UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETROTÉCNICA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

LUCAS KAZUYUKI TAMINATO
MATHEUS PIMENTEL DA SILVA ORTH

BRINSK: SEGUIDOR DE LINHA MUSICAL PARA INTRODUÇÃO À ENGENHARIA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2016

LUCAS KAZUYUKI TAMINATO
MATHEUS PIMENTEL DA SILVA ORTH

BRINSK: SEGUIDOR DE LINHA MUSICAL PARA INTRODUÇÃO À ENGENHARIA

Trabalho de Conclusão de Curso apresentada como requisito parcial à obtenção do título de Bacharel em Engenharia de Controle e Automação, do Departamento Acadêmico de Eletrotécnica, da Universidade Tecnológica Federal do Paraná.
Orientador: Prof. Dr. Marco Antônio Busetti de Paula

CURITIBA

2016

Lucas Kazuyuki Taminato
Matheus Pimentel da Silva Orth

Brinsk: seguidor de linha musical para introdução à engenharia

Este Trabalho de Conclusão de Curso de Graduação foi julgado e aprovado como requisito parcial para a obtenção do Título de Engenheiro de Controle e Automação, do curso de Engenharia de Controle e Automação do Departamento Acadêmico de Eletrotécnica (DAELT) da Universidade Tecnológica Federal do Paraná (UTFPR).

Curitiba, 01 de dezembro de 2016.

_____
Prof. Paulo Sérgio Walenia, Dr.
Coordenador de Curso
Engenharia de Controle e Automação

_____
Profa. Annemarlen Gehrke Castagna, Mestre
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia de Controle e Automação do DAELT

ORIENTAÇÃO

_____
Prof. Marco Antonio Busetti de Paula, Dr.
Universidade Tecnológica Federal do Paraná
Orientador

BANCA EXAMINADORA

_____
Prof. Antonio Carlos Pinho, Dr.
Universidade Tecnológica Federal do Paraná

_____
Prof. Elder Oroski, Dr.
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia de Controle e Automação

AGRADECIMENTOS

À Deus por ser criativo e nos criar em sua semelhança, nos dar força e perseverança para continuarmos focados até o fim.

Ao nosso orientador Prof. Dr. Marco Antônio Busetti de Paula pela paciência, incentivo e instrução, nos mostrando a ideia de lúdico e modular para desenvolvermos nossa criatividade.

Às nossas famílias pelo carinho, amor e apoio nos momentos de dificuldade. Apoio este que foi de fundamental importância para concluirmos nosso trabalho.

Aos nossos amigos pela compreensão e suporte nos fins de semana que não pudemos estar presentes em momentos de confraternização.

À empresa Microduino que acreditou no projeto desde o princípio, oferecendo todo o suporte necessário na parte de *hardware*.

RESUMO

TAMINATO, Lucas Kazuyuki; ORTH, Matheus Pimentel da Silva. **Seguidor de linha musical para introdução à engenharia.** 2016. 58 p. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) - Departamento Acadêmico de Eletrotécnica - DAELT, Universidade Tecnológica Federal do Paraná. Curitiba. 2016.

O presente trabalho tem como objetivo o relato do desenvolvimento de um produto inovador que visa introduzir conceitos básicos de engenharia para jovens, como eletrônica, programação e modelagem 3D, utilizando como atrativo, a música e vice-versa. O projeto consiste em um seguidor de linha musical e um aplicativo móvel conectado por *Bluetooth* ao protótipo para a emissão das notas musicais atribuídas às cores ao lado da linha preta de referência da pista a ser percorrida. Com isso, foram feitas fundamentações teóricas sobre conceitos que seriam de relevante importância para o desenvolvimento desse projeto. Dentre os conceitos utilizados estão música (harmonia, melodia e ritmo), impressão 3D (*driver* e *software*), programação (algoritmos e *softwares*), eletrônica (regulador de tensão, microcontrolador e sensores) e *design* (*software*), ou seja, dos processos, equipamentos e materiais utilizados. Por fim, relatou-se cada passo do desenvolvimento do projeto e a grande dificuldade encontrada foi com relação à fonte de energia escolhida que apresentou resultado adequado para um bom funcionamento, porém com pouca duração de carga.

Palavras-chave: Robótica. Música. Impressão 3D. Educação.

ABSTRACT

TAMINATO, Lucas Kazuyuki; ORTH, Matheus Pimentel da Silva. **Musical follow line for engineering introduction.** 2016. 58 p. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) - Departamento Acadêmico de Eletrotécnica - DAELT, Universidade Tecnológica Federal do Paraná. Curitiba. 2016.

The present work has the objective of reporting on the development of an innovative product that aims to introduce basic concepts of engineering for young people, like eletronics, programming and 3D printing, using music as attractive and vice versa. The project consists of a musical follow line robot and a mobile application connected by Bluetooth to the prototype for the reproduction of the musical notes assigned to the colors next to the black reference line of the track to be followed. Thereby, theoretical foundations were made based on concepts that have relevant importance for the development of this project. Among the concepts used are music (harmony, melody and rhythm), 3D printing (driver and software), programming (algorithms and softwares), electronics (voltage regulator, microcontroller and sensors) and design (software), in other words, process, equipment and materials used. Finally, each step of the project development was reported and the biggest problem was with regard to the chosen source of energy that presented adequate results for a good functioning, but with a short duration of load.

Keywords: Robotic. Music. 3D Printing. Education.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

3D - três dimensões

IDE - Ambiente de Desenvolvimento Integrado (do original *Integrated Development Environment)*

Hz - Hertz

nm - nanometro

USB - Comunicação Serial Universal (do original *Universal Serial Bus*)

Wi-Fi - Fidelidade Sem fio (do original *Wireless Fidelity*)

BLE - *Bluetooth* de Baixa Energia (do original *Bluetooth Low Energy*)

USBTTL - Comunicação Serial Universal Lógica Transistor-Transistor (do original *Universal Serial Bus Transistor-Transistor Logic*)

LED - Diodo Emissor de Luz (do original *Light Emitting Diode*)

RGB - Vermelho, Verde e Azul (do original Red, Green, Blue)

PWM - Modulação por largura de pulso (do original *Pulse Width Modulation*)

iOS - Sistema Operacional do iPhone (do original *iPhone Operational System*)

ISM - Industrial, Cientifica e Médica (do original *Industrial, Scientific and Medical*)

INMETRO - Instituto Nacional de Metrologia, Qualidade e Tecnologia

STL - Linguagem Revestimento/Triangular Padrão (do original *Standard Tessellation/Triagle Language*)

ABS - Acrilonitrila Butadieno Estireno (do original *Acrylonitrile Butadiene Styrene*)

PLA - Ácido Polilático (do original *Polylactic Acid*)

HIPS - Poliestireno de Alto Impacto (do original *High Impact Polystyrene*)

nF - nano Faraday

V - Volt

I2C - Circuito Inter-Integrado (do original *Inter-Integrated Circuit*)

ºC – Celsius

SUMÁRIO

1. INTRODUÇÃO

1.1 TEMA

1.1.1 Delimitação do Tema

A música na educação, juntamente com a ferramenta pedagógica, contribui e muito no desenvolvimento dos adolescentes tanto na questão social, como na cultural e até mesmo profissional, pois auxilia no desenvolvimento psicomotor, raciocínio lógico, memória, trabalho em equipe, concentração, entre muitos outros fatores. A música é "imprescindível na formação da criança para que ela, ao se tornar adulta, atinja sua maioridade intelectual e exerça sua criatividade de maneira crítica e livre" (COELHO, 2013).

Um dos maiores pensadores da história, Platão, afirmou que a música é o instrumento educacional mais potente que existe. "A música é a essência da ordem. Eleva todas as almas para o que é bom, justo e belo, e deve ser para a alma o que para o corpo é a ginástica" (PLATÃO).

Atualmente alguns alunos do ensino fundamental no Brasil são incentivados à robótica com o produto da LEGO Mindstorm EV3, como forma de "motivação para a opção por áreas ligadas à Ciência e à Tecnologia e na capacidade de resolução de problemas, persistência e criatividade" (COIMBRA, 2006, p.8), no entanto percebe-se uma carência de uma continuidade para o fundamental e consequentemente o ensino médio. Isto é, uma exigência maior, para desafios um pouco mais complexos. Tais como um seguidor de linha, o qual consiste, basicamente, em um robô seguir uma linha preta de referência com fundo branco, em volta deste caminho são pintadas diferentes cores, cada uma pré relacionada a uma nota musical. Assim no decorrer do caminho, o robô tocará uma música, desenhada pelo próprio estudante.

Para ocupar os jovens com seus interesses, foi desenvolvido esse produto, o qual é constituído por peças necessárias para a construção de um seguidor de linha que reproduz notas musicais. Permitindo ao jovem aprender a programar, modelar protótipos 3D e teoria musical.

1.2 PROBLEMA E PREMISSAS

O problema principal encontrado foi a falta de integração entre música e engenharia na educação, visando a interdisciplinaridade. A qual pode ser aproveitado de dois modos: o aprendizado de robótica utilizando a música como meio motivacional e também o aprendizado de teoria musical com recursos tecnológicos.

As principais premissas focadas são desenvolver os algoritmos modulares, para possíveis adaptações de códigos pelos jovens, obter um *design* base para a customização pessoal, além do baixo custo do protótipo.

1.2.1 OBJETIVOS

1.2.2 Objetivo Geral

Desenvolver um protótipo de seguidor de linha musical, a fim de integrar a robótica com teoria musical básica.

1.2.3 Objetivos Específicos

- Projetar um seguidor de linha;

- Detectar cores com sensor serial de cor;
- Criação de aplicativo móvel para a reprodução das notas musicais;
- Conectar via *Bluetooth* com aplicativo móvel;
- Utilizar modelagem 3D para prototipagem de *design*.
- Documentar a formulação do projeto do protótipo;

1.3 JUSTIFICATIVA

A sociedade atual é constituída de vários pilares, como: segurança, saúde e educação, muito embora acredita-se que esse último seja a fundação dos demais. Tornando assim a educação como principal formador, não apenas de profissionais, mas de cidadãos, agentes da sociedade. “A cidadania organizada delega ao Estado funções importantes em termos de sedimentação de canais de participação, a começar pela educação."(DEMO, 2007)

No Brasil, o problema com a educação é facilmente observável, o que acaba influenciando na precariedade da saúde, na violência e outros problemas enfrentados no país. Um dos defeitos do sistema educativo é a falta de investimento no ensino extracurricular, como de programação ou da música. Muito embora o aprendizado das duas áreas seja enriquecedor no desenvolvimento de raciocínio lógico e motora, além do entretenimento e trabalho em grupo (GODOI, 2011).

Esse projeto relata o desenvolvimento de um protótipo, o qual serve como meio de aprofundamento não somente em programação, *hardware* e teoria musical básica, mas principalmente incentivar na formação pessoal dos adolescentes.

Para o desenvolvimento de software do seguidor de linha, foi utilizado o Arduíno IDE (*Integrated Developments Environment*) para programação, que utiliza linguagem pseudo C++ com alguns códigos específicos para mapeamento do microcontrolador, já o aplicativo móvel será desenvolvido no *Xcode*, IDE da *Apple*, na linguagem nativa Swift.

Assim, pretende-se motivar a educação em setores pouco explorados e que podem ser discutidos em outras disciplinas, abrindo a possibilidade em projetos

interdisciplinares, buscado em muitas escolas. Contando assim, com o aprimoramento tecnológico e musical de forma educativa e entretida.

1.4 PROCEDIMENTOS METODOLÓGICOS

Inicialmente o projeto buscou por fundamentos teóricos, como dados técnicos, musicais e pedagógicos. Foi pesquisado o funcionamento do Arduino, dos sensores para detecção de cor, códigos para o seguidor de linha, maiores detalhes sobre teoria musical e principalmente métodos pedagógicos para o ensino fundamental (6º ao 9º ano).

