransferData(MCP2515_READ_STATUS_CMD);
117     *status = MCP2515TransferData(MCP2515_DUMMY_DATA);
118     SPICsPin(MCP2515_CS_DACT);
119 }
120 
121 //*****************************************************************************
122 // MCP2515 write data
123 //*****************************************************************************
124 void MCP2515Write(MCP2515Msg_t *mcpMsg)
125 {
126     uint8_t i, tbufdata[4];
127     tbufdata[0] = (mcpMsg->msgID >> 3);
128     tbufdata[1] = ((mcpMsg->msgID & 0x07) << 5);
129     tbufdata[2] = 0;
130     tbufdata[3] = 0;
131 
132     SPICsPin(MCP2515_CS_ACT);
133     MCP2515TransferData(MCP2515_WRITE_TXB_CMD);
134     for(i=0;i<4;i++) MCP2515TransferData(tbufdata[i]);
135     MCP2515TransferData(mcpMsg->msgLen);
136     for(i=0;i<8;i++) MCP2515TransferData(mcpMsg->msgData[i]);
137     SPICsPin(MCP2515_CS_DACT);
138 
139     MCP2515TransferData(0x81);
140 }
141 
142 //*****************************************************************************
143 // MCP2515 read data
144 //*****************************************************************************
145 void MCP2515Read(MCP2515Msg_t *mcpMsg)
146 {
147     uint8_t i, status, tbufdata[4], len;
148     MCP2515ReadStatus(&status);
149 
150     if(status == 3)
151     {
152         SPICsPin(MCP2515_CS_ACT);
153         MCP2515TransferData(MCP2515_Read_RXB_CMD);
154         for(i=0;i<4;i++) tbufdata[i] =
155         MCP2515TransferData(MCP2515_DUMMY_DATA);
156         mcpMsg->msgID = (tbufdata[0] << 3) + (tbufdata[1] >> 5);
157         len = MCP2515TransferData(MCP2515_DUMMY_DATA);
158         mcpMsg->msgLen = len & 0x0F;
159         for(i=0;i<8;i++) mcpMsg->msgData[i] =
160         MCP2515TransferData(MCP2515_DUMMY_DATA);
161         SPICsPin(MCP2515_CS_DACT);
162     }
```

```
}
```

A.8 Código MCP4725

```
/*****************************************************************************//**
* @file mcp4725.c
* @date 15 de set de 2019
* @author Eduardo Henrique Fideles Ribeiro
* @author Wiviane Caroline Maneira
* @brief MCP4725 source file
*******************************************************************************/
//*****************************************************************************
// Includes
//*****************************************************************************
#include "SteeringControl/mcp4725.h"

//*****************************************************************************
// Function to configure MCP4725 DAC
//*****************************************************************************
void MCP4725Configuration(void)
{
    I2CConfiguration();
}

//*****************************************************************************
// Function to send DAC data to MCP4725 module using I2C0 bus
//*****************************************************************************
void MCP4725Write(uint16_t dacValue)
{
    uint8_t high, low;
    high = (dacValue >> 4) & 0x00FF;
    low = ((dacValue & 0x000F) << 4);
    I2CSend(MCP4725_ADDRESS,3,MCP4725_WRITE_REGISTER,high,low);
}
```

A.9 Código MOTOR

```
/*****************************************************************************//**
* @file motor.c
* @date 15 de set de 2019
* @author Eduardo Henrique Fideles Ribeiro
* @author Wiviane Caroline Maneira
* @brief  Motor source file
*******************************************************************************/

//*****************************************************************************
// Update motor parameters
//*****************************************************************************
#include "SteeringControl/motor.h"

//*****************************************************************************
// Update motor parameters
//*****************************************************************************
void UpdateMotor(motor_t *motor)
{
    if(motor->direction == 0)
```

```
20      {
21          GPIOSetMotorRight();
22          DACWrite(motor->voltage);
23      }
24      else if(motor->direction == 1)
25      {
26          GPIOSetMotorLeft();
27          DACWrite(motor->voltage);
28      }
29  }
```

A.10 Código PID

```
1   /*****************************************************************************//**
2   * @file pid.c
3   * @date 5 de nov de 2019
4   * @author Eduardo Henrique Fideles Ribeiro
5   * @author Wiviane Caroline Maneira
6   * @brief  PID source file
7   *****************************************************************************/
8   //***************************************************************************
9   // Include
10  //***************************************************************************
11  #include "SteeringControl/pid.h"
12
13  //***************************************************************************
14  // PID calculation function
15  //***************************************************************************
16  void PIDCalc(pid_t *pid, int16_t setpoint, int16_t actual_position)
17  {
18      if((setpoint != actual_position)
19          && (actual_position < 721)
20          && (actual_position > -721))
21      {
22          // Calculate error
23          pid->error = setpoint - actual_position;
24
25          // Calculate integral
26          pid->integral = pid->integral + pid->error;
27
28          // Calculate derivative
29          pid->derivative = (pid->error - pid->last_error);
30
31          // Calculate output
32          pid->output = Kp*pid->error + Ki*pid->integral + Kd*pid->derivative;
33
34          // Check output limits
35          if(pid->output > MAX_ANGLE) {
36              pid->output = MAX_ANGLE;
37          } else if(pid->output < MIN_ANGLE) {
38              pid->output = MIN_ANGLE;
39          }
40
41          // Clear variables
42          pid->last_error = pid->error;
43          pid->error = 0;
44      }
```

