Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Veículo controlado remotamente por smartphone para divulgação de mensagens de propaganda em eventos públicos

Andrei A. L. Buslik

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Wilson Henrique Veneziano

Brasília
2020

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Veículo controlado remotamente por smartphone para divulgação de mensagens de propaganda em eventos públicos

Andrei A. L. Buslik

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Wilson Henrique Veneziano (Orientador)
CIC/UnB

Prof. Dr.
Edison Ishikawa

Prof. Dr.
João José Costa Gondim

Prof. Dr. João José Costa Gondim
Coordenador do Curso de Engenharia da Computação

Brasília, 15 de dezembro de 2020

Dedicatória

Dedico este trabalho a minha esposa Tainah Lopes Galvão, que esteve ao meu lado desde o início da minha empreitada na faculdade até a entrega deste projeto, sem ela poderia ter desistido há muito tempo. Dedico também a minha mãe Mariza de Lima Mendes e Buslik por todo carinho e atenção e ao meu pai Sérgio Alencar Buslik.

Agradecimentos

Agradeço ao professor dr. Wilson Henrique Veneziano pela oportunidade de fazer este projeto, pelo suporte com os materiais e na montagem final do projeto, bem como pelo conhecimento partilhado e paciência. Agradeço ao colega Rodrigo Alves Carvalho pelo apoio prestado ao longo do projeto. Agradeço também aos amigos e amigas, Vanessa Lima, Lucas Mendonça, Otávio Bravim da Silva, Matheus Avelino que de alguma forma contribuiram para meu bem estar.

Resumo

Foi construído um pequeno carro, controlado remotamente por aplicativo em smartphone, movido a bateria, para divulgação de mensagens de propaganda em eventos públicos, como feiras, palestras e mostras de cursos. A finalidade é apresentar mensagens em displays para divulgar o nome e os cursos do Departamento de Ciência da Computação da Universidade de Brasília. Foram aproveitados a carcaça e os motores elétricos de um pequeno carro de controle remoto comercial. A placa eletrônica de controle original foi substituída por uma placa de Arduino, e foram acrescentados módulo *bluetooth*, *displays* OLED e sensores de ultrassom e infravermelho. Além de toda a montagem dos componentes eletroeletrônicos, foi elaborado o código-fonte para o microcontrolador do Arduino e prototipado um aplicativo de controle do carro para smartphone. O carro é capaz de atender aos comandos do usuário, bem como operar em modo automatizado por meio dos sensores de barreira e de linhas no solo.

Palavras-chave: Arduino, robótica, carro de controle remoto, Virtuino, sensores.

Abstract

This project aims to arouse the interest of new students for IT courses, whether they are young people, veterans or enthusiasts in programming, robotics or both. The project of an application-controlled car that can walk autonomously following lines and detects obstacles will be presented, taking care to be viable and affordable. The modules used, the tests performed and the results and discussions about them will be presented. Ideas for future development of this project will also be proposed.

Keywords: Arduino, robotics, remote control car, Virtuino, sensors

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O uso de tecnologias no cotidiano, como celulares, *desktops* e *laptops*, está fortemente arraigado na nossa sociedade. Essas teconologias, também chamadas de TICs (Tecnologia da Informação e Comunicação), são utilizadas das mais diversas formas, seja na indústria, por meio de processos de automação; no comércio, na parte de gerência e publicidade; no setor de investimentos (informação em tempo real) ou na educação, em processo de ensino e aprendizagem a distância por exemplo.

Cursos da área da computação como Engenharia da Computação, Ciências da Computação e afins, costumam despertar o interesse de quem pretende aprender a entender o funcionamento, manipular ou aprimorar tecnologias.

Segundo o site do departamento de Ciência da Computação da Universidade de Brasília (CIC - UNB) [8]

> ¨O curso de Engenharia de Computação tem como objetivo preparar profissionais para atuarem nas áreas em que os conhecimentos de eletrônica e computação são essenciais e complementares, como sistemas microprocessados, eletrônica embarcada e redes de comunicação de dados e todos os ambientes relacionados. Oferece uma formação sólida em Matemática e Física e nos fundamentos da Engenharia Elétrica e da Computação.¨

Seja para pessoas que já tenham conhecimento na área de programação e eletrônica, ou para quem quer iniciar do zero seus estudos, um curso de ensino superior que abranja ambas as áreas é um diferencial para despertar o interesse de quem busque cursos voltados à tecnologia.

Livros sobre programação para iniciantes ou avançados no assunto são uma fonte de informação, assim como sites e fóruns na *internet*. Seguir exemplos iniciais costumam dar uma visão de como programar ou projetar circuitos e softwares.

O surgimento das plataformas de prototipagem para microcontroladores tem permitido a prática e desenvolvolvimento de soluções eletrônicas por quem se interessar. As

comunidades envolvidas com essas plataformas apoiam o desenvolvimento de soluções próprias, baseadas na estrutura oferecida.

Ainda que não se saiba como programar, é possível a introdução à programação aos poucos, com projetos simples. Tendo essa ideia em mente e no intúito de demonstrar e despertar interesse no curso de Engenharia de Computação, idealizou-se o projeto que aqui apresentado. Será descrito o projeto de um carro controlado por aplicativo em celular e que possa andar seguindo linhas de forma autônoma. Além de tentar despertar o interesse através de apresentações em feiras, pretende-se deixar um projeto que possa ser incrementado ou modificado por outros alunos de graduação.

1.1 Problema

Eventos públicos, como feiras, palestras em escolas e mostras de cursos, são oportunidades interessantes para serem realizadas divulgações de cursos de graduação. Porém, como fazer isso de forma atraente e chamativa ao público-alvo?

1.2 Justificativa

É importante divulgar o nome do departamento e os cursos do Departamento de Ciência da Computação da UnB em eventos públicos. Para isso, é interessante estar disponível algum tipo de artefato robótico que chame a atenção do público. Uma opção é um veículo controlado via *smartphone*, capaz de apresentar em *displays* mensagens alusivas ao Departamento e seus cursos. Entretanto, faz-se necessário construir tal veículo, pois não está disponível no mercado algo exatamente desse tipo.

1.3 Objetivo Geral

Construção de um pequeno carro, controlado remotamente por aplicativo em *smartphone*, movido a bateria, para divulgação de mensagens de propaganda em eventos públicos, como feiras, palestras e mostras de cursos. A ideia é aproveitar somente a carcaça plástica e os motores elétricos de um carrinho de controle remoto comercial; novos sensores, atuadores e placa de controle são incorporados ao veículo. Todos os materiais devem ser de baixo custo e facilmente encontráveis no mercado brasileiro. A carcaça deve passar por um processo de pintura atinente ao objetivo construtivo.

1.4 Objetivo Específico

Para ser alcançado o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

- incorporação de sensores para automatizar o veículo;
- incorporação de displays para apresentação de mensagens;
- incorporação de módulo *bluetooth* para comunicação com *smartphones*;
- desenvolvimento de código-fonte para microcontrolador; e
- prototipação de um aplicativo de controle do veículo via *smartphones*.

1.5 Procedimentos Adotados

O desenvolvimento deste trabalho durou cerca de um ano e meio. Para a construção do *hardware* e desenvolvimento do *software*, foi adotada uma metodologia que compreendeu diversas etapas:

- estudos para definição de métodos para comunicação sem fio entre o veículo e o smartphone controlador;
- estudos para definição de sensores para manobras a serem efetuadas pelo veículo;
- estudos para definição de *displays* para apresentação de mensagens no veículo;
- estudos para a definição de *hardware* de controle do sistema;
- construção dos módulos de sensores e atuadores
- desenvolvimento de código-fonte para controle do *hardware*;
- prototipação de aplicativo para *smartphone*;
- testes funcionais no aplicativo prototipado;
- testes funcionais em cada elemento de *hardware* (sensores e atuadores);
- testes funcionais no código controlador do sistema como um todo;
- testes funcionais do veículo completamente montado;
- pintura do veículo.

1.6 Organização Deste Trabalho

Esta monografia está organizada nos seguintes capítulos:

- Capítulo 2: apresenta a história das placas Arduino;
- Capítulo 3: descreve cada um dos sensores e atuadores utilizados no projeto;
- Capítulo 4: enumera os testes individuais dos sensores e atuadores que compõe este projeto;
- Capítulo 5: descreve o desenvolvimento do projeto em relação ao *hardware*;
- Capítulo 6: descreve o desenvolvimento do *software* por trás do funcionamento do carro; e
- Capítulo 7: apresenta as conclusões e trabalhos futuros.

Capítulo 2

Referencial Teórico

Neste capítulo será apresentado um pouco da história da *Arduino*, sua concepção inicial, sua evolução e opções de desenvolvimento e uso nos dias atuais.

2.1 História da Arduino

No início dos anos 2000, um projeto de pesquisa feito no Instituto de Design de Interação de Ivrea (*Interaction Design Institute Ivrea* - IDII) em Turim, na Itália, por Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis tinha como objetivo encontrar alguma maneira de facilitar e baratear o custo de projetos com componentes eletrônicos para que alunos de *design*, sem conhecimento técnico em eletrônica, pudessem criar dispositivos eletrônicos que reagissem fisicamente conforme fossem estimulados [3].

Neste contexto havia a necessidade de entender a interação entre pessoas e máquinas (ou sistemas) de forma que objetos tivessem um *design* inteligente. Isto levou ao entendimento de que os sistemas precisavam ser interativos, capazes de perceberem eventos do mundo físico e reagirem a eles. Dessa forma, os sistemas recebem estímulos por meio de sensores e respondem por meio de atuadores [3].

Tendo em vista que a idéia original era voltada a estudantes de *design*, seria necessário criar uma linguagem que pudesse codificar o contexto das artes. Assim, Casey Reas desenvolveu um *software* livre, conhecido como *Processing*[9], que criava imagens a partir de simples códigos, chamados de *Sketches*. Utilizando o *Processing* foi possível criar com rapidez descrições visuais de dados e algoritmos na tela do computador, além de ser multiplataforma (MaC, Windows e Linux) [1].

O ambiente de desenvolvimento criado pode ser visto na Figura 2.1 [1], a função setup() inicia as estruturas necessárias da imagem e a função draw() é executada de forma recorrente [1].

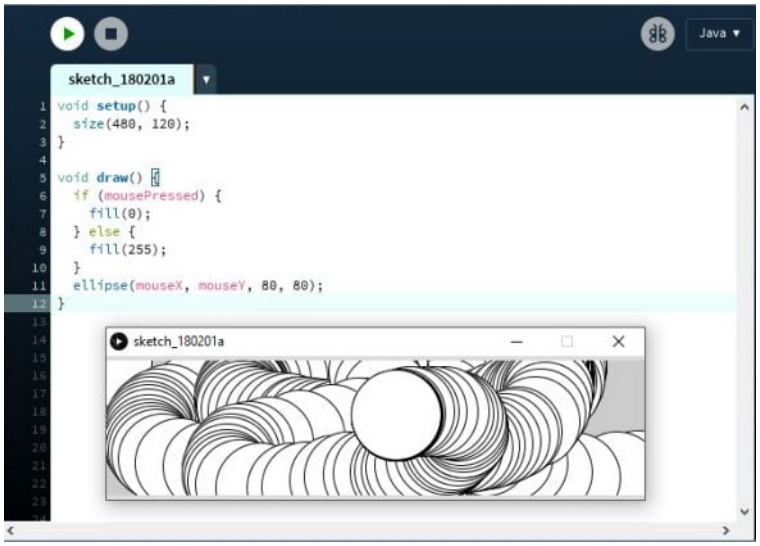

Figura 2.1: Ambiente de desenvolvimento do *Processing* - Fonte: eletronicapratica.com[1].

2.2 Wiring

Vendo o sucesso e facilidade de uso do *Processing* [9], a evolução da idéia de desenhos para dispositivos eletrônicos veio pela tese de mestrado de Hernando Barragán. Uma plataforma de prototipação eletrônica de *hardware* livre composta por uma linguagem de programação e um ambiente de desenvolvimento integrado (IDE).

Modificando o *Processing* [9], o *software* passou a converter código escrito em um programa que um microcontrolador pudesse executar. A IDE do *wiring* pode ser vista na Figura 2.2.

Figura 2.2: Ambiente de desenvolvimento do *Wiring* - Fonte: eletronicapratica.com[1].

Barragán diz em sua tese [10] que as ferramentas de prototipagem eletrônica e de programação à época eram voltadas para engenheiros e técnicos e distantes do público fora destas categorias citadas. Há um artigo em que Báragan [2] conta as razões que o levaram a este projeto e o processo de desenvolvimento do *Wiring* e sua ideia de aproximar a tecnologia para um público mais abrangente. No artigo é dito que:

> ¨*Abstrair os pinos do microcontrolador como números foi, sem dúvida, uma decisão importante, possível porque a sintaxe foi definida antes da implementação em qualquer plataforma de hardware*¨.

Assim, com comandos como pinMode(), digitalRead(), digitalWrite() é abstraido o funcionamento de dispositivos eletrônicos e separou-se a sintaxe que define estes comandos da implementação dos mesmos, abrindo a possibilidade para que diferentes microcontroladores pudessem utiliza-los. Para as placas iniciais, Baragán escolheu o microcontrolador *ATmega128*. O produto final pode ser visto na Figura 2.3.

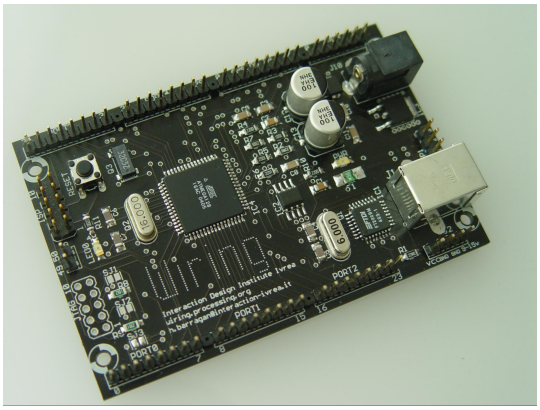

Figura 2.3: Primeira placa *Wiring* feita por Baragán - Fonte: arduinohitstory.github.io[2].

2.3 *Arduino*

As placas feitas por Baragán tinham preços mais acessíveis dos que as placas que existiam no mercado mas ainda longe do ideal. Então, em 2005, Massimo Banzi, juntamente com David Mellis, adicionaram suporte à *Wiring* para o microcontrolador *PIC*, mais barato, e confeccionaram uma placa com menos recursos, a *Programma2003* vista na Figura 2.4.

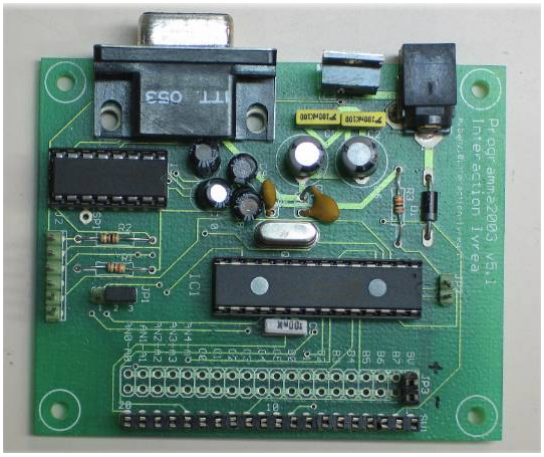

Figura 2.4: Primeira placa feita por Massimo Banzi - Fonte: arduinohitstory.github.io[2].

O problema com os microcontroladores *PIC* era que os mesmos não possuiam ferramentas de código aberto disponível na época, para usar uma linguagem como C para programá-las. Massimo decidiu usar um ambiente chamado JAL (*Just Another Language*) para programar o microcontrolador *PIC*, porém, esta linguagem não era muito usual e só funcionava em *Windows* sendo que muitos dos estudantes na época utilizavam *MacOS*.

Aprimorando a placa e fazendo homenagem ao bar *Bar di Re Arduino* que frequentavam, chegamos a placa mais conhecida atualmente, a *Arduino Uno* vista na Figura 2.5.

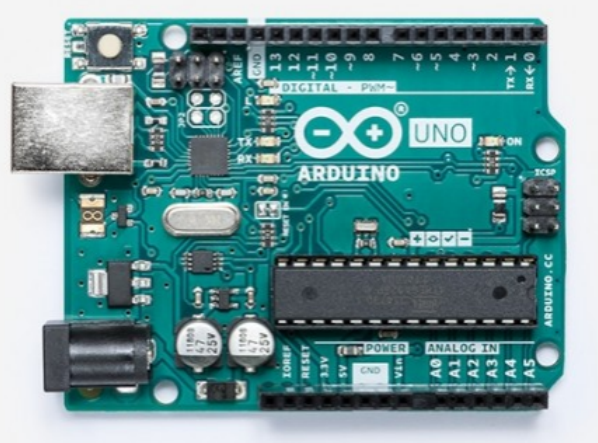

Figura 2.5: *Arduino Uno* - Fonte: store.arduino.cc[3].

Hoje temos diferentes placas para diferentes usos, desde placas para projetos simples, como a própria *Uno* e a placas menores como a *Arduino Nano*, até projetos maiores e mais atuais, como Internet das Coisas (*Internet of Things* - IoT).

Sendo um projeto *open source*, o ambiente de desenvolvimento (IDE da figura 2.6) é disponibilzado para os usuários gratuitamente no site da *Arduino*[3] - `www.arduino.cc/en/Main/Software` - de forma simples. Além de tutoriais e materiais de referência, a comunidade que surgiu em volta da *Arduino* é ativa por meio do fórum do próprio site.

Além das placas existem módulos que podem ser acoplados a elas para diferentes fins, como módulo *bluetooth*, sensor de umidade, telas LCD e OLED. Cada módulo pode ser usado em separado ou em conjunto com outros a depender do projeto, da vontade do usuário e da capacidade da placa.

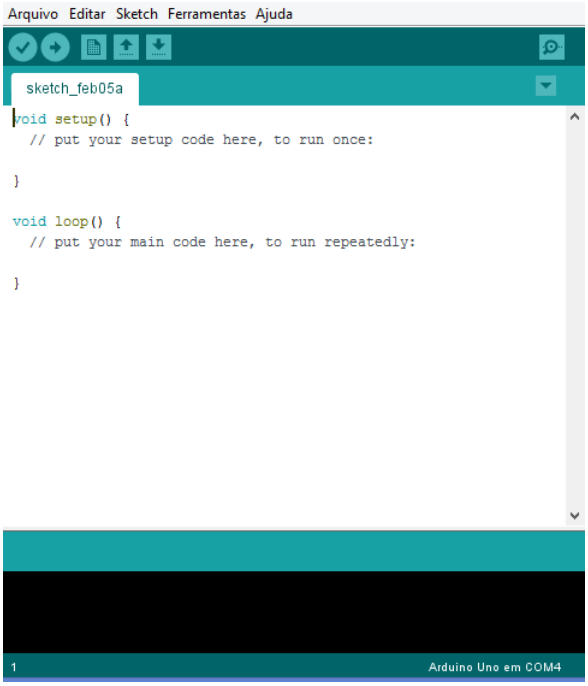

Figura 2.6: Ambiente de desenvolvimento *Arduino*.

2.4 Por que utilizar o *Arduino*?

Além da área de programção, aliar prototipação de circuitos de forma simples, acessível e econômica é interessante. *Arduino* integra essas três características, oferecendo um ambiente de desenvolvimento para programação (IDE) e placas que se comunicam com dispositivos de entrada e saída com diversas funcionalidades.

Por ser *open-source* contribui para ser econômico e ter uma base ampla de usuários que se comunicam por meio de fóruns no próprio site [3].

