UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

ANDERSON AUGUSTO HEINZ

SISTEMA DE DETECÇÃO DE VAGAS PARALELAS E ESTACIONAMENTO AUTOMÁTICO UTILIZANDO SENSORES ULTRASSÔNICOS

Joinville
2014

Anderson Augusto Heinz

SISTEMA DE DETECÇÃO DE VAGAS PARALELAS E ESTACIONAMENTO AUTOMÁTICO UTILIZANDO SENSORES ULTRASSÔNICOS

Monografia submetida ao Curso de Graduação em Engenharia Mecatrônica da Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientador: Anderson Wedderhoff Spengler

Joinville
2014

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Heinz, Anderson Augusto
Sistema de detecção de vagas paralelas e estacionamento automático utilizando sensores ultrassônicos / Anderson Augusto Heinz ; orientador, Anderson Wedderhoff Spengler - Florianópolis, SC, 2014.
69 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Joinville. Graduação em Engenharia Mecatrônica.

Inclui referências

1. Engenharia Mecatrônica. 2. Estacionamento Automático em Vagas Paralelas. 3. Geração de Trajetória para Estacionamento. 4. Sistema Embarcado. 5. Sensores Ultrassônicos. I. Spengler, Anderson Wedderhoff. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Mecatrônica. III. Título.

Anderson Augusto Heinz

SISTEMA DE DETECÇÃO DE VAGAS PARALELAS E ESTACIONAMENTO AUTOMÁTICO UTILIZANDO SENSORES ULTRASSÔNICOS

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro Mecatrônico e aprovado em sua forma final pela comissão examinadora e pelo curso de Engenharia Mecatrônica da Universidade Federal de Santa Catarina.

_____

Prof. Milton Evangelista de Oliveira Filho, Dr. Eng.
Coordenador do Curso

Banca Examinadora:

_____

Prof. Anderson Wedderhoff Spengler, Dr. Eng.
Orientador

_____

Prof. Tiago Vieira da Cunha, Dr. Eng.
Universidade Federal de Santa Catarina

_____

Prof. Luis Fernando Caparroz Duarte, M.e

Centro Universitário Nossa Senhora do Patrocínio

Aos meus pais, Jacó e Joseane, que não mediram esforços para garantir a minha educação e me apoiaram em todos os momentos, tornando a realização desse trabalho possível.

AGRADECIMENTOS

Agradeço a Deus, pela força nos momentos de fraqueza e superação.

Aos meus pais, por terem me incentivado durante todo esse tempo, sempre acreditando na minha capacidade.

À minha amada namorada, Andressa , pelo carinho, paciência e compreensão durante o desenvolvimento deste trabalho.

Ao meu orientador, Anderson, pelo suporte durante todas as fases do desenvolvimento deste trabalho.

Agradeço ao SATE/FEEC/UNICAMP, pela prototipagem da placa de controle, fundamental para realização deste trabalho.

Ao professor Tiago, pela motivação na escolha do tema.

À UFSC Joinville, por ter me autorizado a utilizar o Laboratório de Eletrônica, tornando possível a montagem da placa eletrônica desenvolvida.

Aos colegas e professores, pelos momentos passados juntos em salas de aula e fora.

"A ciência é, portanto, uma perversão de si mesma, a menos que tenha como fim último, melhorar a humanidade."

Nikola Tesla

RESUMO

O presente trabalho foi idealizado considerando a dificuldade que várias pessoas possuem em realizar manobras de estacionamento em vagas paralelas e a crescente aposta do mercado na comercialização de veículos equipados com esse tipo de sistema. O objetivo do projeto consiste em desenvolver um sistema embarcado que possibilite um veículo estacionar em uma vaga paralela de forma automática. O sistema desenvolvido faz uso de um sensor ultrassônico para detectar a vaga. Uma placa de controle foi especialmente desenvolvida para gerenciar todos os componentes utilizados. O firmware implementado é responsável por controlar o veículo para que este siga a trajetória especificada. O sistema foi embarcado em um protótipo para realização de testes. Em todos os testes executados, o sistema comportou-se da maneira esperada sendo capaz de estacionar o carro dentro da vaga sem qualquer tipo de colisão.

Palavras chave: Sistema Embarcado; Estacionamento Automático em Vagas Paralelas; Geração de Trajetória para Estacionamento.

ABSTRACT

This work was conceived considering people's difficult to maneuver a car into a parallel parking slot and the growing market for vehicles with this kind of system. The objective of this project is the development of an embedded system capable to automatically park a vehicle into a parallel slot. The developed system uses an ultrasonic sensor to detect the slot size. A control board has been specially built to manage all the components. The implemented firmware controls the vehicle to make it follow a specified path. A prototype was built to make the tests, in all of tests the system behaved accordingly and it was capable to park in the slot without any kind of collision.

Keywords: Embedded System; Automatic Parallel Parking; Parking Path Planning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABS	Anti-lock Brake System
BOM	Bill Of Materials
CC	Constant Current
CCO	Current-Controlled Oscillator
LED	Light-Emitting Diode
PLL	Phase Locked Loop
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
RAM	Randomic Access Memory
UML	Unified Modeling Language

SUMÁRIO

1 INTRODUÇÃO

Sistemas eletrônicos capazes de auxiliar o motorista a realizar manobras de estacionamento paralelo, consideradas complexas por muitas pessoas, já estão presentes em vários modelos de automóveis comercializados no Brasil.

Neste trabalho é desenvolvido um sistema capaz de realizar esta tarefa utilizando sensores ultrassônicos para identificação da vaga e um sistema de controle que executa a manobra de forma automática, ou seja, sem o auxílio do motorista. Um protótipo foi desenvolvido para validação e realização de testes.

Nas seções seguintes são descritos o modelo de geração de trajetória, os materiais e métodos utilizados, o processo de desenvolvimento do protótipo, os resultados obtidos e, ao final são apresentadas as conclusões e sugestões para trabalhos futuros.

1.1 JUSTIFICATIVA

Segundo Castro (2013) a frota de veículos circulando no Brasil mais que dobrou nos últimos 10 anos. Esse crescimento, comparado no mesmo período, é muito maior do que o crescimento populacional. Entretanto, o investimento federal em infraestrutura de transporte foi muito baixo. Ou seja, houve um aumento expressivo do número de carros, porém pouco se investiu em melhorias e criação de novas estradas.

Esse aumento desenfreado de veículos trafegando nas ruas aliado às dificuldades de estacionar em vagas apertadas pode ser um problema sério. Para muitos motoristas, estacionar o carro em vagas paralelas é considerado um desafio, que geralmente deixa marcas e a maioria evita esse tipo de vaga (OLIVEIRA; VENTURA, 2013).

Sistemas eletrônicos capazes de localizar uma vaga pequena, mas suficiente para estacionar o veículo e ainda auxiliar o motorista a executar a manobra de forma segura e sem preocupações, é algo considerado de grande valor para o usuário.

Os automóveis estão ficando cada vez mais modernos, eficientes, seguros e confortáveis devido à inserção de microcontroladores, câmeras, sensores e sistemas eletrônicos. O desenvolvimento de tecnologias que facilitem a utilização dos automóveis agrega valor ao produto e com o tempo passam a ser itens indispensáveis.

Algumas montadoras já possuem veículos sendo fabricados e vendidos com sistemas capazes de auxiliar ou realizar de forma autônoma manobras de estacionamento como: Frey (2013), Liszewski (2013), Oliveira e Ventura (2013), Achorn (2014) e Cooper (2014). Essa tecnologia é uma tendência para os próximos anos e representa o próximo passo na evolução dos automóveis (GRABIANOWSKI, 2013).

O desenvolvimento desse tema é viável, pois envolve a utilização de diversos conceitos vistos durante a formação do engenheiro mecatrônico. A construção de um protótipo não depende de instrumentos e infraestrutura avançados, podendo ser construído no laboratório de eletrônica da UFSC, campus Joinville.

1.2 OBJETIVOS

Este trabalho tem como objetivo desenvolver um sistema embarcado capaz de identificar uma vaga de estacionamento paralela, verificar se o tamanho da vaga é adequado para estacionar o veículo e, através da leitura dos sensores ultrassônicos e do algoritmo programado em um microcontrolador, efetuar o controle sobre o veículo de forma a realizar a manobra de forma segura.

Pretende-se também aperfeiçoar o conhecimento sobre o funcionamento de cada componente utilizado, bem como adquirir experiência no desenvolvimento de sistemas embarcados através da integração de firmware e hardware.

Verificar se o sistema proposto é capaz de cumprir os objetivos através da utilização de um protótipo de bancada equipado com o sistema desenvolvido.

2 REVISÃO BIBLIOGRÁFICA

2.1 EVOLUÇÃO DA TECNOLOGIA

Sistemas de estacionamento automático em vagas paralelas começaram a equipar os veículos há aproximadamente 10 anos, porém as primeiras ideias e tecnologias começaram a surgir em 1930 (LAUKKONEN, 2014).

Um dos primeiros sistemas foi idealizado de uma forma bem diferente do conceito utilizado atualmente. O sistema utilizava quatro pequenas rodas sendo duas localizadas no eixo dianteiro e as outras duas no eixo traseiro. Para estacionar o veículo era necessário puxar o freio de mão e pressionar um botão no painel. Um sistema hidráulico era acionado erguendo o veículo, que passava a ficar apoiado nas rodas auxiliares (POPULAR SCIENCE, 1934). O próximo passo era pressionar um dos outros dois botões disponíveis no painel para movimentar o veículo para a esquerda ou direita. A energia para acionar estas quatro rodas era fornecida pelo motor do veículo através do sistema de transmissão. Um quarto botão era utilizado para descer o automóvel e concluir a manobra. Esse sistema não chegou a ser um sucesso, porém foi o primeiro passo no desenvolvimento da tecnologia (ibidem, 1934).

Na década de 90 a tecnologia de automação de sistemas robóticos já estava avançada o suficiente para controlar máquinas com o uso de computadores. Foi quando o primeiro sistema de estacionamento paralelo guiado por computadores foi testado com sucesso (LAUKKONEN, 2014). Entretanto, até final do século 20, nenhum veículo com os sistema integrado havia sido produzido em escala comercial. O primeiro modelo a utilizar um sistema de estacionamento automático foi lançado em 2003 no Japão pela Toyota, o Prius. Nos Estados Unidos da América, o primeiro modelo comercializado foi o Lexus, no ano de 2006. Ambos os modelos utilizavam câmeras para realizar a manobra de forma lenta (BELL, 2014).

Em 2010 a Ford lançou no mercado o primeiro sistema a utilizar sensores ultrassônicos com os modelos Lincoln MKS e MKT. De forma rápida a Ford aperfeiçoou o sistema e o disponibilizou na versão *Hatch* do Focus e no SUV Escape (ibidem, 2014).

A segunda geração do sistema auxiliar de estacionamento (*Park Assist 2*) foi lançado em 2010 pela Volkswagen. O sistema consegue estacionar em vagas menores com relação ao anterior, auxilia o motorista a retirar o carro da vaga e também é capaz de estacionar em vagas perpendiculares. A última tecnologia recentemente desenvolvida pela Volkswagen (*Park Assist 3*), apresenta todas as características presentes nas versões anteriores, porém agora também é capaz de auxiliar o motorista a estacionar de frente em vagas paralelas (ACHORN, 2014).

O sistema mais avançado, atualmente disponível no mercado, é capaz de estacionar o veículo sem qualquer intervenção do motorista. O sistema é comercializado pela BMW e equipado no modelo i3. Ao pressionar o botão o sistema aciona os sensores ultrassônicos que identificam a vaga e executam a manobra de forma completamente autônoma (COOPER, 2014).

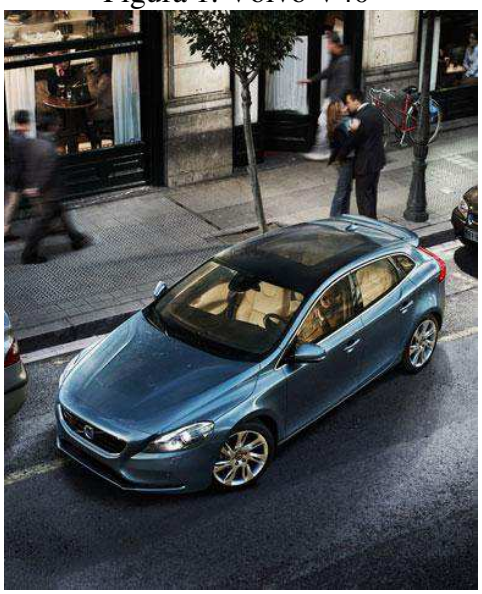
2.2 MODELOS COMERCIAIS

O número de veículos comercializados no Brasil equipados com algum tipo de sistema embarcado capaz de auxiliar o motorista a estacionar o seu veículo em vagas paralelas é relativamente grande e continua crescendo.

Nesta seção são apresentados alguns modelos de diferentes montadoras equipados com sistemas de estacionamento assistido.

O veículo da Volvo modelo V40 (Figura 1) é capaz de detectar e executar a manobra com perfeição em uma vaga apenas 50cm maior que o comprimento do carro. Para ativar o sistema, basta apertar um botão quando estiver ao lado de uma vaga (SIMÕES, 2014). O sistema irá verificar se a vaga é adequada e, caso positivo, apresentará as instruções ao motorista no painel de instrumentos. Para verificar a vaga e executar as manobras com segurança o modelo utiliza sensores ultrassônicos espalhados pelo veículo. O motorista controla os pedais enquanto o sistema controla a direção (VOLVO, 2014.).

Figura 1. Volvo V40



Fonte: Disponível em: http://www.volvocars.com/my/all-cars/volvo-v40/pages/default.aspx, acessado em outubro/2014.

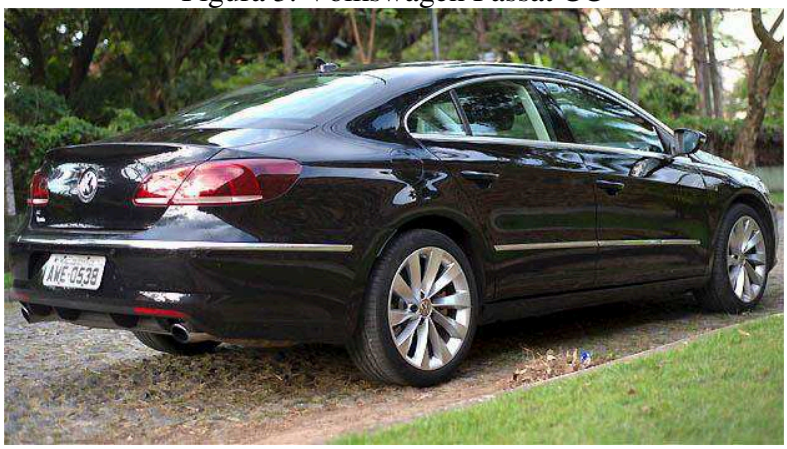
De maneira semelhante, o sistema utilizado pelo Focus (Figura 2) utiliza vários sensores ultrassônicos instalados nos para-choques e laterais do veículo para detectar a vaga. O sistema controla automaticamente a direção do carro enquanto o controle de velocidade e das marchas fica por conta do motorista (FORD, 2012). Para executar a manobra o motorista ativa o sistema através de um botão antes de passar pela vaga. O sistema realiza diversas medições enquanto o carro se desloca para a frente e avisa o motorista quando a vaga é detectada. O sistema solicita ao motorista que engate a marcha ré e controle a embreagem e freio até encaixar na vaga (ibidem, 2012).