```
45      else
46      {
47          pid->error = 0;
48          pid->last_error = 0;
49          pid->derivative = 0;
50          pid->integral = 0;
51          pid->output = 0;
52          pid->motorVoltage = 0;
53          pid->motorDirection = 0;
54      }
55  }
```

A.11 Código SPI

```
1   /*****************************************************************************//**
2   * @file spi.c
3   * @date 25 de ago de 2019
4   * @author Eduardo Henrique Fideles Ribeiro
5   * @author Wiviane Caroline Maneira
6   * @brief SPI source file
7   *******************************************************************************/
8   //*****************************************************************************
9   // Includes
10  //*****************************************************************************
11  #include "SteeringControl/spi.h"
12
13  //*****************************************************************************
14  // SPI master configuration pins
15  //*****************************************************************************
16  void SPIMasterConfiguration(void)
17  {
18      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
19      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3);
20      GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_PIN_3);
21
22      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
23      GPIOPinConfigure(GPIO_PB4_SSI2CLK);
24      GPIOPinConfigure(GPIO_PB6_SSI2RX);
25      GPIOPinConfigure(GPIO_PB7_SSI2TX);
26      GPIOPinTypeSSI(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_6 | GPIO_PIN_7);
27
28      SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);
29      SSIClockSourceGet(SSI2_BASE);
30      SSIConfigSetExpClk(SSI2_BASE, SysCtlClockGet(),
31                         SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER,
32                         4000000, 8);
33      SSIEnable(SSI2_BASE);
34      SPIFifoClear();
35  }
36
37  //*****************************************************************************
38  // SPI slave configuration pins
39  //*****************************************************************************
40  void SPISlaveConfiguration(void)
41  {
42      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
43      SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
```

```
44      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3);
45      GPIOPinConfigure(GPIO_PA2_SSI0CLK);
46      GPIOPinConfigure(GPIO_PA3_SSI0FSS);
47      GPIOPinConfigure(GPIO_PA4_SSI0RX);
48      GPIOPinConfigure(GPIO_PA5_SSI0TX);
49      GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4
50                                      | GPIO_PIN_3 | GPIO_PIN_2);
51      SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(),
52                         SSI_FRF_MOTO_MODE_0,
53                         SSI_MODE_SLAVE, 4000000, 8);
54      SSIEnable(SSI0_BASE);
55      SPIFifoClear();
56  }
57
58  //*****************************************************************************
59  // SPI clear FIFO data
60  //*****************************************************************************
61  void SPIFifoClear(void)
62  {
63      uint32_t byte;
64      while(SSIDataGetNonBlocking(SSI2_BASE, &byte));
65  }
66
67  //*****************************************************************************
68  // SPI function to define the state of CS pin
69  //*****************************************************************************
70  void SPICsPin(uint8_t csCommand)
71  {
72      GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, csCommand);
73  }
74
75  //*****************************************************************************
76  // SPI write function
77  //*****************************************************************************
78  void SPIByteWrite(uint8_t byte)
79  {
80      SSIDataPutNonBlocking(SSI2_BASE, byte);
81  }
82
83  //*****************************************************************************
84  // SPI read function
85  //*****************************************************************************
86  void SPIByteRead(uint32_t *byte)
87  {
88      SSIDataGetNonBlocking(SSI2_BASE, byte);
89  }
```

A.12 Código TIMER

```
1  /*****************************************************************************//**
2  * @file timer.c
3  * @date 8 de ago de 2019
4  * @author Eduardo Henrique Fideles Ribeiro
5  * @author Wiviane Caroline Maneira
6  * @brief TIMER source code
7  *****************************************************************************/
8  //*****************************************************************************
```

```
9  // Includes
10 //*****************************************************************************
11 #include "SteeringControl/timer.h"
12
13 //*****************************************************************************
14 // Timer interrupt configuration to measure the velocity of motor
15 //*****************************************************************************
16 float TimerCalculateRPM(void)
17 {
18     TimerDisable(TIMER0_BASE, TIMER_A);
19     uint32_t timer_value = TimerValueGet(TIMER0_BASE, TIMER_A);
20     float frequency = (float)(SysCtlClockGet()/(timer_value));
21     float rpm = frequency*60;
22     HWREG(TIMER0_BASE + TIMER_O_TAV) = 0x0;
23     TimerEnable(TIMER0_BASE, TIMER_A);
24     return rpm;
25 }
26
27 //*****************************************************************************
28 // Timer interrupt configuration to measure the velocity of motor
29 //*****************************************************************************
30 void TimerIntConfiguration(void)
31 {
32     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
33     TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC_UP);
34     TimerLoadSet(TIMER0_BASE, TIMER_A, TIMER_MAX_VALUE);
35     TimerIntDisable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
36     TimerIntRegister(TIMER0_BASE, TIMER_A, TimerInterruptHandler);
37     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
38     TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
39     TimerEnable(TIMER0_BASE, TIMER_A);
40 }
```

A.13 Código UART