O trabalho seguiu com estudos de seguidores de linha, levando em conta as peças, sua construção e códigos já existentes. Levantando a lista das peças essenciais, procurando por alternativas mais simples para a formulação de um seguidor de linha básico, por meio de testes de motores, sensores e códigos.

Além da pesquisa do seguidor de linha, foram feitas simulações de mil amostras de cada cor com o sensor de cor, e assim ter uma média dos valores para a identificação delas, uma vez que cada uma vai estar relacionada a uma nota musical.

Antes do desenvolvimento do aplicativo móvel, foi utilizado outro aplicativo para verificar a comunicação *Bluetooth* com o *hardware*, tendo isso bem-sucedido, foi então criado o aplicativo móvel.

Os desenhos do chassi foram realizados primeiramente a partir de um esboço aproximado do carrinho desejado, para então serem realizadas melhorias. Assim também com a impressora 3D, foi adquirido experiência nos ajustes de temperatura, materiais e outros parâmetros de configuração com as primeiras tentativas.

Com a conclusão dos testes iniciais e sabendo então as peças escolhidas, foi feito o projeto do protótipo. Considerando principalmente a parte funcional com todas as características de seguir a linha, identificar a cor e a comunicação *Bluetooth* com aplicativo. Com a referência teórica finalizada, foi inicializada a construção do protótipo.

1.5 ESTRUTURA DO TRABALHO

O presente trabalho segue a seguinte estrutura:

- Capítulo 1 - Introdução;
- Capítulo 2 - Fundamentação teórica;
- Capítulo 3 - Desenvolvimento;
- Capítulo 4 - Conclusão.

No primeiro capítulo é apresentado em linhas gerais o estudo do objetivo, além da metodologia aplicada.

A fundamentação teórica começa no capítulo 2, contendo conceitos técnicos necessários para o entendimento do funcionamento e construção do seguidor de linha, com os sensores, motores, a comunicação *Bluetooth*, além da parte de programação. Serão abordados também definições básicas de teoria musical.

No Capítulo 3, é um relato da construção de um protótipo com os acertos e erros, e as melhorias de acordo com os testes realizados, tanto em nível de aprimoramento nos algoritmos do aplicativo e do seguidor de linha quanto no hardware.

As conclusões do estudo, o levantamento das falhas e dos acertos e os próximos passos a serem tomados são apresentados no Capítulo 4.

2. FUNDAMENTAÇÃO TEÓRICA

O seguidor de linha tem a função de despertar o interesse em robótica através do atrativo musical e brincadeiras lúdicas.

> A robótica pode ser definida como “a ciência dos sistemas que interagem com o mundo real com pouca ou mesma nenhuma intervenção humana. ” (ARS CONSULT, 1995, p.21). É uma área multidisciplinar, que integra disciplinas como Matemática, Engenharia Mecânica, Engenharia Elétrica, Inteligência Artificial, entre outras. (ZILLI, 2004, pág. 37).

O interesse pela robótica será tratado através de uma visão chamado robótica educacional. Esse conceito envolve as questões de programação para desenvolvimento de qualquer função de um robô por exemplo, design para o aprendizado da modelagem do protótipo em 3D e eletrônica para utilização de sensores, motores, componentes eletrônicos, entre outros. Além de gerar um aprendizado dinâmico com o envolvimento de conceitos matemáticos e físicos implementados na prática, facilitando a trajetória do aluno em sua vida acadêmica.

2.1 MÚSICA

Bohumill Med (1996, p.9-13) define a música como uma arte sonora por ser de percepção auditiva, com suas características principais: melodia (conjunto de sons sucessivos, dados um após o outro), harmonia (conjunto de sons simultâneos, dados de uma só vez) e ritmo (ordem e proporção em que estão dispostos os sons que constituem melodia e harmonia), tendo como matéria prima o som, este podendo ser representado por somente sete notas (dó, ré, mi, fá, sol, lá, si).

A música no ensino infantil colabora muito para o aprimoramento motor, respeito e faz com que a criança “desenvolva sua criatividade, sua subjetividade e exerça sua liberdade” (COELHO, 2013).

O atrativo musical é o meio pelo qual a criança se interessaria pelo projeto inicialmente, onde será trabalhado parte da teoria musical, atribuindo cores para cada nota musical, oitava na escala maior, que será tocada pelo Brinsk.

Observando-se as frequências das notas musicais e o comprimento de onda das cores, foi feito um mapeamento de uma relação meramente numérica entre cor e nota musical (Tabela 1). Apesar da frequência do som ser uma vibração mecânica e o comprimento de onda das cores ser de origem eletromagnética, foi feita essa relação para utilizar a cor como mais um atrativo visual (MUNIZ, 2014).

Tabela 1 - Associação das notas musicais com as cores

Nota musical	Frequência aproximada da nota Musical (Hz)	Cor	Comprimento de onda aproximado da cor (nm)
Dó	528	Verde	520
Ré	594	Amarelo	580
Mi	660	Laranja	600
Fá	704	Vermelho	700
Sol	396	Violeta	380
Lá	440	Rosa	400
Si	495	Azul	480

Fonte: Adaptado de MUNIZ (2014)

Quando se fala em lúdico, pode-se citar a definição que o dicionário dá.

> Feito através de jogos, brincadeiras, atividades criativas. Que faz referência a jogos ou brinquedos: brincadeiras lúdicas. Divertido; que tem o divertimento acima de qualquer outro propósito. Que faz alguma coisa simplesmente pelo prazer em fazê-la. Psicanálise. Refere-se à manifestação artística ou erótica que aparece na idade infantil e se acentua na adolescência aparecendo sob a forma de jogo (DICIO, 2016).

2.2 ELETRÔNICA

Nos trabalhos acadêmicos e protótipos industriais é comum utilizar o Arduino, que é um *hardware* desenvolvido por uma empresa italiana para uma prototipagem rápida. Baseado em um microcontrolador, permitindo saídas e entradas rápidas e simples, além de disponibilizar uma porta USB, facilitando a compilação dos algoritmos desenvolvidos.

A partir da iniciativa da empresa italiana, diversos acessórios e similares foram desenvolvidos por diferentes empresas, para comunicação *Bluetooth,* Wi-Fi ou *drivers* para motores, dentre outros. Utilizando do mesmo segmento de prototipagem eletrônica, formou-se uma empresa chinesa chamada Microduino.

2.2.1 Microcontrolador e acessórios

A maioria da eletrônica necessária ao projeto foi patrocinada pela empresa chinesa Microduino que possui diferentes linhas de placas com os componentes essenciais (módulos *Bluetooth*, Wi-Fi, *Core, HUB,* sensores de cor, seguidor de linha, ultrassom, entre outros) para protótipos educacionais, e levando em conta a sua estrutura modular, facilita em futuras mudanças e progressos, uma vez que para utilizar mais de uma placa, basta encaixar uma em cima da outra acoplando os *bornes* macho e fêmea.

A placa principal do projeto é a Core (Figura 1), que contém o microcontrolador Atmel Atmega328P onde será armazenado e executado o programa criado para o funcionamento de projeto, ou seja, é o 'cérebro' do robô (MICRODUINO CORE, 2016).

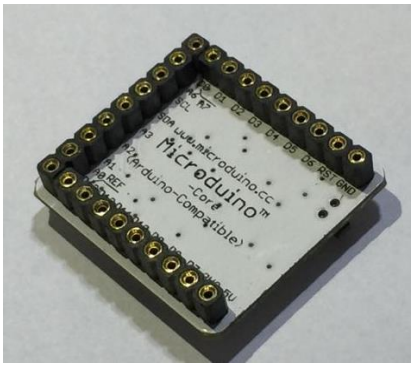

Figura 1 - Microduino *Core*

Fonte: Autores, 2016.

A comunicação *Bluetooth* se dá pelo módulo de *Bluetooth Low Energy* – BLE (MICRODUINO BLE, 2016) da figura 2. Para a compilação do algoritmo criado na IDE do Arduino é necessário acoplar o USBTTL (MICRODUINO USBTTL, 2016) para armazenar o programa criado no Microduino *Core*, pois nele existe uma porta fêmea micro-USB, onde é conectado ao computador. (Figura 3).

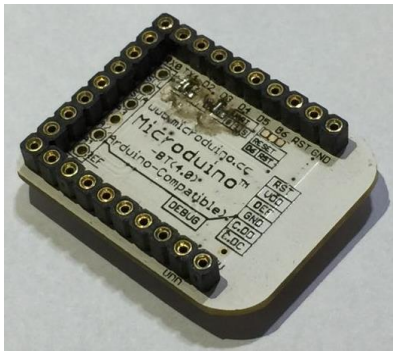

Figura 2 - Microduino *Bluetooth Low Energy*

Fonte: Autores, 2016.

Figura 3 - Microduino *USBTTL*

Fonte: Autores, 2016.

Existe também o HUB (MICRODUINO HUB) representado na figura 4, que será mantida acima das demais, com o objetivo de facilitar a conexão com os sensores e motores, uma vez que os cabos para estes condizem com o formato padronizado ao HUB.

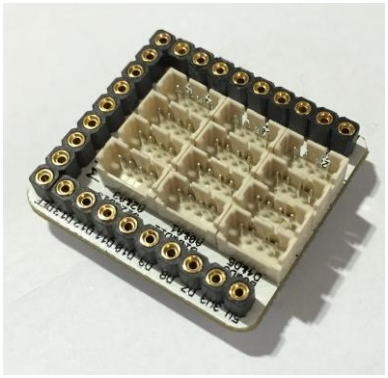

Figura 4 - Microduino *Sensor Hub*

Fonte: Autores, 2016.

2.2.2 Sensores

Para que o processador realize os seus cálculos e acionamento adequado das saídas, são necessários de certos sinais de entradas provindos de três tipos de sensores, são eles:

- Infravermelho (MICRODUINO TOUR, 2015)- Tem como função manter o carrinho na linha. O seu funcionamento se dá a partir da emissão de sinal infravermelho, se o sinal for refletido de volta, ou seja, uma superfície branca, o receptor detecta o mesmo sinal emitido, caso contrário a linha é considerada preta (Figura 5).

Figura 5 - Microduino *Tour Line*

Fonte: Autores, 2016.

- Sensor de Cor (Figura 6) - Localizada ao lado dos infravermelhos, tem a finalidade de detectar as cores desenhadas ao redor da linha de referência do caminho. Este sensor em questão é o TCS3414, o qual possui quatro LED's brancos para a iluminação e reflexo preciso do objeto. Conta-se com quatro filtros de cada referencia RGB, ou seja, vermelho, verde e azul, além de mais 4 sem filtro algum. Os sinais detectados pelos fotodiodos sucedem à conversão analógico-digital para serem transmitidos serialmente para o microcontrolador (MICRODUINO COLOR, 2016).

Figura 6 - Microduino *Color Detector*

Fonte: Autores, 2016.

2.2.3 Motor e *Driver*

Sabendo os dados de entrada oriundos dos sensores, e interpretados pelo microcontrolador, é possível então ligar os motores devidamente (Tabela 2). Para tal é utilizado o *driver* da Microduino (Figura 7), que possui PWM interno, com controle limite de corrente. A princípio será utilizado um *driver* para cada motor, no entanto estuda-se a possibilidade de reduzir para um *driver* para os dois motores.

Tabela 2 - Controle dos motores pelo *Driver*

IN1	IN2	IN3	IN4	OUT1A	OUT1B	OUT2A	OUT2B	Função
0	0	0	0	*Off*	*Off*	*Off*	*Off*	Desligado
1	0	1	0	*High*	*Low*	*High*	*Low*	Frente
0	1	0	1	*Low*	*High*	*Low*	*High*	Para Trás
1	1	1	1	*Low*	*Low*	*Low*	*Low*	Freio

Fonte: Microduino Wiki 2016 (MICRODUINO MOTOR, 2016).