A IDE (Figura 2.6) oferece exemplos de códigos prontos para teste de maneira simples. Também é possível adicionar bibliotecas novas feitas pelos usuários. Também existem livros com projetos detalhados e explicados, como o livro de Simon Munk [11].

Assim, a plataforma *Arduino* se mostra uma via ideal para conhecimento, troca de ideias e introdução para a programação e robótica.

Capítulo 3

Módulos e Componentes de *Hardware* e *Software*

Este capítulo descreve os componentes e módulos utilizados neste trabalho, detalhando o funcionamento e a integração entre eles.

A Figura 3.1 mostra o carro desmontado. A ideia foi partir de um carro comercial, conservar apenas as carcaças interna e externa, além dos motores elétricos de corrente contínua; descartar os demais componentes eletrônicos, acrescentar componentes para controle dos motores, comunicação *bluetooth*, sensores de ultrassom e infravermelho, bem como prototipar um aplicativo para *smartphones* para controle do veículo.

Destacado em vermelho (1) temos os motores que foram reaproveitados no projeto. Já em verde (2) temos a bateria de $9V$. O módulo de som, destacado em azul (3) foi descartado bem como o circuito de recepção e controle original do carro. As demais peças como rodas, eixo, carcaça interna e externa, foram aproveitados no projeto. A seguir serão descritos os componentes adicionados no projeto.

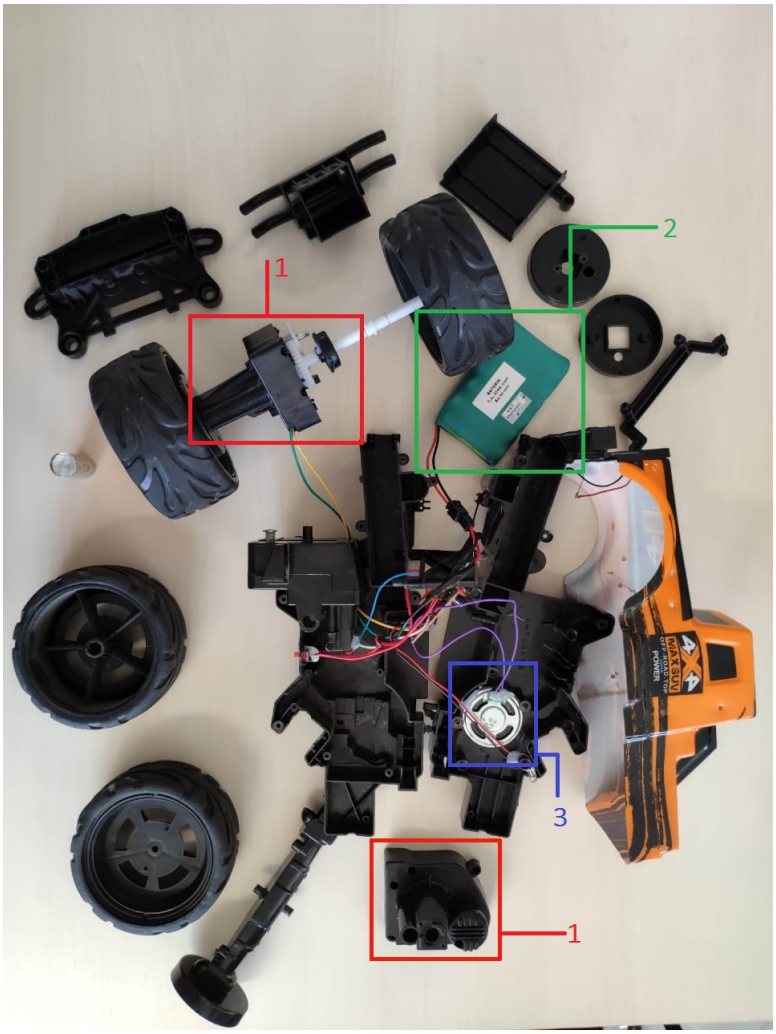

Figura 3.1: Peças originais do carro.

3.1 Arduino Mega

O componente principal deste trabalho é a *Arduino Mega*, vista na Figura 3.2. Destacado em vermelho com o numeral 1 temos os pinos de alimentação entre a *Arduino* e outros módulos. Consistindo em dois pinos terra (*ground* - *GND*), duas portas de 5 V, um regulador de tensão de 3.3 V com fornecimento máximo de 50 mA, pino de *reset* para reiniciar de maneira externa o programa salvo na placa e o pino IOREF (entrada não nomeada, acima da entrada de 5 V) para indicar à módulos externos qual a tensão de referência nos pinos de saída da placa. Há também o pino VIN, quando a placa é alimentada através do conector *Jack* (descrito posteriormente), a tensão da fonte conectada a este estará no VIN, caso a alimentação esteja sendo feita pela porta USB, VIN terá a tensão igual a porta USB. O pino VIN também pode ser usado para alimentar a própria *Arduino*.

Já em azul com o numeral 2, temos 16 portas analógicas nomeadas de A0 à A15, onde cada uma tem a resolução de 10 bits. Por padrão, a referência do conversor AD (*Analogic-Digital*) está ligada internamente ao pino de $5\,V$, ou seja, quando a entrada estiver com $5\,V$ o valor da conversão analógica digital será 1023. O valor da referência pode ser mudado através do pino AREF.

Indicado com a cor amarela e o numeral 3 temos as portas digitais nomeadas de 0 à 53. Estes Pinos operam em $5\,V$, onde cada pino pode fornecer ou receber uma corrente máxima de $40\,mA$. Cada pino possui resistor de *pull-up* interno que pode ser habilitado por software. Dependendo de sua numeração, estes pinos possuem diferentes propriedades, as portas de 0 à 13 são entradas PWM (*Pulse Width Modulation*) de 8 bits que permitem controlar a frequência da variação de tensão nestas portas. As portas 14 à 19 podem ser utilizados para receber (RX) e transmitir (TX) dados seriais TTL. Aqui também temos duas portas SDA (*Serial Data*) e SCL (Serial Clock) para transferir e receber informações pelo protocolo I2C, além de mais dois pinos de terra (GND), localizados abaixo mais a esquerda, e dois pinos de alimentação de $5\,V$, localizados abaixo mais a direita.

As funções dos pinos da *Mega* estão listadas a seguir:

- **comunicação serial** - Serial 0: 0 (RX) e 1 (TX); Serial 1: 19 (RX) e 18 (TX); Serial 2: 17 (RX) e 16 (TX); Serial 3: 15 (RX) e 14 (TX).
- **interrupções externas** - 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), e 21 (interrupt 2).
- **PWM** - pinos 2 à 13 e 44 à 46
- **comunicação I2C** - (TWI): pinos 20 (SDA) and 21 (SCL)

A alimentação da *Mega* pode ser feita pela entrada P4 indicada em verde (4) ou pela entrada USB em marrom (5) para *laptops* e *desktops*. Pela porta USB também transferimos o programa a ser executado. A alimentação externa é feita através do conector *Jack* com o terminal positivo no centro, onde o valor de tensão da fonte externa deve estar entre os limites $6\,V$ a $20\,V$, porém, se alimentada com uma tensão abaixo de $7\,V$, a tensão de funcionamento da placa, que na *Arduino Uno* é $5\,V$, pode ficar instável. Quando alimentada com tensão acima de $12\,V$, o regulador de tensão da placa pode sobreaquecer e danificar a placa. Dessa forma, é recomendado para tensões de fonte externa valores de $7\,V$ à $12\,V$.

A placa possui um regulador de tensão para a entrada de fonte externa, já para a entrada USB não é necessária um regulador de tensão. O circuito da porta USB apresenta alguns componentes que protegem a mesma em caso de picos de tensão. Além disso, a placa conta com um circuito para comutar a alimentação automaticamente entre as tensões

da porta USB e da fonte externa. Caso haja uma tensão no conector DC e a USB esteja conectada, a tensão de $5\,V$ será proveniente da fonte externa e a USB servirá apenas para comunicação com o *desktop/laptop*.

Por fim, em roxo (6) temos o microcontrolador *ATmega 2560* e o botão de *reset* em **preto** (**7**). A Tabela 3.1 mostra as especificações da placa.

A leitura ou escrita de valores nos pinos podem ser feitas por meio de funções como digitalRead(), digitalWrite(), analogRead() ou analogWrite(). O site da *Arduino*[3] possui documentação de referência para o uso das funções internas. A Figura 3.3 mostra algumas dessas funções.

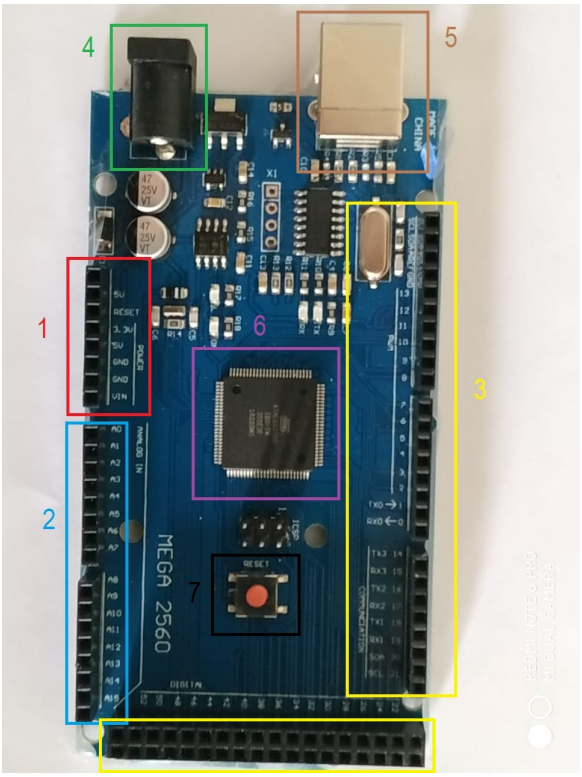

Figura 3.2: Placa *Arduino Mega*.

Figura 3.3: Algumas das funções utilizadas nas placas *Arduino* - Fonte: arduino.cc [3].

Tabela 3.1: Especificações da placa *Arduino Mega*.

Microcontrolador	ATmega2560
Tensão de operação	$5\,V$
Tensão de alimentação (recomendada)	$7\,V$ - $12\,V$
Tensão de alimentação (limite)	$6\,V$ - $20\,V$
Entradas e saídas digitais	54 (sendo 14 do tipo PWM)
Entradas analógicas	16
Corrente contínua I/O (máx.)	$40\,mA$
Corrente contínuia pino $3.3\,V$ **(máx.)**	$50\,mA$
Memória Flash	256 KB / 8 KB (bootloader)
Memória SRAM	8 KB
Memória EEPROM	4 KB
Velocidade do *Clock*	16 MHz
Dimensões	$101,6\,mm$ x $53,34\,mm$
Massa	$150\,g$

Fonte: https://blog.eletrogate.com/guia-completo-do-arduino-mega/ [12]

3.2 Ponte H

O módulo Ponte H é o circuito que permite controlar o giro dos motores presentes no carro e protege o *Arduino* de ser sobrecarregado pela bateria utilizada neste trabalho.

As Pontes H possuem este nome devido ao formato que é montado o circuito que as compõe, semelhantes a letra H. O circuito utiliza quatro chaves (S1, S2, S3 e S4) que são acionadas de forma alternada, ou seja (S1-S3) ou (S2-S4), conforme a Figura 3.4. Dependendo da configuração entre as chaves teremos a corrente percorrendo o motor tanto por um sentido ou por outro. Quando nenhum par de chaves esta acionado, o motor está desligado (circuito à esquerda). Quando para S1-S3 é acionado a corrente que percorre S1-S3 faz com que o motor gire em um sentido (circuito no centro). Já quando o par S2-S4 é acionado à corrente percorrendo outro caminho fazendo com que o motor gire no sentido contrário (circuito à direita).

Podemos ver em vermelho (1) na figura 3.5 as entradas para os motores. Destacado em azul (2) temos a entrada de alimentação de 5V (3), a entrada que suporta tensões de 6V à 35V (3) e o **terra** (**4**). Neste trabalho é utilizada uma bateria de $9\,V$, desta forma a alimentação é feita pela entrada de 6V à 35V.

Indicado em verde (5) vemos a ativa 5V (*Enable 5V*), caso seja utilizado um *jumper* na mesma, a placa utilizará o regulador de tensão integrado para fornecer $5\,V$ (na porta $5\,V$) quando a porta $6\,V - 35\,V$ estiver sendo alimentada por uma tensão entre $6\,V$ e $35\,V$. Neste caso, não se deve alimentar a porta $5\,V$ pois pode danificar os componentes. As portas ativa A (*Enable A*) e ativa B (*Enable B*) são usadas para modular a tensão enviada aos motores e aumentar e diminuir o giro de forma gradativa. Quando estiverem sendo usados *jumpers*, os motores A e B são acionados com velocidade máxima. Para controlar a velocidade do motor é preciso remover os *jumpers* e alimentar o pino com uma tensão entre $0\,V$ e $5\,V$, onde $0\,V$ é a velocidade mínima (parado) e $5\,V$ a velocidade máxima. Neste trabalho, os *jumpers* de todas as portas foram mantidos.

As entradas indicadas em roxo (6) são para controlar o sentido de giro dos motores, IN1 e IN2 para o motor B e IN3 e IN4 para o motor A. Dessa forma, no lugar das chaves S1-S3 temos o pino IN1 e para S2-S4 temos o pino IN2. A Tabela 3.2 mostra como funciona o controle de giro do motor A. O comportamento é o mesmo para as entradas IN3 e IN4 para o motor B.

A Tabela 3.3 mostra as especificações para este módulo.

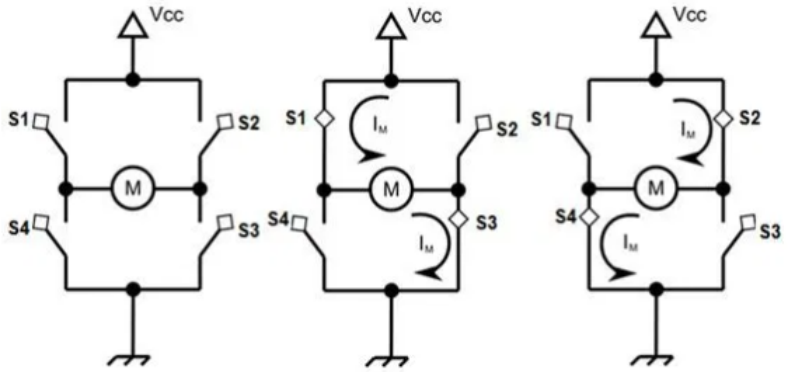

Figura 3.4: Circuito que compõe uma Ponte H - Fonte: https://www.filipeflop.com/[4].

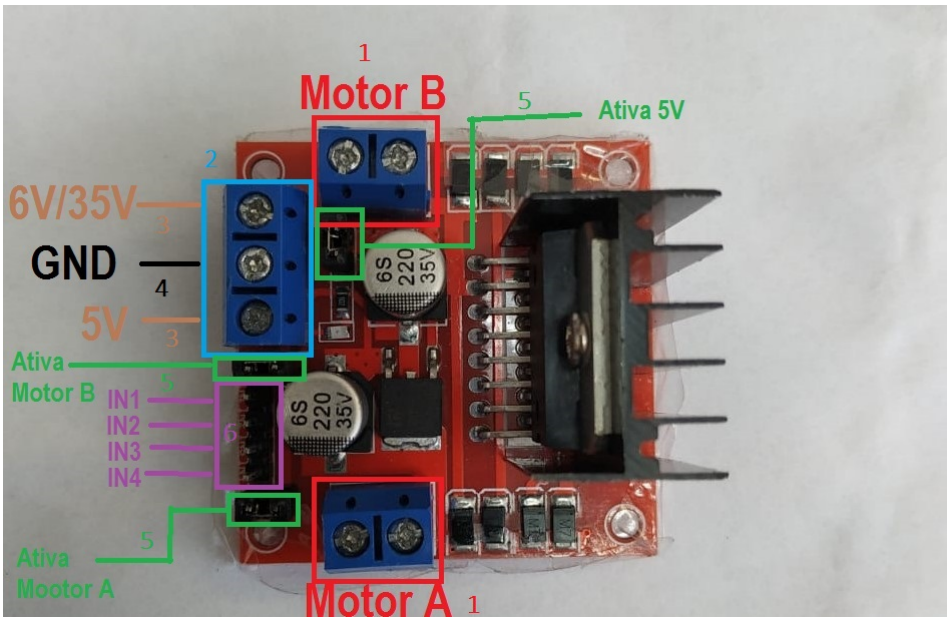

Figura 3.5: Módulo Ponte H - L298N.

Tabela 3.2: Combinações dos pino IN1 e IN2 para controle de giro do motor.

IN1	**IN2**	**Motor**
$0\,V$	$0\,V$	Desligado
$5\,V$	$0\,V$	Giro em um sentido
$0\,V$	$5\,V$	Giro em sentido oposto
$5\,V$	$5\,V$	Freio

Fonte: https://www.filipeflop.com/[13]

Tabela 3.3: Especificações do módulo Ponte H - L298N.

Tensão máxima de alimentação	$7\,V$ - $35\,V$
Tensão de operação	$5\,V$
Corrente máxima por canal	$2\,A$
Corrente máxima por entrada digital	$36\,mA$
Potência máxima dissipada	$25\,W$
Dimensões	$43\,mm$ x $43\,mm$ x $27\,mm$
Massa	$30\,g$

Fonte: https://www.filipeflop.com/[13]

3.3 Módulo *Bluetooth*

O módulo *bluetooth* é utilizado para a comunicação entre o *Arduino* e o aplicativo *Virtuino* utilizado para controlar o carro.

Na Figura 3.6 podemos ver os pinos destacados em roxo (1). Temos o pino de alimentação com recomendação de uso de 3.3V (2), o pino de **terra** (**3**), os pinos TXD (4) (*Transmitter*) e RXD (5) (*Receiver*) para trasmissão e recepção de dados. Destacado em amarelo (6) temos o pino key, utilizado para fazer com que o módulo *blueetooth* possa receber comandos via IDE quando ligado em nível lógico alto e se comunicar com um dispositivo pareado a ele quando estiver em nível lógico baixo; e o pino LED que não foram utilizados neste trabalho.

O módulo recebe as informações do *Virtuino* pelo pino RXD e trasmite dados atraves do pino TXD. As características do módulo *bluetooth* estão na Tabela 3.4.

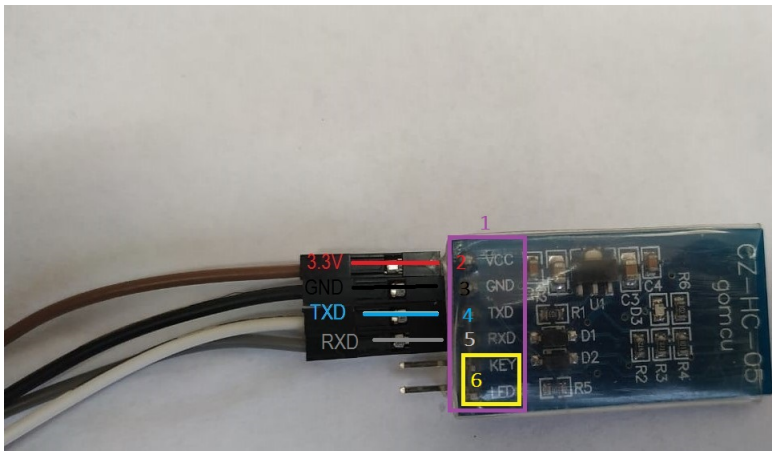

Figura 3.6: Módulo *bluetooth*.