```
1  /*****************************************************************************//**
2  * @file uart.c
3  * @date 25 de ago de 2019
4  * @author Eduardo Henrique Fideles Ribeiro
5  * @author Wiviane Caroline Maneira
6  * @brief Serial communication source file
7  *****************************************************************************/
8  //*****************************************************************************
9  // Includes
10 //*****************************************************************************
11 #include "SteeringControl/uart.h"
12
13 //*****************************************************************************
14 // Configuration of UART1 pins
15 //*****************************************************************************
16 void UART1Configuration(void)
17 {
18     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
19     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
20     GPIOPinConfigure(GPIO_PB0_U1RX);
21     GPIOPinConfigure(GPIO_PB1_U1TX);
22     GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
```

```
23      UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,
24          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
25      UARTEnable(UART1_BASE);
26  }
27
28  //*****************************************************************************
29  // Configuration of UART2 pins
30  //*****************************************************************************
31  void UART2Configuration(void)
32  {
33      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
34      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
35      GPIOPinConfigure(GPIO_PD6_U2RX);
36      GPIOPinConfigure(GPIO_PD7_U2TX);
37      GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_6 | GPIO_PIN_7);
38      UARTConfigSetExpClk(UART2_BASE, SysCtlClockGet(), 9600,
39          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
40      UARTEnable(UART2_BASE);
41  }
42
43  //*****************************************************************************
44  // Configuration of UART5 pins
45  //*****************************************************************************
46  void UART5Configuration(void)
47  {
48      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
49      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
50      GPIOPinConfigure(GPIO_PE4_U5RX);
51      GPIOPinConfigure(GPIO_PE5_U5TX);
52      GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
53      UARTConfigSetExpClk(UART5_BASE, SysCtlClockGet(), 9600,
54          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
55      UARTEnable(UART5_BASE);
56  }
57
58  //*****************************************************************************
59  // Function to write char with UART1
60  //*****************************************************************************
61  void UART1Write(uint8_t c)
62  {
63      UARTCharPutNonBlocking(UART1_BASE, c);
64  }
65
66  //*****************************************************************************
67  // Function to write char with UART2
68  //*****************************************************************************
69  void UART2Write(uint8_t c)
70  {
71      UARTCharPutNonBlocking(UART2_BASE, c);
72  }
73
74  //*****************************************************************************
75  // Function to write char with UART5
76  //*****************************************************************************
77  void UART5Write(uint8_t c)
78  {
79      UARTCharPutNonBlocking(UART5_BASE, c);
```

```
80 }
81
82 //*****************************************************************************
83 // Function to read char with UART5
84 //*****************************************************************************
85 void UART5Read(uint8_t *c)
86 {
87     *c = UARTCharGetNonBlocking(UART5_BASE);
88     SysCtlDelay(25000);
89 }
90
91 //*****************************************************************************
92 // Function to check available data on UART5
93 //*****************************************************************************
94 uint8_t UART5AvailableData(void)
95 {
96     uint8_t ret;
97     ret = UARTCharsAvail(UART5_BASE);
98     return ret;
99 }
```

ANEXO B – Software

B.1 Código Principal

```
1  package com.example.gpstracker;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.os.CountDownTimer;
7  import android.os.Handler;
8  import android.support.design.widget.FloatingActionButton;
9  import android.support.v4.app.FragmentManager;
10 import android.view.View;
11 import android.support.v4.view.GravityCompat;
12 import android.support.v7.app.ActionBarDrawerToggle;
13 import android.view.MenuItem;
14 import android.support.design.widget.NavigationView;
15 import android.support.v4.widget.DrawerLayout;
16
17 import android.support.v7.app.AppCompatActivity;
18 import android.support.v7.widget.Toolbar;
19 import android.view.Menu;
20 import android.widget.Toast;
21
22 import java.util.ArrayList;
23
24 public class MainActivity extends AppCompatActivity
25         implements NavigationView.OnNavigationItemSelectedListener {
26
27     FileFragment fileFragment;
28     MapsFragment mapsFragment;
29     BluetoothFragment bluetoothFragment;
30     FragmentManager fragmentManager;
31
32     @Override
33     protected void onCreate(Bundle savedInstanceState) {
34         super.onCreate(savedInstanceState);
35         setContentView(R.layout.activity_main);
36
37         Toolbar toolbar = findViewById(R.id.toolbar);
38         setSupportActionBar(toolbar);
39
40         final FloatingActionButton fab = findViewById(R.id.fab);
41         fab.setOnClickListener(new View.OnClickListener() {
42             @Override
43             public void onClick(View view) {
44                if(isRecording){
45                     Toast.makeText(getBaseContext(),
46                                    "RECORDING ROUTE STOPPED!",
47                                    Toast.LENGTH_SHORT).show();
48                     fab.setImageResource(R.drawable.ic_play_white_24dp);
49                     //RouteRecorder.isRecording = false;
50                     isRecording = false;
```