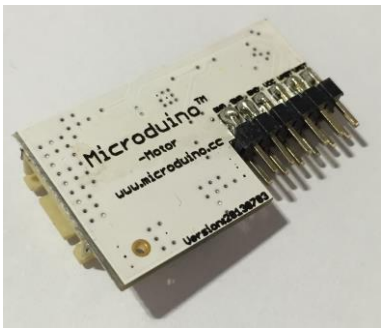

Figura 7 - Microduino *Motor Drive*

Fonte: Autores, 2016.

2.3 PROGRAMAÇÃO

Foram utilizados dois softwares para o desenvolvimento do seguidor de linha musical e o aplicativo móvel.

2.3.1 Arduino

A empresa Arduino também disponibiliza uma IDE própria, que com código-fonte aberto, é na linguagem C++ modificada, a fim de facilitar o desenvolvimento de iniciantes (ARDUINO, 2016).

O algoritmo do seguidor de linha, detecção de cores e transmissão por *Bluetooth* será desenvolvido através da Arduino utilizando uma biblioteca da Microduino (*hardware* utilizado detalhado em seguida), para uma sincronização adequada no momento da compilação.

2.3.2 Xcode

A IDE disponibilizada pela *Apple* para o desenvolvimento de aplicativos *iOS* chama-se *Xcode* e suporta as linguagens *Objective-C* e *Swift*, sendo esta última própria da empresa, e escolhida pela equipe para o projeto, pois é de fácil entendimento por ser de alto nível (DEVELOPER APPLE, 2016). A proposta inclui um aplicativo *mobile* para a execução das notas recebidas pelo seguidor de linha musical.

2.3.3 Topologia

Topologia de rede é a formato em que os dispositivos estão conectados e os dados transmitidos. Existem alguns *layouts* que são normalmente utilizados na indústria como: barramento, estrela, anel e *Token Ring.*

Para a comunicação em *Bluetooth* utiliza-se, genericamente, a topologia de barramento, visto que nela um dispositivo é ligado, sem fio, ao outro dispositivo.

Mais especificamente, o *Bluetooth* trabalha numa faixa de frequência entre 2,4 GHz e 2,48 GHz, denominado banda ISM (Industrial, Cientifica e Médica). Uma rede de *Bluetooth* é denominada *Piconet*, podendo comportar até 8 dispositivos, sendo um deles o Mestre. Este tem a finalidade de garantir o salto da técnica utilizada chamada de espalhamento de espectro por salto de frequência, que consiste no mestre realizar saltos de frequência durante a transmissão em intervalos de tempos curtos o suficiente para não sofrer alguma interferência (FERREIRA, 2005). O objetivo desta técnica é minimizar a concorrência da transferência de dois ou mais dispositivos, com a administração adequada dos *clocks* de cada escravo.

2.4 DESIGN

O *design* do carrinho tem grande importância, uma vez que "um projeto divertido e atraente pode provocar a curiosidade das crianças e motivá-los a começar a usar" (BISSOLOTTI, GONÇALVES, PEREIRA, 2015, p. 8). Além da relevância de ajudar a ter a atenção da criança, foi estudado também as métricas do INMETRO que delimitam a idade mínima de comercialização.

O órgão INMETRO (Instituto Nacional de Metrologia, Qualidade e Tecnologia), além de ser responsável por garantir as devidas medições através da Metrologia, também é incumbido de verificar a qualidade de produtos no mercado com a Avaliação de Conformidade. Esta põe em prova o produto em questão em diferentes aspectos, tais como: inflamável, pontiagudo ou cortante, vazamento químico, corrosivo, dimensões, peças pequenas que podem provocar acidentes. A partir dos resultados, conclui-se a idade mínima adequada para o produto, e ainda se o produto tem o selo de qualidade mínima prevista pelo instituto, tornando-o mais confiável diante dos clientes.

2.4.1 MODELAGEM 3D

O esboço do chassi do carrinho (Figura 8) foi realizado no Tinkercad (TINKERCAD, 2016). Um produto disponível gratuitamente em qualquer *browser*, e muito embora não seja de utilidade para engenharia, o intuito do projeto é disponibilizar para jovens na escola, sem ter, necessariamente, conhecimentos de desenho técnico. Ao projetar o modelo no Tinkercad, o arquivo é convertido para o formato apropriado (.STL) para a impressora 3D.

A estrutura final do objeto é formada por blocos de formatos comuns (paralelepípedos, cilindros, esferas, entre outros) sobrepostos ou subtraídos entre si até chegar ao resultado final desejado.

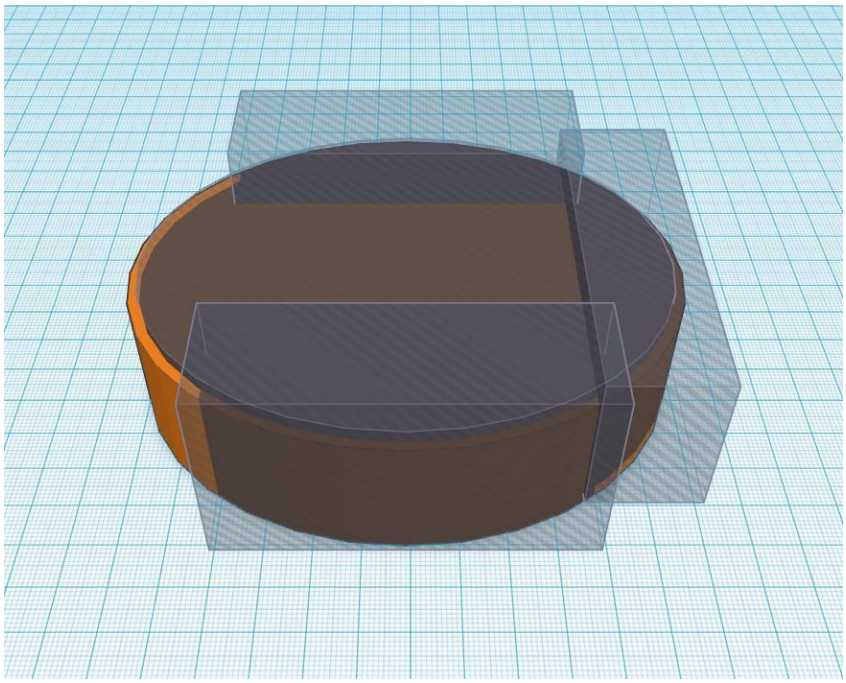

Figura 8 - Visão frontal do desenho

Fonte: Autores, 2016.

A impressora 3D é um meio de prototipagem rápida, considerada uma tecnologia de manufatura aditiva, que lê qualquer arquivo criado em *software* 3D no formato STL, transformando-o num modelo físico a partir de uma série de seções transversais (STRATASYS, 2016).

As impressoras mais utilizadas para modelagem 3D possuem geralmente quatro motores de passo (para controlar a movimentação dos eixos x, y, z e da injeção do filamento no extrusor), um extrusor (para filamentos de 1,75mm de diâmetro) e uma mesa de impressão de alumínio, ambos com controle de temperatura.

Os tipos de filamento para impressão são, em sua maioria derivados do plástico, ABS (Acrolonitrila Butadieno Estireno), PLA (Ácido Polilático), HIPS (Poliestireno de Alto Impacto), Flexível, Madeira, Nylon, etc, sendo o ABS e PLA os mais comercializados.

O software que oferece suporte às impressoras é o *Repetier-Host (Figura 9)* que comanda toda a ação de movimentação do extrusor e da mesa de impressão, além dos controles de temperaturas, como será fatiado o objeto para a impressão de cada camada, tipo de preenchimento, controle de velocidade (quanto mais lento, melhor a qualidade de impressão), espessura de cada camada de material, gerando também o *G-code* que seria

um código com todos os parâmetros de impressão e se a impressora estiver conectada ao computador pode-se fazer um acompanhamento *online*. Todas essas configurações são definidas pelo usuário de acordo com o material e a qualidade de impressão desejada.

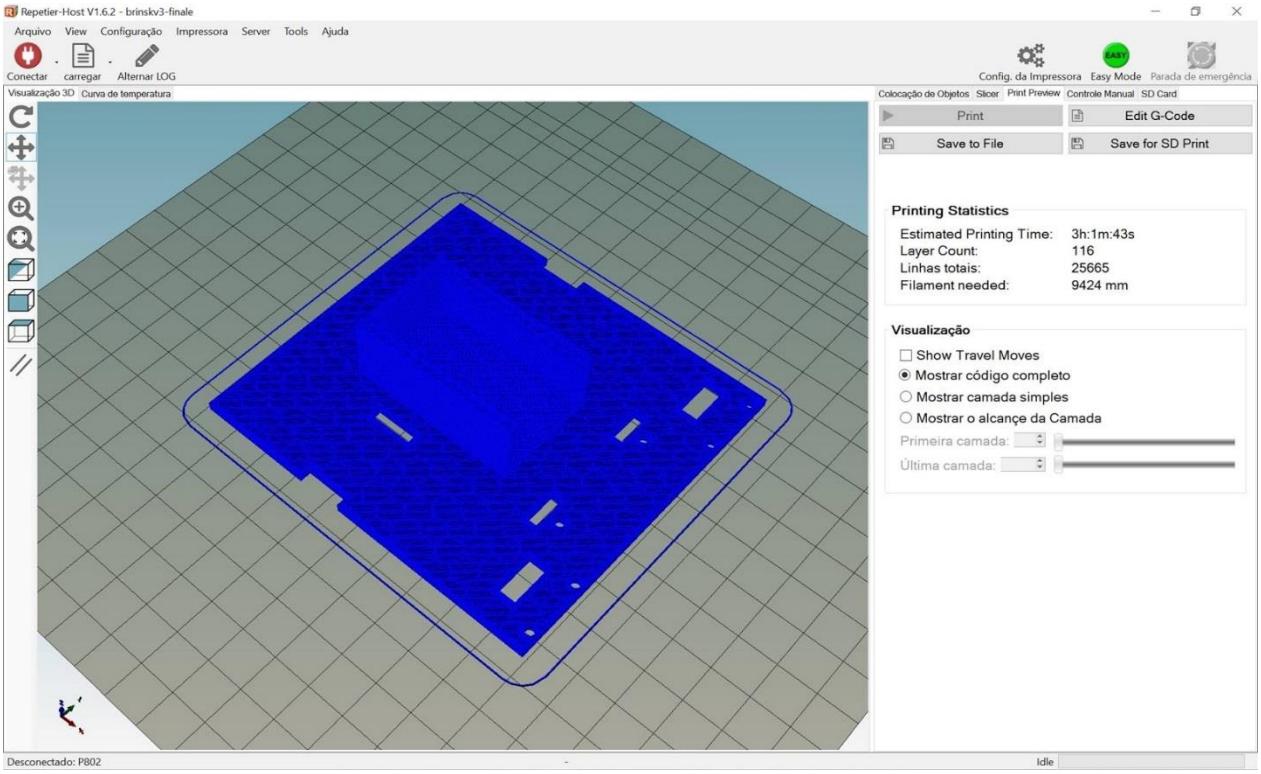

Figura 9 - Interface Repetier-Host

Fonte: Autores, 2016.

Com relação ao preenchimento, temos como principais tipos o *Rectilinear* (linhas retas) que é um padrão de impressão rápida, mas frágil, *Honeycomb* (células hexagonais) que são mais fortes e resistentes por ter a estrutura baseada nos favos de mel e o tipo *Concentric* sendo ideal para impressões com intersecção de um cilindro com um plano paralelo, por exemplo (FAZEDORES, 2014).

3. DESENVOLVIMENTO

O protótipo do seguidor de linha, nomeado Brinsk, foi subdividido em pequenos projetos para um melhor gerenciamento. Primeiramente objetivou-se a construção do seguidor de linha, seguido da detecção de cor e por fim o desenvolvimento do aplicativo móvel e conexão Bluetooth. Para que essas atividades fossem atingidas, o projeto seguiu em três áreas diferentes: eletrônica, *design* e programação.