Tabela 3.4: Especificações do módulo *blueetooth* - HC05.

Tensão de operação	$3.3\,V$ - $6\,V$
Corrente de operação	$30\,mA$
Distância máxima de operação	$10\,m$
Taxa de trasmissão	9600 - 1382400

Fonte: https://www.vidadesilicio.com.br/[14]

3.4 Módulo Seguidor de Linha

O módulo seguidor de linha de 3 canais consiste em três fototransistores, três emissores infravermelhos e três LEDs, destacados em **preto** na Figura 3.7. Os emissores enviam luz em frequências abaixo do vermelho que refletem em superfícies. Caso a superfície refletora seja voltada para cor preta, ou o módulo estiver a uma distância que os fototransistores não detectem reflexão, os LEDs permanecem apagados, caso contrário os LEDs acendem.

A Figura 3.8 mostra a parte posterior do seguidor de linha. Destacado em roxo (1) temos os pinos com fios conectados. Os fios vermelho (2), conectado ao pino R, laranja (3), conectado ao pino C e amarelo (4), conectado ao pino L. Estes pinos see comunicam com o conjunto de fototransistores, emissores e LED direito, central e esquerdo, respectivamente. Já o fio verde (5) está ligado ao terra e o fio azul (6) ao VCC (pino de alimentação). Na Tabela 3.5 temos as especificações deste módulo.

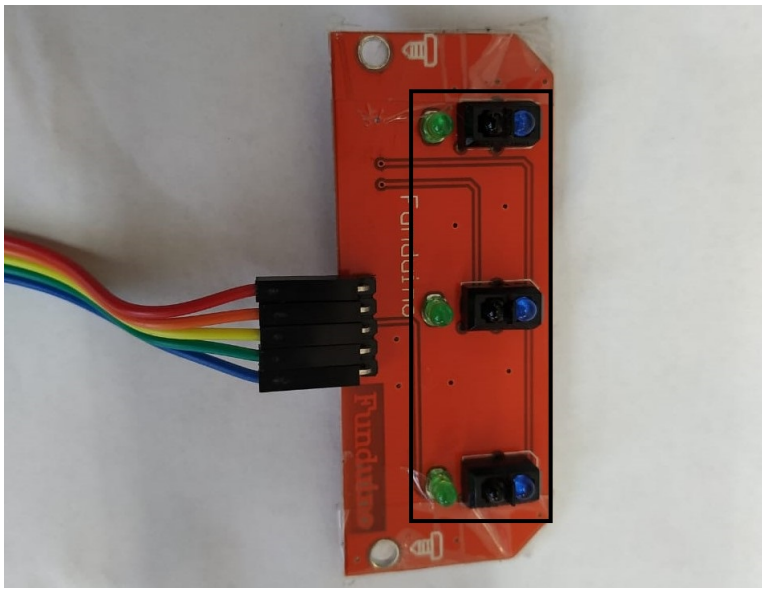

Figura 3.7: Parte frontal do módulo seguidor de linha.

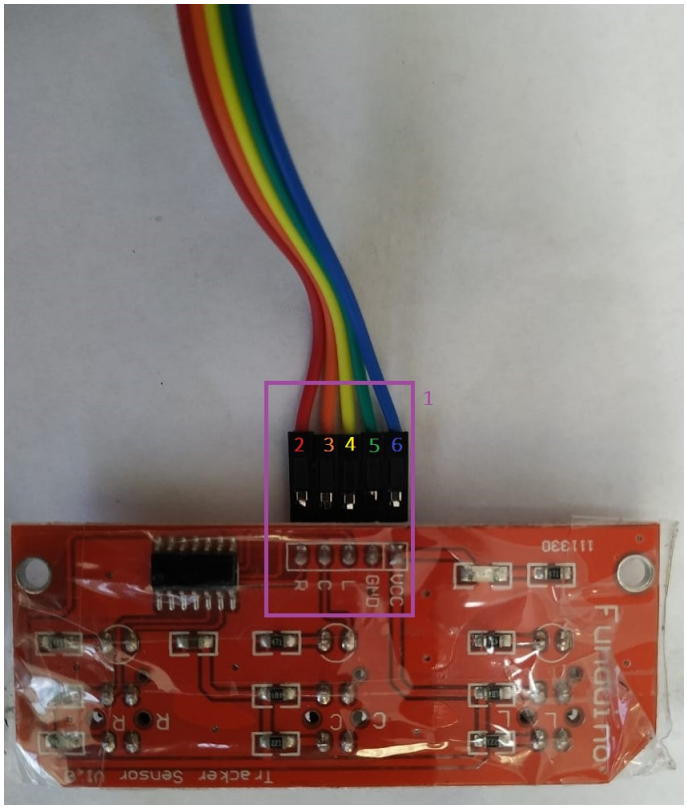

Figura 3.8: Parte posterior do módulo seguidor de linha.

Tabela 3.5: Especificações do módulo seguidor de linha.

Tensão de operação	$5\,V$
Corrente de operação (max.)	$75\,mA$
Distância máxima de detecção	$150\,mm$
Dimensões	$67\,mm$ x $27\,mm$

Fonte: http://www.robotpark.com/[15]

3.5 Módulo Sensor Ultrassônico

O sensor ultrassônico HC-SR04 é composto por um emissor e um receptor ultrassônico. O emissor envia um pulso durante $10\mu s$ indicando o início da trasmissão, em seguida são enviados 8 pulsos com frequência de $40\,kHz$. O sensor aguarda, em nível alto, retorno dos pulsos enviados que, após refletirem no obstáculo, retornam ao sensor e este retorna para nível baixo. Conhecida a velocidade da onda no ar e cronometrado o tempo de retorno, a distância é calculada segunda a equação:

$$d = \frac{v \times t}{2}$$

onde d é a distância do sensor ao obstáculo, v é a velocidade da onda no ar e t o tempo decorrido entre o envio dos 8 pulsos e o retorno dos pulsos refletidos no obstáculo. A Figura 3.9 mostra um exemplo do funcionamento do sensor e a Figura 3.10 mostra o comportamento do mesmo.

Deve-se ter atenção quanto as dimensões do obstáculo e a inclinação do sensor em relação ao obstáculo. Este deve ter ao menos uma área de reflexão de $0.5\,m^2$ e o ângulo em torno de 15°.

Na figura 3.11 se destaca em amarelo (1) os pinos do módulo. Vemos o pino de alimentação de 5V (2), o pino Trigger (3) responsável pelo emissor da onda, o pino Echo (4) responsável pelo receptor da onda e o pino **terra** (**5**).

A Tabela 3.6 mostra as especificações para este módulo.

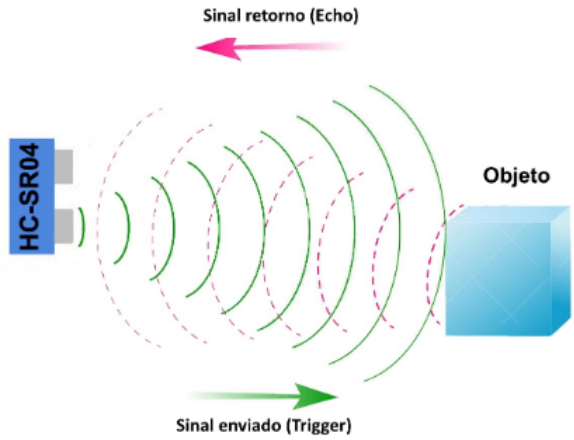

Figura 3.9: Exemplo de detecção de obstáculo - Fonte: www.filipeflop.com[5].

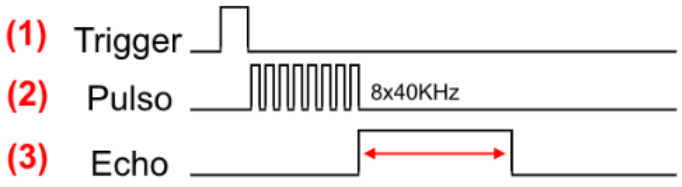

Figura 3.10: Comportamento do módulo sensor ultrassônico - Fonte: www.filipeflop.com[5].

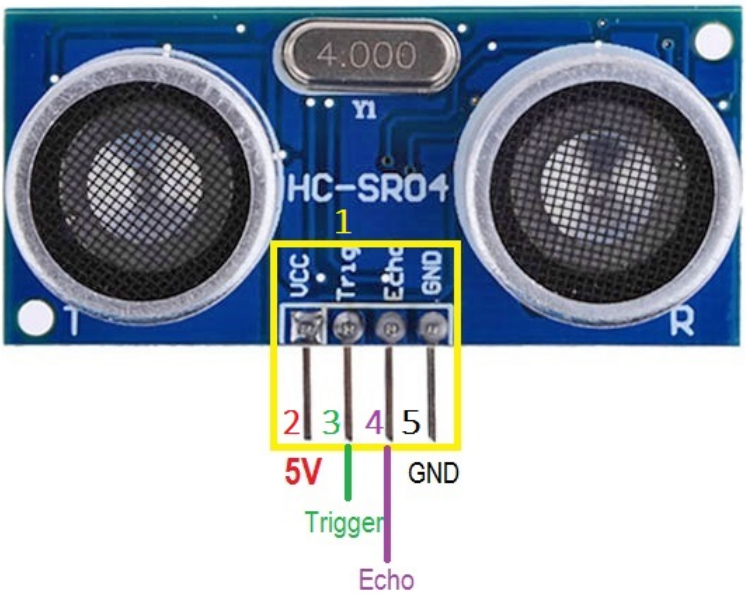

Figura 3.11: Módulo sensor ultrassônico HC-SR04 - Fonte: www.adrobotica.com[6].

Tabela 3.6: Especificações do módulo sensor ultrassônico.

Tensão de operação	$5\,V$
Corrente de operação (max.)	$15\,mA$
Distância de detecção	$2\,cm$ - $400\,cm$
Ângulo eficaz	$\leq 15°$
Resolução	$0.3\,cm$
Dimensões	$45\,mm$ x $20\,mm$ x $15\,mm$

Fonte: https://www.adrobotica.com/[6]

3.6 Telas OLED

Uma tela ou *display* OLED (*organic light-emitting diode*; diodo emissor de luz orgânico) é um Diodo emissor de luz (LED) em que a camada de emissão eletro-luminescente são filmes orgânicos que emitem luz em resposta a uma corrente elétrica que flui entre anodo e catodo. Esta camada de semicondutor orgânico fica situada entre dois eletrodos.

Os OLEDs podem ter duas ou três camadas de material orgânico. A Figura 3.12 mostra a estrutura de um *display* OLED.

As telas OLED utilizadas são vistas frontalmente na Figura 3.13 e suas partes posteriores na Figura 3.14.

Na Figura 3.13 foram destacados os pinos de ambas as telas em roxo (1) e na tela da esquerda são destacados os pinos de **terra** (**2**), alimentação 3V/5V (3), SDL - *Serial Clock* - (4) e SDA - *Serial Data* - (5). O pino SDA é utilizado para receber dados do arduino e o pino SDL para sincronizar a comunicação entre as telas e o *Arduino*.

Conectar as telas ao *Arduino* fará com que ambas recebam os mesmos dados, para que as telas mostrem conteúdos diferentes é necessário mudar o endereço de uma delas. Este endereçamento é feito por meio de um resistor visto na Figura 3.14. Na tela da direita vemos em vermelho o resistor conectado ao endereço original (0x7A), já na tela da direita vemos em **preto** o resistor mudado para outro endereço (0x78). Vale notar que, enquanto nas telas os endereços são os mencionados anteriormente, na *Arduino* devemos utilizar os endereço 0x3C, para o *display* com endereço 0x78, e 0x3D, para o *display* de endereço 0x7A, na programação. A razão para esses endereços contraditórios é a maneira como o Arduino interpreta os endereços I2C. Por padrão, o I2C usa endereços de 7 bits, mas o Arduino converte isso em um valor de 8 bits deslocando os bits uma posição para a esquerda.

Na Tabela 3.7 temos as características dos *displays* utilizados.

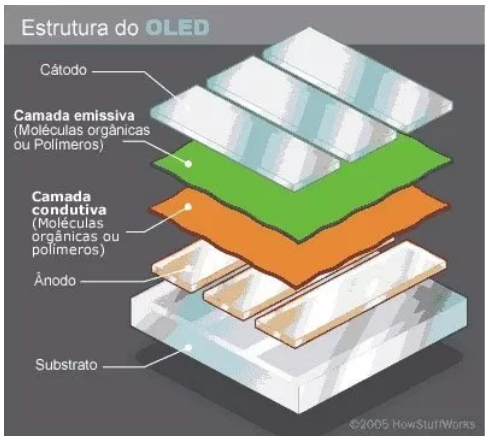

Figura 3.12: Estrutura das telas OLED - Fonte: https://www.arealocal.com.br/[7].

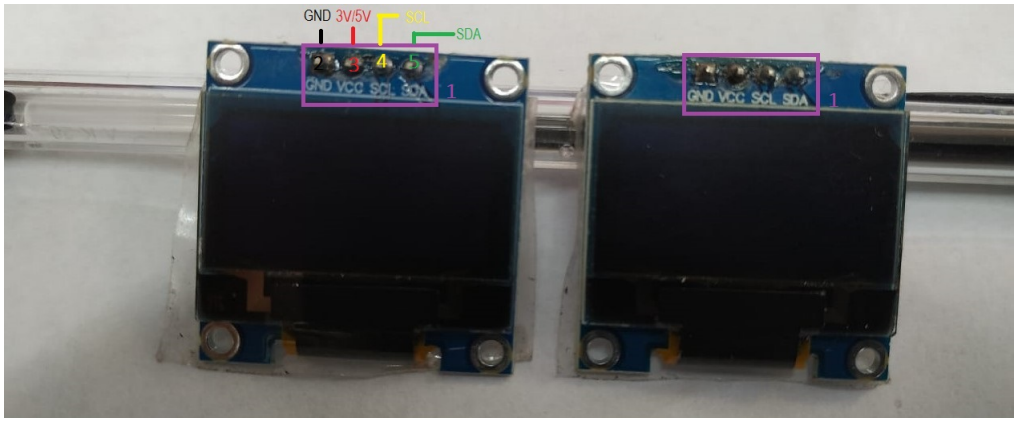

Figura 3.13: Parte frontal das telas OLED.

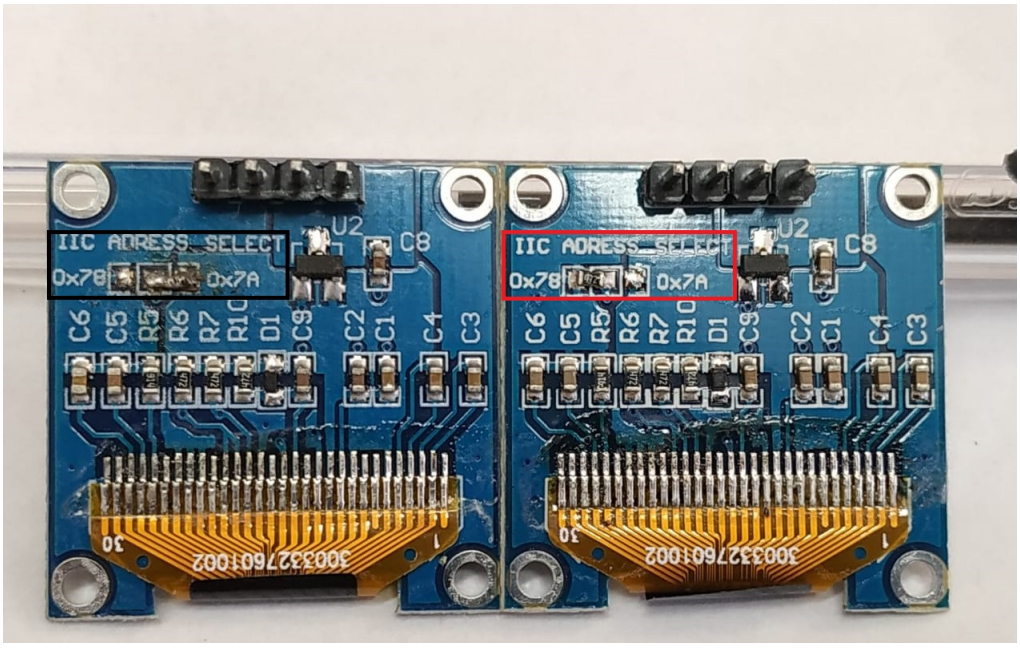

Figura 3.14: Parte posterior das telas OLED.

Tabela 3.7: Especificações dos *displays* OLED.

Conecção	I2C
Área visível	$0.96''$
Resolução	128 x 64
Controlador	SSD1306
Tensão de alimentação	$3.3\,V$ - $5\,V$
Tensão de I/O	$3.3\,V$ - $5\,V$
Dimensões	$28\,mm$ x $27\,mm$

Fonte: http://www.squids.com.br/[16]

3.7 Aplicativo *Virtuino*

Para a comunicação e controle do carro através de *smartphone* foi escolhida a plataforma de prototipação *Virtuino* [17]. Foi escolhido o *Virtuino* por possibilitar prototipações para *smartphones* com sistema operacional *Android*, ter licença gratuita e uma interface amigável.

O aplicativo possui uma interface simples como visto na Figura 3.15. Indicado em vermelho (1) temos a opção de ferramentas (visto nas Figuras 3.21 e 3.22), em laranja (2) temos opções de mudança de *layout* da tela inicial, em cinza (3) temos o bloqueio do painel para terminar as edições e iniciar o uso do aplicativo; em roxo temos um menu de opções (4).

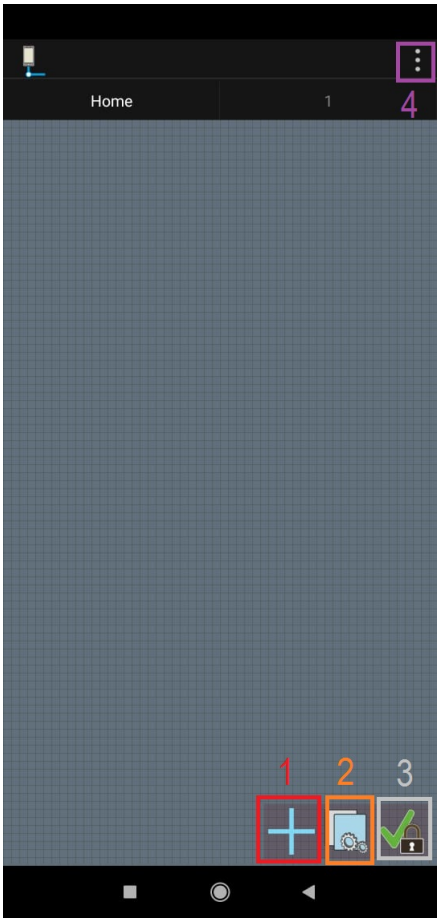

Figura 3.15: Tela inicial do *Virtuino*.

A Figura 3.16 mostra o menu de opções, as mais utilizadas aqui são a abertura de novo projeto (3), conectar (1) a um dispositivo (no caso, a comunicação com a *Arduino*), a configuração de servidor (2) e o **desbloqueio do painel** (**4**) para fazer novas edições caso necessário.

