



Plataforma robótica controlada remotamente para ensino de programação para crianças

Michelle Andrade Valente da Silva

Projeto Final de Graduação

**Centro Técnico Científico
Departamento de Informática
Curso de Graduação em Engenharia da Computação**

Orientador: Prof. Luiz Fernando Bessa Seibel

Rio de Janeiro
Dezembro de 2016



Michelle Andrade Valente da Silva

**Plataforma robótica controlada remotamente
para ensino de programação para crianças**

Relatório de Projeto Final, apresentado ao programa Engenharia de Computação da PUC-Rio como requisito parcial para a obtenção do título de Engenheiro de Computação.

Prof. Luiz Fernando Bessa Seibel
Orientador
Departamento de Informática — PUC-Rio

Rio de Janeiro, 1 de Dezembro de 2016

Resumo

Andrade Valente da Silva, Michelle; Bessa Seibel, Luiz Fernando. **Plataforma robótica controlada remotamente para ensino de programação para crianças.** Rio de Janeiro, 2016. 71p. Relatório de Projeto Final — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Nesse projeto desenvolvemos um aplicativo mobile conectado remotamente a uma plataforma automatizada com o objetivo de ensinar lógica de programação para crianças e jovens. A aplicação possibilita que o usuário crie sequências de comandos que serão executadas por um robô. O aplicativo foi desenvolvido para iOS e conectado via Bluetooth com a plataforma Arduino.

Palavras-chave

Aplicativo Mobile. Arduino. Educação Infantil. Robótica.

Abstract

Andrade Valente da Silva, Michelle; Bessa Seibel, Luiz Fernando.
Remote controlled robotic platform to teach programming for kids. Rio de Janeiro, 2016. 71p. Final Project — Department of Informática, Pontifícia Universidade Católica do Rio de Janeiro.

On this project we developed a mobile application that connects remotely to an automated platform with the goal to teach programming logic to kids. The application allows the user to create sequences of code that it will be executed by a robot. The app was developed for iOS and connected via Bluetooth with the platform Arduino.

Keywords

Arduino. Child Education. Mobile Application. Robotics.

Sumário

1	Introdução	8
2	Situação atual	9
2.1	Soluções existentes	9
2.2	Descrições e Críticas	9
3	Objetivos do Projeto	16
4	Atividades	17
4.1	Estudos premlinares	17
4.2	Testes	18
4.3	Método	21
5	Projeto e especificação	22
5.1	Robô	23
5.2	Protocolo de Comunicação	25
5.3	Aplicativo	26
6	Implementação	33
6.1	Testes funcionais	33
6.2	Testes com usuários	34
7	Considerações finais	36
	Appendices	39
A	Código Arduino	40
B	Código Aplicativo iOS	48

Lista de figuras

1	Scratch	10
2	Tynker	11
3	Move The Turtle	12
4	Lego Mindstorms EV3	13
5	Dash	14
6	Cubetto	15
7	Shield Bluetooth Low Energy	17
8	Esquerda: Aplicativo BlueSerial. Direita: Aplicativo Simple Chat disponibilizado pela Red Bear Lab.	19
9	Protótipo do Aplicativo	20
10	Cronograma	21
11	Arquitetura do projeto	22
12	Robô	23
13	Esquemático Circuito	24
14	Esquerda: Primeira tela do aplicativo (conexão com o robô). Direita: Menu principal.	26
15	Tela Drive.	27
16	Tela Learn.	28
17	Tela Play.	29
18	Esquerda: tela de adição de comando. Direita: tela “Help”.	30
19	Exemplo de trilha de comando preenchida.	31

Lista de tabelas

1	Entradas digitais Arduino	24
---	---------------------------	----

1

Introdução

Para a maioria dos estudantes o primeiro contato com a programação acontece apenas na faculdade. No entanto, os que têm esse contato ainda quando crianças desenvolvem um conhecimento mais profundo da lógica de programação e pensamento computacional. O ensino da programação para crianças é muito mais do que apenas o entendimento da tecnologia, ele possibilita o desenvolvimento das habilidades de resolução de problemas e raciocínio lógico.