A pista feita para o Brinsk tem aproximadamente o tamanho de uma folha A0 e segue o padrão apresentado na figura 10. Onde a linha preta é a pista percorrida pelo Brinsk (círculo cinza) que se move no sentido horário e ao lado esquerdo desta linha preta de referência observa-se as cores que serão detectadas pelo sensor de cor posicionado ao lado esquerdo.

Figura 10 - Exemplo de pista

Fonte: Autores, 2016.

3.1 ELETRÔNICA

A parte da eletrônica pode ser subdividido em seis pequenos blocos (Figura 11), interligados seja para ligação de energia e/ou transferência de dados.

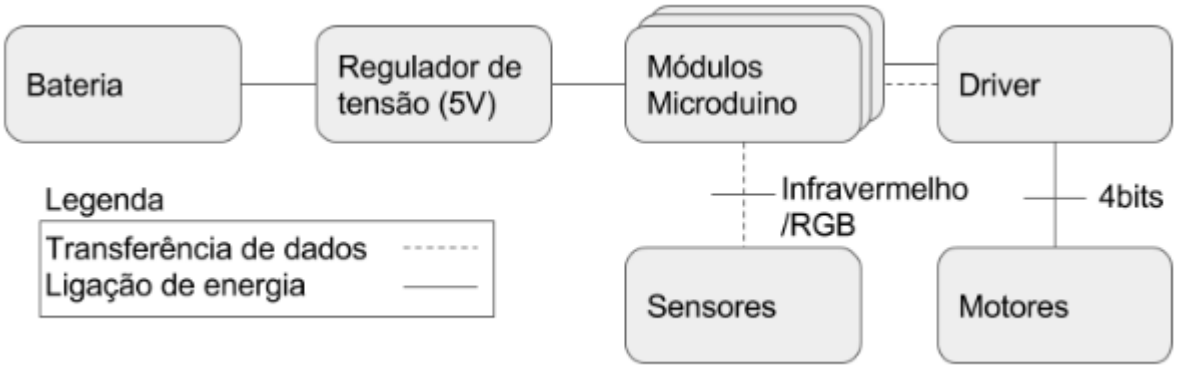

Figura 11 - Esquema da parte eletrônica

Fonte: Autores, 2016.

As placas da Microduino possuem 27 pinos (Tabela 3) divididos em portas analógicas, digitais e alimentação (5V, 3.3V, terra representado pelas letras GND do inglês *ground* e um *reset* - RST).

Tabela 3 - Pinos da placa Microduino

Pin	Original Pin Name	Map Pin Name	Digital Pin	Analog Pin	interrupt	PWM	Serial	SPI	I2C	Power
1	VCC	+5V								+5V
2	VCC	+3V3								+3.3V
3	(AIN1)PD7	D7	D7							
4	(ICP)PB0	D8	D8							
5	(OC1A)PB1	D9	D9			yes				
6	(OC1B/SS)PB2	D10	D10			yes		SS		
7	(OC2A/MOSI)PB3	D11	D11			yes		MOSI		
8	(MISO)PB4	D12	D12					MISO		
9	(SCK)PB5	D13	D13					SCK		
10	AREF	AREF								
11	(ADC0)PC0	A0	D14	A0						
12	(ADC1)PC1	A1	D15	A1						
13	(ADC2)PC2	A2	D16	A2						
14	(ADC3)PC3	A3	D17	A3						
15	(ADC4/SDA)PC4	SDA	D18	A4					SDA	
16	(ADC5/SCL)PC5	SCL	D19	A5					SCL	
17	(ADC6)	A6	D20(only input)	A6						
18	(ADC7)	A7	D21(only input)	A7						
19	(RXD)PD0	D0	D0				0(RX)			
20	(TXD)PD1	D1	D1				0(TX)			
21	(INT0)PD2	D2	D2		0					
22	(OC2B/INT1)PD3	D3	D3		1	yes				
23	(XCK/T0)PD4	D4	D4							
24	(OC0B/T1)PD5	D5	D5			yes				
25	(OC0A/AIN0)PD6	D6	D6			yes				
26	RESET	RST								
27	GND	GND								GND

Fonte: MICRODUINO WIKI 2016 (MICRODUINO CORE, 2016).

Foram utilizados os pinos digitais D4 (motor esquerdo nível lógico 0), D5 (motor direito nível lógico 1), D6 (motor direito nível lógico 0) e D7 (motor esquerdo nível lógico 1) para o controle dos motores, os pinos D2 e D3 para a comunicação serial do sensor de cor, além da utilização das conexões digitais 10 e 12 e de uma analógica I2C utilizando-se conectores mini JST de 4 fios (Figura12) necessários para a parte central do HUB. Todos eles sendo utilizados como saída (exemplo de código: *pinMode*(número do pino digital, *OUTPUT)).*

Figura 12 - Conector mini JST 4 fios

Fonte: Autores, 2016.

Os motores DC utilizados são motores simples feitos para pequenos projetos com Arduino, sendo necessários dois *jumpers* (pequenos cabos conectores) soldados nas duas partes metálicas que aparecem no motor (Figura 13) que representam o sinal lógico alto (1) e baixo (0).

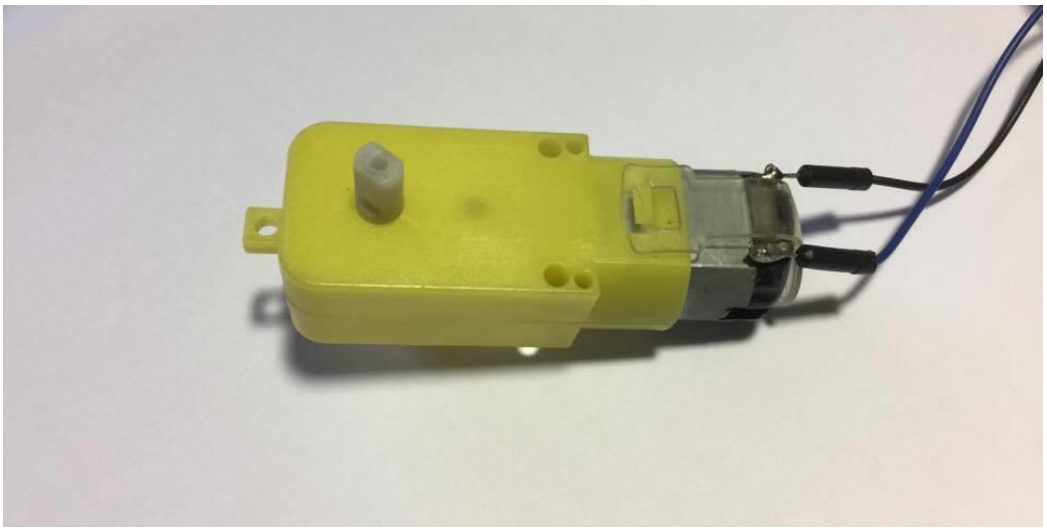

Figura 13 - Motor DC utilizado

Fonte: Autores, 2016.

Para a utilização dos motores DC é muito importante utilizar o *driver* para fazer o controle dos motores, pois estes necessitam de correntes que ultrapassam os valores máximos suportados pelas placas da Microduino. Com isso, a figura 14 ilustra exatamente o significado de cada parte do *driver*.

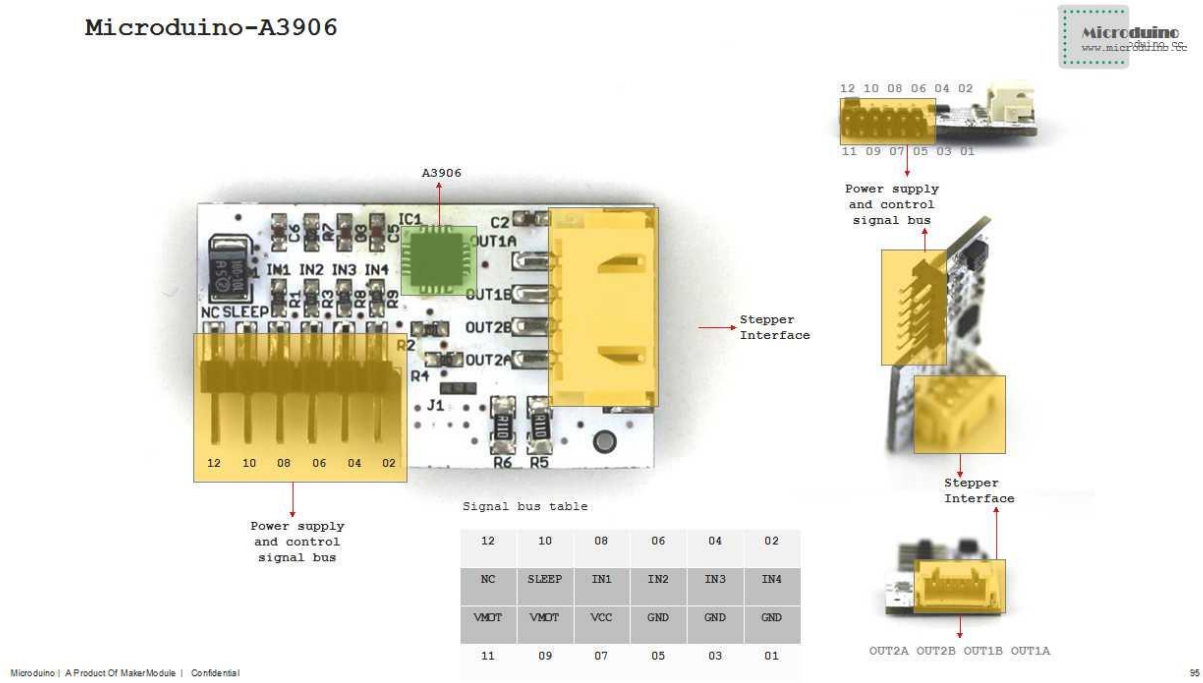

***Figura 14 - Pinagem* driver**

Fonte: MICRODUINO MOTOR, 2016 (site)

Na parte destacada chamada de *Stepper Interface* temos os pinos OUT1A, OUT1B, OUT2B e OUT2A que representam os sinais lógicos alto e baixo de cada motor, sendo assim conectados nos *jumpers* dos dois motores utilizados. Já na parte *Power supply and control signal bus* temos a representação dos pinos de alimentação e de controle de sinal do *driver*, onde alimentamos os pinos 07, 09 e 11 com 5V do pino 5V da Microduino, os pinos 01,03 e 05 com o GND e consequentemente conectamos os pinos 02, 04, 06 e 08 com os pinos digitais D6, D5, D4 e D7 da Microduino, respectivamente.

Para fazer a alimentação de todo o circuito do Brinsk foi utilizado um borne KRE de 2 vias (azul) para conectar duas baterias 9V em paralelo e um botão do tipo *switch* (Figura 15). A utilização de duas baterias 9V em paralelo foi necessária para se obter

uma corrente maior de fornecimento para o circuito, mesmo sabendo que duas baterias nunca possuem a mesma carga (a de maior carga se torna geradora da de menor carga) foi a forma que apresentou resultado mais satisfatório. Além disso, como as placas Microduino suportam apenas 5V, foi utilizado um LM7805 que regula tensões de 7V a 24V para 5V (INSTITUTO NEWTON C. BRAGA, 2014) juntamente com um capacitor 1nF (utilizado somente como um filtro).