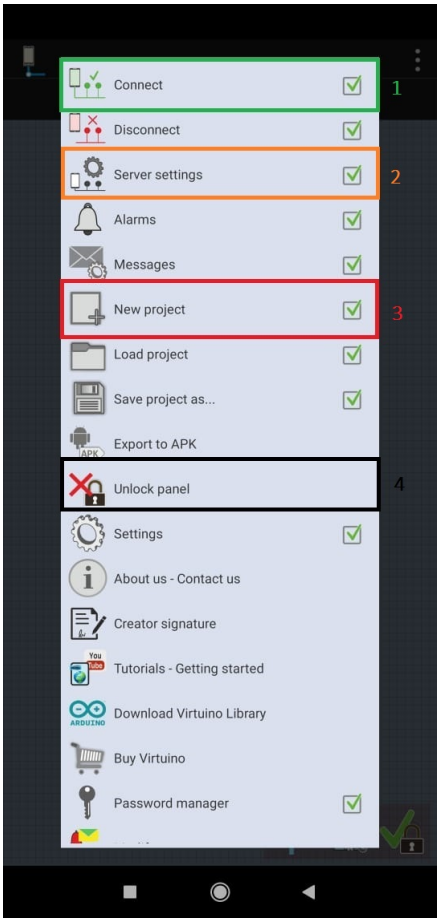

Figura 3.16: Menu de opções.

Para que o aplicativo se comunique com a placa é necessário adiciona-la como opção de servidor. No menu de opções, indo na opção configuração de servidor (2) chegamos na tela de escolha de servidor, vista na Figura 3.17. Na parte inferior, em vermelho, temo a opção de adicionar uma placa, ao seleciona-la chegamos a tela vista na Figura 3.18. Selecionando a opção destacada em amarelo para adicionar uma placa arduino por *bluetooth*, é necessário escolher qual das opções de *bluetooth* disponíveis para o celular será usada; na Figura 3.19 seleciona-se o módulo *bluetooth* ligado à arduino (destacado em **preto**) e por fim, na Figura 3.20 seleciona-se a opção de placa *Arduino Mega* (em roxo).

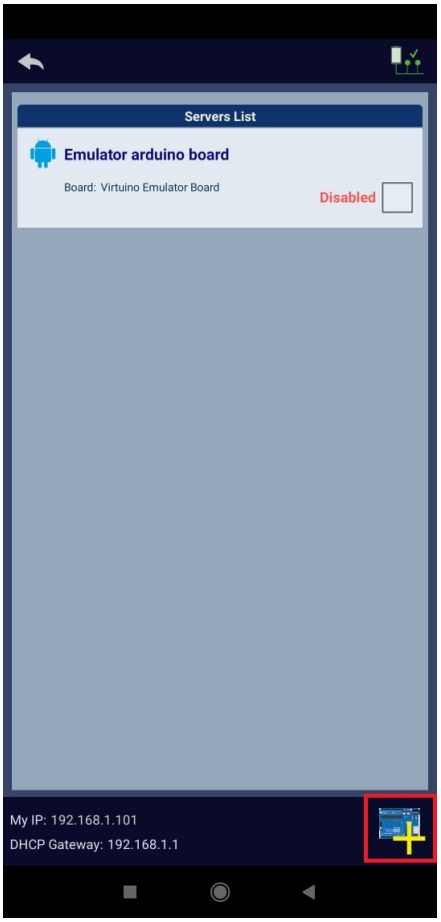

Figura 3.17: Escolha de servidor.

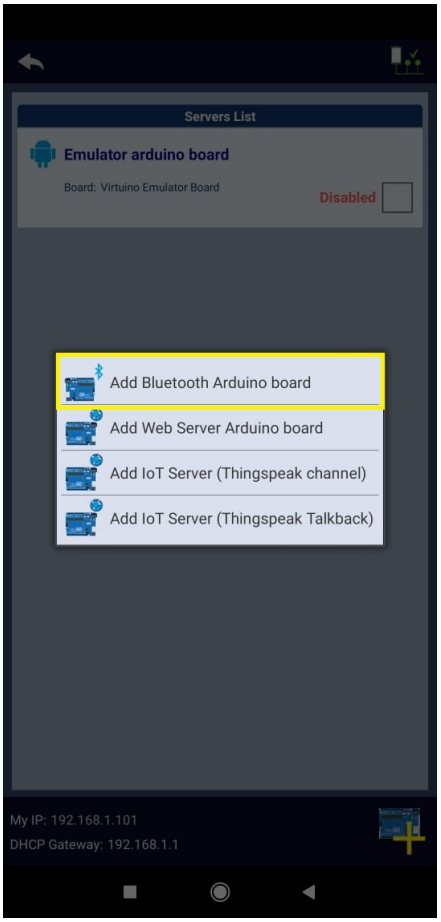

Figura 3.18: Escolha do tipo de placa a ser adicionada.

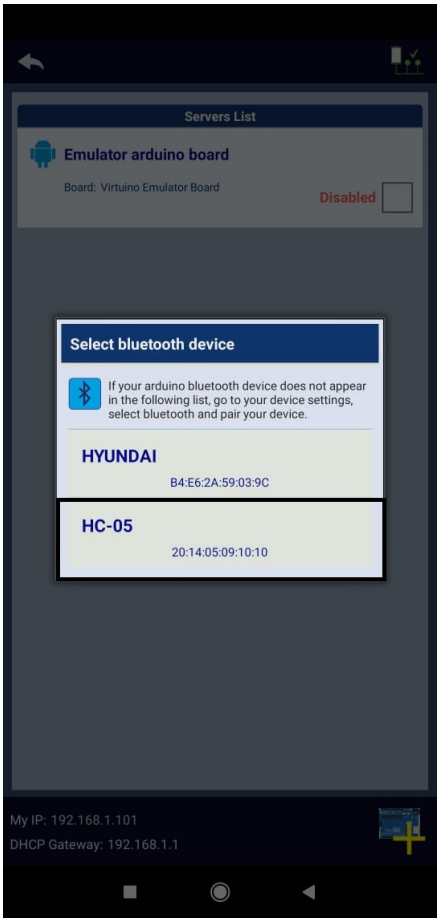

Figura 3.19: Escolha do módulo *bluetooth* para comunicação.

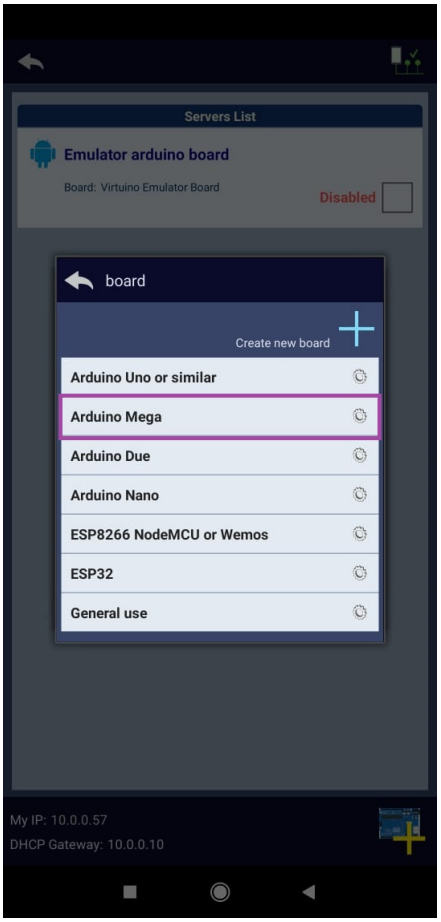

Figura 3.20: Seleção da placa a ser utilizada no projeto do *Virtuino*.

Uma vez que a escolha de servidor foi feita, foi iniciado o desenvolvimento do controle remoto para o carro.

Na Figura 3.21 vemos a tela após a seleção da opção 1 da Figura 3.15. As opções de ferramentas são divididas em duas abas, como destacado em amarelo (*Level 1*) e roxo (*Level 2*). Selecionando a opção em vermelho (3) vamos a tela mostrada na Figura 3.23.

Para a adição de botões podemos ver na Figura 3.23 o que é necessário. Indicado em vermelho (2) está a opção do servidor (no caso a placa *Mega*) em amarelo (3) deve ser selecionado o pino digital ao qual o botão será relacionado (D13 indica que o botão criado no *Virtuino* irá se comunicar com o pino 13 da *Mega*), em **preto** (**4**) está a opção de selo de imagem que irá aparecer no botão. Destacado em roxo (5) está a posição na tela em que o botão irá aparecer, isso pode ser feito também direto no painel, segurando o ícone do botão e arrastando para onde desejarmos, enquanto a tela estiver desbloqueada.

Na cor verde (6) temos o tamanho do botão e em laranja (7) selecionamos o tipo de botão (*switch*, *push button*, etc.). Já na cor azul (8) podemos indicar um atraso entre apertarmos o botão e o aplicativo enviar o sinal para a placa. No canto superior

direito, destacado em marrom (1) temos as opções de salvar a edição do botão montado ou descarta-lo.

Por fim, com o modelo do controle pronto podemos vê-lo finalizado na Figura 3.24, em vermelho (2) temos os direcionais (do tipo *push button*) e em amarelo (1) o botão do tipo *switch* que alterna entre os modos controle e autônomo.

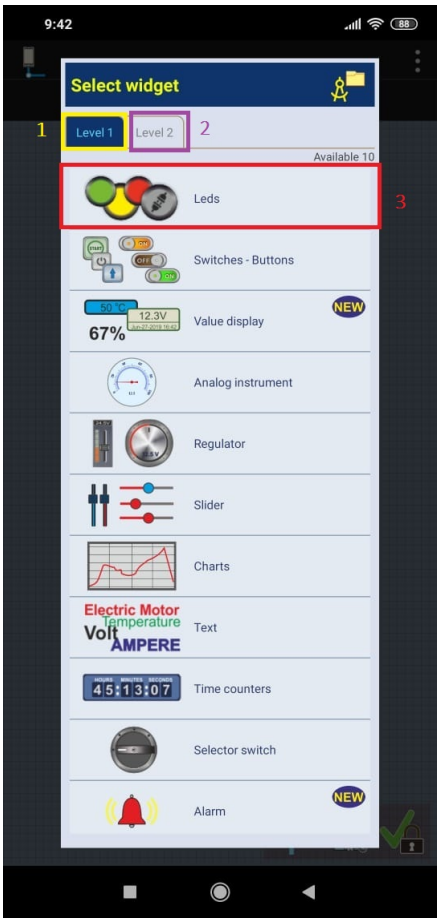

Figura 3.21: Tela de opção para adição de ferramentas.

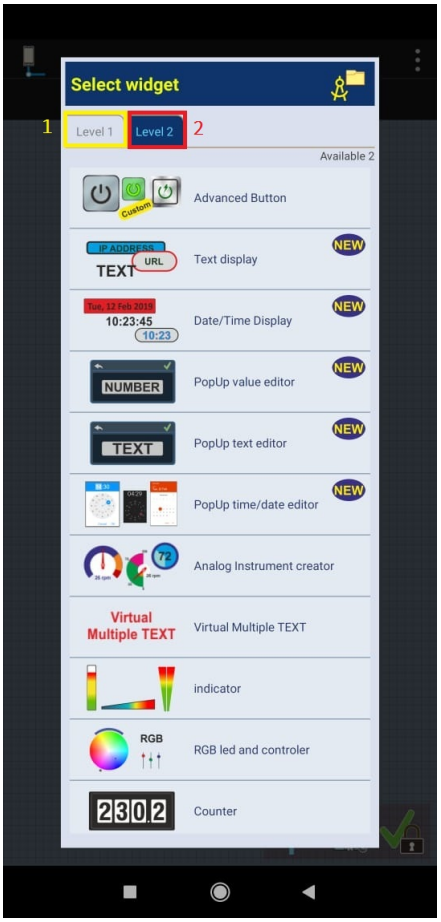

Figura 3.22: Outras opções de ferramentas.

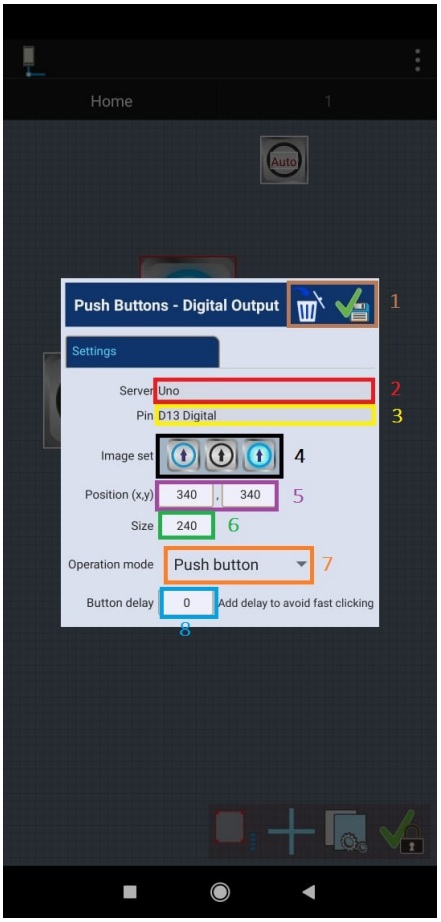

Figura 3.23: Edição de botões.

Figura 3.24: Controle finalizado no *Virtuino*.

3.7.1 Modelagem do Projeto

Conhecidos os componentes do projeto, para fazer uma prospecção de como integra-los temos o diagrama de blocos visto na Figura 3.25. Vemos que o plano é termos a bateria de $9V$ alimentando a *Arduino* e a ponte-H. Por sua vez, a ponte-H controla os motores seguindo comandos enviados a ela pela *Arduino*.

A placa *Arduino* envia à ponte-H infirmação para mover os motores e com qual potência eles devem operar. Ela também envia às telas o que devem a informação a ser impressa nas mesmas. O sensor de ultrassom e o seguidor de linhas devem enviar informação à *Arduino* para que esta atue na ponte-H. Já o módulo *bluetooth* media o envio e recebimento de informação entre a placa *Arduino* e o aplicativo *Virtuino*.

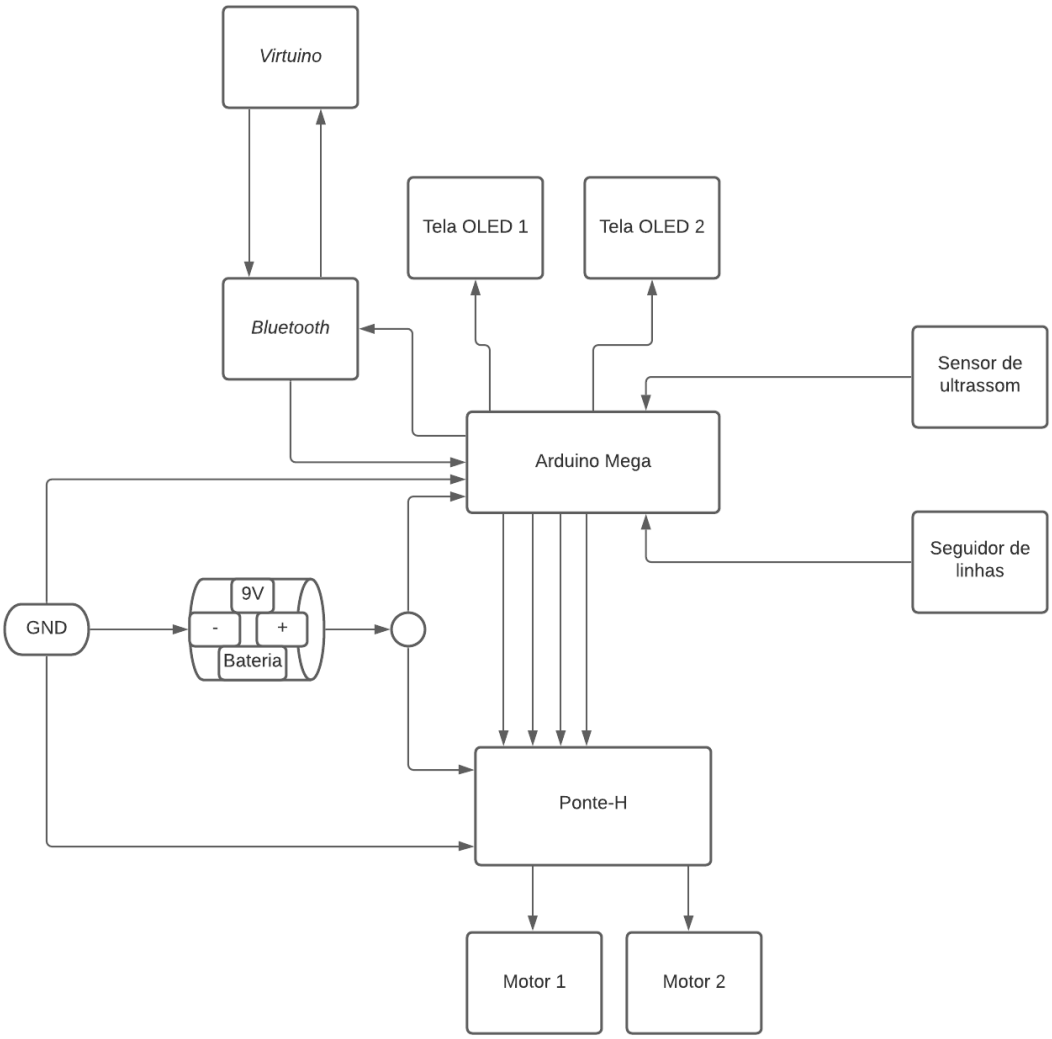

Figura 3.25: Diagrama de blocos para modelar o projeto final.

Capítulo 4

Testes dos Componentes

A fim de atestar o funcionamento de cada um dos módulos, foram feitos testes tanto dos módulos em separado quanto em conjunto. Pequenos programas foram utilizados como testes até que se chegasse ao entendimento do funcionamento de cada um dos componentes e se estes estão de acordo com o que se espera para este trabalho.

4.1 Teste da *Arduino Mega*

Para o teste da placa *Mega* utilizou-se um programa já presente na IDE do *Arduino*. Seguindo o caminho *Arquivo -> Exemplos -> Basics - Blink* encontra-se um código para piscar um LED ligado a porta 13 a cada 1 segundo, visto na figura 4.1. Na linha 4, ao pino 13 (definido como *LED_BUILTIN*) é atribuido o modo saída (função pinMode), nas linhas de 9 à 12 é descrita a lógica para acender e apagar o LED (função digitalWrite) e incluindo um atraso de 1 segundo após cada mudança no estado do LED (função delay).

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Figura 4.1: Código teste para piscar um LED.

4.2 Teste do Seguidor de Linha

O teste do seguidor de linha foi feito seguindo o esboço de circuito visto na Figura 4.2 e com o código visto na Figura 4.3. Nele vemos os pinos 4 à 6 sendo definidos, respectivamente, como os sensores esquerdo, central e direito nas linhas de 2 à 4 e estes pinos são declarados no modo entrada (INPUT) nas linhas 9 à 11. Assim, as leituras são feitas pelos sensores e retornadas para as variáveis declaradas nas linhas 17 à 19 *leftSensor, centerSensor, rightSensor*. As linhas finais foram adicionadas para escreverem na janela serial da IDE o valor 0 quando o sensor não detecta reflexão, e o valor 1 quando detecta. O método na linha 8 (Serial.begin()) "configura a taxa de transferência em bits por segundo (*baud rate*) para transmissão serial"[3].

Assim que o código foi passado para a placa, foi aproximado um objeto até o sensor esquerdo e as respostas foram visualizadas na janela serial.