Figura 2. Ford Focus

Fonte: Disponível em: http://www.ford.com.br/carros/novo-focus-hatch/caracteristicas, acessado em outubro/2014

O sistema oferecido pela Volkswagen no modelo Passat CC (Figura 3) é similar aos modelos apresentados anteriormente. Sensores ultrassônicos são utilizados para detectar a vaga quando o motorista pressiona um botão e sinaliza a manobra com o pisca. O sistema verifica a vaga e mostra o planejamento da rota para concluir a manobra. O motorista controla a troca de marcha e a velocidade do carro (OLIVEIRA; VENTURA, 2013).

Figura 3. Volkswagen Passat CC



Fonte: Disponível em: http://opopular.vrum.com.br/app/301,19/2013/11/27/interna_noticias,48723/carro-que-estaciona-sozinho-ajuda-de-verdade-quem-tem-dificuldades-em.shtml, acessado em outubro/2014.

2.3 TRABALHOS SIMILARES

Nos últimos anos diversos trabalhos relacionados ao tema de estacionamento automático foram desenvolvidos em diversos países. Nesta seção é apresentado um levantamento de alguns trabalhos similares.

O trabalho realizado por VORIA (2010) aborda o desenvolvimento de um sistema de estacionamento automático para ser utilizado em um veículo autônomo. O sistema utiliza uma rede de sensores divididos em dois subsistemas. Um deles é responsável por detectar a vaga e demais obstáculos através do uso de sensores ultrassônicos enquanto o outro faz uso do sensor presente no sistema de freio ABS do veículo para medir a velocidade do carro.

Os dados obtidos pelos sensores são enviados ao microcontrolador PIC18F2550 onde é feito um pré-processamento dos dados que, em seguida, são enviados ao computador de bordo. O computador de bordo analisa os dados recebidos, realiza o processamento e envia ao microcontrolador os comandos para o acionamento dos atuadores (VÓRIA, 2010).

A geração de trajetória implementada no computador de bordo foi desenvolvida com base no modelo de Ackerman que utiliza geometria para determinar a trajetória do veículo durante toda manobra (ibidem, 2010).

O sistema foi instalado em um veículo autônomo que já possuía todos os recursos de atuação sobre acelerador, freio e direção. Esse foi testado e pôde-se concluir que, na maioria das vezes, foi possível estacionar o carro sem qualquer colisão com nenhum obstáculo (ibidem, 2010).

Gupta et al. (2010) apresenta um sistema de estacionamento autônomo para vagas paralelas em carros nos quais a geração de trajetória é definida pela geometria conforme o modelo de Ackerman. A implementação realizada por Gupta et al. (2010) é capaz de calcular a trajetória mesmo que o veículo não esteja completamente alinhado ao veículo ao lado.

O sistema utiliza sensores ultrassônicos para detectar a vaga e possui um *encoder* na roda para determinar a velocidade do veículo. O sistema foi simulado via MATLAB e foi possível concluir que, mesmo em situações onde o veículo não está alinhado, ele é capaz de estacionar de forma adequada e alinhado (GUPTA et al., 2010).

Trabalhos similares foram realizados como em (JEONG et al., 2010), (FAIRUS et al., 2011) que também obtiveram resultados satisfatórios.

2.4 GERAÇÃO DE TRAJETÓRIA

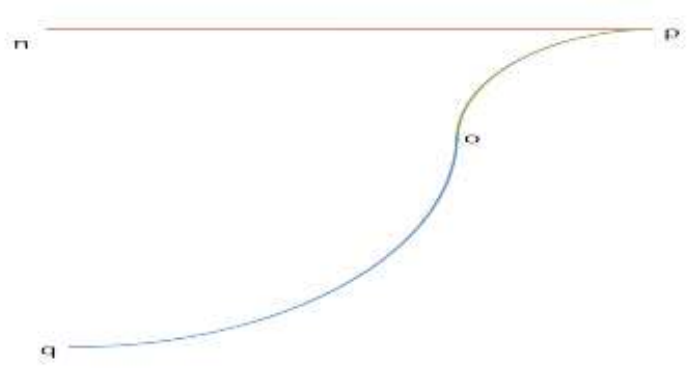
Para a tarefa de estacionar um veículo é necessário gerar uma trajetória que o veículo seja capaz de seguir. O modelo utilizado no desenvolvimento desse trabalho segue uma

abordagem geométrica denominada Modelo de Ackermann. O modelo foi primeiramente introduzido por Erasmus Darwin em 1759 que posteriormente foi patenteado por Rudolph Ackermannem 1817 (GUPTA et al., 2010).

O modelo de Ackermann faz uso de algumas considerações para descrever a trajetória de um veículo: o veículo é representado por um sólido sobre quatro rodas; as rodas dianteiras são movimentadas por um volante enquanto as rodas traseiras são fixas; a tração do veículo é dianteira e a velocidade deve ser determinada através de sensores (VÓRIA, 2010).

Segundo Gupta et al. (2010) o planejamento da trajetória descrito por Ackermann utiliza uma abordagem geométrica simples onde a manobra pode ser dividida em três segmentos (Figura 4). O segmento de reta (n-p) representa o espaço disponível e a orientação da vaga. Os outros dois segmentos (p-o e o-q) descrevem a trajetória do veículo e são dependentes dos parâmetros físicos do veículo.

Figura 4. Trajetória da manobra

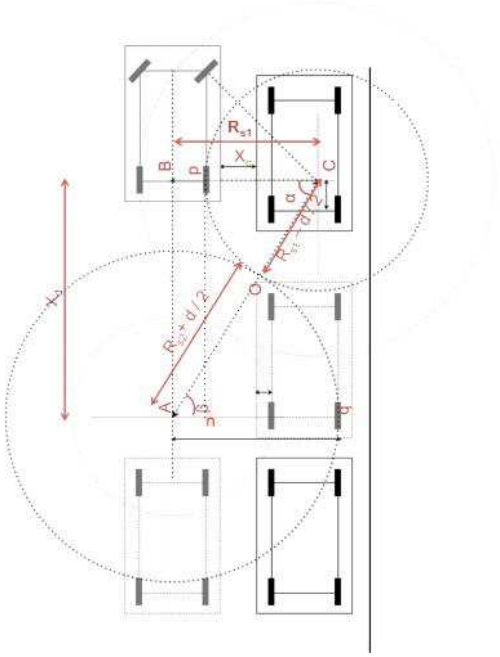


Fonte: GUPTA et al.,2010

Durante toda a trajetória o veículo apenas precisa variar o ângulo de distorção do volante duas vezes, nos pontos *o* e *q*. Todo o percurso é feito apenas à partir do conhecimento da distância entre os veículos estacionados e o veículo que está estacionando. Esses dados são obtidos pelo sensor ultrassônico instalado na lateral do veículo. Todos os outros parâmetros são constantes ou são calculados durante a manobra (GUPTA et al., 2010).

A Figura 5 mostra como os segmentos da trajetória da manobra são obtidos à partir das dimensões do veículo, da vaga e da distância perpendicular do veículo em relação a vaga.

Figura 5. Planejamento geométrico da trajetória. Modelo Ackerman



Fonte: GUPTA et al., 2010

A abordagem utilizada neste trabalho evidencia uma solução utilizando sensores ultrassônicos, uma aproximação do modelo de trajetória de Ackerman e um microcontrolador para realizar a tarefa de estacionamento automático em vagas paralelas. O cálculo da trajetória é apresentado na seção 4.2.

3 MATERIAIS E MÉTODOS

3.1 VEÍCULO

O veículo utilizado como protótipo para validação do conceito foi um carrinho de controle remoto (Figura 6) em escala 1/10 com relação ao veículo Camaro da fabricante Chevrolet. A Tabela 1 mostra um comparativo entre as dimensões do Camaro e do modelo utilizado.

Figura 6. Veículo utilizado como protótipo

Fonte: Próprio autor

Tabela 1: Comparativo entre dimensões do modelo utilizado e veículo comercial

	CAMARO	**PROTÓTIPO**
Altura (m)	1,377	0,13
Comprimento Total (m)	4,836	0,45
Distância entre eixos (m)	2,852	0,255
Largura Total (m)	1,918	0,19
Raio de Curvatura (m)	5,8	0,76[1]

Fonte: Disponível em: http://www.autobytel.com/chevrolet/camaro/2010/specifications/, acessado em outubro/2014.

Com base nos dados apresentados na Tabela 1 é possível notar que o modelo apresenta dimensões próximas à escala de 1:10 porém o Raio de Curvatura é relativamente maior.

A escolha por um veículo rádio controlado deu-se pelo fato deste já possuir o sistema completo de tração e direção elétricos. O que permitiu ter o controle sobre o veículo com poucas alterações mecânicas.

Outros motivos que levaram a esta escolha foram: o baixo investimento com aquisição do veículo e *drivers* para controle dos motores, a facilidade de executar testes e o fato do veículo ser o maior em escala encontrado.

[1]Valor calculado na seção 4.2

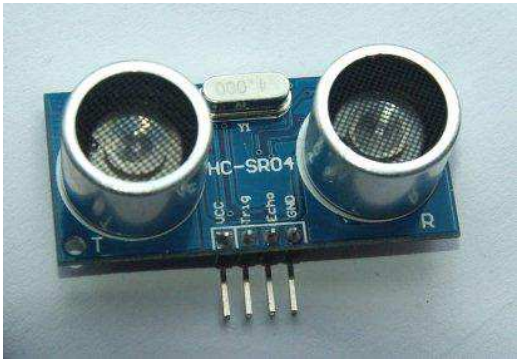
3.2 SENSORES

Os sensores ultrassônicos podem ser utilizados em diferentes aplicações, sendo as mais comuns a detecção de objetos e a medição de distância. Em geral o sensor ultrassônico é constituído de um emissor e um receptor de ondas sonoras em alta frequência. O emissor e o receptor não precisam estar fixados em conjunto porém essa é a configuração ideal para aplicações que necessitam identificar objetos e medir distâncias (SABER ELETRÔNICA, 2006).

O emissor é responsável por emitir as ondas que, quando atingem um obstáculo, são refletidas e captadas pelo receptor que gera um sinal que pode ser processado e fornecer a distância aproximada do objeto. A frequência de operação desse tipo de sensor é normalmente de 42kHz o que resulta em um comprimento de onda da ordem de alguns milímetros, possibilitando assim detectar objetos de alguns centímetros ou maiores (Ibidem, 2006).

O sensor utilizado foi o módulo HC-SR04 (Figura 7). Este módulo apresenta boa precisão sendo capaz de detectar objetos em uma faixa de 2cm a 4m de distância (HC-SR04, 2014).

Figura 7. Sensor ultrassônico HC-SR04



Fonte: Disponível em: http://www.rmcybernetics.com/images/main/eng/HC-SR04.jpg, acessado em outubro/2014.

Este sensor foi utilizado devido sua facilidade de aquisição aliado às suas características elétricas e capacidades de medição. As características elétricas deste módulo são apresentadas na Tabela 2.

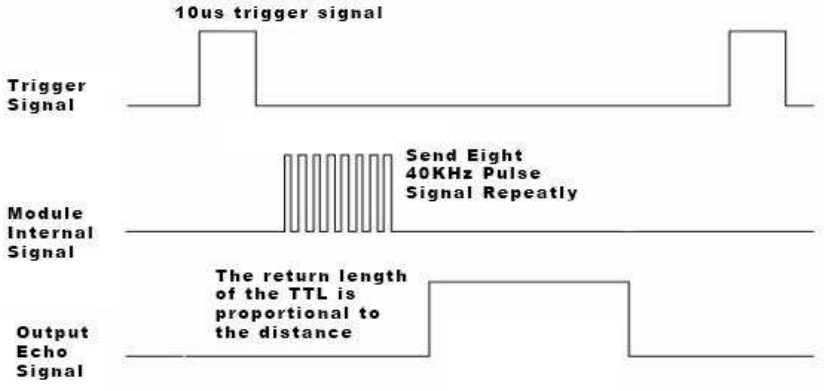
Tabela 2. Características elétricas sensor HC-SR04

Tensão de operação (V)	5
Consumo (mA)	15
Faixa de detecção (cm)	2 - 400
Ângulo de medição (graus)	15
Precisão (mm)	3

Fonte: HC-SR04, 2014

Para utilizar este módulo é necessário enviar um pulso de pelo menos 10µs no pino de *Trigger*. O circuito eletrônico presente no módulo identifica o pulso e inicia um disparo de 8 pulsos de ondas ultrassônicas com frequência de aproximadamente 40kHz pelo transmissor. Após o disparo, o pino de *Echo* é colocado em nível lógico alto e permanecerá assim até que o receptor receba o sinal de eco dos pulsos enviados. O processo é ilustrado na Figura 8 (HC-SR04, 2014).

Figura 8. Sinais de acionamento e leitura do sensor HC-SR04



Fonte: Disponível em: http://www.electrodragon.com/w/images/8/84/Frequency..jpg, acessado em outubro/2014.

A distância do objeto será diretamente proporcional ao tempo desse pulso gerado no pino de *Echo*. Portanto, basta medir a largura do pulso para calcular a distância do objeto. A largura do pulso varia entre 150µs a 20ms, porém, caso nenhum obstáculo seja detectado, o tamanho do pulso é limitado em 38ms (TECHNOLOGIES, 2014).

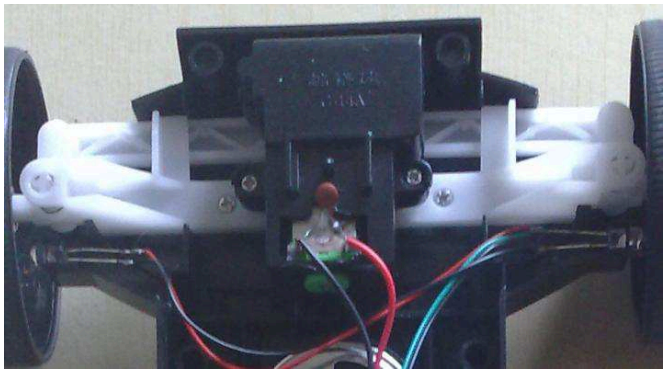
O pulso enviado de *Trigger* possui nível lógico de 3.3V (Determinado pela placa de controle). Embora o sensor opere com 5V, ele é capaz de identificar o sinal de 3.3V e funciona de forma adequada.