```
51                }else {
52                    Toast.makeText(getBaseContext(),
53                                   "RECORDING ROUTE STARTED!",
54                                   Toast.LENGTH_SHORT).show();
55                    fab.setImageResource(R.drawable.ic_stop_white_24dp);
56                    //RouteRecorder.isRecording = true;
57                    isRecording = true;
58                }
59            }
60        });
61
62        DrawerLayout drawer = findViewById(R.id.drawer_layout);
63        NavigationView navigationView = findViewById(R.id.nav_view);
64        ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
65            this, drawer, toolbar, R.string.navigation_drawer_open,
66            R.string.navigation_drawer_close);
67        drawer.addDrawerListener(toggle);
68        toggle.syncState();
69        navigationView.setNavigationItemSelectedListener(this);
70
71        mapsFragment = new MapsFragment();
72        bluetoothFragment = new BluetoothFragment();
73        fileFragment = new FileFragment();
74        fragmentManager = getSupportFragmentManager();
75        fragmentManager.beginTransaction()
76        .replace(R.id.linearLayout, mapsFragment).commit();
77
78        handlerSimulation.postDelayed(new Runnable() {
79            @Override
80            public void run() {
81                mainRoutine();
82                handlerSimulation.postDelayed(this,3000);
83            }
84        },3000);
85    }
86
87    @Override
88    public void onBackPressed() {
89        DrawerLayout drawer = findViewById(R.id.drawer_layout);
90        if (drawer.isDrawerOpen(GravityCompat.START)) {
91            drawer.closeDrawer(GravityCompat.START);
92        } else {
93            super.onBackPressed();
94        }
95    }
96    @Override
97    public boolean onCreateOptionsMenu(Menu menu) {
98        // Inflate the menu;
99        // this adds items to the action bar if it is present.
100       getMenuInflater().inflate(R.menu.main, menu);
101       return true;
102   }
103
104   @Override
105   public boolean onOptionsItemSelected(MenuItem item) {
106       // Handle action bar item clicks here. The action bar will
107       // automatically handle clicks on the Home/Up button, so long
```

```
108         // as you specify a parent activity in AndroidManifest.xml.
109         int id = item.getItemId();
110         return super.onOptionsItemSelected(item);
111     }
112 
113     @SuppressWarnings("StatementWithEmptyBody")
114     @Override
115     public boolean onNavigationItemSelected(MenuItem item) {
116         // Handle navigation view item clicks here.
117         int id = item.getItemId();
118 
119         if (id == R.id.nav_routes) {
120             fragmentManager.beginTransaction()
121             .replace(R.id.linearLayout, fileFragment).commit();
122         } else if (id == R.id.nav_maps) {
123             fragmentManager.beginTransaction()
124             .replace(R.id.linearLayout, mapsFragment).commit();
125         } else if (id == R.id.nav_bluetooth) {
126             fragmentManager.beginTransaction()
127             .replace(R.id.linearLayout, bluetoothFragment).commit();
128         } else if (id == R.id.nav_start_route) {
129             if(!initRoute && isBluetoothConnected && routeChoosed != -1){
130                 fragmentManager.beginTransaction()
131                 .replace(R.id.linearLayout, mapsFragment).commit();
132                 initRoute = true;
133                 initialOperation = true;
134                 contGps = 0;
135                 Toast.makeText(getBaseContext(),
136                     "ROUTE OPERATION STARTED!",
137                     Toast.LENGTH_SHORT).show();
138             }
139             else if(!initRoute && !isBluetoothConnected && routeChoosed != -1)
140             {
141                 Toast.makeText(getBaseContext(),
142                     "ROUTE OPERATION PROBLEM\n\n
143                     CHECK BLUETOOTH CONNECTION!",
144                     Toast.LENGTH_LONG).show();
145             }
146             else if(!initRoute && isBluetoothConnected && routeChoosed == -1)
147             {
148                 Toast.makeText(getBaseContext(),
149                     "ROUTE OPERATION PROBLEM\n\n
150                     CHECK ROUTE SELECTED!",
151                     Toast.LENGTH_LONG).show();
152             }
153             else if(!initRoute && !isBluetoothConnected && routeChoosed == -1)
154             {
155                 Toast.makeText(getBaseContext(),
156                     "ROUTE OPERATION PROBLEM\n\n
157                     CHECK BLUETOOTH CONNECTION AND ROUTE SELECTED!",
158                     Toast.LENGTH_LONG).show();
159             }
160             else if(initRoute)
161             {
162                 Toast.makeText(getBaseContext(),
163                     "ROUTE OPERATION ALREADY RUNNING!",
164                     Toast.LENGTH_LONG).show();
```