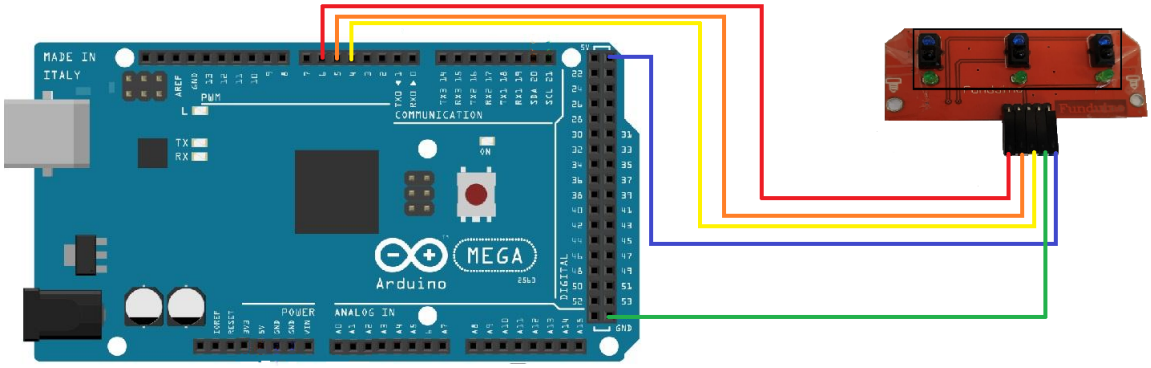

Figura 4.2: Esboço do circuito com seguidor de linha.

```
// connect the sensors to digital pins
#define LEFT_SENSORPIN 4
#define CENTER_SENSORPIN 5
#define RIGHT_SENSORPIN 6

void setup()
{
  Serial.begin(9600);
  pinMode(LEFT_SENSORPIN,INPUT);
  pinMode(CENTER_SENSORPIN,INPUT);
  pinMode(RIGHT_SENSORPIN,INPUT);
}

void loop()
{
  // read input from sensors
  byte leftSensor=digitalRead(LEFT_SENSORPIN);
  byte centerSensor=digitalRead(CENTER_SENSORPIN);
  byte rightSensor=digitalRead(RIGHT_SENSORPIN);

  Serial.print(" Left : ");
  Serial.print(leftSensor);
  Serial.print(" Centre : ");
  Serial.print(centerSensor);
  Serial.print(" Right : ");
  Serial.print(rightSensor);
  Serial.println();
  delay(1000);
}
```

Figura 4.3: Código teste para testar o módulo seguidor de linha.

4.3 Teste do Módulo *Bluetooth* e *Virtuino*

O teste do módulo *bluetooth* foi feito em conjunto com o *Virtuino*[17], para isso foi montado um circuito com um LED ligado à porta 13 e ao terra, dado o esboço na Figura 4.4, e feito o código visto na Figura 4.5. Como visto nesta última, foi necessário incluir duas bibliotecas para a utilização do aplicativo (#include¨VirtuinoBluetooth.h¨) e do módulo *bluetooth* (#include <SoftwareSerial.h>), além disso é necessário declarar em quais pinos da placa serão conectados os pino TXD e RXD do *bluetooth* como é visto na linha 3, e declarar qual a taxa de transferência em bits por segundo para o aplicativo se comunicar via *bluetooth* (linha 4).

Seguindo temos a definição das portas 8 e 13 para as variáveis *Switch* e *Resposta*, ambas foram colocadas no modo de saída (OUTPUT, linhas 13 e 14). O método *virtuino.run()* é necessário para que o aplicativo seja inicializado. Na lógica de funcionamento, a função digitalRead(Switch) lê o valor da tensão na porta 8, caso esteja em tensão alta (HIGH), indica que o botão no aplicativo que se comunica com esta porta foi acionado, então a função digitalWrite(LED, HIGH) passa a porta 13 para tensão alta (5V), caso contrário a porta 13 fica em tensão baixa (0V).

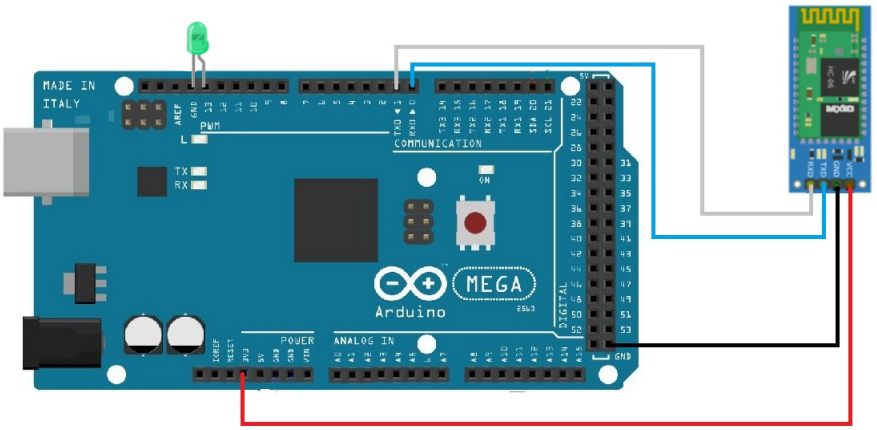

Figura 4.4: Esboço do circuito para piscar um LED via comando do celular pelo *Virtuino*.

```
#include "VirtuinoBluetooth.h"
#include <SoftwareSerial.h>
SoftwareSerial bluetoothSerial =  SoftwareSerial(0,1);
VirtuinoBluetooth virtuino(bluetoothSerial,9600);
#define Switch 8
#define LED 13

void setup() {
  // put your setup code here, to run once:
  virtuino.DEBUG = true;              // Set this value
  //Serial.begin(9600);               // Habilitar esta
  bluetoothSerial.begin(9600);        // Disable this li
  virtuino.vPinMode(Switch, OUTPUT);
  pinMode(LED, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

  virtuino.run();

 if(digitalRead(Switch) == HIGH){
     digitalWrite(LED, HIGH);
 }

 else{
     digitalWrite(LED, LOW);
 }
```

Figura 4.5: Código teste para piscar um LED via comando do celular pelo *Virtuino*.

4.4 Teste do Sensor Ultrassônico

O sensor ultrassônico foi testado conectandoo à *Mega* com o código da Figura 4.7, obtido no *site filipeflop*[5]. O esboço do circuito montado pode ser visto na Figura 4.6. Novamente foi necessária a inclusão de uma biblioteca (#include <Ultrassonic.h>) para a comunicação da *Mega* com o sensor. Os pinos *Echo* e *Trigger* do sensor foram ligados às portas 4 e 5 respectivamente e inicializados na linha 8.

As variáveis *cmMsec* e *inMsec* recebem o valor da distância lida no método *ultrassonic.convert*, a primeira recebe a distância em centímetros e a segunda em polegadas, o método *ultrassonic.convert* utiliza o tempo de resposta entre a onda enviada e recebida pelo sensor, calcula a distância e converte para a unidade desejada. As medições são mostradas na tela serial da IDE a cada 1 segundo.

Aproximando e afastando um objeto do sensor, as medições de distância foram observadas na janela serial.

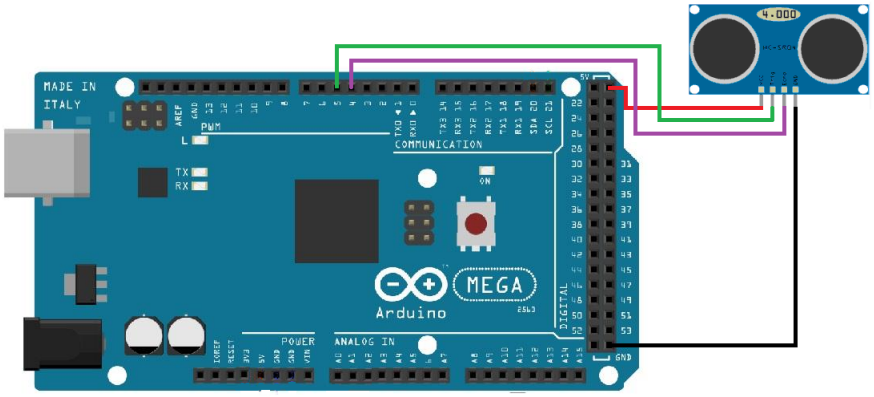

Figura 4.6: Esboço do circuito com sensor ultrassônico.

```
#include <Ultrasonic.h>

//Define os pinos para o trigger e echo
#define pino_trigger 4
#define pino_echo 5

//Inicializa o sensor nos pinos definidos acima
Ultrasonic ultrasonic(pino_trigger, pino_echo);

void setup()
{
  Serial.begin(9600);
  Serial.println("Lendo dados do sensor...");
}

void loop()
{
  //Le as informacoes do sensor, em cm e pol
  float cmMsec, inMsec;
  long microsec = ultrasonic.timing();
  cmMsec = ultrasonic.convert(microsec, Ultrasonic::CM);
  inMsec = ultrasonic.convert(microsec, Ultrasonic::IN);
  //Exibe informacoes no serial monitor
  Serial.print("Distancia em cm: ");
  Serial.print(cmMsec);
  Serial.print(" - Distancia em polegadas: ");
  Serial.println(inMsec);
  delay(1000);
}
```

Figura 4.7: Código teste para leitura de distâncias feita pelo sensor ultrassônico.

4.5 Teste da Ponte H

O teste com a ponte H necessitou de dois motores DC, então foram usados os dois motores do carro, conforme visto na Figura 4.8. O código para o teste foi cedido pelo professor Wilson e pode ser visto na Figura 4.9. As portas 8 e 9 foram designadas para o motor das rodas (linhas 2 e 3) e as portas 10 e 11 para o motor de giro da carcaça do carro (linhas 6 e 7).

As variáveis associadas a estas 4 portas foram postas no modo saída (linhas 11 à 14). Na lógica de funcionamento fez-se os motores girarem em um sentido, onde vemos indicado na função digitalWrite(MA_DIR1, LOW) e digitalWrite(MA_DIR2, HIGH) indica que o motor da direita gira em um sentido, em seguida este código foi modifica

trocando-se os argumentos das funções nas linhas 19 à 23 (LOW para HIGH e HIGH para LOW) para inverter o sentido de giro de ambos os motores.

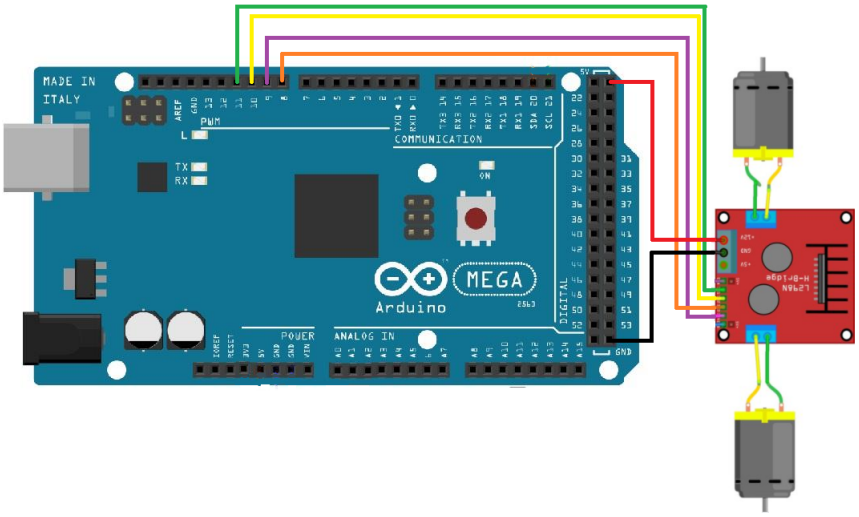

Figura 4.8: Esboço do circuito da ponte H junto com motores DC.

```
// MOTOR DA DIREITA
#define MA_DIR1 8
#define MA_DIR2 9

// MOTOR DA ESQUERDA
#define MB_DIR1 10
#define MB_DIR2 11

void setup()
{
pinMode(MA_DIR1,OUTPUT);
pinMode(MA_DIR2,OUTPUT);
pinMode(MB_DIR1,OUTPUT);
pinMode(MB_DIR2,OUTPUT);
}
void loop()
{

// motor A e B - frente
digitalWrite(MA_DIR1,LOW);
digitalWrite(MA_DIR2,HIGH);

digitalWrite(MB_DIR1,LOW);
digitalWrite(MB_DIR2,HIGH);

delay(500);

}
```

Figura 4.9: Código teste para utilizar a ponte H junto com motores DC.

4.6 Teste das Telas OLED

O teste decidido para as telas OLED foi que as mesmas mostrassem conteúdos diferentes para atestar que a mudança do resistor em uma delas está conectado a um endereço diferente e para ter a certeza de que a tela que teve o resistor mudado não tivesse sido avariada no processo.

A Figura 4.10 mostra o esboço da montagem do circuito. O código usado para o teste é visto na Figura 4.11, onde vemos a inclusão de uma nova biblioteca para a comunicação das telas com a *Mega* (#include <Adafruit_SSD1306.h>)[18]. Na linha 3, a porta 4 da *Mega* foi definida para reiniciar as telas caso necessário. Nas linhas 4 e 5 cada tela é inicializada. As linhas 12 à 14 indicam o endereço ligado à uma das telas (no caso o endereço 0x3D), limpa qualquer informação que possa ter na tela e chama o método

Display1.display(), este método mostra na tela o último conteúdo passado para a tela (no caso, como a tela foi limpa não mostrará nada). Estes procedimentos são repetidos para a segunda tela nas linhas 16 à 18 mas com o endereço 0x3C e com o objeto Display2.

Nas linhas 23 à 30 indicamos, respectivamente, o tamanho do texto a ser escrito em uma das telas, a cor das letras e do plano de fundo, a indicação de posição nos eixos horizontal (eixo X) e vertical (eixo Y) em que será iniciado a escrita de informação na tela (os valores crescem da direita para a esquerda no eixo horizontal e de cima para baixo no eixo vertical) e a escrita na tela da frase ¨Display 1¨. O método *fillCircle* desenha um círculo na tela usando como argumentos a posição no eixo X e a posição no eixo Y para o centro do círculo, o raio da circunferência em pixels e a cor. Um procedimento idêntico é feito para a segunda tela. A variável ***i*** é utilizada para mover o centro do círculo para que dê a impressão de movimento na primeira tela e passando para a segunda tela.

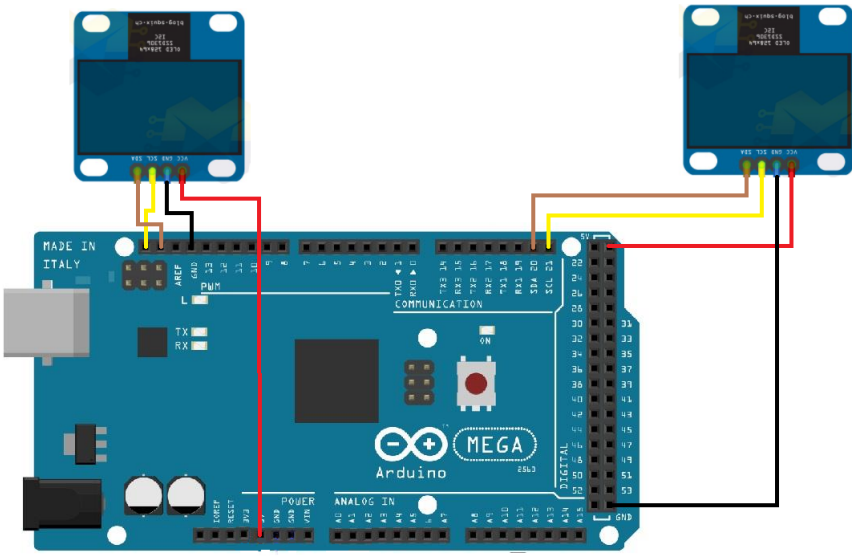

Figura 4.10: Esboço do circuito de telas OLED.

```
#include <Adafruit_SSD1306.h>

#define OLED_RESET 4
Adafruit_SSD1306 Display1(OLED_RESET);
Adafruit_SSD1306 Display2(OLED_RESET);

int i, j;

void setup(){
  //Serial.begin(9600);

  Display1.begin(SSD1306_SWITCHCAPVCC, 0x3D);
  Display1.clearDisplay();
  Display1.display();

  Display2.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  Display2.clearDisplay();
  Display2.display();
}

void loop() {
  for (i = 0; i < 270; i+=10) {

    Display1.setTextSize(2);
    Display1.setTextColor(WHITE, BLACK);
    Display1.setCursor(0, 0 );
    Display1.println("Display 1");
    Display1.fillCircle(i, 30,  10, 1);
    Display1.display();
    Display1.clearDisplay();

    Display2.setTextSize(2);
    Display2.setTextColor(WHITE, BLACK);
    Display2.setCursor(0, 0);
    Display2.println("Display 2");
    Display2.fillCircle(i - 127, 30,  10, 1);
    Display2.display();
    Display2.clearDisplay();
  }
}
```

Figura 4.11: Código teste para as telas OLED.

4.7 Integração dos Módulos

Feitos os testes com os módulos em separado, foi montado o circuito com todos os módulos ao mesmo tempo na *Mega*, como visto na Figura 4.12. Foi passado para um só arquivo os códigos de testes de cada módulo com pequenas auterações iniciais e prosseguiu-se o teste de funcionamento de todos os módulos integrados.

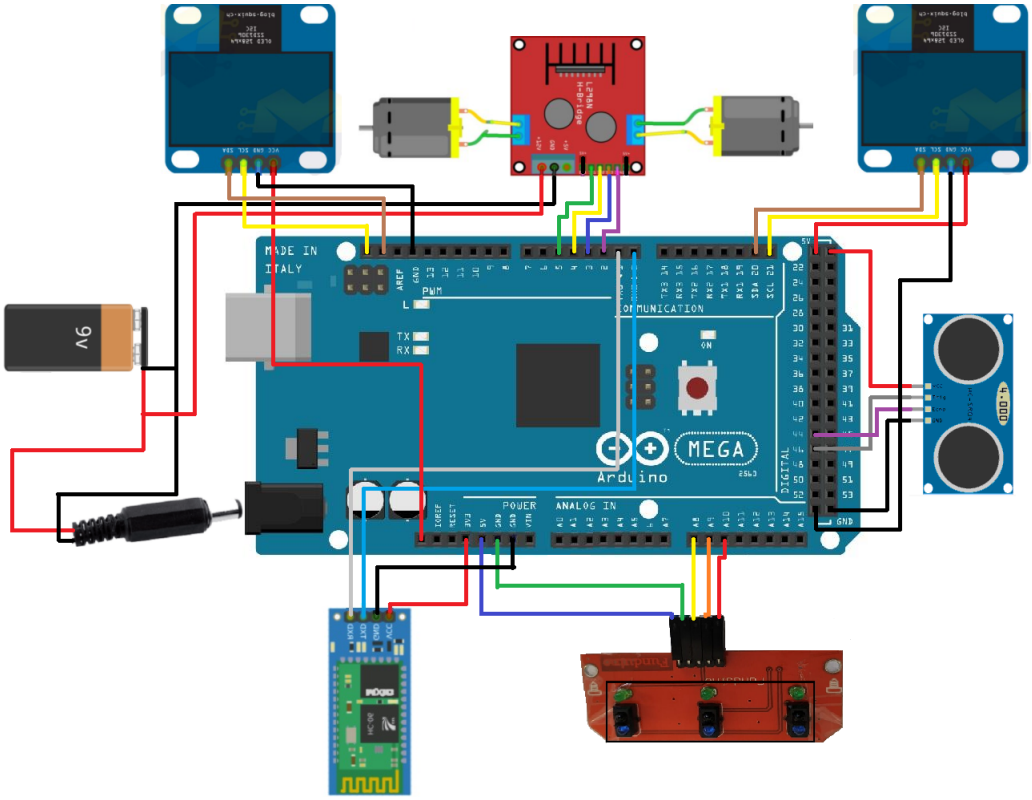

Figura 4.12: Esboço do circuito final do projeto.