Entretanto, o sinal enviado pelo sensor pelo pino *Echo* possui nível lógico de 5V devido sua alimentação. O microcontrolador não suporta esse nível de tensão de entrada e, dessa forma foi necessário utilizar um divisor resistivo de tensão para adequar aos 3.3V da placa, que foi montado na própria placa do sensor.

3.3 MOTORES

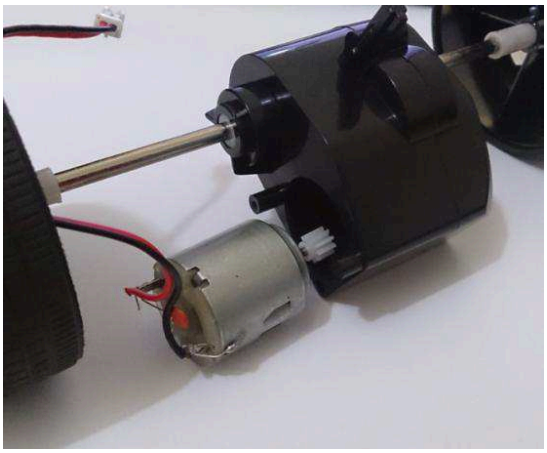
O veículo utilizado possui dois motores CC (Corrente Contínua) em seu interior. Um dos motores é utilizado para controle da direção (Figura 9) enquanto o outro é utilizado para a tração do veículo (Figura 10).

Figura 9. Motor acoplado ao sistema de direção



Fonte: Próprio autor

Figura 10. Motor e sistema de tração

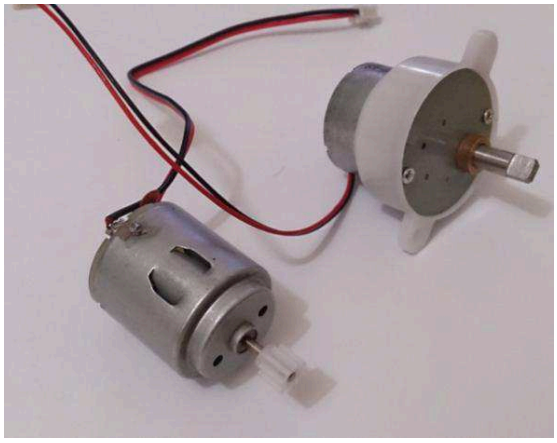


Fonte: Próprio autor

O motor utilizado para tracionar o carro funciona muito bem em velocidade normal de operação, porém não possui torque suficiente para mover o veículo quando utilizado em baixas rotações. Isto se deve, principalmente, ao fato do motor não possuir um conjunto de engrenagens interno e ao fato da relação de engrenagens da caixa de redução não ter sido projetada para baixas velocidades.

É fundamental que o veículo se desloque em uma velocidade baixa para garantir uma boa amostragem do sensor lateral e conseguir identificar a vaga e os possíveis obstáculos com maior precisão. Assim, foi necessário trocar o motor de tração por um motor que entregue um bom torque em baixas rotações. O motor escolhido é ilustrado ao lado do motor original na Figura 11. Este motor possui uma caixa de redução acoplada, o que permite que o motor desempenhe uma baixa velocidade angular e ao mesmo tempo um torque muito maior comparado ao motor original, mesmo sendo de menor dimensão.

Figura 11. Motor original (esquerda) e motor novo (direita)



Fonte: Próprio autor

As modificações realizadas para adaptar o novo motor entre outras informações mecânicas são apresentados na seção 4.3.2.

3.4 DRIVER

O *driver* escolhido para acionar os dois motores utilizados no protótipo é um módulo baseado no circuito integrado L9110 capaz de acionar 2 motores CC, independentemente, em ambos os sentidos de rotação. O módulo é ilustrado na Figura 12.

Este módulo foi escolhido por atender os requisitos mesmo possuindo dimensões bem compactas. Em se tratando de um sistema embarcado, essa é uma característica muito positiva, pois não ocupa muito espaço e é muito leve. Outra vantagem é o custo reduzido devido o baixo número de componentes e ao fato de não possuir dissipadores metálicos.

Figura 12. Módulo ponte H L9110

Fonte: Disponível em http://miniimg.rightinthebox.com/images/384x384/201311/mkoope1385538334197.jpg, acessado em outubro/2014.

As características elétricas do módulo são apresentadas na Tabela 3.

Tabela 3. Características Elétricas Módulo L9110

	MÍNIMO	**TÍPICO**	**MÁXIMO**
Tensão de alimentação (V)	2,5	6	12
Corrente de operação (µA)	200	350	500
Corrente da carga (mA)	-	800	850
Corrente de pico (A)	-	1,5	2

Fonte: Disponível em http://nvhs.files.wordpress.com/2013/02/datasheet-l9110.pdf, Acessado em outubro/2014.

4 DESENVOLVIMENTO DO PROTÓTIPO

O protótipo deve simular o comportamento do veículo durante toda etapa de estacionamento paralelo. Dessa forma, faz-se necessário o uso de um veículo no qual o sistema será embarcado, uma unidade de processamento onde toda a lógica será processada e convertida em sinais de controle e o firmware que descreve o comportamento do veículo com base nos sensores e algoritmos de trajetória.

4.1 DESCRIÇÃO E REQUISITOS DO PROTÓTIPO

O objetivo de implementar o sistema proposto em um protótipo é verificar a capacidade do sistema desempenhar suas funções na prática.

Um bom protótipo deve representar o sistema real da melhor forma possível, podendo assim fazer com que o desenvolvedor consiga identificar possíveis falhas e corrigi-las ainda durante a fase de desenvolvimento.

O protótipo aqui desenvolvido deve ser capaz de identificar uma vaga paralela à sua direita, verificar se o tamanho da vaga é suficiente e, dependendo dessa análise, executar a manobra ou não.

O veículo utilizado como protótipo está descrito em detalhes na seção 3.1. A unidade de processamento é apresentada na seção 4.3.1 enquanto o firmware é descrito na seção 4.4. Os testes realizados com o protótipo são apresentados na seção 4.5.

4.2 CÁLCULO DA TRAJETÓRIA

Conforme mencionado na seção 2.4, o cálculo da trajetória aqui apresentado utiliza como base o modelo de Ackerman. Entretanto, algumas condições impostas pelo modelo não se fazem presente no veículo utilizado: a tração do veículo é traseira ao invés de dianteira e a velocidade do veículo não é determinada por sensores conforme sugere o modelo.

Sabendo das diferenças entre o veículo e o modelo, optou-se por determinar geometricamente os parâmetros necessários para realizar a manobra e utilizá-los como ponto de partida nos testes com o veículo.

O cálculo da trajetória do veículo considera que o veículo já identificou e determinou o comprimento da vaga e encontra-se parado com orientação paralela à calçada. A orientação do veículo neste momento irá determinar a orientação final do veículo dentro da vaga, característica intrínseca do modelo de Ackerman.

Algumas informações do veículo devem ser conhecidas previamente. São elas: largura do veículo e raio de curvatura. A largura do veículo foi obtida através de simples medição,

porém o raio de curvatura teve que ser calculado com base no ângulo de esterçamento do veículo. Segundo Gupta (2010), o raio de curvatura pode ser determinado pela equação 1.

$$R = \frac{D}{\tan \beta} + \frac{W}{2} \tag{1}$$

onde:

R - Raio de curvatura

D - Distância entre eixos

β - Ângulo de esterçamento máximo

W - Largura total do veículo

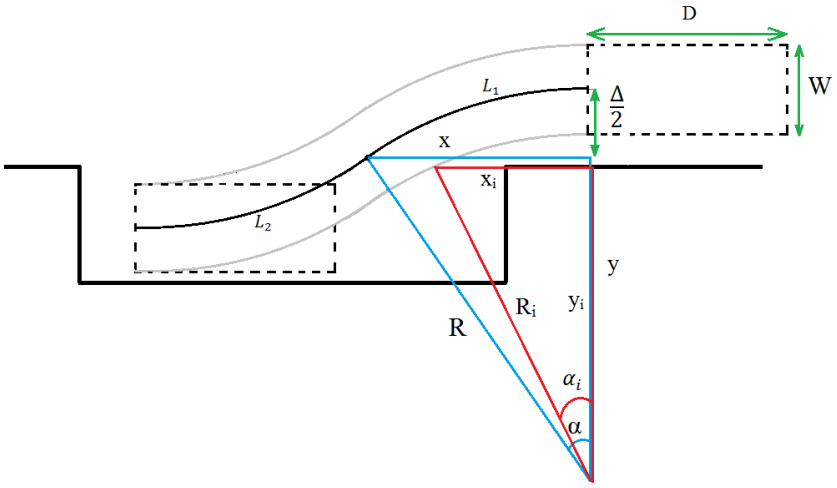
$$R = \frac{25{,}5}{\tan(21)} + \frac{19}{2}$$

$$R = 76cm$$

O raio de curvatura calculado é um valor aproximado, pois a medição do ângulo de esterçamento e demais medidas não é muito precisa. Considerando os valores obtidos é possível iniciar os cálculos de trajetória.

A Figura 13 ilustra a trajetória prevista para que o veículo utilize para realizar a manobra de estacionamento paralelo. Os cálculos que seguem são, em sua maior parte, deduzidos a partir dos dois triângulos retângulo destacados na imagem.

Figura 13. Cálculo da trajetória: Modelagem do problema



Fonte: Próprio autor

O deslocamento que o veículo deve percorrer de forma paralela em relação a vaga é determinado pela equação 2.

$$\Delta = W + s \tag{2}$$

onde:

Δ - Deslocamento paralelo

W - Largura total do veículo

s - Distância medida pelo sensor lateral

A distância medida pelo sensor será considerada nos cálculos como uma constante de valor igual a 5cm. Dessa forma, substituindo os valores, encontra-se o valor do deslocamento.

$$\Delta = 24cm$$

Uma vez determinado o valor do deslocamento é possível calcular o valor de y, necessário para determinar o ângulo α, através da equação 3:

$$y = R - \frac{\Delta}{2} \tag{3}$$

$$y = 76 - \frac{24}{2}$$

$$y = 64cm$$

Sabendo o valor de y e conhecendo o valor de R, é possível determinar o ângulo α utilizando trigonometria básica.

$$\alpha = \cos^{-1}\left(\frac{y}{R}\right) \tag{4}$$

$$\alpha = \cos^{-1}\left(\frac{64}{76}\right)$$

$$\alpha = 32{,}6^{o}$$

Com o valor de α determinado é possível calcular, de forma similar, o valor de x:

$$x = R * \sin(\alpha) \tag{5}$$

$$x = 76 * \sin(32{,}6)$$

$$x = 41cm$$

O raio interno é calculado para que seja possível estimar o tamanho mínimo necessário que o veículo consegue estacionar.

$$R_i = R - \frac{W}{2} \tag{6}$$

onde:

R_i - Raio interno

$$R_i = 76 - \frac{19}{2}$$

$$R_i = 66{,}5cm$$

A partir do raio interno pode-se utilizar o mesmo raciocínio para determinar o ângulo interno que representa o raio de curvatura considerando a lataria do veículo.

$$\alpha_i = \cos^{-1}\left(\frac{y_i}{R}\right) \tag{7}$$

$$\alpha_i = \cos^{-1}\left(\frac{R_i - s}{R}\right)$$

$$\alpha_i = \cos^{-1}\left(\frac{66{,}5 - 5}{66{,}5}\right)$$

$$\alpha_i = 22{,}4^{\circ}$$

De forma similar pode-se determinar o valor de x_i que representa a maior distância que o eixo do veículo deve estar em relação ao final da vaga para evitar um choque entre os veículos.

$$x_i = R_i * \sin\alpha_i \tag{8}$$

$$x_i = 66{,}5 * \sin(22{,}4)$$

$$x_i = 25{,}3cm$$

Um valor menor será adotado para obter uma folga e evitar que pequenas variações ocasionem contato entre os veículos

$$x_i = 20cm$$

Considerando D_t como a distância entre o eixo traseiro e a extremidade traseira da lataria do veículo, é possível estimar o tamanho mínimo necessário para estacionar o veículo de forma segura.

$$Vaga_{min} = 2x - x_i + D_t$$
$$Vaga_{min} = 2 * 41 - 20 + 9 \quad (9)$$
$$Vaga_{min} = 71cm$$

Acrescentando a distância que se deseja que o veículo mantenha no final da manobra (3cm), com relação ao veículo estacionado atrás, obtêm-se o valor que deve ser utilizado no firmware como condição para efetuar a manobra ou não.

$$Vaga_{min} = 74cm$$

O próximo passo é determinar a distância do arco L_1, que representa a distância que o veículo deve percorrer até atingir o ponto de inflexão. Essa informação será utilizada no firmware como condição para inverter a direção das rodas dianteiras.

$$L_1 = \frac{\pi * R * \alpha}{180} \quad (10)$$
$$L_1 = \frac{\pi * 76 * 32{,}6}{180}$$
$$L_1 = 43{,}24cm$$

O arco L_2 tem mesmo comprimento que L_1

$$L_2 = 43{,}24cm$$

4.3 HARDWARE

4.3.1 Eletrônica

A eletrônica presente neste trabalho consiste de uma placa de controle que foi desenvolvida especialmente para esta aplicação. Para o desenvolvimento do diagrama elétrico, lista de materiais e *layout,* foi utilizado o firmware *Altium Designer*. A placa possui um microcontrolador *ARM7* que é responsável por interpretar os dados recebidos pelos sensores e comandar o protótipo para que este estacione de forma automática.

4.3.1.1 Requisitos de hardware

A especificação de hardware foi definida com base no objetivo final que o protótipo deveria alcançar. Dessa forma é fundamental que a especificação esteja de acordo com as características físicas e elétricas do veículo utilizado.

Especificações da placa de controle:

-Tensão de operação: 6V (Tensão da bateria)

-Regulagem de tensão para 5V e 3,3V

-Conexão para até 3 sensores ultrassônicos

-Saída para acionar 2 motores CC

-3 Leds com cores distintas

-Possibilidade de comunicação wireless

-Botão de reset

-Botão configurável

4.3.1.2Microcontrolador

O modelo utilizado é o *LPC2132FBD64* da *NXP*. Este modelo possui 64 pinos em um encapsulamento *LQFP64*, barramento de 32bits, frequência do núcleo de até 60MHz, 64kb de memória flash, 16kb de memória RAM e possui 6 saídas para PWM.

Este microcontrolador foi escolhido devido suas funcionalidades suprirem todos os requisitos necessários para o funcionamento do protótipo.

O mapeamento dos pinos levou em consideração a funcionalidade exigida para cada pino, dependendo da aplicação. Para utilizar o módulo L9110 são necessários 4 sinais PWM, conforme citado na seção 3.4. Desta forma, escolheu-se 4 pinos com possibilidade de serem configurados como saída PWM.