Esse trabalho é motivado por esse pensamento, criando uma ferramenta de simples utilização para que crianças a partir de 6 anos possam aprender conceitos de lógica e programação.

Com esse propósito foi desenvolvido um aplicativo iOS conectado via Bluetooth a um robô utilizando a plataforma Arduino. O aplicativo ensina de forma lúdica conceitos de programação e o pensamento lógico. Além disso, o usuário pode observar o resultado de suas criações sendo executado em tempo real no robô.

No mercado encontramos diversas soluções que possuem o mesmo objetivo. As que são mais acessíveis são em formato de aplicativos mobile ou sites que possuem diferentes jogos para o ensino do pensamento computacional. Também é possível encontrar opções que unem hardware e software, porém elas possuem um preço alto no Brasil.

Pelo alto custo e baixa divulgação no país as crianças e jovens desconhecem essas ferramentas e ao crescerem encontram dificuldades ao começar a aprender a programar. Além disso, essas aplicações quando utilizadas podem gerar o interesse dos usuários pela computação e pela engenharia, gerando futuros engenheiros.

Esse projeto possui três principais objetivos: tornar o ensinamento da programação mais acessível, ensinar o raciocínio lógico para crianças e desenvolver uma forma de aprendizado divertida e lúdica.

2

Situação atual

2.1

Soluções existentes

Foram separadas em duas categorias as soluções encontradas no mercado, plataformas que disponibilizam apenas o software e plataformas que unem software e hardware.

1. Software

- (a) Scratch (1)
- (b) Tynker (2)
- (c) Move the Turtle (3)

2. Software e Hardware

- (a) Lego Mindstorms (4)
- (b) Dash - Wonder Workshop (5)
- (c) Cubetto (6)

2.2

Descrições e Críticas

Todos os projetos estudados que serão apresentados nesta seção possuem o mesmo objetivo do projeto final, o ensino do pensamento computacional para crianças e jovens.

Diferentes aspectos, como usabilidade e design, foram analisados e serviram de inspiração no desenvolvimento do projeto. Cubetto foi o único projeto que não foi testado, porém sua menção é válida uma vez que é o único que utiliza Arduino para seu controle. Nos demais projetos a aluna pode ter contato e pode observar as ferramentas sendo utilizadas por crianças.



Figura 1: Scratch

2.2.1 Scratch

Scratch é uma linguagem de programação desenvolvida em 2007 pelo Media Lab do MIT [7]. Por ela funcionar de maneira simples e visual crianças a partir de 8 anos conseguem aprender a programar facilmente. Com essa linguagem é possível criar facilmente animações, jogos, histórias interativas e muito mais.

Além de ser uma linguagem ótima para o aprendizado de pensamento computacional, Scratch também possui uma comunidade online onde projetos são compartilhados. Por exemplo podemos encontrar um jogo criado por outra pessoa, testá-lo e ainda ver o código que está por trás dele.

Um dos pontos fortes dessa aplicação é a possibilidade de apresentar visualmente os resultados do projeto criado em tempo real. Por exemplo, podemos programar um simples jogo e testar seu funcionamento na própria plataforma online. Essa característica foi utilizada para o projeto final, uma vez que com o robô conseguimos também observar em tempo real o resultado do que foi programado.

A interface a partir de blocos de comandos predefinidos que se conectam entre si é uma forma muito interessante de mostrar o caminho da resolução do problema para o usuário. Porém o grande número de comandos faz com que o jogo seja complexo para crianças menores. Por isso para o projeto final foi selecionado um grupo pequeno de comandos que facilitam o entendimento.

Custo da ferramenta: Gratauta (8)



Figura 2: Tynker

2.2.2 Tynker

Tynker é uma plataforma online muito similar à mencionada anteriormente, Scratch. Sua principal diferença é a parte de tutoriais presentes no website. Possuindo a mesma característica de programação a partir de blocos para criar jogos e histórias interativas, Tynker faz com que crianças entendem a lógica de programação sem precisar focar na sintaxe, regras e símbolos presentes na programação.