No próximo capítulo serão apresentados e discutidos os resultados dos testes feitos até então.

Capítulo 5

O Veículo Desenvolvido

Os resultados de cada teste realizado no capítulo 4 serão mostrados neste capítulo. Serão feitas discussões sobre os resultados, análises dos problemas encontrados e suas soluções.

5.1 Resultado dos Testes dos Módulos em Separado

Os testes de cada módulo em separado foram simples em geral, o teste com a placa *Mega* correreu sem nenhum problema, o LED ligado à porta 13 acendeu e apagou a cada um segundo como esperado. Não houve qualquer comportamento inesperado.

O seguidor de linha também não apresentou qualquer falha ou erro. As respostas do sensor foram mostradas na tela serial e podem ser vistas na Figura 5.1. Nela o sensor esquerdo está em 1, detectando reflexão, e os sensores central e direito em 0. A Figura 5.2 mostra um objeto (borracha) próximo ao seguidor de linha, em frente aos sensores esquerdo e frontal e os LEDs referenste a eles estão acesos. Os três sensores foram testados e funcionaram.

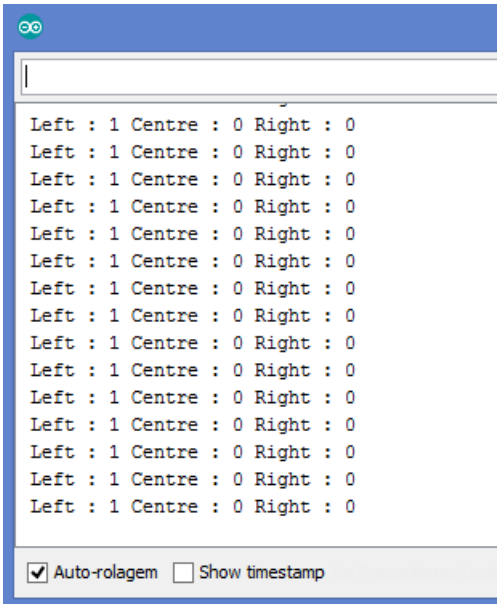

Figura 5.1: Janela serial mostrando algumas detecções do seguidor de linha.

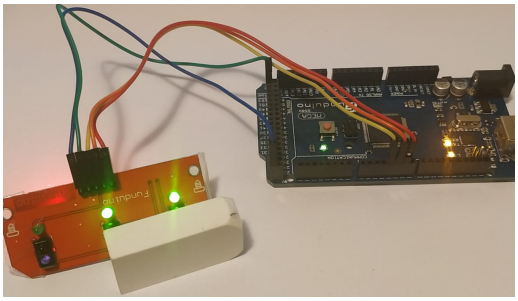

Figura 5.2: Circuito para o teste do módulo seguidor de linha.

O teste com o sensor ultrassônico também foi satisfatório, as medições feitas são mostradas na Figura 5.3. Foi utilizada uma trena para comparar as medidas feitas em centímetros, como podemos ver na Figura 5.4. A figura também destaca papéis com valores indicativos de distância naquele ponto, em vermelho (1) a posição 75 cm em relação à parede, em azul (2) a posição 35 cm e em verde (3) a posição 10 cm. As medições observadas na janela serial foram bem próximas aos valores indicados na trena.

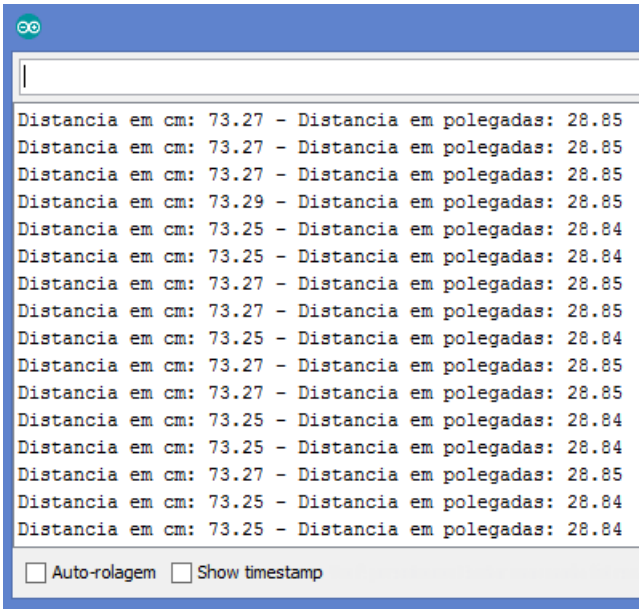

Figura 5.3: Medições feitas pelo sensor ultrassônico.

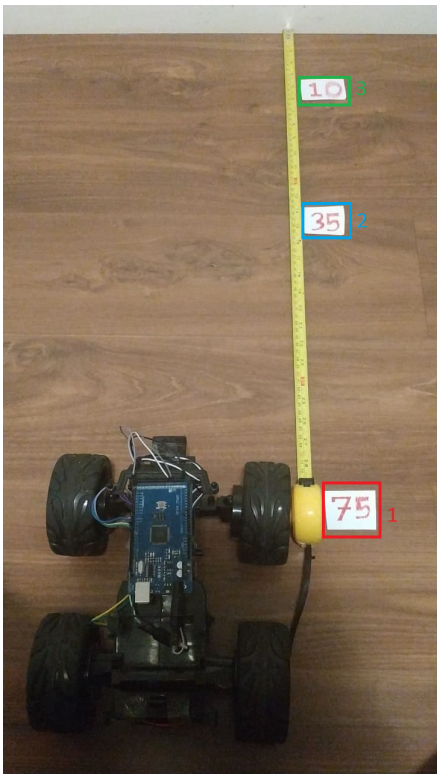

Figura 5.4: Medição feita con uma trena para comparação das distâncias lidas pelo sensor ultrassônico.

A ponte H teve uma resposta correta nos testes de giro de direção, os motores giram em um sentido e após a mudança no código giraram no sentido oposto.

No início, as telas OLED mostravam o mesmo conteúdo usando o código da Figura 4.11. Após feitas pesquisas, foi encontrada a solução de se mudar o resistor de uma delas como foi mencionado na seção 3.6 e obteve-se o comportamento esperado do código. A Figura 5.5 mostra o resultado deste teste. Nela podemos ver as frases distintas *Display 1* e *Display 2*. O código também desenha um círculo passando da direita para a esquerda nas telas. Pela lógica implementada, o círculo chega ao fim de uma das telas e passa a aparecer no início da outra.

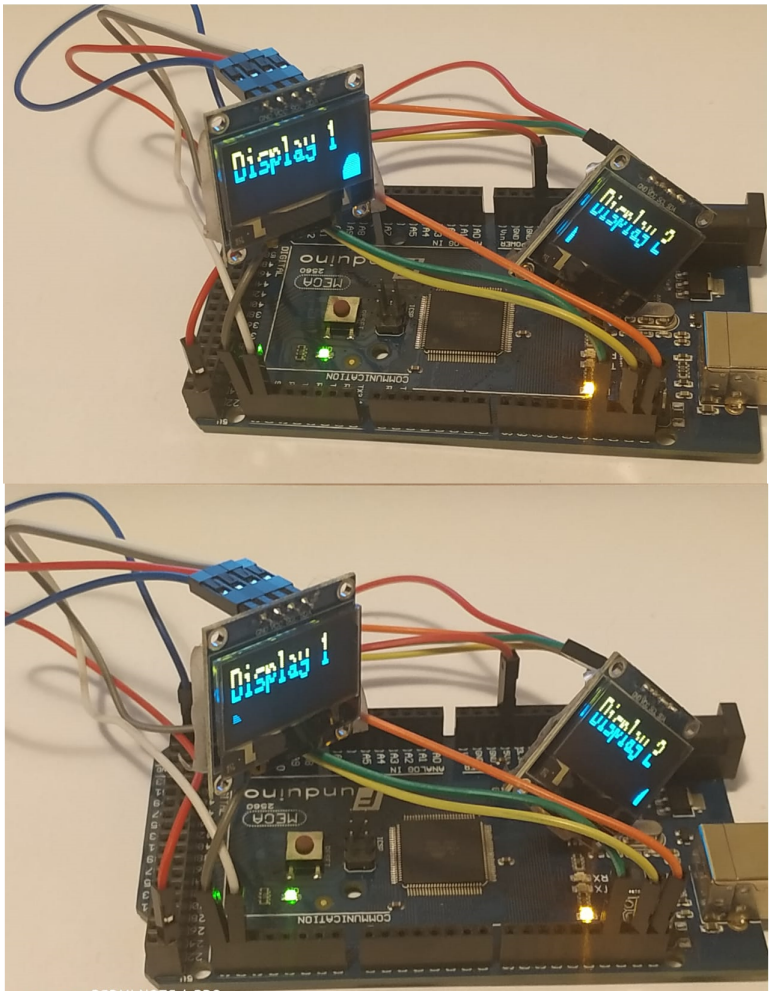

Figura 5.5: Telas OLED mostrando conteúdos diferentes.

O teste envolvendo o módulo *bluetooth* encontrou alguns obstáculos. O circuito não respondia. Não era possível ligar ou ascender o LED e nem controlar os motores nos testes. Uma vez que o teste com a ponte H sem o *bluetooth* havia funcionado, não havia razão para crer ser um problema na *Mega* ou na ponte H. Após pesquisas foi descoberto que os pinos para comunicação serial nativos são os pinos 0, 1, 15, 16, 17, 18 e 19 e a biblioteca *SoftwareSerial.h* foi desenvolvida para permitir a comunicação serial em outros

pinos digitais do *Arduino*, usando o *software* para replicar a funcionalidade (daí o nome "SoftwareSerial"). Dessa forma, a comunicação entre a *Mega* e o *Virtuino* não ocorria.

Analisando os arquivos da biblioteca *VirtuinoBluetooth.h* foi descoberto que para a comunicação ser possível era necessário desabilitar uma linha de código (visto na Figura 5.6) para que a comunicação seja feita via *hardware*, sem uso da biblioteca *SoftwareSerial.h*. Assim, os testes foram refeitos e tanto o LED quanto os motores responderam aos comandos. A Figura 5.7 mostra um LED verde apagado (acima) e aceso depois de um *switch* no *Virtuino* ser ativado (abaixo).

```
#ifndef VirtuinoBluetooth_h
#define VirtuinoBluetooth_h

//#define BLUETOOTH_USE_SOFTWARE_SERIAL  // disable this line if you want to use hardware serial


#include "Arduino.h"
#ifdef BLUETOOTH_USE_SOFTWARE_SERIAL
#include "SoftwareSerial.h"
#endif
```

Figura 5.6: Desabilitação de linha na biblioteca *VirtuinoBluetooh.h*.

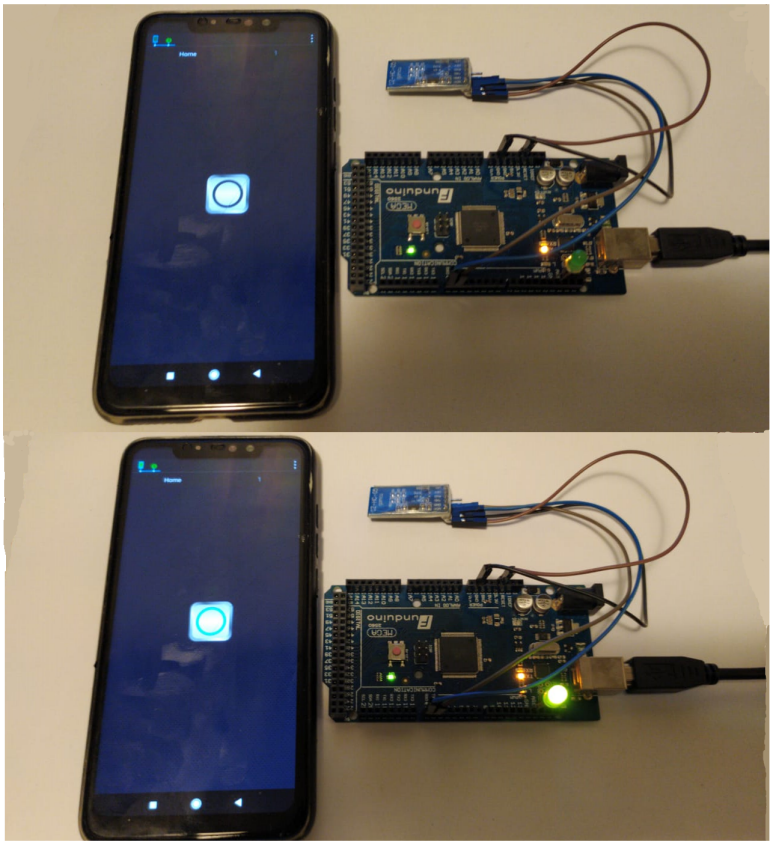

Figura 5.7: LED senso aceso através do *Virtuino*.

5.2 Resultado dos Testes do Veículo Integrado

Os testes dos módulos integrados sofreu alterações nos códigos de cada módulo seprado para que pudesse funcionar. A lógica para o carro foi imaginada da seguinte forma, uma parte do código dedicada a controlar o carro pelo celular, chamado de modo controle, e outra dedicada a andar seguindo linhas, modo autônomo.

Os modos são alternados pressionando no *Virtuino* o botão auto (ver Figura 3.24). Inicialmente foi feito o teste apenas com o módulo *bluetooth* e o seguidor de linha. O carro seguiu os comandos feitos no aplicativo e seguiu uma linha reta, feita com fita isolante, ao alternar para modo autônomo.

Seguindo, foi adicionada uma lógica para que, no modo autônomo, quando o sensor ultrassônico detectasse um obstáculo a uma distância de 30 cm ou menos, o carro deixasse de seguir a linha, parar os motores e andar para trás. Porém, o carro passou a não responder a qualquer comando do celular. Após feitas pesquisas nos fóruns do site do Arduino [3], houve a suspeita de um problema na biblioteca Ultrasonic.h, o que levou a necessidade de atualiza-la.

Tendo atualizado a bilbioteca Ultrasonic.h o carro voltou a responder aos comandos, contudo, ou o carro não detectava o obstáculo ou ele não parava a tempo. Para entender o que ocorria, foi usado o LED ligado ao pino 13 da *Mega*, o mesmo acendia ao entrar no bloco de detecção de obstáculo e apagava ao sair deste trecho de código. Dessa forma, notou-se que o LED piscava inúmeras vezes. Um teste adicional foi feito, com um código enxuto contendo somente o seguidor de linha e o sensor ultrassônico. Este novo teste consistia em manter o trecho de código que fazia a chamada para a função da biblioteca Ultrasonic.h que calcula as distâncias e, caso fosse detectado um obstáculo a 30 cm ou menos, o carro deveria freiar, andar para trás, freiar e voltar a seguir a linha. Isso ocorreu, mas não de maneira correta. Houve engasgos do motor de giro das rodas.

A esse código implementado foi adicionado o módulo *bluetooth* e sua biblioteca, com a adição deste o carro mantinha o comportamento descrito no parágrafo anterior. Quando a conexão era feita pelo *Virtuino* o carro passava a ter um comportamento ainda mais errático, dando ré aos poucos e parava. Para compreender esse comportamento foi adicionado um *display* no *Virtuino* para receber os valores das distâncias calculadas. Estes valores mudavam constantemente, algumas vezes chegando a zero, e em certos momentos, mesmo quando o *display* mostrava valores acima de 30 cm, o carro tentava fazer a correção.

Lendo o *datasheet* do sensor ultrassônico e lembrando que a função *loop* faz com que o *sketch* seja executado de forma recursiva percebeu-se que o sensor poderia estar fazendo várias leituras em um curto período. Assim, a lógica no trecho de cálculo de distância foi modificada, passou a fazer cinco leituras de distâncias e calcular a média dessas distâncias para só depois decidir se há um obstáculo a frente e tomar alguma

decisão. A resposta do carro se tornou muito melhor, sem engasgos e nem acionamento dos motores em momentos inesperados. A Figura 5.8 mostra a *Mega* e os módulos integrados ao carro. O seguidor de linha foi instalado na parte inferior do carro (1) no eixo dianteiro, o sensor ultrassónico foi instalado na parte frontal (**2**). As telas OLED (3) receberam trecho de código para mostrar uma pequena logo da Universidade de Brasília. O módulo *bluetooth* (4) também é evidenciado na figura.

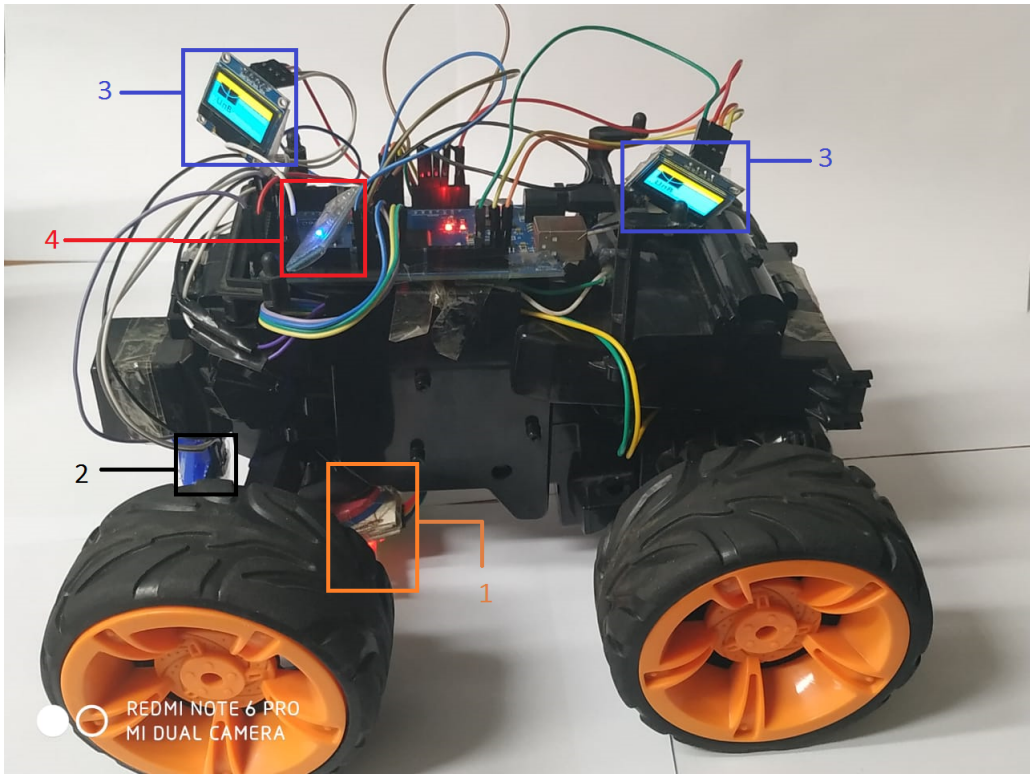

Figura 5.8: *Mega* e módulos integrados ao carro.