Para utilizar o sensor ultrassônico durante a aplicação foi previsto que poderia ser necessário utilizar uma interrupção externa para cada sensor no pino de Echo. Desta maneira foram conectados os pinos de Echo dos três sensores previstos em hardware em pinos com capacidade para atuar como interrupção externa.

Os demais pinos não requerem funções especiais, logo foram conectados aos demais pinos disponíveis do microcontrolador. O mapeamento da utilização dos pinos do microcontrolador é apresentado na Tabela 4.

Tabela 4. Mapeamento dos pinos utilizados

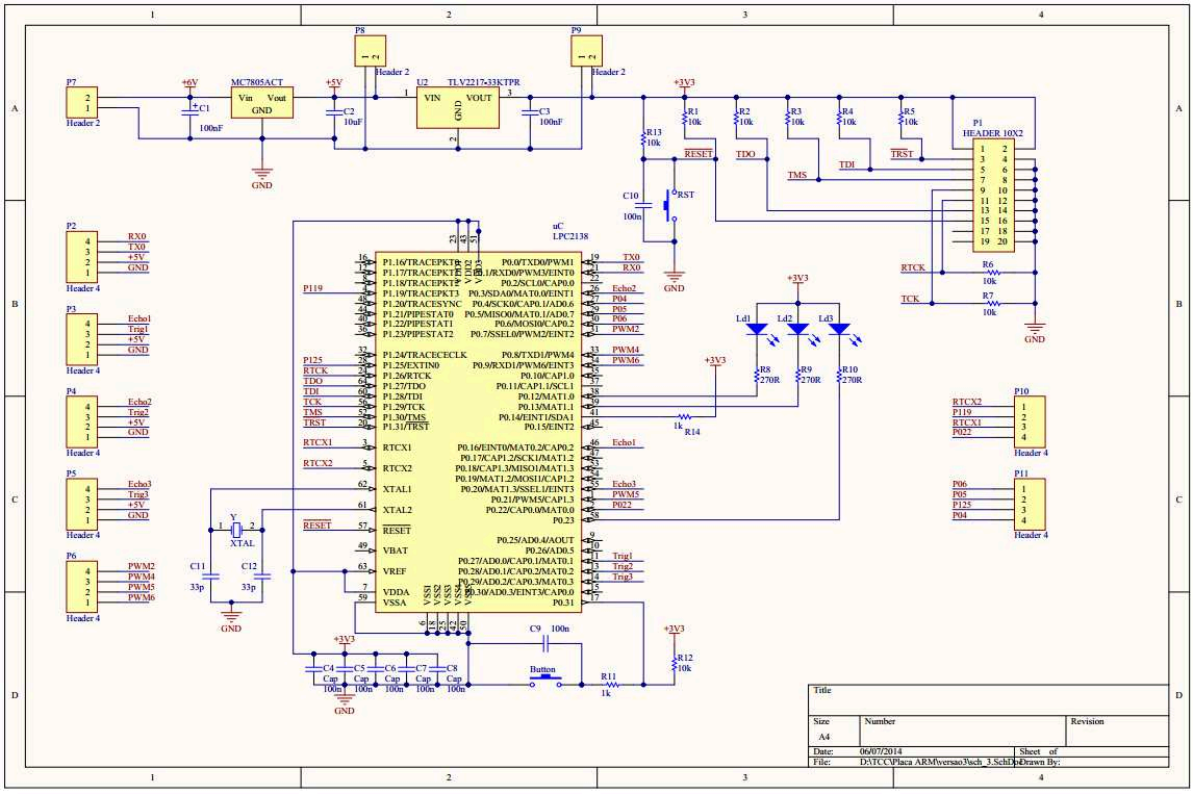
PINO	FUNÇÃO PADRÃO	FUNÇÃO UTILIZADA	SINAL
1	P0.21	PWM5	PWM5
6	VSS1	VSS1	GND
7	VDDA	VDDA	3V3
11	P0.27	P0.27	Trig1
13	P0.28	P0.28	Trig2
14	P0.29	P0.29	Trig3
17	P0.31	P0.31	Button
18	VSS2	VSS2	GND
20	TRST	TRST	TRST
23	VDD1	VDD1	3V3
24	RTCK	RTCK	RTCK
25	VSS3	VSS3	GND
26	P0.3	EINT1	Echo2
31	P0.7	PWM2	PWM2
33	P0.8	PWM4	PWM4
34	P0.9	PWM6	PWM6
38	P0.12	P0.12	Ld1
39	P0.13	P0.13	Ld2
42	VSS3	VSS3	GND
43	VDD2	VDD2	3V3
46	P0.16	EINT0	Echo1
50	VSS5	VSS5	GND
51	VDD3	VDD3	3V3
52	TMS	TMS	TMS
55	P0.20	EINT3	Echo3
56	TCK	TCK	TCK
57	RESET	RESET	RESET
58	P0.23	P0.23	Ld3
59	VSSAA	VSSA	GND
60	P1.28	TDI	TDI
61	XTAL2	XTAL2	XTAL2
62	XTAL1	XTAL1	XTAL1
63	VREF	VREF	3V3
64	P1.27	TDO	TDO

Fonte: Próprio autor

4.3.1.3 Diagrama elétrico - Esquemático

Com base nos requisitos e características dos atuadores e sensores utilizados foi possível desenhar o diagrama elétrico da placa de controle (Figura 14). O esquemático descreve todos os componentes utilizados na placa bem como suas conexões elétricas.

Figura 14. Esquemático da placa de controle



Fonte: Próprio autor

Na parte superior da Figura 14 está o circuito de alimentação da placa. A bateria é conectada ao conector P7 e logo em seguida a tensão da bateria é regulada em 5V com o uso do regulador linear LM7805. A regulagem em 5V é necessária para alimentar os sensores ultrassônicos.

Capacitores eletrolíticos são utilizados para atenuar possíveis oscilações de tensão que normalmente ocorrem devido a picos de corrente ou flutuações da fonte. Já os capacitores cerâmicos de 100nF são utilizados como filtros passa baixa para atenuar eventuais ruídos. Outro regulador linear (TLV2217-33KTPR) é utilizado para obter os 3,3V necessários para alimentar o microcontrolador.

O conector P1 é utilizado para gravação do firmware no microcontrolador. Este conector utiliza o padrão *J-Tag* de 20 pinos utilizado pela *ARM*. Este padrão foi escolhido devido a compatibilidade com o gravador adquirido (Figura 15).

Figura 15. Gravador ARM padrão *J-Tag*

Fonte: Disponível em https://www.segger.com/admin/uploads/imageBox/J-Link_BASE.jpg, acessado em outubro 2014.

4.3.1.4 Lista de materiais - BOM

A partir do esquemático foi possível gerar a BOM (Lista de materiais, do inglês - *Bill Of Materials*) da placa. Como o nome sugere, a BOM contém todos os componentes utilizados no diagrama elétrico. Ela é normalmente utilizada no processo de cotação e compra dos componentes, pois reúne de forma organizada todos os componentes utilizados com uma breve descrição de suas características.

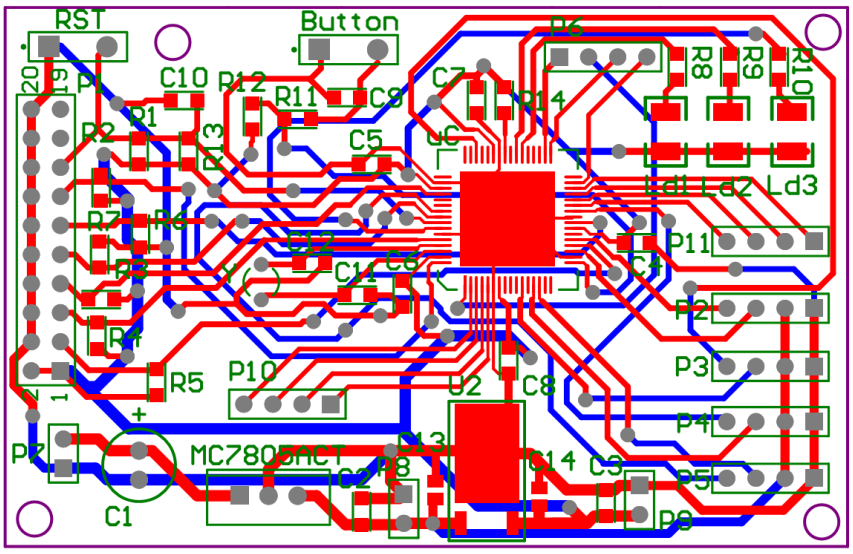
A BOM gerada com o auxílio do *Altium Designer* é apresentada no Apêndice A.Neste trabalho ela foi utilizada na seleção dos componentes, antes da montagem da placa de controle.

4.3.1.5 *Layout*

A disposição dos componentes foi escolhida para facilitar a conexão com os sensores ultrassônicos de tal forma que cada sensor possui um conector de 4 posições com os pinos de sinal e alimentação. De maneira similar, os pinos de acionamento dos motores e demais pinos também foram agrupados em conectores de 4 posições. O *layout* finalizado é ilustrado na Figura 17.

Pinos adicionais do microcontrolador foram previstos no esquemático e *layout* para eventuais necessidades futuras. Quatro furos foram incluídos para fixação da placa.

Figura 16. *Layout* da placa de controle

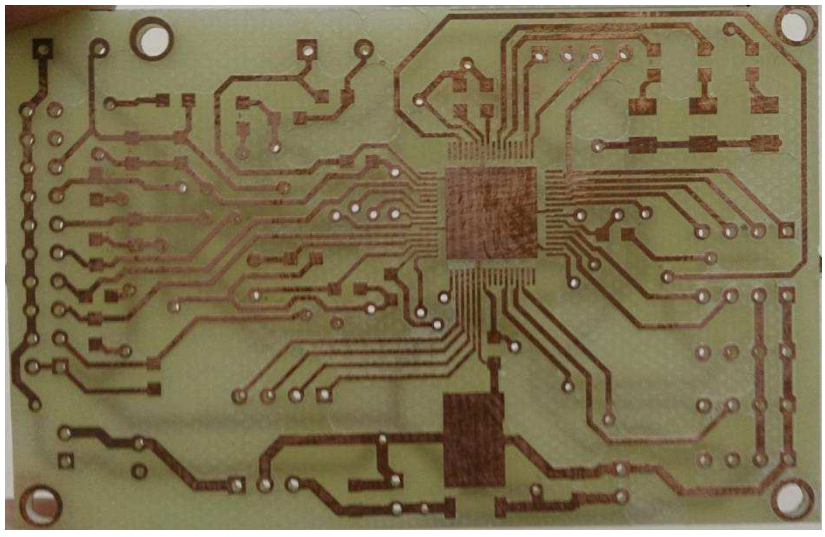


Fonte: Próprio autor

A partir do *layout* é possível gerar os arquivos *Gerber* necessários para confeccionar a placa em uma prototipadora de PCBs. Os arquivos *Gerber* contém todas as informações presentes no *layout* como: notas de fabricação, dimensões, informações mecânicas, furos, *pads*, trilhas, planos e *silkscreen*.

A Figura 18 ilustra o resultado obtido após a prototipagem.

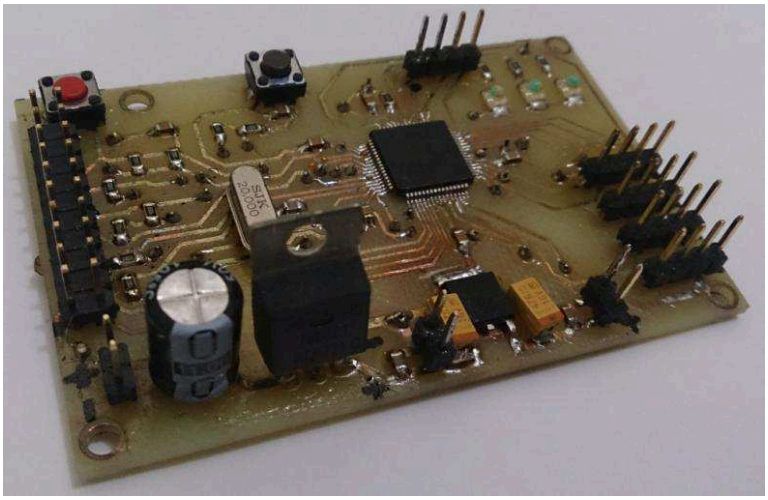
Figura 17. Placa de controle antes da montagem



Fonte: Próprio autor

A montagem da placa deu-se no laboratório de eletrônica da UFSC Campus Joinville, aonde todos os componentes utilizados puderam ser encontrados. A Figura 19 apresenta o resultado após a montagem da placa de controle.

Figura 18. Placa de controle após a montagem



Fonte: Próprio autor

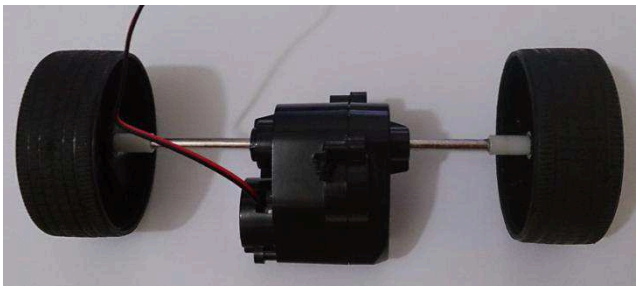
4.3.2 Mecânica

A parte mecânica do protótipo consiste do chassis, sistema de direção, sistema de tração e lataria. Toda a lógica de controle original do veículo foi removida para suportar o novo conjunto desenvolvido, da configuração original só restaram os dois motores e a bateria.

4.3.2.1 Sistema motriz

Conforme mencionado na seção 3.3, foi necessário trocar o motor de tração pois o mesmo não possui torque suficiente quando acionado em baixas rotações. O sistema de tração original é ilustrado na Figura 20.

Figura 19. Sistema original de tração



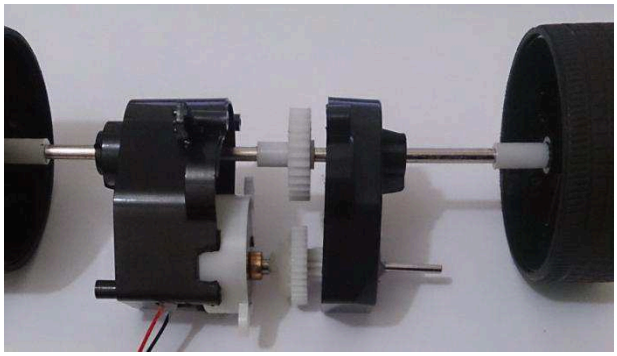
Fonte: Próprio autor

Não foi possível encontrar um motor de mesmas dimensões com rotação menor que permitisse uma substituição direta. Desta forma fez-se necessário modificar o sistema de transmissão mecânico do conjunto de tração.

Para fixar o novo motor foi utilizado o mesmo espaço aonde estava posicionado o motor original. Porém para ajustar o motor de forma adequada foi necessário cortar a peça até a posição ideal.