O sistema de tutoriais se divide em dois grupos um para familiares e um para a escola. Os dois possuem diferentes pacotes que podem ser comprados na plataforma online. Como Tynker é um produto comercial, o ponto fraco é de que para acessar as melhores funcionalidade dessa aplicação é necessário pagar alguma mensalidade.

O diferencial mais interessante é de que ao contrário do Scratch que é feito em Adobe Flash, Tynker é desenvolvido em HTML5 e JavaScript. Isso faz com que não seja necessário instalar plugins no navegador e também com que seja possível ser utilizado em tablets e smartphones.

Custo da ferramenta: USD 0,00 até USD 2.000,00 (dependendo do módulo escolhido) (9)

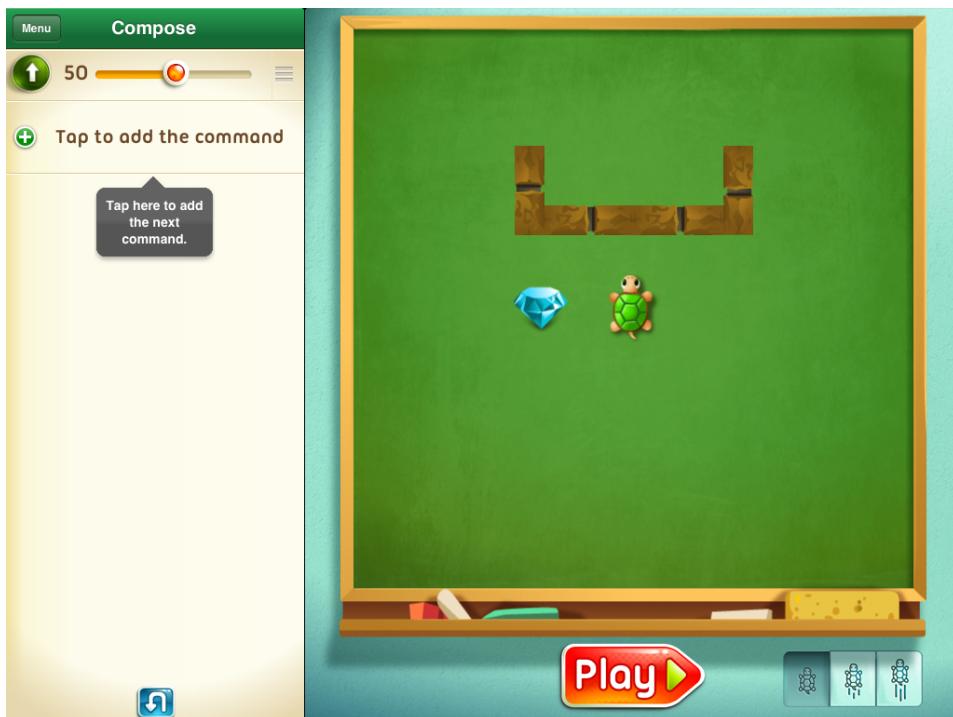


Figura 3: Move The Turtle

2.2.3

Move The Turtle

Dos exemplos citados anteriormente Move The Turtle é a primeira plataforma voltada completamente para smartphones e tablets. O aplicativo possui uma interface muito simples para adicionar sequências de comandos que serão executados pela tartaruga para que ela chegue até um diamante. Esse aplicativo foi uma das principais inspirações para o projeto, uma vez que ele é um aplicativo lúdico voltado para crianças a partir de 5 anos aprenderem a lógica de programação.

A parte de escolha de comandos foi uma ótima inspiração para o projeto uma vez que apresenta comandos simples, como andar para frente e para os lados, que são adicionados criando uma sequência que será executada pela tartaruga.

O ponto fraco do jogo são as diferentes fases, os cenários podem ser cansativos e não muito interessantes, fazendo com que o usuário perca o interesse pelo jogo.

A análise dessa plataforma foi extremamente interessante do ponto de vista da interface de usuário voltada para um aplicativo mobile. Pela falta de espaço físico ao apresentar as informações em um smartphone é necessário tomar cuidado com as escolhas do design do aplicativo.