```
165             }
166         }
167
168         DrawerLayout drawer = findViewById(R.id.drawer_layout);
169         drawer.closeDrawer(GravityCompat.START);
170         return true;
171     }
172 }
```

B.2 Código Bluetooth

```
1  package com.example.gpstracker;
2
3  import android.Manifest;
4  import android.bluetooth.BluetoothAdapter;
5  import android.bluetooth.BluetoothDevice;
6  import android.content.BroadcastReceiver;
7  import android.content.Context;
8  import android.content.Intent;
9  import android.content.IntentFilter;
10 import android.os.Build;
11 import android.os.Bundle;
12 import android.os.Handler;
13 import android.support.annotation.RequiresApi;
14 import android.support.v4.app.Fragment;
15 import android.support.v4.content.LocalBroadcastManager;
16 import android.util.Log;
17 import android.view.LayoutInflater;
18 import android.view.View;
19 import android.view.ViewGroup;
20 import android.widget.AdapterView;
21 import android.widget.Button;
22 import android.widget.ListView;
23 import android.widget.Toast;
24
25 import java.util.ArrayList;
26 import java.util.UUID;
27
28 /**
29  * A simple {@link Fragment} subclass.
30  */
31 public class BluetoothFragment extends Fragment {
32
33 // ------------------------------------------------------------------ //
34 //  Global variables
35 // ------------------------------------------------------------------ //
36     private static final String TAG = "MainActivity";
37     Context bluetoothFragmentContext;
38     Button btOnOff, btDiscoverable, btStartConnection, btDiscover;
39     BluetoothConnectionService mBluetoothConnection;
40     private static final UUID UUID_INSECURE =
41     UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
42     BluetoothDevice mBTdevice;
43     BluetoothAdapter bluetoothAdapter;
44     public ArrayList<BluetoothDevice> devicesBT =
45     new ArrayList<BluetoothDevice>();
46     public DeviceListAdapter deviceListAdapter;
```

```
47      ListView lvNewDevides;
48      Boolean isBluetoothConnected = false;
49      BluetoothParser bluetoothParser;
50
51      BroadcastReceiver broadcastReceiver1;
52      BroadcastReceiver broadcastReceiver2;
53      BroadcastReceiver broadcastReceiver3;
54      BroadcastReceiver broadcastReceiver4;
55      BroadcastReceiver mReceiver;
56
57      IntentFilter btOnOffIntent;
58      IntentFilter btDiscoverableIntent;
59      IntentFilter btDiscoverIntent;
60
61      private View view;
62  // ------------------------------------------------------------ //
63
64      public BluetoothFragment() {
65          // Required empty public constructor
66      }
67
68      @Override
69      public View onCreateView(LayoutInflater inflater, ViewGroup container,
70                               Bundle savedInstanceState) {
71          // Inflate the layout for this fragment
72          bluetoothFragmentContext = this.getContext();
73          View view =
74          inflater.inflate(R.layout.fragment_bluetooth, container, false);
75
76          btOnOff = view.findViewById(R.id.buttonOnOff);
77          btStartConnection = view.findViewById(R.id.buttonStartConnection);
78          btDiscoverable = view.findViewById(R.id.buttonDiscoverable);
79          btDiscover = view.findViewById(R.id.buttonDiscover);
80          lvNewDevides = view.findViewById(R.id.listViewNewDevices);
81          devicesBT = new ArrayList<>();
82          bluetoothParser = new BluetoothParser();
83
84          bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
85
86          lvNewDevides.setOnItemClickListener(
87              new AdapterView.OnItemClickListener() {
88              @Override
89              public void onItemClick(AdapterView<?> parent, View view,
90                                      int position, long id) {
91                  bluetoothAdapter.cancelDiscovery();
92                  Log.d(TAG,"onItemClick: Clicked on a device");
93
94                  String deviceName = devicesBT.get(position).getName();
95                  String deviceAddress = devicesBT.get(position).getAddress();
96                  if(Build.VERSION.SDK_INT > Build.VERSION_CODES.JELLY_BEAN_MR2){
97                      Log.d(TAG,
98                      "onItemClick: Trying to pair with " + deviceName);
99                      devicesBT.get(position).createBond();
100
101                     mBTdevice = devicesBT.get(position);
102                     mBluetoothConnection =
103                     new BluetoothConnectionService(bluetoothFragmentContext);
```