A transmissão do torque do motor para o eixo foi inicialmente planejada considerando o mesmo sistema de engrenagem da caixa original, entretanto não foi possível realizar o acoplamento devido a alteração feita na caixa que impossibilitou de utilizar a engrenagem intermediária. A Figura 21 mostra o resultado da nova montagem do sistema de tração.

Figura 20. Sistema de tração após modificação



Fonte: Próprio autor

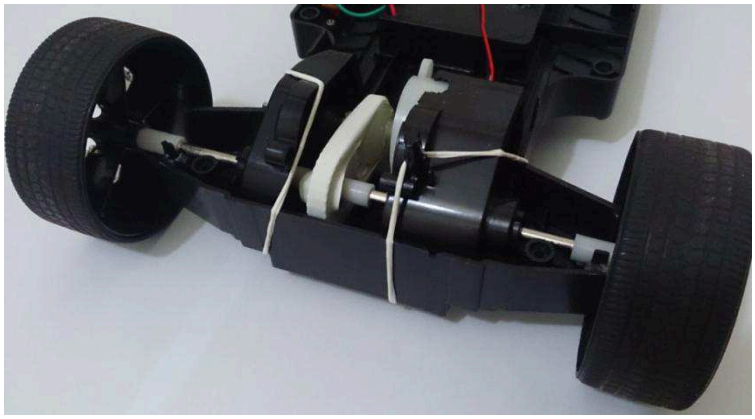
Como não foi possível utilizar o sistema original, partiu-se para um acoplamento considerando o uso de uma correia. Uma correia dentada exige uma relação perfeita entre seus dentes, geometria das engrenagens e comprimento, tal correia não foi encontrada. A solução utilizada foi um anel de borracha.

De modo a garantir o espaçamento constante entre as engrenagens e, assim, evitar que a redução da capacidade de transmissão da correia foi adicionado um pino metálico na outra extremidade da caixa conforme pode ser observado na Figura 21. O pino encaixa perfeitamente no furo central da engrenagem do motor e assim garante um apoio em dois pontos.

Como toda a estrutura do sistema de tração foi alterada, não é mais possível utilizar o sistema de fixação original no chassis do veículo. Para contornar este problema foi necessário fazer alguns cortes no chassis de modo a garantir o espaço necessário do novo conjunto motor.

Para fixar o novo conjunto foi elaborado uma estratégia que permite a remoção do mesmo caso seja necessário algum ajuste ou manutenção. A solução adotada foi utilizar dois elásticos conforme Figura 22, onde é ilustrado também a solução da correia de transmissão.

Figura 21. Sistema de tração fixado ao chassis



Fonte: Próprio autor

4.4 FIRMWARE

O firmware foi implementado em linguagem de programação C utilizando o firmware *Keil uVision4* como ambiente de desenvolvimento. O processo de desenvolvimento pode ser dividido em 4 etapas: Modelagem, Configuração do Microcontrolador, Periféricos e Aplicação.

Nesta seção são apresentadas as etapas em detalhes e exemplificando com trechos do firmware desenvolvido.

4.4.1 Modelagem

A modelagem utilizada para o desenvolvimento do firmware é baseada na Linguagem de Modelagem Unificada (UML, do inglês *Unified Modeling Language*). De acordo com Guedes (2011) a UML é uma linguagem de modelagem de propósito geral que foi adotada como linguagem-padrão para desenvolvedores de firmware. O objetivo da linguagem é auxiliar o desenvolvedor de firmware a definir e representar as características do sistema a ser desenvolvido.

4.4.1.1 Requisitos de firmware

O levantamento dos requisitos consiste, normalmente, na primeira fase do desenvolvimento do firmware. É nesta fase que deseja-se determinar tudo que o firmware deve ser capaz de realizar depois de finalizado (GUEDES, 2011).

Os requisitos foram levantados com base no processo real de estacionamento em vagas paralelas. O detalhamento dos requisitos foi dividido em quatro etapas: Início, Procura Vaga, Verifica Vaga e Manobra.

1. INÍCIO: Ao pressionar o botão *start/stop* o sistema deve iniciar o estacionamento.
   Restrições:
   a. Distância lateral menor que 5cm
   b. Veículo parado ($v = 0$)
   c. Não estar a frente do veículo ao lado

2. PROCURA VAGA: O veículo desloca-se para frente com velocidade constante enquanto o sensor lateral identifica a vaga.
   Restrições:
   a. O sensor dianteiro não indica presença de obstáculos
   b. Deve parar na posição apropriada ao final da manobra
   c. Motorista não interrompe manobra

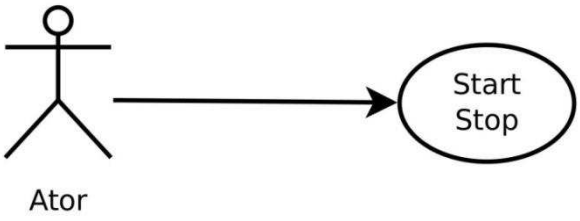
3. VERIFICA VAGA: Determina o tamanho da vaga, compara com valores pré-determinados de acordo com as dimensões do veículo. Se o tamanho não for suficiente aborta a manobra, caso contrário inicia manobra de estacionamento.
   Restrições:
   a. Veículo parado ($v = 0$)
   b. Tamanho da vaga medido corretamente
   c. Motorista não interrompe manobra

4. MANOBRA: O sistema controla a direção, velocidade e sentido do veículo. Deve concluir a operação sem acidentes, determinar o ponto do inflexão e ponto final da manobra. Deixar o carro no meio da vaga
   Restrições:
   a. Sensores não indicam obstáculos
   b. Motorista não interrompe manobra

4.4.1.2 Diagrama de casos de uso

A partir dos requisitos é possível desenhar o diagrama de casos de uso que representa a interação entre os atores e as funções desempenhadas pelo sistema. A Figura 23 mostra o diagrama de casos de uso em que só há um ator e este apenas atua sobre o sistema através do botão *start/stop* pois o veículo é autônomo. O diagrama gerado é ilustrado na Figura 23.

Figura 22. Diagrama Casos de Uso

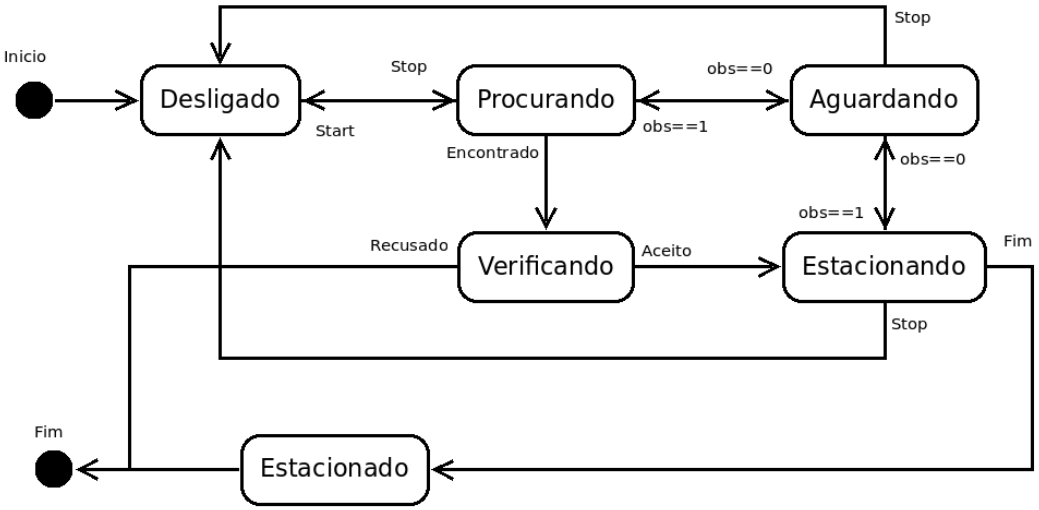


Fonte: Próprio autor

4.4.1.3 Diagrama de máquina de estados

Este diagrama é obtido a partir do diagrama de casos de uso e do funcionamento do sistema através das restrições previamente definidas. O diagrama apresenta todos os possíveis estados do sistema bem como as possíveis formas de transição entre os estados. A Figura 24 mostra o diagrama obtido.

Figura 23. Máquina de Estados



Fonte: Próprio autor

Com o auxilio da modelagem do sistema fica muito mais fácil o desenvolvimento do firmware pois o comportamento que o sistema deve respeitar já está bem definido e de uma forma intuitiva similar à lógica de programação.

4.4.2 Configuração do microcontrolador

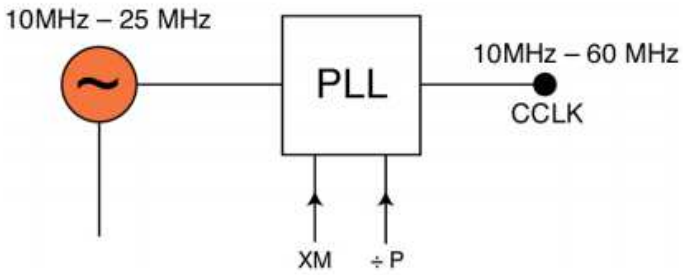
Com base na modelagem do sistema o próximo passo no desenvolvimento do firmware é a configuração do microcontrolador. É nesta etapa que são configurados os modos de operação, funções dos pinos e demais ajustes do microcontrolador para que este opere da maneira desejada.

Algumas configurações são feitas em um arquivo Startup.s que varia de acordo com a família do microcontrolador utilizado. O firmware Keil automaticamente importa o arquivo correspondente ao microcontrolador quando um novo projeto é iniciado. Neste arquivo são definidos os vetores de interrupção, são configuradas as frequências de operação do núcleo e dos periféricos, reserva e inicializa as pilhas para todos os modos de operação, entre outras funções.

Para esta aplicação apenas foi necessário configurar as frequências do núcleo e periféricos através do PLL (*Phase Locked Loop*). Esta configuração leva em conta a

frequência do cristal externo utilizado que é de 20MHz. A frequência de trabalho do núcleo, de acordo com o *datasheet*, é limitada em 60MHz. Para obter o máximo desempenho foi definido que o núcleo deve operar em 60MHz enquanto os periféricos devem operar em 30MHz. O PLL funciona de acordo com dois controles: M (Multiplicador) e P (divisor). Ajustando os valores desses dois parâmetros é possível definir a frequência de operação do núcleo do microcontrolador. A Figura 25 ilustra o diagrama do PLL.

Figura 24. Representação do PLL



Fonte: MARTIN, 2005, Pg. 60.

É possível determinar o valor do multiplicador M dividindo o valor desejado de frequência (CCLK) pela frequência do cristal (Osc), como segue:

$$M = \frac{CCLK}{Osc} \tag{11}$$

$$M = \frac{60}{20}$$

$$M = 3$$

Para determinar o valor do divisor P é necessário levar em conta o oscilador controlado por corrente (CCO do inglês, *Current-Controlled Oscillator*) que trabalha dentro da faixa 156MHz a 320MHz. O cálculo é dado pela equação 12:

$$FCCO = CCLK \times 2 \times P \tag{12}$$

$$156 < FCCO < 320 = 60 \times 2 \times P$$

O valor de P deve ser um número inteiro, logo é possível verificar que o único valor que atende a condição da equação 12 é P=2

$$156 < FCCO < 320 = 240$$

Uma vez que os parâmetros M e P estão definidos apropriadamente, o próximo passo foi alterar o registrador PLLCFG configurando os bits de acordo com a Figura 26.

Figura 25. Configuração do registrador PLLCFG

Table 27. PLL Divider values

PSEL Bits (PLLCFG bits [6:5])	Value of P
00	1
01	2
10	4
11	8

Table 28. PLL Multiplier values

MSEL Bits (PLLCFG bits [4:0])	Value of M
00000	1
00001	2
00010	3
00011	4
...	...
11110	31
11111	32

Fonte: User manual, pg 39.

Conforme mencionado anteriormente, esse registrador é alterado no arquivo "Startup.s", onde o valor deve ser escrito no registrador em hexadecimal.

```
PLL_SETUP       EQU         1
PLLCFG_Val      EQU         0x00000022
```

Para determinar a frequência de operação dos periféricos é necessário configurar o divisor VPB. Este divisor é utilizado para reduzir a frequência após o PLL por um fator de 1, 2 ou 4. O registrador que deve ser alterado é o VPBDIV que foi alterado para dividir o CCLK por 2.

```
VPBDIV_SETUP    EQU         1
VPBDIV_Val      EQU         0x00000002
```

Dentro do arquivo principal, "main.c", foram criadas funções para configuração e inicialização de todos pinos e periféricos utilizados. Todas essas funções são detalhadas na próxima seção.

4.4.3 Periféricos

Com o microcontrolador configurado conforme desejado é possível iniciar o desenvolvimento das funções e definições necessárias para o uso do *driver* e dos sensores.

Para facilitar o uso desses componentes foi criado um arquivo "Hardware.H" contendo todas as definições de hardware. Dessa forma é possível organizar todas as definições dos

pinos utilizados do microcontrolador em um arquivo separado, mantendo uma estrutura de código organizada.

O uso de definições possibilita uma rápida visualização e entendimento do firmware uma vez que permite atribuir um nome ao valor do pino. Um trecho do código aonde são definidos os pinos que estão conectados aos 3 leds presentes na placa é apresentado a seguir.

```
#define  LED_GREEN_PIN        0x00001000
#define  LED_BLUE_PIN         0x00002000
#define LED_RED_PIN           0x00800000
```

Da mesma forma foram utilizadas mais definições para diversas constantes utilizadas no decorrer do firmware dentro do arquivo "main.c".

É preciso definir a função de cada pino quando esta não for a função padrão após o reset do microcontrolador, dessa forma uma função é utilizada para configurar todos os pinos. A rotina utilizada pela função é apresentada a seguir.

```
IODIR0 = 0x38A03380;      //Pinos são configurados como saída
PINSEL0 = 0x000A80C5;     //Seleciona as funções alternativas de cada pino
PINSEL1= 0x00000701;      //Seleciona as funções alternativas de cada pino
```

Os dois *timers* disponíveis são utilizados pela aplicação desenvolvida. O timer 0 é utilizado para realizar a medição de distância através dos sensores ultrassônicos enquanto o timer 1 é utilizado para medir a distância da vaga e manobra através da velocidade constante do veículo.