Resolvido os problemas de integração com o sensor ultrassônico o código ficou estruturado da seguinte maneira:

- função *setup* com a inicialização de variáveis e definição dos pinos como entrada ou saída;
- função *loop* com a chamada da função de comunicação entre o *Arduino* e o aplicativo, e alternar entre modo controle e autônomo;
- função *autonomo* com decisão para testar se há obstáculo ou andar;
- função *controle* para controlar o carro através do *Virtuino*;
- função *teste* para verificar se há um obstáculo a frente ou não;

- funçao *desvia* para, caso haja um obstáculo no modo autônomo, desviar e dar a oportunidade do controlador mudar para o modo controle;
- função *norte* para andar para frente;
- função *nordeste* para andar para frente e esquerda;
- função *noroeste* para andar em frente e a direita;
- função *sul* para andar para trás;
- função *sudeste* para andar para trás e esquerda;
- função *sudoeste* para andar trás e a direita;
- função *parado* para que nenhum dos motores gire;
- função *freio* para impedir que o carro siga andando por inércia;
- função *norte_VelMid_2* para seguir em frente com o motor de giro das rodas em potência baixa;
- função *nordeste_VelMid_2* para andar em frente e a esquerda com o motor de giro das rodas em potência baixa;
- função *noroeste_VelMid_2* para andar em frente e a direita com o motor de giro das rodas em potência baixa;
- função *sudeste_VelMid_2* para andar para trás e esquerda com o motor de giro das rodas em potência baixa;
- função *sudoeste_VelMid_2* para andar trás e a direita com o motor de giro das rodas em potência baixa.

A modularização do código em funções tornou mais simples o entendimento e, na ocorrência de novos problemas ou erros, facilita na identificação e correção.

Antes do desenvolvimento do código fonte, o carro teve sua carcaça externa colocada para um vislumbre do carro em sua versão final. As telas serão utilizadas para chamar a atenção do público, bem como permitir que as pessoas possam escrever frases próprias durante as demonstrações. As Figuras 5.9, 5.10 e 5.11 mostram uma visão final do carro.

Figura 5.9: Carro em sua versão final com alusão ao CIC e a UnB.

Figura 5.10: Carro em sua versão final com alusão ao curso de Engenharia da Computação.

Figura 5.11: Carro em sua versão final com o logotipo da UnB.

Capítulo 6

O *Software* Desenvolvido

Concluídos os testes com os módulos, os mesmos foram inseridos no veículo. Este capítulo discorrerá sobre o uso do carro em sua montagem final, os novos testes feito e seus resultados.

6.1 Protótipo Final

Os testes dos módulos integrados e as adaptações iniciais ao código encontraram novos problemas. Após alguns minutos de uso no modo controle o carro parava e não respondia mais. Caso estivesse no modo autônomo não era percebido nenhum problema. Desse modo foi feito um estudo sobre a lógica desenvolvida para o código. Na seção 5.2 foram mencionadas as funções e parte da lógica inicial idealizada.

A Figura 6.1 mostra um fluxograma para a lógica inicial. Vemos que, caso o botão AUTO (Figura 3.24) esteja desativado, o programa faz o teste para ver se há um obstáculo a frente e segue para o modo controle; caso esteja ativado faz o teste de obstáculo e irá para o modo autônomo. Caso a placa seja desligada, o programa encerra, caso contrário ira retornar para o ponto de leitura do botão AUTO (conforme compora tamento da função *loop* nativa do *Arduino*).

Na Figura 6.2 temos o fluxograma do modo controle. Nele é feita a leitura dos comandos de direção vistos na Figura 3.24. Se o usuário apertar somente para o norte, o carro se move nesta direção e o programa retorna para a função *loop*. Os outros casos em que são acionados apenas um botão de direção são vistos em seguida, caso apenas um botão seja acionado o carro se move para a direção desejada.

No fluxograma da Figura 6.3 temos a lógica inicial para o modo autônomo. Caso nenhum dos três sensores detectasse reflexão, o carro se moveria para a frente. Caso o sensor esquerdo sozinho ou em conjunto com o central detectassem que o carro saiu da linha, o carro se moveria para a direita. Se o sensor direito sozinho ou em conjunto com o

sensor central detectassem saída de linha o carro deveria se mover para a esquerda. Caso nenhum dos casos anteriores ocorresse o carro deveria ficar parado (motores com potência zero).

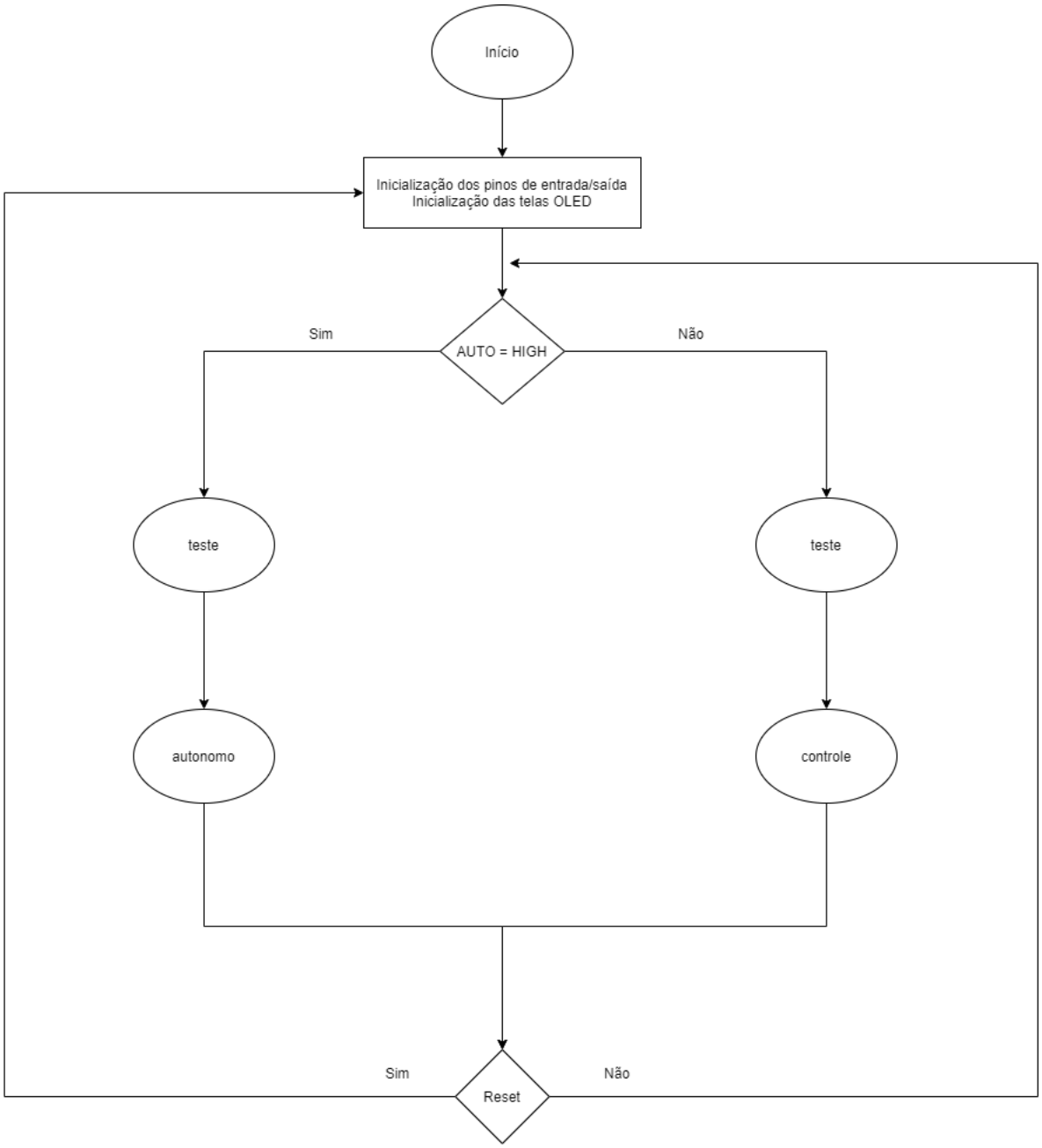

Figura 6.1: Fluxograma para a primeira lógica implementada do programa principal.

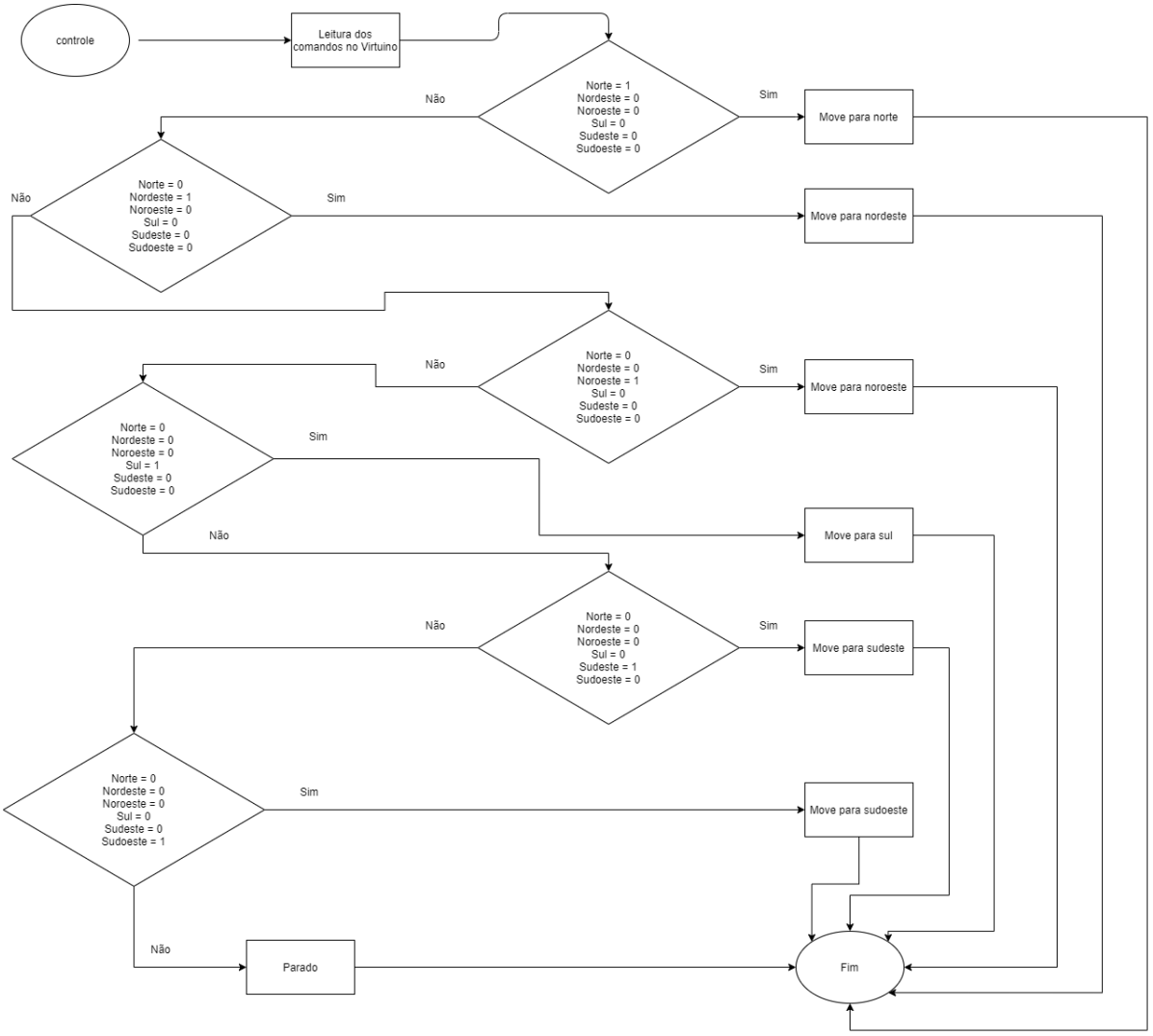

Figura 6.2: Fluxograma inicial para o modo controle.

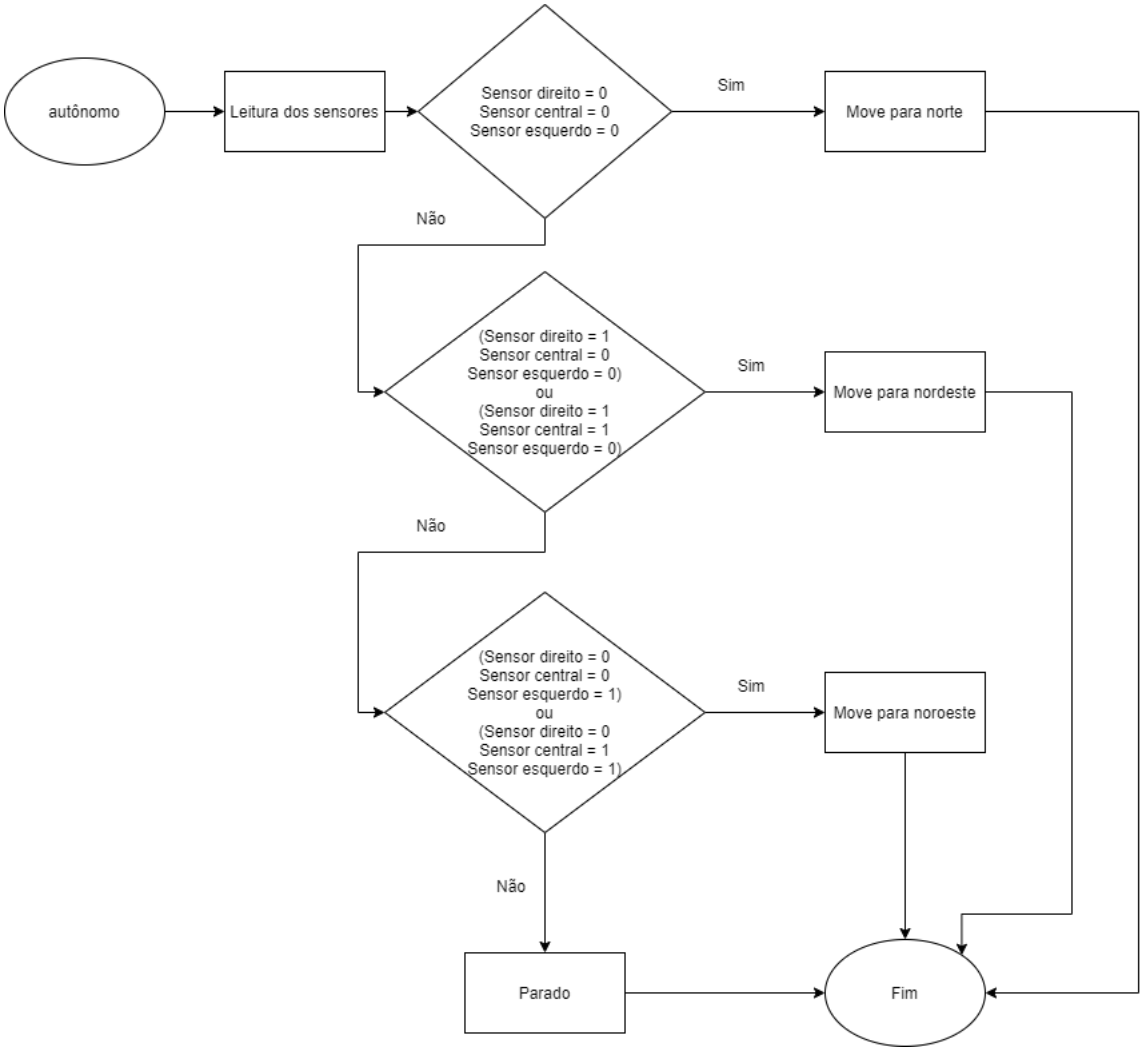

Figura 6.3: Fluxograma inicial para o modo autônomo.

Inicialmente foi desconectado o sensor ultrassom para testar se o comportamento anômalo de travamento iria desaparecer, o que não ocorreu. Depois de inumeras tentativas foi mudada a lógica de funcionamento. Como foi descrito, a lógica para o modo controle testava se apenas um botão estivesse acionado e fazia um movimento, caso contrário permanecia parado. Porém, todos os casos diferentes de um botão acionado também contemplava casos inesperados além de aconamento simultâneo de mais de um botão. Os pinos da placa *Mega* permanecem em estado de alta impedância, o que faz com que os valores variem muito. Dessa forma, o modo controle entraria no caso em que fica parado e poderia não mais sair de lá (como ocorreu diversas vezes). Portanto, foi necessário mudar a lógica. O fluxograma da Figura 6.4 mostra a nova ideia implementada.

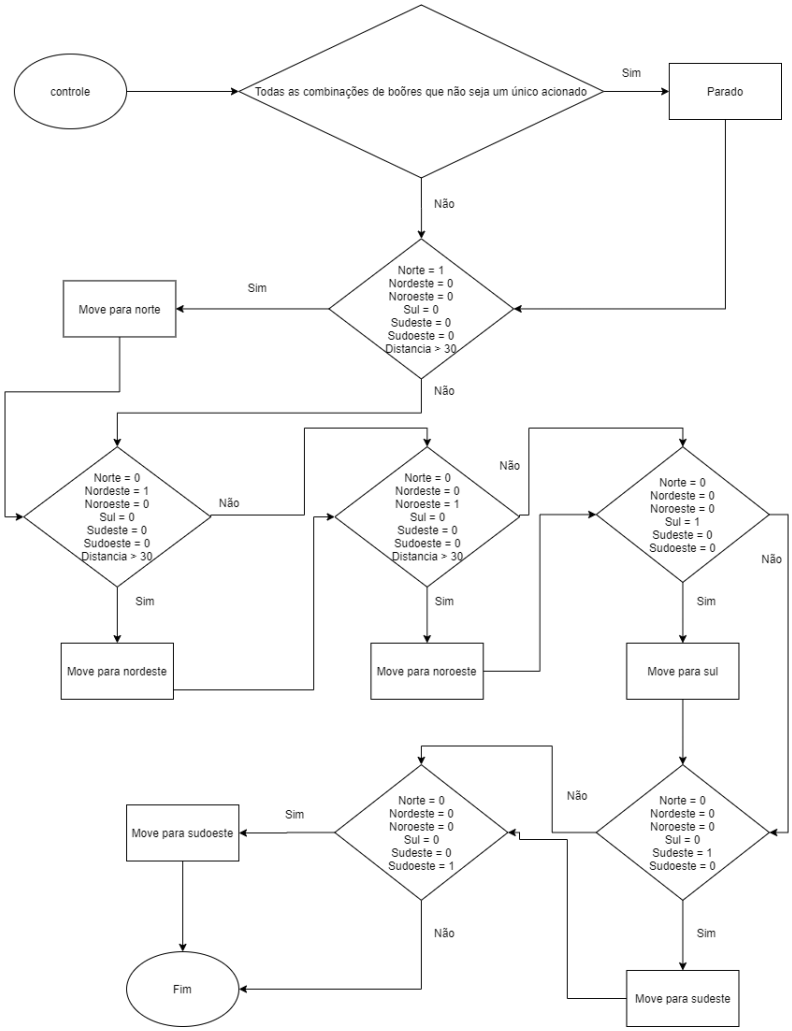

Figura 6.4: Fluxograma modificado para o modo controle.

A nova implementação declara manualmente cada uma das combinações possíveis de botões acionados simultâneamente. Caso alguma dessas combinações ocorrra (dois, três, quatro, cinco ou os seis botões acionados), o carro não deve imprimir potência no motor de giro das rodas e passar para o próximo teste de acionamento, não mais encerrando. Caso este estado não ocorra, segue para os próximos testes de caso.

Adicionalmente, conectando-se novamente o sensor de ultrassom, com a lógica de teste de obstáculo, vista no fluxograma da Figura 6.5, se um obstáculo for detectado a $30cm$ ou menos, então o carro não irá para norte, noroeste e nem nordeste.

Esta nova lógica se mostrou eficiente, o carro não entrava indevidamente no caso em que devia ficar parado e também ficou mais responsivo aos comandos do *Virtuino*.