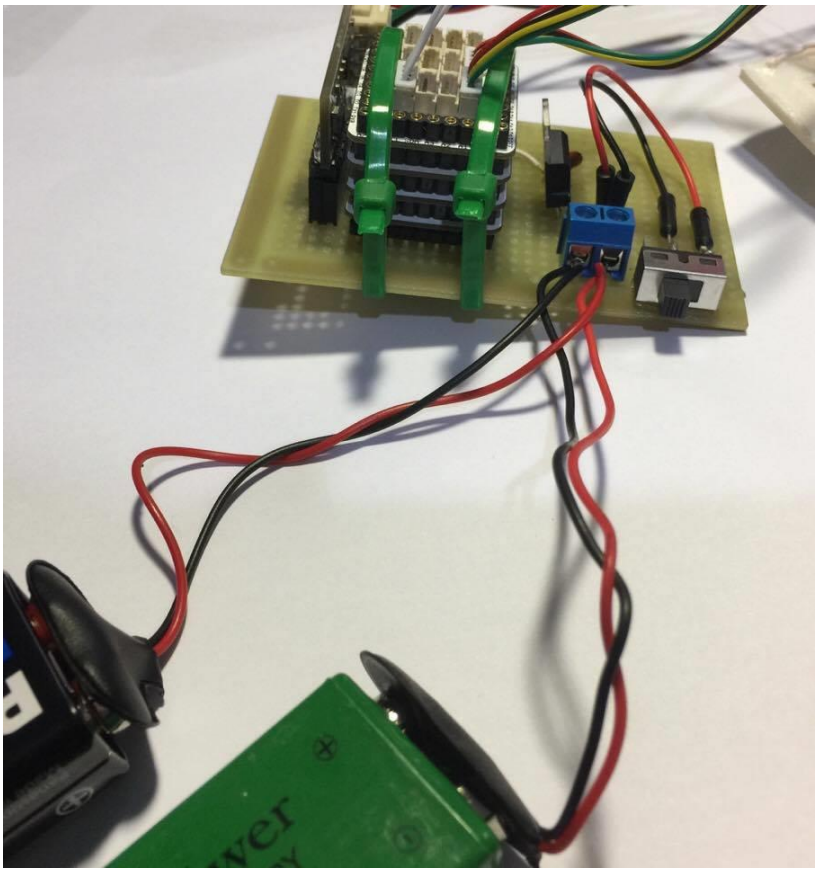

Figura 15 - Ilustração dos componentes de alimentação da placa

Fonte: Autores, 2016.

Todo o circuito foi feito em uma placa de circuito impresso utilizando-se soquetes para conectar o Microduino e o *driver* dos motores, fazendo as trilhas das conexões conforme especificado acima e com o exemplo simplificado da figura 16.

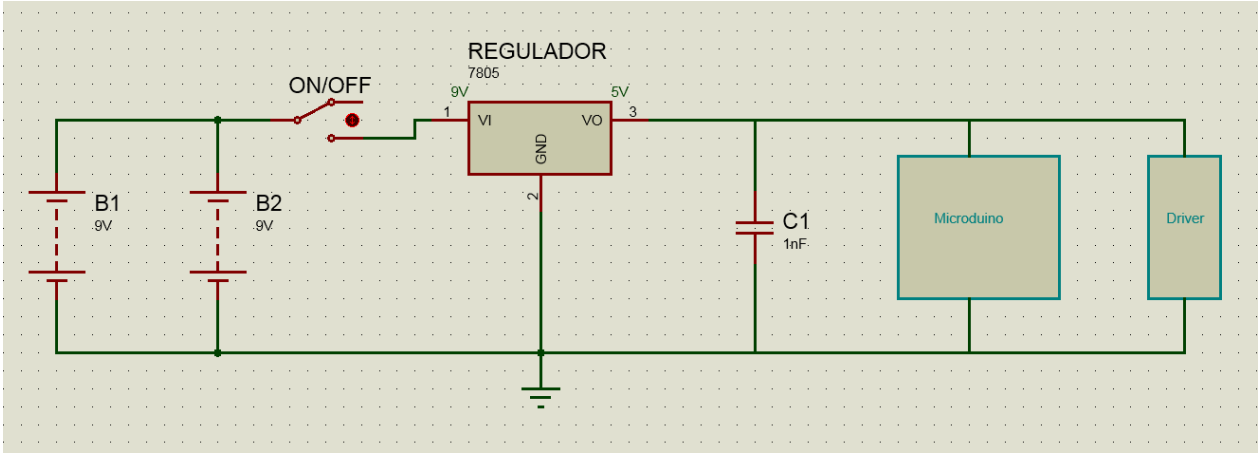

Figura 16 - Diagrama elétrico da placa

Fonte: Autores, 2016.

3.2 PROGRAMAÇÃO

Os programas foram desenvolvidos em dois ambientes distintos: Arduino e Xcode. No primeiro foi programado os três algoritmos (seguidor de linha, sensor de cor e *Bluetooth*) e a integração destes. No segundo foi criado o aplicativo móvel para a reprodução do som das notas musicais enviados pelo Brinsk via *Bluetooth*.

3.2.1 Arduino

A programação no Arduino foi dividida nas inicializações como a abertura da porta serial para o *Bluetooth*, as variáveis globais, inclusão de bibliotecas, dentre outras. Seguindo então para o *loop* (parte do código que fica repetindo), no qual existem três funções: *followLine()* (seguidor de linha), *readRGB* (leitura do sensor de cor) e *bluetoothWrite()* (envio de dado ao aplicativo móvel) (Figura 17).

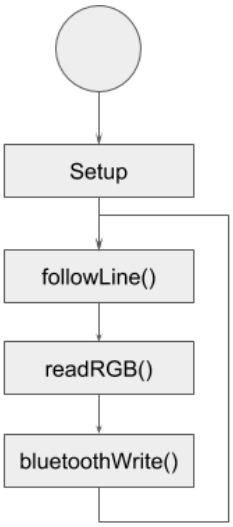

Figura 17 - Representação geral do programa

Fonte: Autores, 2016.

3.2.1.1 Seguidor de linha

O primeiro módulo de programação desenvolvido foi o seguidor de linha, o qual consiste basicamente na leitura dos sinais digitais dos dois sensores infravermelho e na análise se existe algum deles que está diferente de 0, ou seja, detecção de cor escura. Se essa condição for verdadeira, é verificado, então, qual dos sensores detectou a linha preta, acionando apenas o motor do lado contrário deste sensor. Assim se o sensor esquerdo detectar, o motor direito será acionado e o esquerdo desligado, mantendo o Brinsk na linha (Tabela 4). Caso ambos sensores detectarem a cor preta, o Brinsk fica girando no sentido horário até que outra condição seja acionada.

As funções de acionamento do motor foram padronizadas em motorFrente(), motorDireita() e motorEsqueda(), que já formata os quatro bits (dois para cada motor) de entrada do driver Microduino, onde é realizada então a técnica de PWM.

Tabela 4 - Tabela de estados dos sensores de linha

Sensor		Ação
Esquerdo	Direito	
0	0	motorFrente()
0	1	motorEsquerda()
1	0	motorDireita()
1	1	motorDireita()

Fonte: Autores, 2016.

3.2.1.2 Sensor de Cor

Para a utilização do sensor de cor da Microduino, foi utilizado uma biblioteca disponibilizada pela própria empresa, chamada *color.h,* que junto com a biblioteca local *Wire.h* facilita a comunicação serial I2C utilizada pelo sensor. Assim, a função para o módulo do sensor (Figura 18) consiste na leitura das cores utilizando a função *readData* para cada tonalidade de RGB (vermelho, verde e azul), que retorna um valor entre 0 e 32767. Este valor deriva dos quinze sensores e seus dois estados possíveis, ou seja, $2^{15} - 1$.

Após a leitura dos valores de vermelho, verde e azul, estes são convertidos para o formato padrão atribuído a cores neste formato, que varia entre 0 e 255. Este cálculo é realizado por uma regra de três básica.

Uma vez os valores de cores calculados e formatados, eles são estão armazenados em variáveis globais até a próxima detecção, para serem enviados ao aplicativo móvel por Bluetooth.

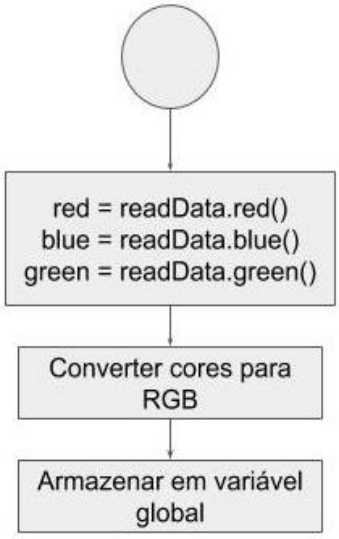

Figura 18 - Representação do sensor de cor

Fonte: Autores, 2016.

3.2.1.3 Bluetooth

Com a porta serial aberta na parte inicial do programa (*Setup*), o bloco responsável por transmitir as cores lidas para o aplicativo móvel (Figura 19), primeiramente verifica se existe alguma porta serial conectada com a função *serial.available()* e verificar se esta retorna algum número maior que zero.

Após essa verificação, é formatado a *string* (palavra) enviada ao aplicativo móvel por *Bluetooth*. Essa formatação consiste em separar as variáveis globais referentes às tonalidades RGB, previamente lidas pelo bloco de código responsável pela leitura das cores com o sensor de cor, em vírgulas.

Com a *string* devidamente formatado com os dados necessários, é então enviada ao aplicativo móvel utilizando da função *serial.print (string).*

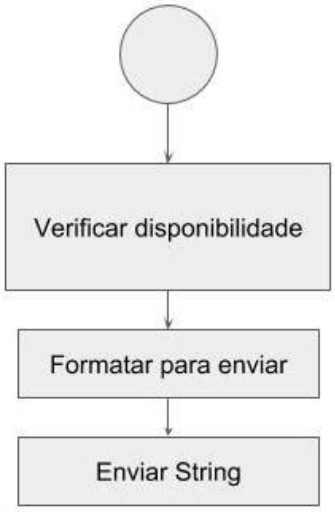

Figura 19 - Representação do funcionamento do *Bluetooth*

Fonte: Autores, 2016.

3.2.2 Xcode

Após testes de comunicação *Bluetooth* com o Microduino e aplicativos de terceiros, foi desenvolvido um simples aplicativo em Swift, no Xcode, para conectar e receber os dados das cores lidas e emitir os sons de acordo de cada cor.

O aplicativo móvel é constituído de duas telas (Figura 20):

- Tela iniciar: na tela responsável em mostrar todos os dispositivos *Bluetooth* disponíveis é primeiramente inicializado o gerenciador central (var activeCentralManager = CBCentralManager(delegate: self, queue: nil)), o qual é responsável em descobrir, conectar e atualizar os status dos dispositivos descobertos. Foi programado que assim que uma linha da lista seja clicada, o dispositivo referente aquela linha será conectada (activeCentralManager.connect(peripheralDevice!, options: nil), onde peripheralDevice é o dispositivo na linha), e se esse dispositivo for Microduino, será então direcionado para a tela seguinte;

- Tela Microduino-Brinsk: nesta tela, o carrinho já esta conectado ao aplicativo móvel e este recebe as informações enviadas pelo hardware atribuindo a cor de fundo à mesma recebida em RGB. Para a conversão dos valores de RGB para a nota musical, foram feitas sete verificações (uma para cada nota) para saber se os valores definidos pelos testes de cada cor e os valores lidos são compatíveis, com uma tolerância de 20 unidades. Por exemplo, foi definido que a nota 'Si' é referente aos valores de vermelho (R) igual a 23, verde (G) 83 e azul (B) 139, assim se algum valor lido entre vermelho igual a 3 ou 43, verde igual 63 ou 103 e azul 119 ou 159, sabe-se que foi lido a cor referente a nota 'Si'

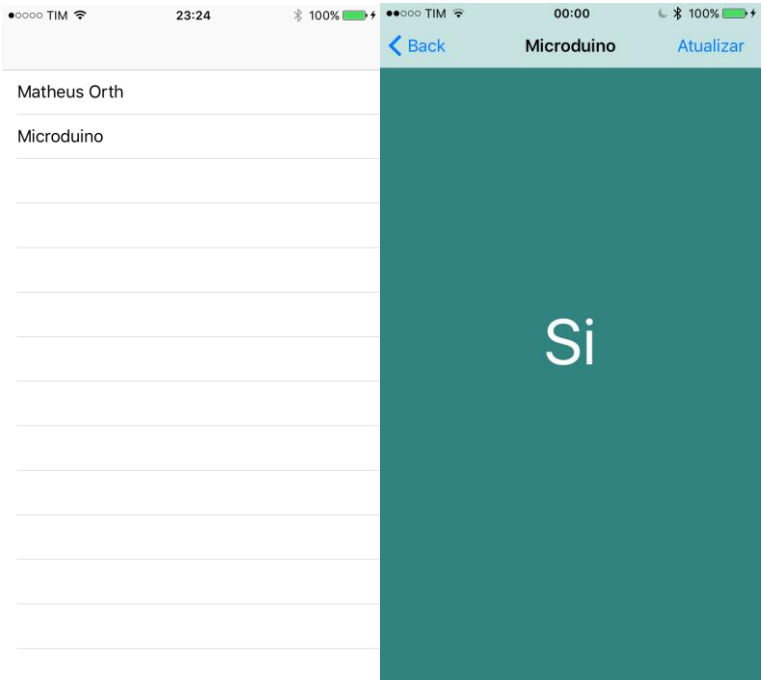

Figura 20 - Interface do aplicativo móvel

Fonte: Autores, 2016.