O timer 0 é configurado para realizar um incremento na contagem do tempo a cada 1µs, esse tempo foi escolhido pois permite uma boa resolução na medição do tempo de eco além de facilitar os cálculos. O ajuste do timer 0 é feito configurando o *prescaler* através do registrador T0PR. O valor utilizado como *prescaler* é obtido pela multiplicação do tempo de incremento desejado pela frequência de operação do timer, conforme equação 13:

$$T0PR = Tempo\_incremento * PCLK \quad (13)$$

$$T0PR = 1\,s * 30MHz$$

$$T0PR = 30$$

De forma similar é possível configurar o timer 1 para obter um incremento a cada 1ms, tempo suficiente para realizar a medição da distância da vaga com ótima resolução. O ajuste é feito através do registrador T1PR.

$$T1PR = Tempo\_incremento * PCLK \quad (14)$$
$$T1PR = 1ms * 30MHz$$
$$T1PR = 30000$$

A rotina de firmware utilizada para configuração de ambos os *timers* é apresentada a seguir. Os valores calculados foram convertidos para hexadecimal e foi necessário substituir 1 do *prescaler* pois a contagem é iniciada em 0.

```
T0CTCR = 0x00000000;     //Configura o timer0 como contador
T0PR= 0x0000001D;        //Configura prescaler do timer0 para incremento em 1us
T0TCR= 0x00000002;       //Reinicia o timer e o prescaler
T1CTCR = 0x00000000;     //Configura o timer1 como contador
T1PR= 0x0000752F;        //Configura prescaler do timer0 para incremento em 1ms
T1TCR= 0x00000002;       //Reinicia o timer e o prescaler
```

4.4.3.1 Driver L9110

Para poder utilizar o módulo L9110 com controle de velocidade independente nos dois motores é necessário utilizar 4 sinais PWM, sendo 2 sinais para cada motor.

Os sinais de PWM foram configurados para operar com uma frequência de aproximadamente 58kHz. A função que configura e habilita as 4 saídas é ilustrada abaixo.

```
PWMPR = 0x00000001;      //Carrega o prescaler
PWMPCR = 0x00007400;     //Configura todos os sinais de PWM como SINGLE EDGE
PWMMCR = 0x00000003;     //Reinicia contador quando PWMTC iguala a PWMMR0.
PWMMR0 = PERIOD;         //Define a frequência do PWM
PWMMR2 = 0x00000000;     //Valores são iniciados em 0 e alterados na aplicação
PWMMR4 = 0x00000000;
PWMMR5 = 0x00000000;
PWMMR6 = 0x00000000;
PWMLER = 0x00000075;     //Habilita os comparadores respectivos a cada sinal
PWMTCR = 0x00000002;     //Reinicia contador
PWMTCR = 0x00000009;     //Habilita PWM
```

4.4.3.2 Sensor ultrassônico

Para conseguir realizar a medição utilizando os sensores foram utilizadas interrupções externas. Dessa forma foi criado uma função para configurar o tipo de interrupção e definir as funções de tratamento de cada interrupção, conforme apresentado a seguir.

```
EXTMODE = 0x0F;            //Configura interrupções para detectar pela borda
EXTPOLAR = 0;              //Define a borda de descida
//----Sensor1----
//        VICVectCntl1  =  0x20 | 14;          //Canal usando EINT0
//        VICVectAddr1  =  (unsigned) IRQ1_interrupt;      //Função  de tratamento
//----Sensor2----
          VICVectCntl2  =  0x20 | 15;          // Canal usando EINT1
          VICVectAddr2  =  (unsigned) IRQ2_interrupt;      //Função  de tratamento
//----Sensor3----
//        VICVectCntl3  =  0x20 | 17;          // Canal usando EINT3
//        VICVectAddr3  =  (unsigned) IRQ3_interrupt;      //Função  de tratamento
VICIntEnable = 0x0002C000;                   //Habilita as três interrupções externas
```

A função que é chamada quando a interrupção ocorre, ou seja, quando o sensor terminou de receber o pulso de eco, é apresentada a seguir. Esta rotina apenas guarda o valor do timer na variável T3 e limpa a *flag* de interrupção.

```
T3 = T0TC;                 //Armazena o valor do timer0 na variável T3
EXTINT = 0x02;             //Limpa a flag da interrupção EINT2
VICVectAddr = 0x00;        //Termina a interrupção
```

4.4.4 Aplicação

Em se tratando do firmware, a aplicação é a camada de mais alto nível, são as rotinas de firmware que realmente implementam a funcionalidade do código. A aplicação faz uso das rotinas de utilização dos sensores e atuadores que normalmente já estão desenvolvidas para tomar alguma decisão e assim executar todas suas funcionalidades.

Neste trabalho, a aplicação contém toda a lógica responsável pela manobra de estacionamento automático em vagas paralelas. O modelo de geração de trajetória é implementado em total sincronia com a leitura dos sensores e os comandos enviados aos motores.

A aplicação descrita nos próximos parágrafos foi implementada com base na máquina de estados (Figura 24), porém, com algumas modificações. O estado aguardando não foi implementado nesta versão do firmware.

Após todas as rotinas iniciais de configuração o firmware entra em um loop infinito onde a aplicação é implementada. Dentro desse loop uma estrutura do tipo *Switch-Case* é utilizada para verificar qual estado o sistema se encontra.

O estado inicial é definido como *desligado*. Neste estado o sistema apenas altera o estado atual para *Procurando*. Uma rotina poderia ser implementada para aguardar o motorista pressionar o botão de iniciar, porém foi considerado que ao chegar neste estado o estacionamento deveria ser iniciado.

```
ESTADO_ANTERIOR = ESTADO_ATUAL
ESTADO_ATUAL = PROCURANDO
```

O firmware passa a verificar qual rotina será a próxima a ser executada e entra no estado *Procurando*. Então, uma função é chamada para iniciar o processo de busca pela vaga.

A função faz a leitura do sensor lateral, armazena a distância medida e compara com um valor limite para verificar se há um carro ou obstáculo ao lado e com base no resultado toma alguma ação. Enquanto as leitura são feitas, o veículo está seguindo em linha reta. Esse rotina se repete até que a vaga termina quando o estado é alterado. A rotina simplificada da função é apresentada a seguir.

```
LÊ_SENSOR_LATERAL
CALCULA_DISTÂNCIA_OBJETO
ACIONA_MOTOR_PARA_FRENTE
SE (DISTANCIA >= DISTANCIA_MINIMA)              //Sem carro ao lado
        SE (CONTADOR == 0)                      //Inicio da vaga
                LIGA_TIMER
                INCREMENTA_CONTADOR
        SENÃO                                   //Durante a vaga
                CONTINUA
SENÃO                                           //Carro ao lado
        SE (CONTADOR == 1)                      //Já passou pela vaga
                ARMAZENA_VALOR_TIMER
                AJUSTA_POSIÇÃO_CARRO
                DESLIGA_MOTOR
                ESTADO_ANTERIOR = ESTADO_ATUAL
                ESTADO_ATUAL = VERIFICANDO
        SENÃO                                   //Antes da vaga
                CONTINUA
```

Neste momento o sistema identificou a vaga e capturou o tempo que levou para o veículo passar pela vaga. O próximo estado é *Verificando,* onde o sistema irá calcular o tamanho da vaga com base na velocidade constante do veículo. Neste estado também é feito a

comparação do tamanho da vaga com o tamanho mínimo necessário para poder estacionar o veículo.

```
CALCULA_TAMANHO_VAGA
SE (TAMANHO > TAMANHO_MINIMO)                    //Vaga é suficiente
        ESTADO_ANTERIOR = ESTADO_ATUAL
        ESTADO_ATUAL = ESTACIONANDO_1
SENÃO                                            //Vaga não é suficiente
        ESTADO_ANTERIOR = ESTADO_ATUAL
        ESTADO_ATUAL = CANCELAR
```

Caso o tamanho da vaga seja menor do que o necessário o sistema irá indicar o estado através dos leds presentes na placa e não irá fazer mais nada, indicando ao motorista que assuma o controle do carro. Caso a vaga seja suficiente, o sistema irá iniciar a manobra que foi dividida em 4 etapas.

A primeira etapa da manobra aciona o motor na direção reversa e aciona a direção para a direita e liga o timer. Enquanto o veículo se desloca, a rotina fica verificando a distância percorrida pelo veículo para detectar o ponto em que o veículo deve parar e mudar sua direção.

```
SE (CONTADOR == 0)
        LIGA_TIMER
        INCREMENTA_CONTADOR

ACIONA_MOTOR_PARA_TRAS
ACIONA_DIREÇÃO_PARA_DIREITA
DELAY
CALCULA_DISTÂNCIA_PERCORRIDA

SE (DISTÂNCIA >= DISTÂNCIA_INFLEXÃO)
        DESLIGA_MOTOR
        ESTADO_ANTERIOR = ESTADO_ATUAL
        ESTADO_ATUAL = ESTACIONANDO_2
```

O estado *Estacionando2* apenas inverte a direção do veículo e muda o estado para *Estacionando3*. A rotina presente no estado *Estacionando3* é muito similar à rotina *Estacionando2*. A última parte do estado *Estacionando* consiste apenas de um ajuste do veículo para que este fique em uma posição mais centralizada dentro da vaga. Ao final desse estado o sistema considera-se a tarefa de estacionamento concluída e altera-se o estado para

Estacionado. Este último estado indica ao motorista que a manobra foi concluída com sucesso.

O código completo é apresentado no Apêndice B.

5 RESULTADOS E DISCUSSÕES

Os testes realizados validam o sistema desenvolvido verificando seu funcionamento e desempenho mediante diferentes condições de variação.

5.1 TESTE1: TESTE FUNCIONAL

O primeiro teste foi realizado utilizando o firmware com os parâmetros calculados na seção 4.2. O tamanho da vaga é ajustado para 76cm, ligeiramente maior que o valor mínimo calculado, e o veículo é posicionado à 5cm de distância do obstáculo ao lado.

O objetivo deste teste é verificar o comportamento do sistema como um todo, se o sistema está executando todos os estados corretamente e se todos os subsistemas estão funcionando em sintonia.

O resultado do teste pode ser visualizado na Figura 27.

Figura 26. Resultado com os parâmetros calculados

Fonte: Próprio autor

5.2 TESTE2: VERIFICAR PRECISÃO DE MEDIÇÃO E MANOBRA

Com base no resultado do teste 1 foi verificado que, com alguns ajustes no firmware, seria possível melhorar o desempenho do sistema permitindo que este obtenha resultados melhores.

A tabela 5 mostra os parâmetros que foram alterados em comparação com os valores originais.

Tabela 5. Parâmetros modificados

PARÂMETRO	VALOR ORIGINAL	VALOR MODIFICADO
Tempo de ajuste [s]	4.23	4.1
$VAGA_{min}$ [cm]	74	70
L1 [cm]	43,24	50
L2 [cm]	43,24	53

Fonte: Próprio autor

Este teste tem como objetivo quantificar a capacidade do sistema de identificar, medir e estacionar em uma vaga paralela. Para isto são realizadas medições considerando três casos com relação a distância da vaga: 2cm menor, tamanho calculado e 2cm maior.

Para minimizar ruídos que possam mascarar os resultados do teste são executadas 3 repetições para cada caso, mantendo as mesmas condições. Os dados obtidos após a realização dos testes são apresentados na Tabela 6.

Tabela 6. Resultados: Variação do tamanho da vaga

VAGA 68CM	**IDENTIFICAÇÃO DA VAGA**	**RESULTADO DA MANOBRA**	**DISTÂNCIA DA CALÇADA [CM]**
1	Ok	Abortado	-
2	Ok	Abortado	-
3	Ok	Abortado	-
Vaga 70cm	**Identificação da vaga**	**Resultado da manobra**	**Distância da calçada [cm]**
1	Ok	Estacionado	3
2	Ok	Estacionado	3
3	Ok	Estacionado	4
Vaga 72cm	**Identificação da vaga**	**Resultado da manobra**	**Distância da calçada [cm]**
1	Ok	Estacionado	3.5
2	Ok	Estacionado	2
3	Ok	Estacionado	2.5

Fonte: Próprio autor

O resultado final obtido com os valores otimizados pode ser visualizado na Figura 28.

Figura 27. Resultado final - Parâmetros otimizados

Fonte: Próprio autor

6 CONCLUSÕES

6.1 CONCLUSÃO

Através do desenvolvimento deste trabalho foi possível adquirir experiência no desenvolvimento de um projeto de sistema embarcado que integra firmware e hardware. A placa de controle desenvolvida bem como o firmware apresentaram resultados muito bons, cumprindo todos os requisitos especificados.

Analisando os resultados obtidos com o protótipo é possível verificar que o sistema proposto é capaz de identificar uma vaga paralela e determinar seu comprimento com precisão menor que 2cm. Considerando o método utilizado para realizar este cálculo e comparando este valor com o tamanho da vaga, pode-se concluir que é um bom resultado. O sistema é capaz de analisar o tamanho da vaga, decidir se o tamanho da vaga é suficiente para realizar a manobra ou não e, caso a vaga tenha tamanho suficiente, realizar a manobra de forma automática.

O menor tamanho de vaga em que o veículo foi capaz de estacionar possui 70cm de comprimento, ou seja, é necessário uma vaga 35% maior que o comprimento do protótipo para que o sistema consiga estacionar. A maioria dos modelos comerciais é capaz de estacionar em vagas com tamanho 20% maior. O fato do sistema desenvolvido necessitar de um espaço maior está relacionado às características físicas do protótipo, principalmente devido ao pequeno ângulo de esterçamento que resulta em um raio de curvatura grande.

Embora o sistema tenha sido modelado considerando o uso de três sensores ultrassônicos, que permitiria detectar obstáculos dinâmicos durante a manobra, o sistema foi implementado utilizando apenas um sensor.

O sistema de transmissão de torque desenvolvido para tracionar o veículo não apresenta uma boa confiabilidade, saindo do lugar e ocasionando perda de tração. Durante a fase de testes foi necessário desmontar o protótipo algumas vezes para encaixar a correia na posição desejada.

6.2 TRABALHOS FUTUROS

As sugestões propostas são no sentido de aperfeiçoar o sistema desenvolvido a fim de obter resultados ainda melhores. O firmware pode ser modificado, sem grandes esforços, para incluir os outros dois sensores ultrassônicos previstos na modelagem do firmware e na placa de controle. A mudança permitiria detectar obstáculos dinâmicos que eventualmente podem

surgir durante a manobra, resultando em um sistema mais seguro e próximo dos modelos comerciais.

Outra melhoria interessante seria a inclusão de um módulo de comunicação sem fio *wifi* que poderia ser usado para controlar o protótipo ou ainda realizar a telemetria do mesmo. A placa já está preparada para ligar um módulo *wifi*.

O sistema também pode ser adaptado e testado em um protótipo maior de modo a verificar o seu desempenho em uma condição mais realista. A possibilidade de testar o sistema em um veículo com características mecânicas similares aos veículos comerciais seria de grande importância na verificação do algoritmo de geração de trajetória.

REFERÊNCIAS

ACHORN, George. **IN DETAIL: THE NEW PASSAT – GENERATION 8, TECHNOLOGY PREVIEW**. Disponível em: <http://www.vwvortex.com/news/volkswagen-news/detail-new-passat-generation-8-2/?COLLCC=2281205935>. Acesso em: 15 nov. 2014.