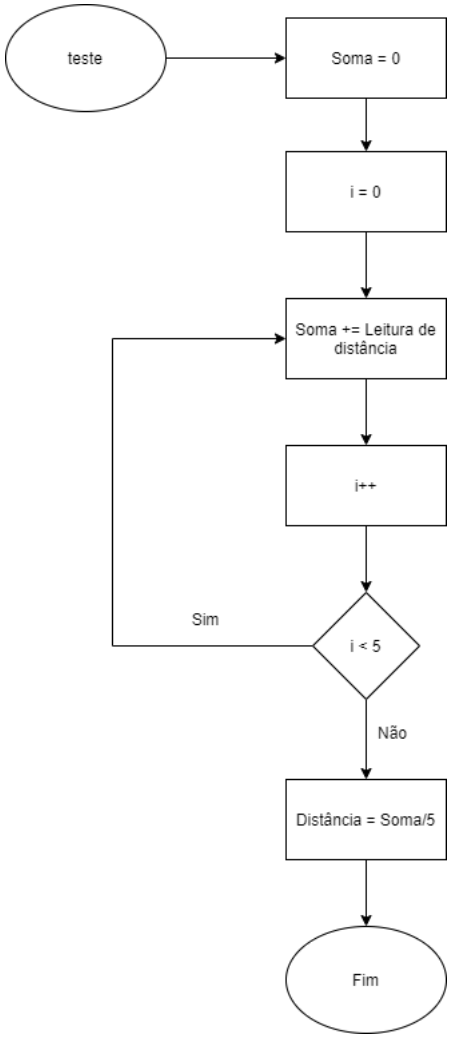

Figura 6.5: Fluxograma para testar se há um obstáculo a frente.

Seguindo a ideia desenvolvida para o modo controle, o modo autônomo teve sua lógica modificada, como visto no fluxograma da Figura 6.6. Os casos em que o carro não deve se mover são os casos em que os sensores laterais detectem saída de linha e o central não, ou os sensores laterais não indiquem saída de linha e o central sim, ou ainda quando os três sensores detectarem saída de linha.

Após algum movimento ser feito ou um caso não ser observado nos sensores, o caso subsequente deve ser testado. Só após esgotar os testes é que deve se encerrar e retornar para a parte principal do programa.

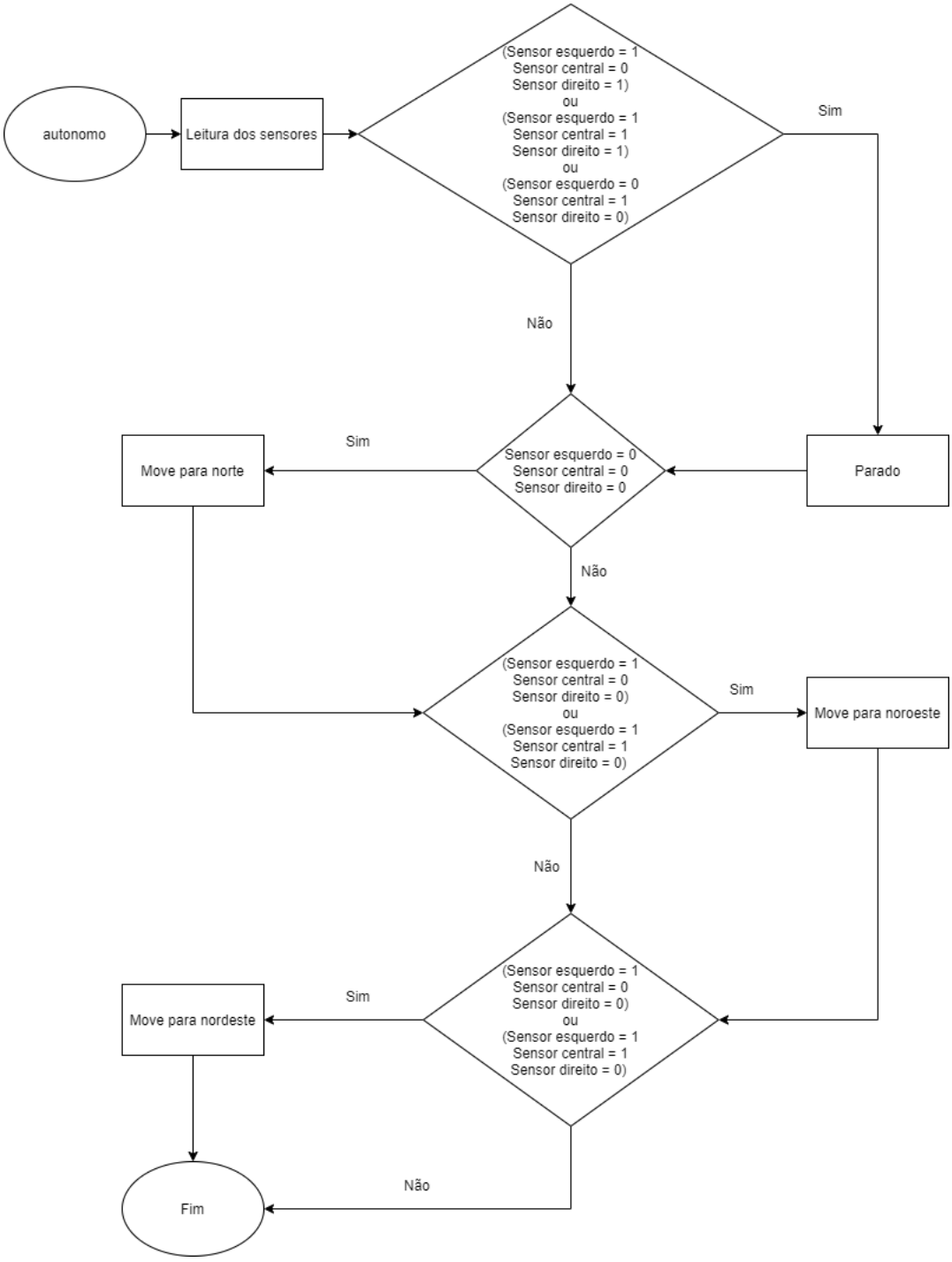

Figura 6.6: Fluxograma modificado para o modo autônomo.

Tendo sido alteradas as lógicas dos modos controle e autômomo, foi feita a lógica para o caso em que houver um obstáculo no caminho quando o carro estiver no môdo autônomo. O carro deve freiar, mover para trás durante 2 segundos, sair do caminho e parar, dando a oportunidade do usuário retomar o controle do carro e recoloca-lo no caminho. O fluxograma da Figura 6.7 mostra a lógica descrita.

Assim, a lógica do programa ficou como visto no fluxograma da Figura 6.8. Um temporizador foi adicionado antes de entrar na parte de teste de obstáculo. Depois de algum uso notou-se que o carro ficaria mais responsivo caso o teste não fosse feito sempre. Então, caso a última leitura de distância tenha sido feita em menos de 0.4 segundos, não será necessário fazer uma nova leitura.

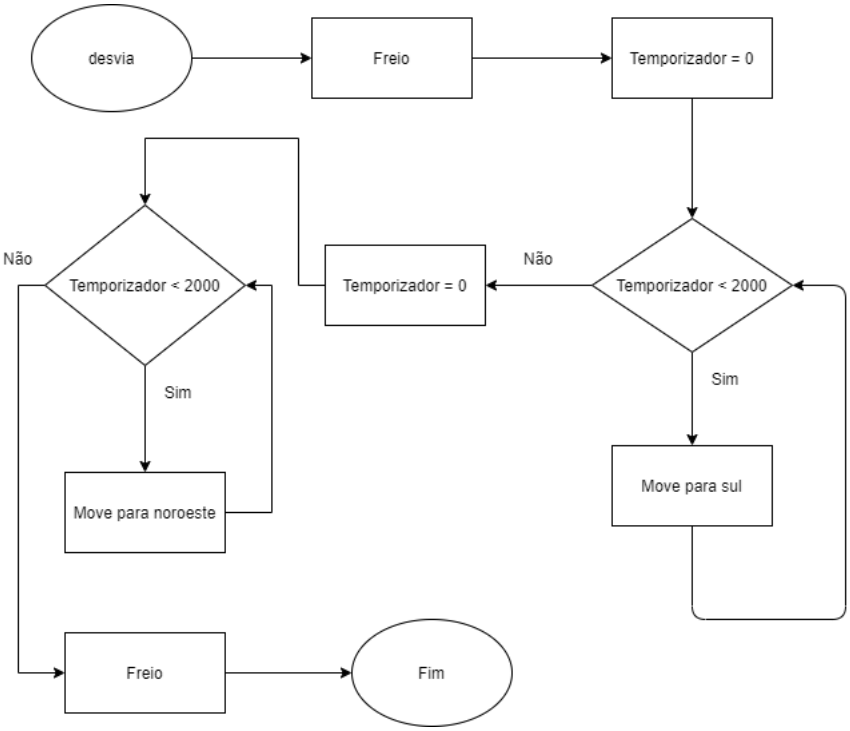

Figura 6.7: Fluxograma para o desvio de obstáculo no modo autônomo.

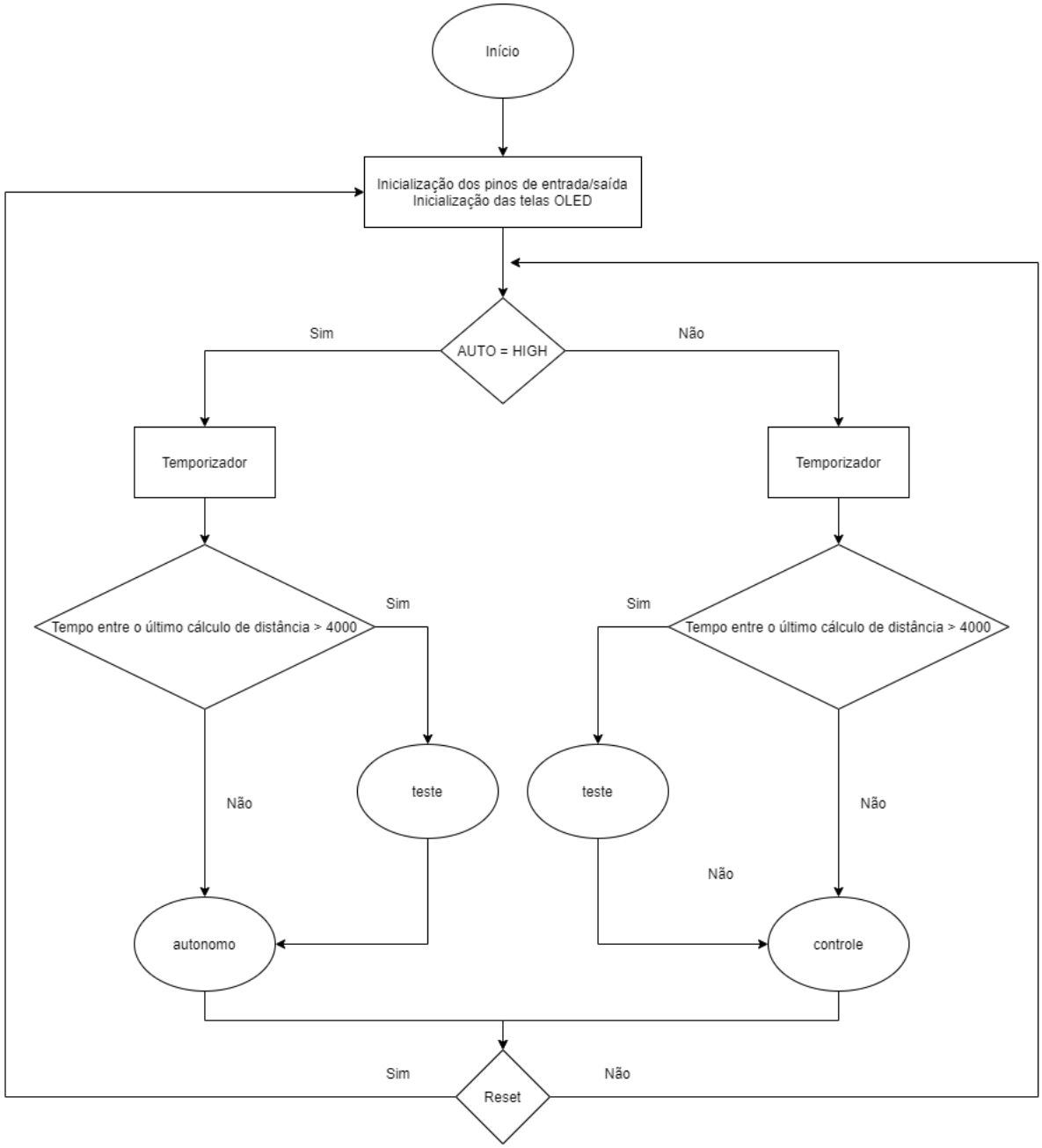

Figura 6.8: Fluxograma para a lógica final do programa principal.

Desenvolvidas as lógicas do programa principal, dos modos controle e autônomo, do teste de obstáculo e do desvio de obstáculo foram feitos testes adicionais para o seguidor de linhas e do *bluetooth*.

O seguidor de linha nota que o carro saiu da linha a uma certa distância do piso, dependendo da cor do piso. Foram feitos testes em pisos escuros e em pisos claros totalizando quatro pisos com cores diferentes, bem como em cima de papel branco A4. Para o piso mais escuro (de cor voltada para o marrom) a distância máxima entre o sensor e o piso para que houvesse detecção foi $1.5cm$, enquanto que a maior distância medida

, de $3cm$, foi na folha branca de papel A4. Dessa forma, foi necessário fazer ajustes na altura do sensor para que ele conseguisse medir pisos mais escuros mas com o cuidado de que, ao fazer um movimento de curva, o sensor não bata no chão e não seja forçado.

Os úiltimos testes com o *bluetooth* envolveram controlar o carro até que ele ficasse a uma distância em que o mesmo parasse de responder aos comandos, indicando que o *bluetooth* ficasse fora do alcance do sinal do celular. Este teste foi feito com e sem a carcaça externa. O carro deixou de seguir comandos cerca de $10m$ de distância de onde o controle se encontrava (dentro do parâmetro indicado pela fabricante), não sendo encontrada diferença substâncial com ou sem a carcaça externa.

Capítulo 7

Conclusão

Neste trabalho de conclusão de curso de graduação foi desenvolvido um pequeno veículo a ser utilizado para se fazer divulgação do nome e dos cursos do Departamento de Ciência da Computação da Universidade de Brasília (UnB). A partir de um carro plástico de controle remoto, foram aproveitados a carcaça e os motores elétricos. Foram incorporados nova placa de controle (*Arduino*), módulo *bluetooth*, sensores ultrassônico e infravermelho, e *displays* OLED para apresentação de mensagens. O veículo é capaz de operar em modo automatizado, desviando-se de barreiras e seguindo linhas no solo, além de atender aos comandos do usuário via *smartphone*. Foi elaborado o código-fonte para o controle via *Arduino*, além de ter sido prototipado um aplicativo para *smartphone*. O custo total do projeto, desconsiderando o preço do carro original, foi de *R*$370. Foram encontrados preços variádos para cada componente o que sugere que o custo pode ser mais baixo.

Os testes feitos com cada um dos componentes mostrou a versatilidade de plataformas de prototipação de *hardwares* com baixo custo. A descrição e discussão do desenvolvimento da lógica para o funcionamento do circuito mostra a importância de se entender o funcionamento do *hardware* e sua sincronização com a programação. Enviar e receber informações de módulos e tomar decisões ou realizar ações são o cerne de projetos em *Arduino*.

Os desafios de se lidar com *hardware* e adicionar uma lógica via *software* para que o mesmo funcione como o esperado não deve ser impeditivo para quem quer que tente projetos desta natureza, mas sim parte do processo de aprendizagem e crescimento.

Conclui-se que plataformas como a *Arduino* se mostram como ótima alternativa para ensino e aprendizagem tanto de programação como prototipação de circuitos, com a vantagem de um custo relativamente baixo frente à outras plataformas. Havendo possibilidade de com apenas uma placa, criar um projeto finalizado e abrindo espaço para incrementa-lo, atualiza-lo ou ainda refaze-lo.

Como trabalhos futuros, sugere-se tentar aumentar a quantidade de *displays* e de atuadores, para serem possíveis mais efeitos chamativos ao público. Ainda, há a possibilidade de ampliar o uso do *bluetooth* para comunicação entre dois carros, ou entre o carro e outro dispositivo; e novas decisões segundo informações obtidas pelo sensor ultrassônico.

Adicionar outra placa *Arduino*, como a modelo *Uno*, para processar os sinais do sensor de ultrassom e passar um aviso de obstáculo para a placa *Mega* pode melhorar a responsividade do carro nos modos controle e autônomo. Também podem ser adicionados mais sensores à placa adicional, como um módulo *bluetooth* de modelo HC-06 para comunicação entre dois carros seguindo este projeto.

Referências

[1] Costa, Fabio: *História do arduino – como surgiu esta incrível plataforma de prototipagem eletrônica.* https://eletronicapratica.com/historia-do-arduino/, acesso em 2020-02-04. ix, 5, 6, 7

[2] Barragán, Hernando: *The untold history of arduino.* https://arduinohistory.github.io/, acesso em 2020-02-04. ix, 7, 8

[3] *Site oficial da arduino.* https://www.arduino.cc/, acesso em 2019-03-26. ix, 5, 9, 10, 14, 15, 38, 54

[4] *Athos eletronics.* https://athoselectronics.com/ponte-h/, acesso em 2019-08-13. ix, 17

[5] *Filipeflop.* https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/, acesso em 2019-06-25. ix, 21, 41

[6] *Adrobótica.* https://www.adrobotica.com/produto/sensor-ultrasonico-hc-sr04/, acesso em 2020-01-11. ix, 22

[7] *o que são oleds?* https://www.arealocal.com.br/o-que-sao-os-oleds/, acesso em 2019-08-13. ix, 23

[8] *Departamento de ciências da computação.* https://cic.unb.br/cursos/engenharia-de-computacao/, acesso em 2020-02-04. 1

[9] *Site processing.* https://processing.org/, acesso em 2020-01-30. 5, 6

[10] Barragán, Hernando: *Wiring: Prototyping physical interaction design.* http://people.interactionivrea.org/h.barragan/thesis/thesis_hi_res.pdf, acesso em 2020-02-04. 7

[11] Monk, Simon: *Programação com Arduino » Começando com Sketches.* Bookman Editora Ltda., 2017. 10

[12] Murta, Gustavo: *Guia completo da arduino mega.* https://blog.eletrogate.com/guia-completo-do-arduino-mega/, acesso em 2019-06-18. 15

[13] *Filipeflop.* https://www.filipeflop.com/produto/driver-motor-ponte-h-l298n/, acesso em 2019-08-13. 17, 18

[14] *Vida de silício*. https://www.vidadesilicio.com.br/hc-05-modulo-bluetooth, acesso em 2019-08-13. 19

[15] *Robotic technology center*. http://www.robotpark.com/TCRT5000-3-Channel-Line-Tracking-Sensor-Module, acesso em 2019-08-13. 20

[16] *Squids arduino*. http://www.squids.com.br/arduino/index.php/projetos-arduino/projetos-squids/basico/297-projeto-90-como-controlar-um-display-oled-ssd1306-no-arduino, acesso em 2019-08-13. 25

[17] *Site oficial da plataforma de prototipação virtuino*. https://virtuino.com/, acesso em 2019-05-12. 25, 39

[18] *Biblioteca adafruit*. https://github.com/adafruit/Adafruit_SSD1306, acesso em 2019-06-25. 45