3.3 DESIGN

3.3.1 Tinkercad

A estrutura física do seguidor foi toda projetada utilizando o Tinkercad (TINKERCAD, 2016), que utiliza de formas geométricas já pré-estabelecidas para as modelagens. O desenho do carrinho variou de formato e tamanho do primeiro ao último modelo (Figura 21). Isso, devido à falta ou sobra de espaço no carrinho, visando encontrar o tamanho ideal, ou seja, a largura mínima, uma vez que a distância entre os sensores de linha exige um tamanho mínimo para a detecção da linha preta. E para a melhor detecção da cor, concluiu-se que é melhor o sensor de cor ao lado do sensor de linha do que atrás, pois assim não haveria a possibilidade da cor passar despercebida, caso os sensores estivessem em cima da linha preta de referência. Tendo a largura mínima, calculou-se o espaço mínimo para os motores, os quais servem de apoio para a estrutura da eletrônica, ainda sobrando um espaço entre eles para suportar as duas baterias 9V.

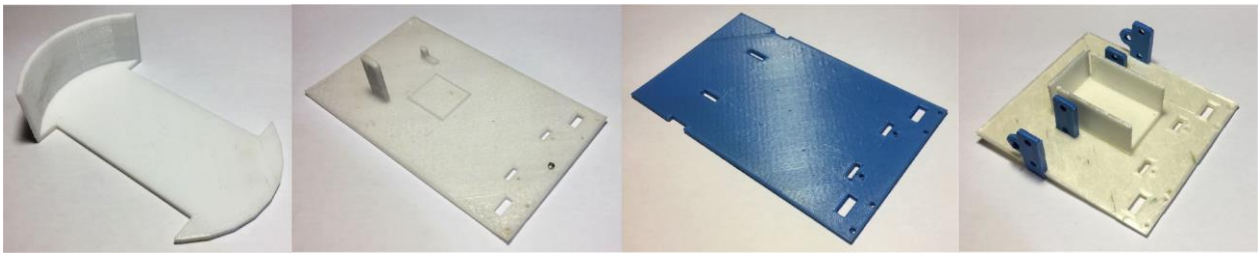

Figura 21 - Evolução da estrutura do Brinsk

Fonte: Autores, 2016.

No primeiro protótipo foi idealizado o formato redondo e um espaço lateral para as rodas (medidas 120 mm x 75 mm com espessura de 3 mm como padrão para todos os modelos), no entanto a largura foi insuficiente para comportar os quatro sensores e a detecção da linha preta de referência.

Levando em conta a largura mínima necessária (90 mm), foi desenhado o segundo seguidor de linha (150 mm x 90 mm), já com os orifícios para os cabos dos sensores (mantida nos seguintes desenhos) e os suportes para os motores (10 mm x 30 mm). Muito embora, a fragilidade de apenas um suporte por motor, com uma base de pequena área e material delicado, um deles se quebrou. Repara-se também na tentativa de ter uma base fixa para o Microduino.

Com o aprendizado do segundo projeto, foi redesenhado os suportes separadamente (visível no último rascunho) para encaixar com os novos orifícios, além da retirada do suporte para o Microduino, mantendo-se as medidas do segundo chassi.

No último projeto, foi reduzido o tamanho consideravelmente (100mm x 90mm), uma vez que os módulos Microduino foram instalados em cima dos motores. Vê-se o suporte da bateria de 9V entre os suportes dos motores, servindo também de apoio para a eletrônica.

Além do chassi do protótipo, também foram desenhadas as rodas e o apoio frontal (que faz o papel da roda da frente) em dois modelos. Para ambas foram realizadas duas tentativas de impressão até chegar nos modelos finais (Figura 22).

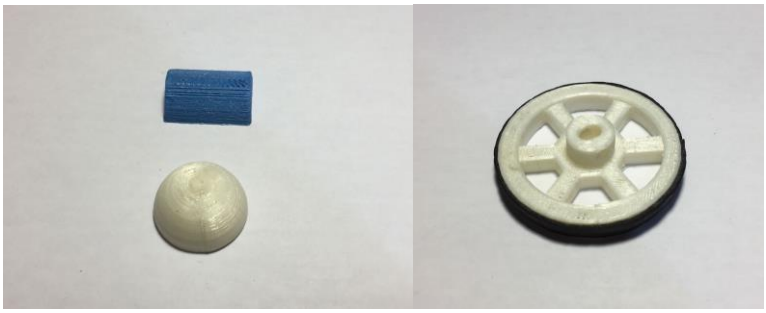

Figura 22 – Modelo final dos apoios frontais e da roda

Fonte: Autores, 2016.

Primeiramente foi testado o apoio frontal no formato de meio cilindro (com diâmetro de 9 mm) e depois de meia esfera (com diâmetro de 12 mm). Devido, principalmente, a altura e a indiferença do atrito, decidiu-se utilizar o apoio de meio cilindro. Na primeira roda ocorreu um erro de impressão, definida na próxima sessão, além do mau dimensionamento do eixo do motor, fazendo com que o encaixe na roda sobrasse espaço. Com as rodas (45 mm de diâmetro e 5 mm de espessura) devidamente

impressas e dimensionadas, foi acoplado uma borracha utilizada nas raquetes de tênis para fazer a função do pneu no carrinho (Figura 23).

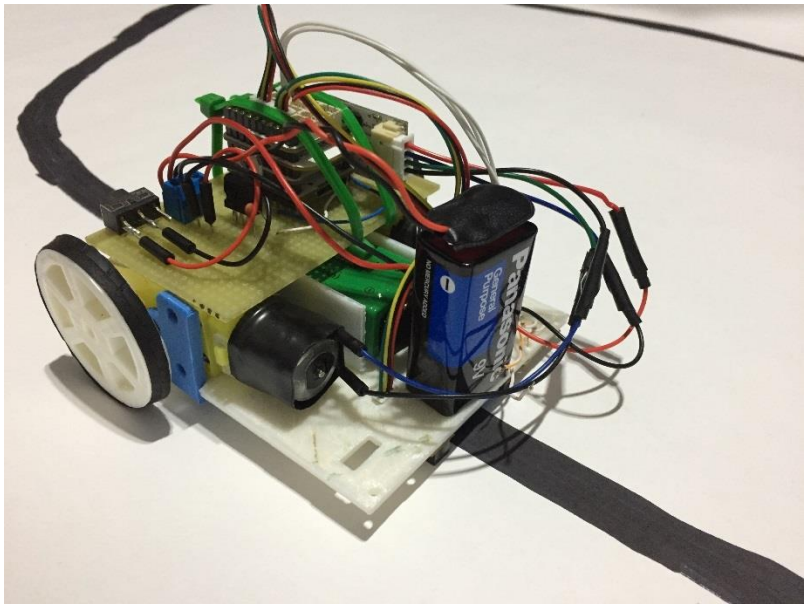

Figura 23 - Brinsk modelo final

Fonte: Autores, 2016.

3.3.2 Impressora 3D

A impressora 3D utilizada foi a Reprap Prusa I3 que tem suporte de temperatura de extrusor de 170ºC a 275ºC e temperatura de mesa de até 110ºC. Os materiais utilizados para impressão do protótipo e das peças foram o ABS e o PLA, ambos termoplásticos, porém um sendo derivado do petróleo (ABS) e o outro derivado de fontes renováveis (PLA) como amido de milho, raízes de mandioca e cana.

A utilização do ABS gerou alguns problemas de “entortamento” nas bordas das peças, chamado de efeito *warp* (Figura 24), devido à falta de uma estufa em volta da impressora (presente somente em impressoras profissionais) para diminuir o choque térmico causado na peça ao esfriar, além é claro de ter que trabalhar com a mesa de impressão acima dos 105ºC, valor muito próximo à máxima temperatura suportada pela

impressora (110ºC). Com isso, o PLA, que se apresentou mais resistente, sem os efeitos *warp* e sem a exigência de uma temperatura elevada na mesa, foi o material escolhido para a confecção final de todo o carrinho.

Figura 24 - Ilustração do efeito *warp*

Fonte: Autores, 2016.

Das diversas formas de preenchimento, foi utilizado o *Honeycomb (Figura 25)*, uma vez que é o mais utilizado por garantir uma estrutura mais forte.

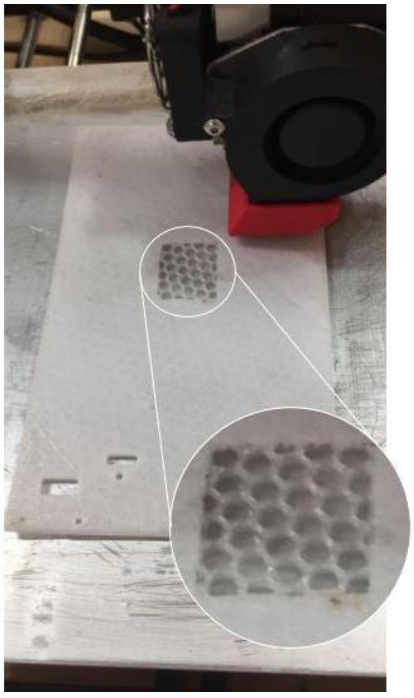

Figura 25 - Preenchimento *Honeycomb*

Fonte: Autores, 2016.

3.4 TESTES

Os testes por parte de hardware sempre tiveram a tela de monitoramento serial, disponível na *IDE*, como referência. Para isso é necessário programar para que exista a impressão desejada (*Serial.print(variável)*). Assim, os primeiros testes foram com o algoritmo do seguidor de linha, que consistiu em verificar os sinais de retorno para o branco e preto, além de configurar os motores. Antes de utilizar o *driver* para os testes com os motores, foram utilizados quatro *LEDs* simulando os quatro bits que seriam enviados ao *driver*.

Com o algoritmo e *hardware* do seguidor de linha devidamente prontos, foi testado, em um novo arquivo para o programa, o algoritmo e sensor responsáveis pela leitura das cores. A utilização da biblioteca *color.h* fez com que o teste precisasse de apenas um ensaio, verificando, mais uma vez, no monitoramento serial.

Antes de ser desenvolvido o aplicativo, foi testado a conexão *Bluetooth.* Para essa foi primeiramente feito com um aplicativo de terceiro e a placa Microduino *Core*+BLE, que consiste em *Core* e BLE na mesma placa, no entanto por falta de material no site da Microduino, a conexão não teve sucesso. Estima-se que, assim como o *Bluetooth* necessita de uma porta serial para a comunicação e a compilação do algoritmo também se dá por uma porta serial via USB com o computador, talvez possa existir alguma interferência e assim a compilação não é completada. Trocando então esta placa para dois módulos distintos, um sendo o Core e outro BLE, foi compilado o programa sem a placa do *Bluetooth,* sendo acoplado apenas quando a compilação fosse completa (instruções do site Microduino Motor). E assim foi feita a comunicação tanto do *hardware* para o aplicativo de terceiro como do aplicativo para o *hardware.* Essa verificação foi feita através do monitoramento serial e o clássico liga/desliga de *LED*.