```
104                 }
105             }
106         });
107
108         return view;
109     }
110
111     @Override
112     public void onResume() {
113         super.onResume();
114
115         // ---------------------------------------------------------- //
116         //  Broadcast receiver to check bluetooth connecting state
117         // ---------------------------------------------------------- //
118         broadcastReceiver1 = new BroadcastReceiver() {
119             @Override
120             public void onReceive(Context context, Intent intent) {
121                 final String action = intent.getAction();
122
123                 if(action.equals(bluetoothAdapter.ACTION_STATE_CHANGED)){
124                     final int state =
125                     intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
126                                        bluetoothAdapter.ERROR);
127
128                     switch (state){
129                         case BluetoothAdapter.STATE_OFF:
130                             Log.d(TAG, "onReceive: STATE_OFF");
131                             break;
132                         case BluetoothAdapter.STATE_TURNING_OFF:
133                             Log.d(TAG,"broadcastReceiver1: STATE_TURNING_OFF");
134                             break;
135                         case BluetoothAdapter.STATE_ON:
136                             Log.d(TAG,"broadcastReceiver1: STATE_ON");
137                             break;
138                         case BluetoothAdapter.STATE_TURNING_ON:
139                             Log.d(TAG, "broadcastReceiver1: STATE_TURNING_ON");
140                             break;
141                     }
142                 }
143             }
144         };
145
146         // ---------------------------------------------------------- //
147         //  Broadcast receiver to check bluetooth connection state
148         // ---------------------------------------------------------- //
149         broadcastReceiver2 = new BroadcastReceiver() {
150             @Override
151             public void onReceive(Context context, Intent intent) {
152                 final String action = intent.getAction();
153
154                 if(action.equals(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED)){
155                     final int mode =
156                     intent.getIntExtra(BluetoothAdapter.EXTRA_SCAN_MODE,
157                                        BluetoothAdapter.ERROR);
158
159                     switch (mode){
160                         case BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE:
```

```
161                         Log.d(TAG,
162                         "broadcastReceiver2: Discoverability Enabled");
163                         break;
164                     case BluetoothAdapter.SCAN_MODE_CONNECTABLE:
165                         Log.d(TAG,
166                         "broadcastReceiver2: Discoverability Disabled.
167                                              Able to receive connections");
168                         break;
169                     case BluetoothAdapter.SCAN_MODE_NONE:
170                         Log.d(TAG,
171                         "broadcastReceiver2: Discoverability Disabled.
172                                            Not able to receive connections");
173                         break;
174                     case BluetoothAdapter.STATE_CONNECTING:
175                         Log.d(TAG,
176                         "broadcastReceiver2: Connecting...");
177                         break;
178                     case BluetoothAdapter.STATE_CONNECTED:
179                         Log.d(TAG,
180                         "broadcastReceiver2: Connected");
181                         break;
182                 }
183             }
184         }
185     };
186
187     // ---------------------------------------------------------------- //
188     //  Broadcast receiver to check bluetooth nearby devices
189     // ---------------------------------------------------------------- //
190     broadcastReceiver3 = new BroadcastReceiver() {
191         @Override
192         public void onReceive(Context context, Intent intent) {
193             final String action = intent.getAction();
194             Log.d(TAG, "onReceive: ACTION FOUND");
195
196             if(action.equals(BluetoothDevice.ACTION_FOUND)){
197                 BluetoothDevice device =
198                 intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
199                 devicesBT.add(device);
200                 Log.d(TAG, "onReceive: "
201                       + device.getName()
202                       + ": " +
203                       device.getAddress());
204                 deviceListAdapter = new DeviceListAdapter(context,
205                                         R.layout.device_adapter_view, devicesBT);
206                 lvNewDevides.setAdapter(deviceListAdapter);
207             }
208         }
209     };
210
211     // ---------------------------------------------------------------- //
212     //  Broadcast receiver to check bluetooth bound state
213     // ---------------------------------------------------------------- //
214     broadcastReceiver4 = new BroadcastReceiver() {
215         @Override
216         public void onReceive(Context context, Intent intent) {
217             final String action = intent.getAction();
```

```
218
219                 if(action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)){
220                     BluetoothDevice device =
221                     intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
222
223                     if(device.getBondState() == BluetoothDevice.BOND_BONDED){
224                         Log.d(TAG, "broadcastReceiver4: BOND_BONDED");
225                         mBTdevice = device;
226                         Toast.makeText(bluetoothFragmentContext,
227                         "DEVICES PAIRED!",Toast.LENGTH_SHORT).show();
228                     }
229                     if(device.getBondState() == BluetoothDevice.BOND_BONDING){
230                         Log.d(TAG, "broadcastReceiver4: BOND_BONDING");
231                     }
232                     if(device.getBondState() == BluetoothDevice.BOND_NONE){
233                         Log.d(TAG, "broadcastReceiver4: BOND_NONE");
234                     }
235                 }
236             }
237         };
238
239         // ------------------------------------------------------------- //
240         // Creating Button actions
241         // ------------------------------------------------------------- //
242         // ------------------------------------------------------------- //
243         // ON/OFF bluetooth
244         // ------------------------------------------------------------- //
245         btOnOff.setOnClickListener(new View.OnClickListener(){
246             @Override
247             public void onClick(View v) {
248                 Log.d(TAG, "onClickOnOff: enabling/disabling bluetooth");
249                 if(bluetoothAdapter == null){
250                     Log.d(TAG, "onClickOnOff: Does not have BT capabilities");
251                 }
252                 if(!bluetoothAdapter.isEnabled()){
253                     Log.d(TAG, "onClickOnOff: enabling BT");
254                     Intent enableBTIntent =
255                     new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
256                     startActivity(enableBTIntent);
257                 }
258                 if(bluetoothAdapter.isEnabled()){
259                     Log.d(TAG, "onClickOnOff: disabling BT");
260                     bluetoothAdapter.disable();
261                 }
262             }
263         });
264
265         // ------------------------------------------------------------- //
266         // Bluetooth start connection
267         // ------------------------------------------------------------- //
268         btStartConnection.setOnClickListener(new View.OnClickListener(){
269             @Override
270             public void onClick(View v) {
271                 Log.d(TAG,
272                 "startConnection: Initializing RFCOM bluetooth connection");
273                 mBluetoothConnection.startClient(mBTdevice, UUID_INSECURE);
274                 isBluetoothConnected = true;
```