Custo do aplicativo: USD 3.99 (10)



Figura 4: Lego Mindstorms EV3

2.2.4 Lego Mindstorms EV3

Desde dos anos 30 a Lego vem sendo um dos brinquedos preferidos de crianças no mundo inteiro. No fim dos anos 90 como resultado de uma parceria com o Media Lab do MIT (7) o produto Lego Mindstorms foi lançado. Seu objetivo é a criação de um robô que irá auxiliar o desenvolvimento intelectual de crianças e jovens.

O Lego Mindstorms é constituído de peças originais da linha Lego e da linha LEGO Technic, com diferentes sensores (toque, intensidade luminosa e temperatura) e controlado por um processador programável.

O Lego MindStorms EV3 é a terceira geração da linha de robôs lançada em 2013. Ele permite que o usuário utilize a criatividade para criar diferentes robôs com diferentes funcionalidades. Além de entender os componentes utilizados na construção o usuário aprende a programar utilizando um ambiente de desenvolvimento feito para jovens. Utilizando blocos de maneira semelhante ao Scratch, é possível criar diferentes programas para serem executados pelo robô. Dessa maneira conceitos de programação como loops, condições e até concorrência são ensinados de uma maneira divertida.

Custo: USD 349.99 (11)

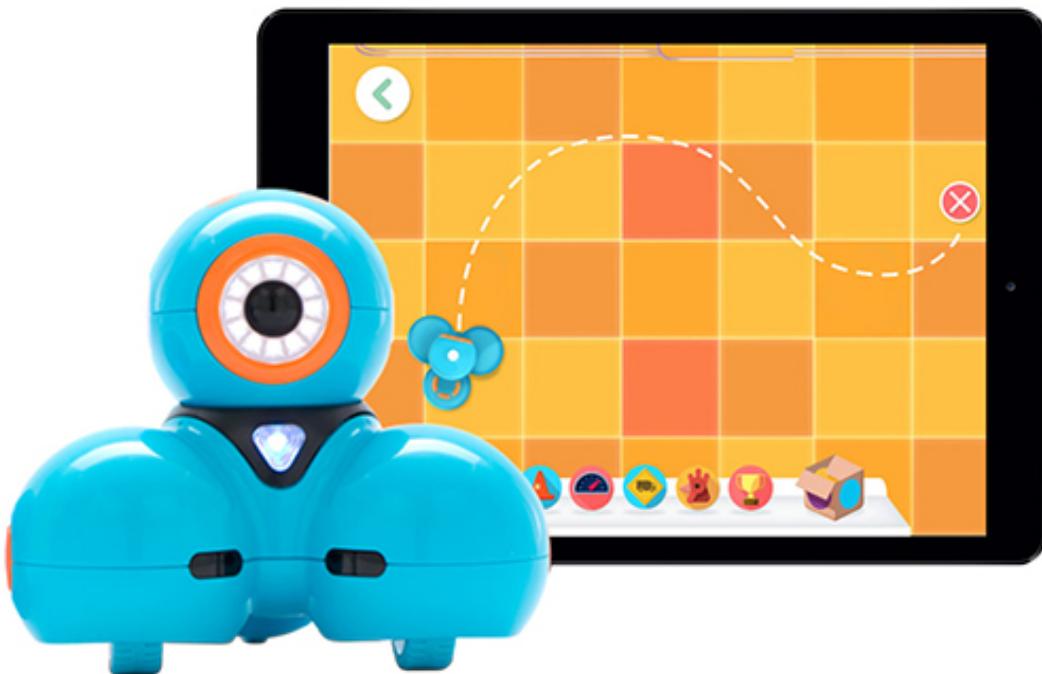


Figura 5: Dash

2.2.5

Dash - Wonder Workshop

Dash é um dos robôs da Wonder Workshop (12), uma empresa voltada para o desenvolvimento de plataformas de aprendizado de programação para crianças e jovens.

Ao contrário do Lego Mindstorms seu objetivo não é de montagem do robô, a ideia é usar seus motores, sensores, câmeras e alto falantes já embutidos para diferentes atividades encontradas nos aplicativos mobile. O robô se conecta através de uma interface bluetooth com os aplicativos que permitem controlar seus movimentos.