Confirmando ser possível a comunicação *Bluetooth, foi* desenvolvido o aplicativo móvel. O qual, não teve algum problema na parte da comunicação. O único recurso que ficou a desejar foi o controle da duração das notas musicais, uma vez que as amostras de piano têm duração fixa, ou seja, em vez de ter um som contínuo de uma nota, tem-se pulsos da mesma nota.

Na parte de impressão 3D, tivemos um certo trabalho em cada impressão realizada, devido à falta de ajuste automático da impressora tendo que ser realizada manualmente e a cada impressão, somando-se ainda as impressões que não tiveram um resultado adequado. Tudo isso demandou um tempo razoável na elaboração do projeto, visto que a impressão de cada chassi levou cerca de três horas para ser concluída e as rodas levaram em média trinta minutos cada. Já os suportes, tanto dos motores como o apoio frontal, tiveram uma impressão mais rápida por serem peças menores, levando em média de dez a quinze minutos.

Nos testes realizados com cada função do Brinsk, conseguimos resultados satisfatórios, sem qualquer problema com relação à *bugs* (pequenas falhas que fazem com que não funcione com fluidez) ou por *hardware.*

Porém, quando foram colocadas para funcionar em conjunto, apresentaram falhas, principalmente pela falta de uma fonte de energia que oferecesse suporte para toda a corrente e carga necessária para um funcionamento bom e duradouro do Brinsk.

A única fonte em que conseguimos resultados satisfatórios foram as baterias 9V, visto que elas forneceram a corrente necessária para o funcionamento de todo o Brinsk, porém com o único defeito de possuírem pouca duração, fazendo com que funcionasse por apenas alguns segundos quando as baterias estavam com a carga completa.

4. CONCLUSÃO

Ao projetar o seguidor de linha, foi obtido um resultado bom e sem nenhum problema quanto ao funcionamento dos sensores de linha, dos motores e do drive, apresentando uma autonomia satisfatória de bateria.

Na detecção de cores, apesar do impasse com relação à utilização de dois sensores de cor, conseguimos atingir os objetivos com apenas um sensor, optando-se assim pela utilização dessa configuração.

A criação do aplicativo móvel teve como grande dificuldade a implementação dos sons de piano referentes a cada cor escolhida, uma vez que os sons tinham uma duração fixa não permitindo a flexibilidade de ritmo. Além disso, ainda existe a oportunidade de desenvolver mais alguns recursos, como permitir ao usuário fazer a relação da cor com a nota musical.

Na conexão *Bluetooth* apesar dos muitos problemas para iniciarmos essa etapa, conseguimos com sucesso fazer a conexão do Brinsk com o Microduino.

Na modelagem 3D, apesar dos problemas com a utilização do ABS em que algumas peças ficaram tortas ou até mesmo com uma estrutura frágil, não afetou em grandes proporções o andamento do projeto, pois mesmo com esses problemas as peças se mostraram funcionais para a realização dos testes.

Toda a documentação para a formulação do projeto foi realizada com sucesso, pois conseguimos desenvolver o Brinsk com todas as suas funções apesar dos desafios, possibilitando ainda uma continuidade de melhorias no aplicativo e na estabilidade com a integração de algoritmos no futuro.

REFERÊNCIAS

ARDUINO. **What is Arduino?** Disponível em: <http://www.arduino.org/learning/getting-started/what-is-arduino>. Acesso em 17 nov. 2016 3:00.

ARS CONSULT. **Apostila de Introdução a Robótica**. Recife, 1995.

BISSOLOTTI, Katielen; GONÇALVES, Berenice; PEREIRA, Alice Theresinha Cybis. **Design Centrado na Criança: Estudo de Recomendações para uma boa Experiência.** Santa Catarina, jun. 2015. 15 ERGODESIGN. Disponível em <http://pdf.blucher.com.br.s3-sa-east-1.amazonaws.com/designproceedings/15ergodesign/90-U058.pdf>. Acesso em: 21 jun. 2016 01:12.

COELHO, Christina T. **A música na educação infantil.** 2013**.** Disponível em <http://pedagogiaaopedaletra.com/a-musica-na-educacao-infantil>. Acesso em 20 out. 2015 17:38.

COIMBRA, José C. T. **Aplicações da robótica no ensino secundário: O sistema LEGO MINDSTORMS e a física.** Faculdade de Ciências e Tecnologia da Universidade de Coimbra. Disponível em: <http://fisica.uc.pt/data/20072008/apontamentos/apnt_220_3.pdf>. Acesso em 19 out. 2015 20:44.

DEMO, Pedro. **Política social, educação e cidadania.** 10. ed. Campinas: Papirus, 2007. p. 131.

DEVELOPER APPLE. **Xcode.** Disponível em: <https://developer.apple.com/xcode/ide>. Acesso em 15 set. 2016 21:10.

DICIO. **Dicionário Online de Português**. 2016. Definição de lúdico. Disponível em <http://www.dicio.com.br/ludico/>. Acesso em: 20 jun. 2016 00:16.

FAZEDORES. **O que é o quê nas Slicing settings para impressão 3D.** Disponível em: <http://blog.fazedores.com/o-que-e-o-que-nas-slicing-settings-para-impressao-3d/>. Acesso em 18 nov. 2016 15:10.

FERREIRA, Sueli de A.; PEREIRA, Magno C.; Oliveira, Alexsander X.; FÉO, Alexandre E. **Bluetooth.** Resende, 2005. SEGeT. Disponível em <http://www.aedb.br/seget/arquivos/artigos05/16_artigo_Bluetooth.pdf>. Acesso em: 21 jun. 2016 02:05.

GODOI, Luiz R. **A importância da música na educação infantil.** 2011. 36 pág. Trabalho de conclusão de curso. UEL, Londrina, 2011. Disponível em: <http://www.uel.br/ceca/pedagogia/pages/arquivos/LUIS%20RODRIGO%20GODOI.pdf >. Acesso em: 19 jun. 2016 20:40.

INSTITUTO NEWTON C. BRAGA. **Reguladores de tensão 7800 (ART156).** 2014. Disponível em < http://www.newtoncbraga.com.br/index.php/como-funciona/1076-art156>. Acesso em 10 nov. 2016 17:41

MED, Bohumil. **Teoria da música.** 4. ed. Brasília. Musimed, 1996.

MICRODUINO BLE, **Microduino-Module BLE.** 2016. Disponível em: <https://wiki.microduino.cc/index.php/Microduino-Module_BLE>. Acesso em 11 out. 2016 20:20.

MICRODUINO COLOR, **Sensor-Color Detector.** 2016. Disponível em: <https://wiki.microduino.cc/index.php/Sensor-Color_Detector>. Acesso em 15 set. 2016 20:30

MICRODUINO CORE, **Microduino-Module Core.** 2016. Disponível em: <https://wiki.microduino.cc/index.php/Microduino-Module_Core>. Acesso em 20 jun. 2016 18:50.

MICRODUINO HUB, **Microduino-Module Sensor Hub.** 2016. Disponível em: <https://wiki.microduino.cc/index.php/Microduino-Module_Sensor_Hub>. Acesso em 10 jul. 2016 17:10.

MICRODUINO MOTOR, **Microduino-Module Motor.** 2016 Disponível em: <https://wiki.microduino.cc/index.php/Microduino-Module_Motor>. Acesso em 20 jun. 2016 19:53.

MICRODUINO TOUR, **Microduino-Line-track.** 2015. Disponível em: <https://wiki.microduino.cc/index.php/Microduino-Line-track>. Acesso em 10 jul. 2016 17:00.

MICRODUINO USBTTL, **Microduino-Module USBTTL.** 2016. Disponível em: <https://wiki.microduino.cc/index.php/Microduino-Module_USBTTL>. Acesso em 20 jun. 2016 19:07.

MUNIZ, Aline. **Sete cores e sete sons.** Rede Record, 2014. Disponível em <http://entretenimento.r7.com/blogs/aline-muniz/sete-cores-e-sete-sons-20140428>. Acesso em: 20 jun. 2016 23:20.

PLATÃO. **Frase site.** Disponível em:<http://www.frase.site/platao/a-musica-e-a-essencia-da-ordem-eleva-todas-as-almas-para-o-que-e-bom-justo-e-belo-e-deve-ser-para-a-alma-o-que-para-o-corpo-e-a-ginastica.html>. Acesso em 10 nov. 2016 22:57

STRATASYS. **O que é prototipagem rápida?** Disponível em: <http://www.stratasys.com/br/resources/rapid-prototyping>. Acesso em 17 nov. 2016 02:43.

TINKERCAD. **Tinkercad**. 2016. Disponível em: <https://www.tinkercad.com>. Acesso em 20 maio 2016 16:15.

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ. **Sistema de Bibliotecas. Normas para elaboração de trabalhos acadêmicos**. Curitiba: UTFPR, 2008. Disponível em: <http://www3.utfpr.edu.br/dibib/normas-para-elaboracao-de-trabalhos-academicos/normas_trabalhos_utfpr.pdf>. Acesso em: 22 out. 2015 16:15.

ZILLI, Silvana do Rocio. **A robótica educacional no ensino fundamental: Perspectivas e prática.** 2004. 89 pág. Dissertação de Mestrado. UFSC, Florianópolis, 2004. Disponível em <https://repositorio.ufsc.br/xmlui/handle/123456789/86930>. Acesso em: 20 out 2015 22:10.