```
275                 Handler msgHandler = new Handler();
276                 msgHandler.postDelayed(new Runnable() {
277                     @Override
278                     public void run() {
279                         MainActivity.isBluetoothConnected = true;
280                         Toast.makeText(bluetoothFragmentContext,
281                                 "DEVICES CONNECTED!",
282                                 Toast.LENGTH_SHORT).show();
283                     }
284                 }, 2000);
285
286             }
287         });
288
289         // ------------------------------------------------------------- //
290         // Turn on bluetooth discoverable to another devices
291         // ------------------------------------------------------------- //
292         btDiscoverable.setOnClickListener(new View.OnClickListener() {
293             @Override
294             public void onClick(View v) {
295                 Log.d(TAG,
296                 "onClickEnableDiscoverable:
297                 Making device discoverable for 300 seconds");
298
299                 Intent discoverableIntent =
300                 new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
301                 discoverableIntent.putExtra(
302                     BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
303                 startActivity(discoverableIntent);
304             }
305         });
306
307         // ------------------------------------------------------------- //
308         // Discover another bluetooth device
309         // ------------------------------------------------------------- //
310         btDiscover.setOnClickListener(new View.OnClickListener() {
311             @Override
312             @RequiresApi(api = Build.VERSION_CODES.M)
313             public void onClick(View v) {
314                 Log.d(TAG, "btDiscover: Looking for unpaired devices");
315
316                 if(bluetoothAdapter.isDiscovering()){
317                     bluetoothAdapter.cancelDiscovery();
318                     Log.d(TAG, "btDiscover: Canceling discovery");
319
320                     checkBTPermissions();
321
322                     Log.d(TAG, "btDiscover: Starting discovery");
323                     bluetoothAdapter.startDiscovery();
324                 }
325                 if(!bluetoothAdapter.isDiscovering()){
326                     checkBTPermissions();
327
328                     Log.d(TAG, "btDiscover: Starting discovery");
329                     bluetoothAdapter.startDiscovery();
330                 }
331             }
```

```
332         });
333
334         // ------------------------------------------------------------- //
335         // Creating Intent filters
336         // ------------------------------------------------------------- //
337         IntentFilter filter =
338         new IntentFilter(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
339         bluetoothFragmentContext.registerReceiver(broadcastReceiver4,
340                                                   filter);
341
342         btOnOffIntent =
343          new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
344         bluetoothFragmentContext.registerReceiver(broadcastReceiver1,
345                                                   btOnOffIntent);
346
347         btDiscoverableIntent =
348         new IntentFilter(bluetoothAdapter.ACTION_SCAN_MODE_CHANGED);
349         bluetoothFragmentContext.registerReceiver(broadcastReceiver2,
350                                                   btDiscoverableIntent);
351
352         btDiscoverIntent = new IntentFilter(BluetoothDevice.ACTION_FOUND);
353         bluetoothFragmentContext.registerReceiver(broadcastReceiver3,
354                                                   btDiscoverIntent);
355
356         LocalBroadcastManager.getInstance(bluetoothFragmentContext)
357         .registerReceiver(mReceiver, new IntentFilter("incomingMessage"));
358     }
359
360     @RequiresApi(api = Build.VERSION_CODES.M)
361     private void checkBTPermissions() {
362         if(Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP){
363             int permissionCheck =
364             bluetoothFragmentContext.checkSelfPermission(
365                 "Manifest.permission.ACCESS_FINE_LOCATION");
366             permissionCheck +=
367             bluetoothFragmentContext.checkSelfPermission(
368                 "Manifest.permission.ACCESS_COARSE_LOCATION");
369             if(permissionCheck != 0){
370                 this.requestPermissions(
371                     new String[]{Manifest.permission.ACCESS_FINE_LOCATION,
372                     Manifest.permission.ACCESS_COARSE_LOCATION}, 1001);
373             }else{
374                 Log.d(TAG, "checkBTPermissions:
375                 No need to check permissions.
376                 SDK version < LOLLIPOP");
377             }
378         }
379     }
380
381     public void sendData(int angle){
382         if(isBluetoothConnected){
383             byte[] bytes = bluetoothParser.parceMotorData(angle);
384             mBluetoothConnection.write(bytes);
385         }
386     }
387 }
```

B.3 Código Arquivos