Existem 5 aplicativos com diferentes atividades, como criação de rotas e até produção de músicas que serão tocadas pelo Dash. Seus aplicativos são extremamente atrativos para crianças por suas cores e atividades divertidas.

Entre os exemplos apresentados esse é o que mais se assemelha ao projeto final desenvolvido. Uma vez que ele é focado no aplicativo para desenvolver o pensamento computacional e a possibilidade de observar os resultados facilmente no robô.

Custo: USD 149.99 (13)

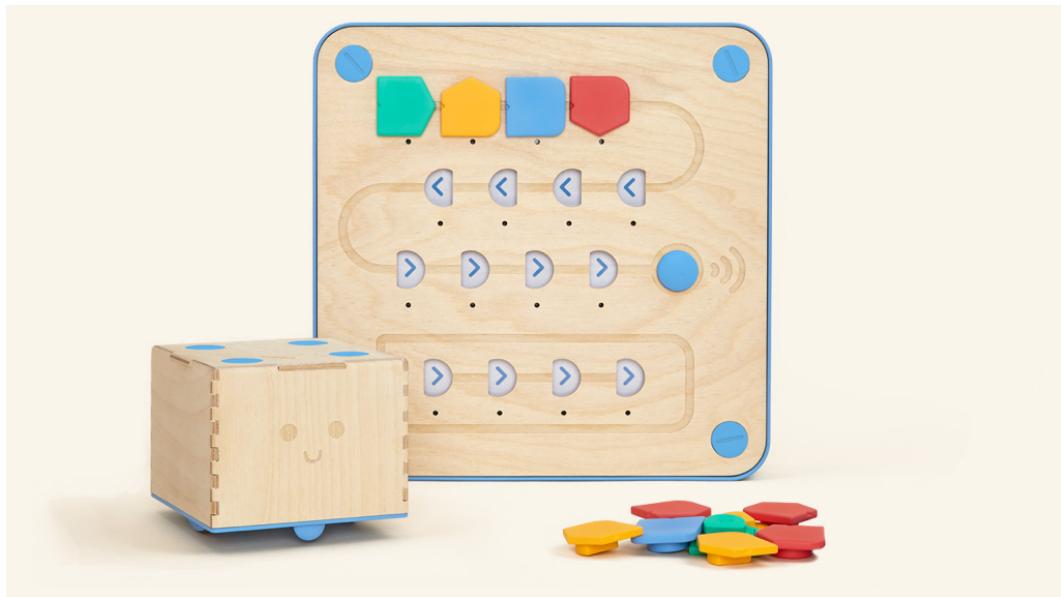


Figura 6: Cubetto

2.2.6 **Cubetto**

Cubetto é um robô de madeira acompanhado por uma tábua para a sua programação. Ele foi desenvolvido pela startup Primo Toys (14) e obteve financiamento participativo no website Kickstarter (15). Como no exemplo anterior, eles não tem como interesse para o usuário o desenvolvimento da parte de Hardware do robô. Eles escondem essa parte com um material amigável para crianças a partir de 3 anos.

Para a sua construção foram utilizados dois Arduinos um para o controle do robô e um para a tábua de programação. Os comandos do robô são adicionados a tábua criando uma fila de ações que o robô executará após apertar o botão “GO”. Cada comando (como mover para frente, para esquerda, etc) é representado por uma cor diferente nos blocos e eles são encaixados na tábua.

Esse projeto semelha ao projeto final apresentado pela utilização do microprocessador Arduino e a iteração a partir de simples comandos para o robô. Todo o projeto do Cubetto é Open-Source e disponibilizado online.

Custo: USD 199.99 (16)

3

Objetivos do Projeto

Como apresentado na seção 1.0, esse projeto possui três principais objetivos: tornar o ensinamento da programação mais acessível, ensinar o raciocínio lógico para crianças e desenvolver uma forma de aprendizado divertida e lúdica.

Com esse intuito, propõe-se a criação de um aplicativo mobile conectado remotamente a um robô. Com ele é possível ensinar a lógica de programação e o pensamento computacional para crianças a partir de 6 anos. Para o desenvolvimento do aplicativo foi escolhido o ambiente iOS, e para a plataforma automatizada a utilização de um Arduino Uno ligado a um chassi.