APÊNDICE A – CÓDIGO DO APLICATIVO

Tela lista de dispositivos Bluetooth ao redor

```
//
//  DevicesViewController.swift
//  Brinsk
//
//  Created by Matheus Orth on 04/11/16.
//  Copyright © 2016 Orth. All rights reserved.
//

import UIKit
import CoreBluetooth
import ExternalAccessory

class DevicesViewController: UIViewController, CBCentralManagerDelegate,
CBPeripheralDelegate, UITableViewDelegate, UITableViewDataSource {

   var activeCentralManager: CBCentralManager?
   var peripheralDevice: CBPeripheral?
   var devices: Dictionary<String, CBPeripheral> = [:]
   var deviceName: String?
   var devicesRSSI = [NSNumber]()
   var devicesServices: CBService!
   var deviceCharacteristic: CBCharacteristic!
   var deviceCharacteristicFFF1: CBCharacteristic!
   var deviceCharacteristicsArray: [CBCharacteristic]!

   @IBOutlet weak var tableView: UITableView!

   override func viewDidLoad() {
```

```
        super.viewDidLoad()
    }

    override func viewDidAppear(_ animated: Bool) {
        devices.removeAll(keepingCapacity: false)
        devicesRSSI.removeAll(keepingCapacity: false)
        activeCentralManager = CBCentralManager(delegate: self, queue: nil)

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func centralManagerDidUpdateState(_ central: CBCentralManager) {
        if central != nil{
            if central.state == .poweredOn{
                central.scanForPeripherals(withServices: nil, options: nil)
                print("Searching for BLE Devices")
            }
            else {
                print("Bluetooth switched off or not initialized")
            }
        }
    }

    func centralManager(_ central: CBCentralManager, didDiscover peripheral:
CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {

        if let name = peripheral.name{
            if(devices[name] == nil){
```

```
            devices[name] = peripheral
            devicesRSSI.append(RSSI)
            self.tableView.reloadData()
        }
    }
}

func centralManager(_ central: CBCentralManager, didConnect peripheral:
CBPeripheral) {
    if let peripheralDevice = peripheralDevice{
        peripheralDevice.discoverServices(nil)
        if navigationController != nil{
            navigationItem.title = "Connected to \(deviceName!)"
        }
    }
}

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "homeSegue"{
        let destVC = segue.destination as! HomeViewController
        destVC.deviceCharacteristicFFF1 = deviceCharacteristicFFF1
        destVC.deviceCharacteristics = deviceCharacteristic
        destVC.peripheralDevice = peripheralDevice
    }
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    for service in peripheral.services! {
        let thisService = service as CBService
        if thisService.uuid.uuidString == "FFF0"{
            peripheral.discoverCharacteristics(nil,for: thisService)
```

```
        }
        print(thisService)
    }
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service:
CBService, error: Error?) {

    if (error == nil){
        for charateristic in service.characteristics! {
            let thisCharacteristic = charateristic
            peripheral.setNotifyValue(true, for: thisCharacteristic)
            if navigationController != nil{
                navigationItem.title = "Discovered Characteristic for \(deviceName!)"
            }

            if thisCharacteristic.uuid.uuidString == "FFF1"{
                deviceCharacteristicFFF1 = thisCharacteristic
            }
            deviceCharacteristic = thisCharacteristic
        }

        if let navigationController = navigationController{
            let _ = navigationController.popViewController(animated: true)
        }
        performSegue(withIdentifier: "homeSegue", sender: self)
    }
}

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic:
CBCharacteristic, error: Error?) {
```

```
        print("got some")
    }

    func cancelConnection(){
        if let activeCentralManager = activeCentralManager{
            if let peripheralDevice = peripheralDevice{
                print(peripheralDevice)
                activeCentralManager.cancelPeripheralConnection(peripheralDevice)
            }
        }
    }

    func centralManager(_ central: CBCentralManager, didDisconnectPeripheral
peripheral: CBPeripheral, error: Error?) {
        print("Disconnected")
        central.scanForPeripherals(withServices: nil, options: nil)
    }

    func writeValue(data: String){
        let data = (data as NSString).data(using: String.Encoding.utf8.rawValue)
        if let peripheralDevice = peripheralDevice{
            if let deviceCharacteristics = deviceCharacteristic{
                peripheralDevice.writeValue(data!, for: deviceCharacteristics, type:
CBCharacteristicWriteType.withoutResponse)
            }
        }
    }


    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
```

```
  }
  func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
{
      return devices.count
  }

  func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {

      let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath as
IndexPath)

      let discoveredPeripheralArray = Array(devices.values)
      if let name = discoveredPeripheralArray[indexPath.row].name{
          if let textLabelText = cell.textLabel{
              textLabelText.text = name
          }
      }

      return cell
  }

  func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
      if (devices.count > 0){
          let discoveredPeripheralArray = Array(devices.values)
          peripheralDevice = discoveredPeripheralArray[indexPath.row]
          if let peripheralDevice = peripheralDevice{
              peripheralDevice.delegate = self
              deviceName = peripheralDevice.name!
          }
          else{
```

```
            deviceName = " "
        }
        if let activeCentralManager = activeCentralManager{
            activeCentralManager.stopScan()
            activeCentralManager.connect(peripheralDevice!, options: nil)
            if navigationController != nil{
                navigationItem.title = "Connecting \(deviceName!)"
            }
        }
    }
  }
}
```

_____

Tela de emitir os sons

```
//
//  HomeViewController.swift
//  Brinsk
//
//  Created by Matheus Orth on 04/11/16.
//  Copyright © 2016 Orth. All rights reserved.
//

import UIKit
import CoreBluetooth


class HomeViewController: UIViewController, CBCentralManagerDelegate,
CBPeripheralDelegate {
```

```
    var peripheralDevice: CBPeripheral?
    var deviceCharacteristics: CBCharacteristic!
    var deviceCharacteristicFFF1: CBCharacteristic!

    var redRGB = Float()
    var greenRGB = Float()
    var blueRGB = Float()

    var colorDetected = UIColor()

    @IBOutlet weak var notaLabel: UILabel!



    override func viewDidLoad() {
        super.viewDidLoad()

        peripheralDevice?.delegate = self
        self.navigationItem.title = "Microduino"
        self.navigationItem.rightBarButtonItem = UIBarButtonItem(title: "Atualizar", style:
.plain, target: self, action: #selector(atualizarAction))

        notaLabel.text = "Si"
        if let services = peripheralDevice?.services{
            for service in services{
                peripheralDevice?.discoverCharacteristics(nil, for: service)
            }
        }
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
```

```
    // Dispose of any resources that can be recreated.
  }

  func centralManagerDidUpdateState(_ central: CBCentralManager) {
        if central.state == .poweredOn {
            print("Bluetooth ON")
        }
        else {
            print("Bluetooth switched off or not initialized")
        }
  }


  func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic:
CBCharacteristic, error: Error?) {
      if let characteristicValue = characteristic.value{
          let datastring = NSString(data: characteristicValue, encoding:
String.Encoding.utf8.rawValue)
          if let datastring = datastring{
              //navigationItem.title = datastring as String
              print(datastring)
              if datastring.contains(","){
                  let rgbArray = datastring.components(separatedBy: ",")
                  redRGB = rgbArray[0].floatValue
                  greenRGB = rgbArray[1].floatValue
                  blueRGB = rgbArray[2].floatValue
                  setColor()
                  getNota()
              }
          }
      }
```

```
    }

    func setColor(){


        colorDetected = UIColor(red: CGFloat(redRGB/255), green:
CGFloat(greenRGB/255), blue: CGFloat(blueRGB/255), alpha: 1.0)
        self.view.backgroundColor = colorDetected
    }

    func writeValue(data: String){
        let data = (data as NSString).data(using: String.Encoding.utf8.rawValue)
        if let peripheralDevice = peripheralDevice{
            if let deviceCharacteristics = deviceCharacteristics{
                peripheralDevice.writeValue(data!, for:
                    deviceCharacteristics, type: CBCharacteristicWriteType.withoutResponse)
                readValue()
            }
        }
    }

    func readValue(){
        peripheralDevice?.readValue(for: deviceCharacteristics)
    }

    func atualizarAction(){
        writeValue(data: "Refresh")
    }

    func getNota(){
```

```
    var notaString = "_"

    if betweenRGB(red: 30, green: 109, blue: 55){//Dó
        print("Dó")
        notaString = "Dó"
    }else if betweenRGB(red: 153, green: 237, blue: 59){//Ré
        print("Ré")
        notaString = "Ré"
    }else if betweenRGB(red: 167, green: 109, blue: 51){//Mi
        print("Mi")
        notaString = "Mi"
    }else if betweenRGB(red: 65, green: 45, blue: 39){//Fá
        print("Fá")
        notaString = "Fá"
    }else if betweenRGB(red: 65, green: 71, blue: 83){//Sol
        print("Sol")
        notaString = "Sol"
    }else if betweenRGB(red: 205, green: 103, blue: 125){//La
        print("La")
        notaString = "La"
    }else if betweenRGB(red: 23, green: 83, blue: 139){//Si
        print("Si")
        notaString = "Si"
    }else{//Pausa
        print("Pausa")

    }

    notaLabel.text = notaString
}
```

```
  func betweenRGB(red: Float, green: Float, blue: Float) -> Bool{

      let tolerance : Float = 20.0

      if redRGB <= (red + tolerance) && redRGB >= (red - tolerance) && greenRGB <=
(green + tolerance) && greenRGB >= (green - tolerance) && blueRGB <= (blue +
tolerance) && blueRGB >= (blue - tolerance) {
          return true
      }
      return false
  }
}
extension String {
  var floatValue: Float {
      return (self as NSString).floatValue
  }
}
```

APÊNDICE B – CÓDIGO DO BRINSK

```
#include <Wire.h>
#include <math.h>
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2, 3);

#define my_Serial mySerial

#define redMax 32767
#define greenMax 32767
#define blueMax 32767

String msg = "";
String comdata = "";
int redValue, greenValue, blueValue;
int lineLeft = 10;
int lineRight = 12;

int MotorLUp = 13;    //D13 - IN1
int MotorLDown = 4;  //D4 - IN2
int MotorRUp = 5;    //D5 - IN3
int MotorRDown = 6;  //D6 - IN4

#include "colorSensor.h"

void setup()
{
 Serial.begin(9600);
 my_Serial.begin(9600);
 Wire.begin(); // join i2c bus (address optional for master);
 pinMode(MotorLUp, OUTPUT);
```

```
 pinMode(MotorLDown, OUTPUT);
 pinMode(MotorRUp, OUTPUT);
 pinMode(MotorRDown, OUTPUT);
 pinMode(lineLeft, INPUT);
 pinMode(lineRight, INPUT);
 //pinMode(speedMotor,OUTPUT);
}

void loop()
{
 setTimingReg(INTEG_MODE_FREE);
 setInterruptSourceReg(INT_SOURCE_GREEN);
 setInterruptControlReg(INTR_LEVEL | INTR_PERSIST_EVERY);
 setGain(GAIN_1 | PRESCALER_4);
 setEnableADC();
 while (1)
 {
   followLine();
   readRGB();
   calculateCoordinate();
   bluetoothWrite();
   delay(1);
   clearInterrupt();
   //colorWipe(strip.Color(redValue, greenValue, blueValue), 0);
 }
}

void readRGB()
{
 Wire.beginTransmission(COLOR_SENSOR_ADDR);
 Wire.write(REG_BLOCK_READ);
```

```
 Wire.endTransmission();

 Wire.beginTransmission(COLOR_SENSOR_ADDR);
 Wire.requestFrom(COLOR_SENSOR_ADDR, 8);
 //delay(500);
 delay(15);
 if (8 <= Wire.available())  // if two bytes were received
 {
   for (i = 0; i < 8; i++)
   {
     readingdata[i] = Wire.read();
   }
 }
 green = readingdata[1] * 256 + readingdata[0];
 red = readingdata[3] * 256 + readingdata[2];
 blue = readingdata[5] * 256 + readingdata[4];
 clr = readingdata[7] * 256 + readingdata[6];


 // redValue=map(red, 0, 29888, 0, 255);
 // greenValue=map(green, 8, 31048, 0, 255);
 // blueValue=map(blue, 0, 21888, 0, 255);

 redValue = map(red, 0, redMax, 0, 255);
 greenValue = map(green, 0, greenMax, 0, 255);
 blueValue = map(blue, 0, blueMax, 0, 255);
}

void bluetoothWrite(){
 if (my_Serial.available() > 0){
  Serial.println("Depoisdoif");
```

```
    if (digitalRead(5) == LOW){
        digitalWrite(5,HIGH);
   }else{
        digitalWrite(5,LOW);
   }
   my_Serial.print(redValue, DEC);
   my_Serial.print(", ");
   my_Serial.print(greenValue, DEC);
   my_Serial.print(", ");
   my_Serial.print(blueValue, DEC);
 }
}

void followLine(){
 int lineLeftState = digitalRead(lineLeft);
   delay(1);

 int lineRightState = digitalRead(lineRight);
 delay(1);

 if (lineLeftState != 0 || lineRightState != 0){
   if (lineLeftState == 1){
     motorDireita();
   }else if(lineRightState == 1 ){
     motorEsquerda();
   }
 }else{
   motorFrente();
 }
}
```

```
void motorFrente(){
 Serial.println("motorFrente");
 setMotors("1010");
}

void motorDireita(){
 Serial.println("motorDireita");
 setMotors("1000");
}

void motorEsquerda(){
 Serial.println("motorEsquerda");
 setMotors("0010");
}

void setMotors(String comdata){
 if(comdata.length() > 0)
 {
   for(int i = 0; i < comdata.length(); i++)
   {
     if(comdata[i]=='0'||comdata[i]=='1')
     {
       digitalWrite(MotorLUp, comdata[0] - '0');
       digitalWrite(MotorLDown, comdata[1] - '0');
       digitalWrite(MotorRUp, comdata[2] - '0');
       digitalWrite(MotorRDown, comdata[3] - '0');
         Serial.print(digitalRead(MotorLUp));
         Serial.print(digitalRead(MotorLDown));
         Serial.print(digitalRead(MotorRUp));
         Serial.println(digitalRead(MotorRDown));
     }
```

```
  }
  Serial.println(comdata);
  comdata = "";
 }
}
```