BELL, Lyndon. **What is active park assist?** Disponível em: <http://www.autobytel.com/car-ownership/advice/what-is-active-park-assist-113908/>. Acesso em: 27 out. 2014.

CASTRO, Juliana. **Frota de veículos mais que dobra em 10 anos.** Disponível em: <http://oglobo.globo.com/brasil/frota-de-veiculos-mais-que-dobra-em-10-anos-10934030>. Acesso em: 16 dez. 2013.

COOPER, Sean. **Hands-on with BMW's self-parking i3.** Disponível em: <http://www.engadget.com/2014/01/09/bmw-i3-self-parking/>. Acesso em: 15 nov. 2014.

FAIRUS, M.A. ; SALIM, S.N.S. ; JAMALUDIN, I.W. ; KAMARUDIN, M.N.**Development of an automatic parallel parking system for nonholonomic mobile robot**. Electrical, Control and Computer Engineering (INECCE), 2011 International Conference.

FORD. **Active Parking Assist**. Disponível em: <http://corporate.ford.com/doc/APA_Lexus.pdf>. Acesso em: 04 nov. 2014.

FREY, Marc. **Ford prepara sistema que estaciona o carro sozinho.** Disponível em: <http://carros.ig.com.br/noticias/ford+prepara+sistema+que+estaciona+o+carro+sozinho/6792.html>. Acesso em: 28 nov. 2013.

GUEDES, Gilleanes T. A.. **UML: Uma Abordagem Prática**. 2. ed. São Paulo: Novatec, 2011.

GUPTA, Ankitet al. **Autonomous Parallel Parking System for Ackerman steering four wheelers**. Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference.

GRABIANOWSKI, Ed. **How Self-parking Cars Work.** Disponível em: <http://auto.howstuffworks.com/car-driving-safety/safety-regulatory-devices/self-parking-car2.htm>. Acesso em: 28 nov. 2013.

HC-SR04, Datasheet. **Ultrasonic Ranging Module HC - SR04.** Disponível em: <http://users.ece.utexas.edu/~valvano/Datasheets/HCSR04b.pdf>. Acesso em: 26 jul. 2014.

JEONG, S. H. et al. **Low cost design of parallel parking assist system based on an ultrasonic sensor**. International Journal of Automotive Technology, v. 11, n. 3, p. 409-416, 2010.

LAUKKONEN, Jeremy. **Automatic Parking Systems**. Disponível em: <http://cartech.about.com/od/Safety/a/Automatic-Parking-Systems.htm>. Acesso em: 27 out. 2014.

LISZEWSKI, Andrew. **O carro da Volvo que estaciona sozinho.** Disponível em: <http://gizmodo.uol.com.br/video-carro-volvo-estaciona-sozinho/>. Acesso em: 27 nov. 2013.

MARTIN, Trevor. **TheInsider's Guide to the Philips ARM7-Based Microcontrollers**. U.k: HitexUk, 2005. Disponível em: <http://docweb.khk.be/Patrick%20Colleman/ARM7/lpc-ARM-book_srn.pdf >. Acessoem: 27 jul. 2014.

OLIVEIRA, Marcello; VENTURA, Thiago. **Carro que estaciona sozinho ajuda quem tem dificuldades na baliza?** Disponível em: <http://estadodeminas.vrum.com.br/app/noticia/noticias/2013/11/27/interna_noticias,48723/carro-que-estaciona-sozinho-ajuda-de-verdade-quem-tem-dificuldades-em.shtml>. Acesso em: 15 dez. 2013.

POPULAR SCIENCE. New York: **Popular Science** Publishing Co., v. 125, n. 3, 1934. Mensal. Disponível em: <http://books.google.com.br/books?id=HCgDAAAAMBAJ&pg=PA58&dq=Popular+Science+1931+plane&hl=en&ei=dvsKTffpK8ShnAfqofGVDg&sa=X&oi=book_result&ct=result&redir_esc=y#v=onepage&q&f=true>. Acesso em: 27 out. 2014.

SABER ELETRÔNICA. São Paulo: Editora saber - Ano 42 - Nº 400 - Maio/2006. Disponível em <http://www.sabereletronica.com.br/artigos-2/1753-sensores-ultra-snicos> Acessado em 24 nov. 2013.

SIMÕES, Karina. **IG avaliou o Park Assist, sistema que estaciona o carro sozinho**. 2014. Disponível em: <http://carros.ig.com.br/especiais/ig+avaliou+o+park+assist+sistema+que+estaciona+o+carro+sozinho/7403.html>. Acesso em: 13 set. 2014.

SOUSA, Daniel Rodrigues de. **Microcontroladores ARM7 (Philips, Família LPC213x): o poder dos 32 bits: teoria e prática**. São Paulo: Érica, 2006.

TECHNOLOGIES, Cytron. **ProductUser's Manual – HC-SR04 Ultrasonic Sensor.** Disponível em: <https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit?pli=1>. Acesso em: 26 jul. 2014.

VOLVO. **Assistente de Estacionamento**. Disponível em: <http://www.volvocars.com/br/all-cars/volvo-v40/Pages/top-accessories.aspx>. Acesso em: 28 out. 2014.

VÓRIA, Frederico Saraiva. **Sistema de Estacionamento Automático para um Veículo Autônomo.** 2010. 75 f. Monografia (Graduação) - UFMG, Belo Horizonte, 2010.

APÊNDICE A – LISTA DE MATERIAIS (BOM)

Bill of Materials

ARM

Source Data From: arm_v3.PrjPcb
Project: arm_v3.PrjPcb
Variant: None

Creation Date: 13/10/2014 21:37:10
Print Date: 41925 41925,90086

Footprint	Comment	LibRef	Designator	Description	Quantity
SPST-2	SW-PB	SW-PB	Button, RST	Switch	2
cap100	Cap Pol1	Cap Pol1	C1	Polarized Capacitor (Radial)	1
0805	SMD	Cap, Cap, Res1 Res1, Res1, Res1, Res1, Res1, Res1, Res1, Res1, Res1, Res1, Res1, Res1	C2, C3, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14	Capacitor, Capacitor, Resistor, Resistor, Resistor, Resistor, Resistor, Resistor, Resistor, Resistor, Resistor, Resistor, Resistor, Resistor	16
0805	Cap	Cap	C4, C5, C6, C7, C8, C9, C10, C11, C12	Capacitor	9
C0805	Cap Pol3	Cap Pol3	C13, C14	Polarized Capacitor (Surface Mount)	2
1210	LED SMD	LED0	Ld1, Ld2, Ld3	Typical INFRARED GaAs LED	3
D2PAK_M	Volt Reg	Volt Reg	MC7805ACT	Voltage Regulator	1
HDR2X10	HEADER 10X2	HEADER 10X2	P1	Header, 10-Pin, Dual row	1
HDR1X4	Header 4	Header 4	P2, P3, P4, P5, P6, P10, P11	Header, 4-Pin	7
HDR1X2	Header 2	Header 2	P7, P8, P9	Header, 2-Pin	3
KTP	TLV2217-33KTPR	TLV2217-33KTPR	U2		1
ARM7	LPC2138	LPC2138	uC		1
R38	XTAL	XTAL	Y	Crystal Oscillator	1
					48