A aplicação irá permitir que o usuário crie sequências de comandos para serem executadas pelo robô. Primeiramente essa sequência será codificada para ser enviada via Bluetooth para o Arduino, que irá traduzir o código recebido e fazer com que os comandos sejam realizados. Dessa forma, a criança ao brincar com o aplicativo irá desenvolver o pensamento sistemático e entender conceitos como loops e interrupções.

Para que possam ser criados um número considerável de comandos e opções para o robô, serão adicionados os seguintes componentes: servomecanismos para controlar as rodas, LEDs que poderão ser acesos e apagados, sensores de detecção para realizar interrupções e um buzzer para servir de buzina.

Para que possa feita a comunicação entre o componente Bluetooth e o aplicativo é necessário a criação de um protocolo que permitirá que os dois se entendam. Logo, após as escolhas de comandos pelo usuário, o aplicativo irá traduzir os comandos para o formato do protocolo, ao receber o Arduino irá interpretar o formato e fazer com robô execute os comandos. Além disso, para que a comunicação funcione bem é preciso que o sistema seja reativo, ou seja, ele deve ser diretamente responsável tanto ao usuário enviando comandos pelo aplicativo, quanto as informações provenientes de seu sensor.

Finalmente, é fundamental tornar a utilização do aplicativo uma atividade lúdica para os usuários. Logo, é necessário ter uma interface atrativa para crianças, que melhore a experiência de aprendizado e que propicie diversão ao utilizar o projeto.

4 Atividades

4.1 Estudos preliminares

Os estudos preliminares relacionados ao projeto descrito neste documento são:

- Arduino
 - Já se possuía o conhecimento e experiência com a plataforma e a linguagem utilizada (semelhante a linguagem C). Porém foi necessário estudar a utilização de bibliotecas que ajudaram no desenvolvimento do projeto, como por exemplo a biblioteca do BLE Shield desenvolvido pela Red Bear Lab (17) para a conexão bluetooth e a biblioteca de Servos para o controle dos servomecanismos.
- Bluetooth
 - A comunicação entre o Android e o Arduino é feita através de uma interface serial Bluetooth onde se passam os comandos que serão interpretados pelo Arduino. Foi necessário estudar os diferentes tipos de Bluetooth e quais são aceitos pelas plataformas mobiles. Foi definido para o projeto o Bluetooth Low Energy (BLE) por ser compatível com o sistema iOS.

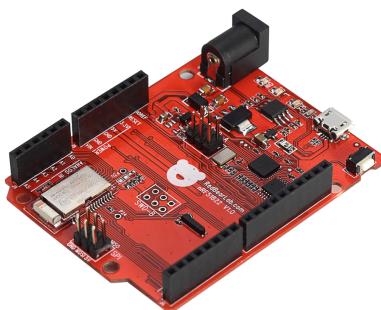


Figura 7: Shield Bluetooth Low Energy

- Programação Mobile

- Não se possuía nenhum conhecimento de programação de aplicativos mobile. Inicialmente o projeto começou a ser desenvolvido na plataforma Android e uma versão inicial do aplicativo foi finalizada nesta plataforma. Para a versão final se escolheu aprender Swift para poder desenvolver o projeto para iOS. Essa escolha se deu pelo interesse pela linguagem e também pela facilidade de se ter disponível um celular iOS.
- Essa experiência foi interessante pois foi obtido o conhecimento básico das duas formas de programação mobile, Java utilizando o ambiente Android Studio e Swift utilizando o ambiente Xcode.

- Componentes do Robô

- Foi feito um estudo de como funcionam os servomecanismos utilizados no robô. Para isso foi necessário entender os diferentes tipos de servos, e no final foram escolhidos dois servos de rotação contínua para as duas rodas.
- A primeira versão do robô possuía dois sensores infravermelhos para a detecção de obstáculos. Foi estudado o funcionamento desse tipo de sensor e a programação para obter a distância ao obstáculo.
- Para melhorar a precisão da detecção de obstáculo os sensores foram modificados para sensores ultrassônicos. Para isso novamente foi feito um estudo da tecnologia e da implementação no Arduino.