```
1  package com.example.gpstracker;
2
3
4  import android.content.Context;
5  import android.os.Bundle;
6  import android.support.annotation.NonNull;
7  import android.support.annotation.Nullable;
8  import android.support.v4.app.Fragment;
9  import android.view.LayoutInflater;
10 import android.view.View;
11 import android.view.ViewGroup;
12 import android.widget.AdapterView;
13 import android.widget.ArrayAdapter;
14 import android.widget.ListView;
15 import android.widget.Toast;
16
17 import java.util.ArrayList;
18
19
20 /**
21  * A simple {@link Fragment} subclass.
22  */
23 public class FileFragment extends Fragment {
24
25     ArrayAdapter<String> adapter;
26     ArrayList<String> routes;
27     ListView lvFiles;
28     Context fileFragmentContext;
29
30     public FileFragment() {
31         // Required empty public constructor
32     }
33
34     @Override
35     public View onCreateView(LayoutInflater inflater, ViewGroup container,
36                              Bundle savedInstanceState) {
37         View view = inflater.inflate(R.layout.fragment_file, container, false);
38         fileFragmentContext = this.getContext();
39
40         lvFiles = view.findViewById(R.id.listviewFiles);
41         routes = new ArrayList<>();
42
43         routes.add("ROUTE 1");
44         routes.add("ROUTE 2");
45         adapter = new ArrayAdapter<>(fileFragmentContext,
46                     android.R.layout.simple_list_item_1, routes);
47         lvFiles.setAdapter(adapter);
48
49         lvFiles.setOnItemClickListener(
50             new AdapterView.OnItemClickListener() {
51             @Override
52             public void onItemClick(AdapterView<?> parent, View view,
53                                     int position, long id) {
54                 MainActivity.routeChoosed = position;
```

```
55                  Toast.makeText(fileFragmentContext,
56                  "ROUTE "+ (position + 1) + " SELECTED!",
57                  Toast.LENGTH_SHORT).show();
58              }
59          });
60
61          return view;
62      }
63  }
```

B.4 Código Mapa

```
1   package com.example.gpstracker;
2
3
4   import android.os.Bundle;
5   import android.support.annotation.NonNull;
6   import android.support.annotation.Nullable;
7   import android.support.v4.app.Fragment;
8   import android.view.LayoutInflater;
9   import android.view.View;
10  import android.view.ViewGroup;
11
12  import com.google.android.gms.maps.CameraUpdateFactory;
13  import com.google.android.gms.maps.GoogleMap;
14  import com.google.android.gms.maps.OnMapReadyCallback;
15  import com.google.android.gms.maps.SupportMapFragment;
16  import com.google.android.gms.maps.model.LatLng;
17  import com.google.android.gms.maps.model.Marker;
18  import com.google.android.gms.maps.model.MarkerOptions;
19
20
21  /**
22   * A simple {@link Fragment} subclass.
23   */
24  public class MapsFragment extends Fragment implements OnMapReadyCallback {
25
26      static GoogleMap map;
27      static MarkerOptions options;
28      static Marker marker;
29
30      public MapsFragment() {
31          // Required empty public constructor
32      }
33
34
35      @Override
36      public View onCreateView(LayoutInflater inflater, ViewGroup container,
37                               Bundle savedInstanceState) {
38          // Inflate the layout for this fragment
39          View v = inflater.inflate(R.layout.fragment_maps, container, false);
40          return v;
41      }
42
43      @Override
44      public void onViewCreated(@NonNull View view,
45                                @Nullable Bundle savedInstanceState) {
```

```
46          super.onViewCreated(view, savedInstanceState);
47          SupportMapFragment mapFragment =
48          (SupportMapFragment)getChildFragmentManager().findFragmentById(R.id.map1);
49          mapFragment.getMapAsync(this);
50      }
51
52      @Override
53      public void onMapReady(GoogleMap googleMap) {
54          map = googleMap;
55          options = new MarkerOptions();
56
57          LatLng latLng = new LatLng(-11.362918, -52.885738);
58          options.position(latLng).title("Marker");
59          marker = map.addMarker(options);
60          map.setMinZoomPreference(3.6f);
61          map.moveCamera(CameraUpdateFactory.newLatLng(latLng));
62      }
63
64      static public void setMapPosition(double latitude, double longitude){
65          LatLng latLng = new LatLng(latitude, longitude);
66          marker.setPosition(latLng);
67          map.moveCamera(CameraUpdateFactory.newLatLng(latLng));
68          map.setMinZoomPreference(19.0f);
69      }
70  }
```