Approved **Notes**

APÊNDICEB – FIRMWARE DESENVOLVIDO

```
1  /*****************************************************************
2  *
3  * hardware.h : Hardware Definitions
4  *
5  * By: Anderson Augusto Heinz
6  *
7  *****************************************************************/
8
9
10 //LEDS
11 #define LED_GREEN_PIN          0x00001000        // P0.12 - LED1
12 #define LED_BLUE_PIN           0x00002000        // P0.13 - LED2
13 #define LED_RED_PIN            0x00800000        // P0.23 - LED3
14
15 #define LED_GREEN_ON           IOCLR0 = LED_GREEN_PIN
16 #define LED_GREEN_OFF          IOSET0 = LED_GREEN_PIN
17 #define LED_BLUE_ON            IOCLR0 = LED_BLUE_PIN
18 #define LED_BLUE_OFF           IOSET0 = LED_BLUE_PIN
19 #define LED_RED_ON             IOCLR0 = LED_RED_PIN
20 #define LED_RED_OFF            IOSET0 = LED_RED_PIN
21
22
23 //WIFI
24 #defineTX0_PIN 0x00000001      // P0.0 - UART0 TXD0
25 #defineRX0_PIN 0x00000002      // P0.1 - UART0 RXD0
26
27
28 //PWM PONTE_H
29 #definePWM2_PIN 0x00000080     // P0.7  - PWM2
30 #definePWM4_PIN 0x00000100     // P0.8 - PWM4
31 #definePWM5_PIN 0x00200000     // P0.21 - PWM5
32 #definePWM6_PIN 0x00000200     // P0.9  - PWM6
33
34
35 //ULTRASONICOS
36 #defineECHO1_PIN 0x00010000    // P0.16 - ECHO1 / EINT0
37 #defineECHO2_PIN 0x00000008    // P0.3 - ECHO2 / EINT1
38 #defineECHO3_PIN 0x00100000    // P0.20 - ECHO3 / EINT3
39
40 #defineTRIG1_PIN 0x08000000    // P0.27 - TRIG1
41 #defineTRIG2_PIN 0x10000000    // P0.28 - TRIG2
42 #defineTRIG3_PIN 0x20000000    // P0.29 - TRIG3
43
```

```
/*****************************************************************************
2  *
3  * main.c : Main program
4  *
5  * By: Anderson Augusto Heinz
6  *
7  *****************************************************************************/
8
9  /* Headers *///*****************************************************************
10
11 #include        <LPC213X.H>
12 #include        <stdio.H>
13 #include        "hardware.h"
14
15
16 /* Definitions *///*************************************************************
17
```

```
18 #define PERIOD          0x00000100      // Frequency of PWM modulation
19 #define DUTY_SPEED   0x00000FFF      // Falling edge. Vehicle speed
20 #define DUTY_TURN    0x00000090      // Falling edge. Speed to turn tires
21 #define Delay_Echo       500             // Delay [us] from ultrasonic sensor from end of trig pulse to start of echo
pulse
22 #definePRESCALLER_0          30      // Prescaler for timer 0. Defined to 1us increment in timer count mode
23 #definePRESCALLER_1          30000   // Prescaler for timer 1. Defined to 1ms increment in timer count mode
24 #defineCAR_SPEED_FWD         4.72    // Forward Speed [cm/s]
25 #defineCAR_SPEED_BWD         4.55    // Backward Speed [cm/s]
26
27 #defineDist_Obstacle    5       // The distance [cm] to consider an obstacle
28 #defineDist_Side            10      // The distance [cm] to consider a car in the side [cm]
29 #defineSetting_Time       4100    // Calculated: 4230 = 20/4.72; Best: 4100. The time [ms] to adjust the position of the
car
30 #defineDist_Spot_Min     70      // Calculated: 74; Best: 70. The shortest space [cm] necessary to park the car
31 #defineDist_Inflection    50      // Calculated: 43.24; Best: 50. The distance [cm] to change the direction of the wheels
32 #defineDist_End_Point    53      // Calculated: 43.24; Best: 53. The distance [cm] to stop inside the space.
33
34 //States
35 #define  OFF                     1
36 #define  SEARCHING               2
37 #define  WAITING                 3
38 #define  CHECKING                4
39 #define  PARKING_PART1           5
40 #define  PARKING_PART2           6
41 #define  PARKING_PART3           7
42 #define  PARKING_PART4           8
43 #define  PARKED                  9
44 #define  ABORTED                 10
45
46
47 /* Variables *///***************************************************************************
48 unsignedintTime_Echo_S1 = 0;
49 unsignedintTime_Echo_S2 = 0;
50 unsignedintTime_Echo_S3 = 0;
51 unsignedintT1 = 0;
52 unsignedintT2 = 0;
53 unsignedintT3 = 0;
54 unsignedintT4 = 0;
55 unsignedintT5 = 0;
56
57 unsignedintSpot_Counter = 0;
58 unsignedintManeuver_Counter = 0;
59 unsignedintTime_Maneuver = 0;
60
61 unsignedintCurrent_State = OFF;
62 unsignedintPrevious_Sate = OFF;
63
64 floatDist_S1= 0;            //Front Sensor
65 floatDist_S2= 0;            //Side Sensor
66 floatDist_S3= 0;            //Rear Sensor
67 floatDist_Spot = 0;
68 floatTime_Spot = 0;
69 floatDist_Maneuver = 0;
70
71 /* Functions *///***************************************************************************
72 voidDelay_ms(longtemp);
73 voidPWM_Config(void);
74 voidIO_Config(void);
75 voidForward(void);
76 voidBackwards(void);
77 voidStop(void);
78 voidLeft(void);
79 voidRight(void);
80 voidStraight(void);
81 voidDelay_10us(void);
82 voidEXTINT_Config(void);
```

```
83  voidIRQ1_interrupt(void)__irq;
84  voidIRQ2_interrupt(void)__irq;
85  voidIRQ3_interrupt(void)__irq;
86  voidDefault_int(void)__irq;
87  voidStartup(void);
88  voidTimer_Config(void);
89  voidRead_Ultra(void);
90
91  voidSearch(void);
92  voidCheck(void);
93  voidPark_1(void);
94  voidPark_2(void);
95  voidPark_3(void);
96  voidPark_4(void);
97  voidRead_Ultra_Side(void);
98  voidRead_Ultra_Backwards(void);
99  voidWait_Searching(void);
100  voidWait_Parking(void);
101
102  /* MAIN *///*****************************************************************************
103
104  intmain(void)
105  {
106  //Configurations
107  IO_Config();
108  PWM_Config();
109  EXTINT_Config();        //Configures ECHO pins as external interrupt triggered by the rising edges
110  Timer_Config();
111
112  //Startup after all the Configurations
113  Startup();
114
115  while(1)
116  {
117  switch(Current_State)
118  {
119  //*****************************************************************************************
120  case ABORTED: //Abortado
121  LED_RED_ON;
122  LED_BLUE_ON;
123  LED_GREEN_ON;
124  Delay_ms(250);
125  LED_RED_OFF;
126  LED_BLUE_OFF;
127  LED_GREEN_OFF;
128  Delay_ms(250);
129  LED_RED_ON;
130  LED_BLUE_ON;
131  LED_GREEN_ON;
132
133  while(1); //The End
134
135  //*****************************************************************************************
136  caseOFF://Desligado
137  //Wait start command
138  Previous_Sate = Current_State;
139  Current_State = SEARCHING;
140  break;
141
142  //*****************************************************************************************
143  casePARKED: //Estacionado
144  LED_GREEN_ON;
145  Delay_ms(250);
146  LED_GREEN_OFF;
147  Delay_ms(250);
148  LED_GREEN_ON;
149  Delay_ms(250);
```

```
150 LED_GREEN_OFF;
151 Delay_ms(250);
152 LED_GREEN_ON;
153 Delay_ms(250);
154 LED_GREEN_OFF;
155 Delay_ms(250);
156 LED_GREEN_ON;
157
158 while(1);//The End
159
160 //*****************************************************************************************
161 caseSEARCHING://Procurando
162 Search();
163 break;
164
165 //*****************************************************************************************
166 caseCHECKING: //Verificando
167 Check();
168 break;
169
170 //*****************************************************************************************
171 casePARKING_PART1: //Estacionando parte 1
172 Park_1();
173 break;
174
175 //*****************************************************************************************
176 casePARKING_PART2: //Estacionando parte 2
177 Park_2();
178 break;
179
180 //*****************************************************************************************
181 casePARKING_PART3: //Estacionando parte 3
182 Park_3();
183 break;
184
185 //*****************************************************************************************
186 casePARKING_PART4: //Estacionando parte 4
187 Park_4();
188 break;
189
190 }
191 }
192 }
193
194 //*****************************************************************************************
195 voidSearch(void)
196 {
197 LED_GREEN_ON;
198 Read_Ultra_Side();       //Will read side sensor
199 Delay_ms(60);            //Time between two measurements. Recommendation from datasheet
200
201 //Converting values from timer count to time
202 Time_Echo_S2 = (T3-Delay_Echo);          //the number of increments is equal to the time in [us]
203                                           //Delay_Echo is the void time until Echo is high level
204
205 //Converting time to distance of the objects
206 Dist_S2 = (Time_Echo_S2/58);             //Dist_S2 will be [cm] if Time_Echo is given in [us]
207
208 Forward();
209
210 if(Dist_S2 >= Dist_Side)          //there is no car/obstacle on the side
211 {
212 if(Spot_Counter == 0)             //Is it the first time? Means the begin of the space
213 {
214 T1TCR = 0x00000002;               //Restart counter and prescaler
215 T1TCR = 0x00000001;               //Start timer to measure the time and calculate the size of the space.
216
```

```
217 LED_BLUE_ON;
218 Delay_ms(100);
219 LED_BLUE_OFF;
220 Spot_Counter = 1;
221 }
222
223 }
224 else                                    //there is a car/obstacle on the side
225 {
226 if(Spot_Counter == 1)                   //After space
227 {
228 Time_Spot=T1TC;                         //Get the timer counter value
229 T1TCR=0x00000002;                       //Restart counter and prescaler
230
231 LED_BLUE_ON;
232
233 Delay_ms(Setting_Time);                 //Go forward for some time to adjust the position of the car
234 Stop();
235 LED_BLUE_OFF;
236
237 Previous_Sate = Current_State;
238 Current_State = CHECKING;
239
240 }
241
242 }
243 LED_GREEN_OFF;
244
245 }
246
247
248 //*********************************************************************************************
249 voidCheck(void)
250 {
251 Stop();          //Redundancy
252
253 LED_BLUE_ON;
254 Delay_ms(50);
255 LED_GREEN_ON;
256 Delay_ms(50);
257 LED_RED_ON;
258 Delay_ms(50);
259 LED_BLUE_OFF;
260 LED_GREEN_OFF;
261 LED_RED_OFF;
262
263 Time_Spot=Time_Spot/1000;                       //Converts to time in [s]
264 Dist_Spot=Time_Spot*CAR_SPEED_FWD;              //Calculates the size of the space using the velocity [cm]
265
266 LED_BLUE_ON;
267 Delay_ms(70);
268 LED_BLUE_OFF;
269
270 if(Dist_Spot>Dist_Spot_Min)       //Can the car park in this space?
271 {
272 LED_GREEN_ON;
273 Delay_ms(100);
274 LED_GREEN_OFF;
275 Previous_Sate=Current_State;
276 Current_State=PARKING_PART1;
277 }
278 else
279 {
280 LED_RED_ON;
281 Delay_ms(200);
282 LED_RED_OFF;
283 Previous_Sate=Current_State;
```

```
284 Current_State=ABORTED;
285 }
286 }
287
288
289 //*****************************************************************************************
290 voidPark_1(void)
291 {
292 if(Maneuver_Counter==0)          //shall be defined with 0 when created
293 {
294 LED_BLUE_ON;
295 Delay_ms(50);
296 LED_BLUE_OFF;
297 T1TCR=0x00000002;                //Restart counter and prescaler
298 T1TCR=0x00000001;                //Start
299 Maneuver_Counter=1;
300 }
301
302 Right();
303 Backwards();
304 LED_GREEN_ON;
305 Delay_ms(50);
306 LED_GREEN_OFF;
307 Delay_ms(50);
308
309 //Check distance traveled
310 Time_Maneuver=T1TC;                                  // Get the timer counter value
311 Time_Maneuver=Time_Maneuver/1000;                    //Convert to time in [s]
312 Dist_Maneuver=Time_Maneuver*CAR_SPEED_BWD;
313
314 if(Dist_Maneuver>=Dist_Inflection)                   //Has the car reached the distance to change to part 2?
315 {
316 Stop();
317 Previous_Sate=Current_State;
318 Current_State=PARKING_PART2;
319
320 LED_RED_ON;
321 Delay_ms(100);
322 LED_RED_OFF;
323 }
324 //else //Keep going
325 }
326
327 //*****************************************************************************************
328 voidPark_2(void)
329 {
330 Stop();                  //Redundancy
331 LED_RED_ON;
332 Delay_ms(70);            //Time to mechanical inerce
333 LED_RED_OFF;
334 Delay_ms(70);            //Time to mechanical inerce
335 LED_RED_ON;
336 Straight();
337 Delay_ms(70);            //Time to mechanical inerce
338 LED_RED_OFF;
339 Previous_Sate=Current_State;
340 Current_State=PARKING_PART3;
341
342 T1TCR=0x00000002;    //Restart counter and prescaler
343 T1TCR=0x00000001;    //Start
344
345 }
346
347 //*****************************************************************************************
348 voidPark_3(void)
349 {
350 Left();
```

```
Backwards();
LED_GREEN_ON;
Delay_ms(50);
LED_GREEN_OFF;
Delay_ms(50);

Time_Maneuver=T1TC;                     // Get the timer counter value
Time_Maneuver=Time_Maneuver/1000;       //Convert to time [s]
Dist_Maneuver=Time_Maneuver*CAR_SPEED_BWD;

if(Dist_Maneuver>=Dist_End_Point)       //Has the car reached the distance to change to part 4?
{
Stop();
Straight();
Previous_Sate=Current_State;
Current_State=PARKING_PART4;
}
//else //Keep going
}

//*****************************************************************************************
voidPark_4(void)
{
Stop();                                 //Redundancy
LED_BLUE_ON;
Straight();                             //Align the wheels
Forward();
Delay_ms(Setting_Time*0.4);
Stop();
LED_BLUE_OFF;
Previous_Sate=Current_State;
Current_State=PARKED;
T1TCR=0x00000002;                       //Restart counter and prescaler

}

//*****************************************************************************************
voidRead_Ultra_Side(void)
{
//Reads Sensor2 - Side of the car
T0TCR=0x00000002;                       //Restart counter and prescaler

IOSET0=TRIG2_PIN;
Delay_10us();
IOCLR0=TRIG2_PIN;
T0TCR=0x00000001;                       //Start

}

//*****************************************************************************************
voidDelay_ms(longtemp)                  //Works with 20MHz oscillator & PCLK=30MHz
{
longi;
longj;
for(i=0;i<temp;i++)
{
for(j=0;j<7400;j++);
}
}

//*****************************************************************************************
voidDelay_10us(void)                    //Works with 20MHz oscillator & PCLK=30MHz
{
longi;
for(i=0;i<118;i++);

}
```

```
418
419 //*****************************************************************************
420 voidPWM_Config(void)
421 {
422 PWMPR =0x00000001;              //Load prescaler.
423 PWMPCR =0x00007400;             //PWM2 PWM4 PWM5 PWM6 ARE SINGLE EDGE CONTROLLES.
424 PWMMCR =0x00000003;             //Resets counters when PWMTC matches with PWMMR0.
425 PWMMR0 =PERIOD;                 //Defines the Frequency
426 PWMMR2 =0x00000000;             //Falling edge is defined accordingly with desired speed
427 PWMMR4 =0x00000000;             //Falling edge is defined accordingly with desired speed
428 PWMMR5 =0x00000000;             //Falling edge is defined accordingly with desired speed
429 PWMMR6 =0x00000000;             //Falling edge is defined accordingly with desired speed
430 PWMLER =0x00000075;             //ENABLES COMPARATORS 0, 5 e 6
431 PWMTCR =0x00000002;             //RESTART COUNTER
432 PWMTCR =0x00000009;             //ENABLES PWM COUNTER
433
434 }
435
436 //*****************************************************************************
437 voidIO_Config(void)
438 {
439 IODIR0=0x38A03380;              // PORT0 pins are OUTPUT
440 IOSET0=0x00803000;              // KEEPS ALL THE LEDS OFF
441 IOCLR0=0x00110008;              // Echo pins are cleared
442
443 PINSEL0=0x000A80C5;             //p0.0-TX_:_p0.1-RX_:_p0.3-EINT1_:_p0.7-PWM2_:_p0.8-PWM4_:_p0.9-PWM6
444 PINSEL1=0x00000701;             //p0.16-EINT0_:_p0.20-EINT3_:_p0.21-PWM5
445
446 }
447
448 //*****************************************************************************
449 voidTimer_Config(void)
450 {
451 T0CTCR=0x00000000;              //Timer mode --> Counter
452 T0PR =0x0000001D;               //Prescaler timer0 period of incrementation = 1us
453 T0TCR =0x00000002;              //Restart counter and prescaler
454
455 T1CTCR=0x00000000;
456 T1PR =0x0000752F;               //Prescaler timer1 period of incrementation = 1ms
457 T1TCR =0x00000002;              //Restart counter and prescaler
458
459 }
460
461 //*****************************************************************************
462 voidForward(void)
463 {
464 PWMMR5 =DUTY_SPEED;             //Configures the falling edge ~ DUTY CYCLE 50%
465 PWMMR6 =0;                       //Configures the falling edge ~ DUTY CYCLE 0%
466 PWMLER =0x00000075;             //Refresh
467
468 }
469
470 //*****************************************************************************
471 voidBackwards(void)
472 {
473 PWMMR5 =0;                      //Configures the falling edge ~ DUTY CYCLE 0%
474 PWMMR6 =DUTY_SPEED;             //Configures the falling edge ~ DUTY CYCLE 50%
475 PWMLER =0x00000075;              //Refresh
476
477 }
478
479 //*****************************************************************************
480 voidStop(void)
481 {
482 PWMMR5 =0;                      //Configures the falling edge ~ DUTY CYCLE 0%
483 PWMMR6 =0;                      //Configures the falling edge ~ DUTY CYCLE 0%
484 PWMLER =0x00000075;             //Refresh
```

```
485
486 }
487
488 //*****************************************************************************
489 voidLeft(void)
490 {
491 PWMMR2=0;                       //Configures the falling edge ~ DUTY CYCLE 0%
492 PWMMR4=DUTY_TURN;               //Configures the falling edge ~ DUTY CYCLE %
493 PWMLER= 0x00000075;             //Refresh
494
495 }
496
497 //*****************************************************************************
498 voidRight(void)
499 {
500 PWMMR2=DUTY_TURN;               //Configures the falling edge ~ DUTY CYCLE %
501 PWMMR4=0;                       //Configures the falling edge ~ DUTY CYCLE 0%
502 PWMLER= 0x00000075;             //Refresh
503
504 }
505
506 //*****************************************************************************
507 voidStraight(void)
508 {
509 PWMMR2=0;                       //Configures the falling edge ~ DUTY CYCLE 0%
510 PWMMR4=0;                       //Configures the falling edge ~ DUTY CYCLE 0%
511 PWMLER= 0x00000075;             //Refresh
512
513 }
514
515 //*****************************************************************************
516 voidEXTINT_Config(void)
517 {
518 VICDefVectAddr=(unsigned)Default_int;           //handler default - Bad interruptions
519
520 EXTMODE=0x0F;                   //INTERRUPTS DEFINED BY LEVEL
521 EXTPOLAR=0;                     //CHOOSING FALLING EDGE
522
523 //----Sensor1----
524 // VICVectCntl1 = 0x20 | 14;     //CHANEL 1, EINT0
525 // VICVectAddr1 = (unsigned) IRQ1_interrupt;
526
527 //----Sensor2----
528 VICVectCntl2 = 0x20|15;         //CHANEL 2, EINT1
529 VICVectAddr2 = (unsigned)IRQ2_interrupt;
530
531 //----Sensor3----
532 // VICVectCntl3 = 0x20 | 17;     //CHANEL 3, EINT3
533 // VICVectAddr3 = (unsigned) IRQ3_interrupt;
534
535 VICIntEnable=0x0002C000;        //ENABLES EINT0, EINT1, EINT3
536
537 }
538
539 //*****************************************************************************
540 voidIRQ1_interrupt(void)__irq     //----Sensor1----
541 {
542 //do this every interruption
543 T1=T0TC;                //Gets timer 0 value
544
545 EXTINT=0x01;            //Clean the flag BIT 0 -- EINT1
546 VICVectAddr=0x00;       //Terminates the interrupt
547
548 }
549
550 //*****************************************************************************
551 voidIRQ2_interrupt(void)__irq     //----Sensor2----
```

```
552 {
553 //do this every interruption
554 T3=T0TC;                    //Gets timer 0 value
555
556 EXTINT=0x02;                //Clean the flag BIT 1 -- EINT2
557 VICVectAddr=0x00;      //Terminates the interrupt
558
559 }
560
561 //*********************************************************************************************
562 voidIRQ3_interrupt(void)__irq       //----Sensor3----
563 {
564 //do this every interruption
565 T5=T0TC;                    //Gets timer 0 value
566
567 EXTINT=0x08;                //Clean the flag BIT 3 -- EINT3
568 VICVectAddr=0x00;      //Terminates the interrupt
569
570 }
571
572 //*********************************************************************************************
573 voidDefault_int(void)__irq
574
575 {
576 ;//Do nothing. Used to clean interruptions
577 //não faz nada, usada para limpar interrupções quando a original esta desligada
578 //Errata
579 //http://www.keil.com/support/docs/2888.htm
580 }
581
582 //*********************************************************************************************
583 voidStartup(void)
584 {
585 //Initial animation to confirm success during configuration
586 LED_GREEN_ON;
587 Delay_ms(350);
588 LED_BLUE_ON;
589 Delay_ms(350);
590 LED_RED_ON;
591 Delay_ms(350);
592 LED_GREEN_OFF;
593 LED_BLUE_OFF;
594 LED_RED_OFF;
595 Delay_ms(300);
596 LED_GREEN_ON;
597 LED_BLUE_ON;
598 LED_RED_ON;
599 Delay_ms(300);
600
601 LED_GREEN_OFF;
602 LED_BLUE_OFF;
603 LED_RED_OFF;
604
605 }
606
607 //*********************************************************************************************
608 //*********************************************************************************************
609 //*********************************************************************************************
610
```