4.2 Testes

Uma vez que a plataforma foi montada, foram desenvolvidos programas para testar o comportamento do robô. Foram estes os testes:

4.2.1 Sensores de Distância

Primeiramente foram feitos os testes dos sensores infravermelhos, a fim de entender o seu funcionamento, a distância alcançada e a precisão. Para tal foi feito com que os LEDs servissem de indicação se o sensor estava indicando proximidade com algo. Dessa forma, caso alguém ou algum objeto se aproximasse do robô perto do sensor da esquerda, o LED da esquerda se acenderia, e caso se aproximasse pela direita o LED da direita se acenderia.

Após esses testes foi notado a instabilidade do sensor, sendo necessário trocar ele pelo sensor ultrassônico. Os mesmos testes foram feitos com os novos sensores e melhores resultados foram encontrados.

4.2.2

Movimentação do robô

Após confirmar o funcionamento do circuito foram feitos testes com o robô em movimento, fazendo com que ele reagisse caso o sensor detectasse algo. Por exemplo um dos códigos feitos foi o seguinte: o carro se movimentava em direção reta até que o sensor detectasse que algo estava na sua frente, se isso ocorresse o carro parava, buzinava (som vindo do Buzzer), verificava novamente se ainda existia algo na sua frente, caso existisse ele virava para um dos lados e seguia em frente, caso estivesse livre ele simplesmente seguia em frente.

4.2.3

Conexão sem fio

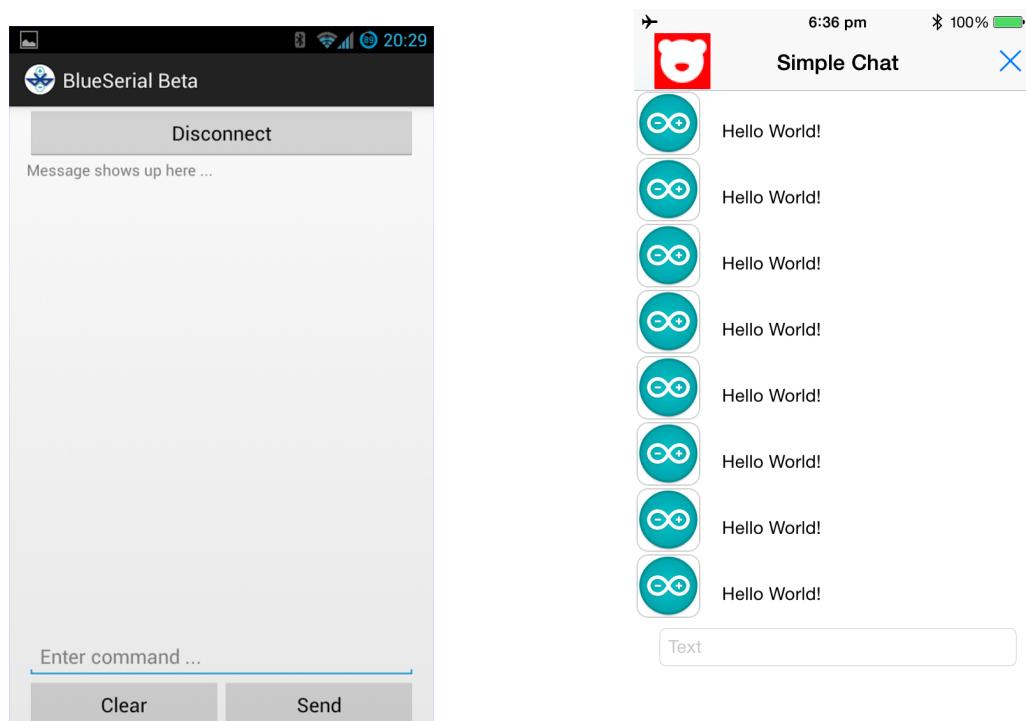


Figura 8: Esquerda: Aplicativo BlueSerial. Direita: Aplicativo Simple Chat disponibilizado pela Red Bear Lab.

A conexão sem fio foi feito por Bluetooth, primeiramente utilizando a biblioteca SoftwareSerial do próprio Arduino. Para isso foi adicionado um módulo Bluetooth ao circuito.

Para testar essa funcionalidade foi usado o aplicativo BlueSerial, que realiza o pedido de conexão para o módulo a partir de um celular. Neste aplicativo além de se conectar, é possível receber e enviar mensagens por comunicação serial. Com isso foi possível testar essa funcionalidade enviando e recebendo mensagens do Arduino. O aplicativo funciona de maneira simples, como um aplicativo de conversa, onde

o usuário digita a mensagem desejada e aperta enviar. Essa mensagem é recebida pelo Arduino, mostrada no monitor serial e enviada de volta para o celular.

Após a mudança da plataforma do aplicativo de Android para iOS foi descoberto que iOS não aceita todas as versões do Bluetooth. Para resolver esse problema a documentação disponibilizada pela Apple (18) foi estudada e foi descoberta a necessidade de uso de um módulo Bluetooth diferente. Em consequência disso ocorreu a mudança para o Bluetooth Low Energy Shield da Red Bear Lab (21) que é compatível com o sistema iOS.

Novos testes foram feitos com esse módulo, utilizando um aplicativo disponibilizado pela própria Red Bear Lab e foi observado que a conexão com iOS funcionava a contento. O aplicativo, chamado Simple Chat, funciona da mesma forma que o BlueSerial, possibilitando enviar mensagens para o Arduino e receber-las de volta.

4.2.4 Protótipo do aplicativo

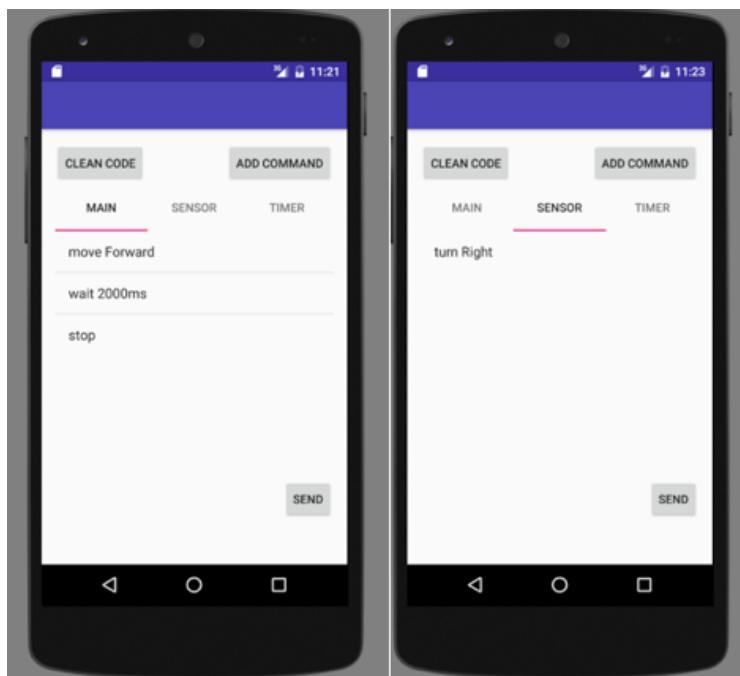


Figura 9: Protótipo do Aplicativo

Durante o primeiro semestre foi feito um protótipo do aplicativo para testar o envio de comandos do aplicativo para o robô. Nessa primeira versão não foi levado em consideração o design do aplicativo voltado para crianças. Foram apenas desenvolvidas as funções de adição de comandos e o envio para o robô.

Com o protótipo foi possível testar a parte de interpretação de comandos enviados para o Arduino e ver os resultados sendo executados em tempo real.

4.3 Método

As atividades desenvolvidas no projeto foram separadas em três tipos. Primeiramente as atividades que se dedicam a criação do robô, como o desenvolvimento do circuito, a montagem do carro e o código do Arduino. O segundo tipo são as atividades relacionadas ao aplicativo, primeiramente o desenvolvimento do protótipo em Android e depois a versão do aplicativo final em iOS. E por fim foram classificados como “outras” atividades como estudo, documentação e pesquisa com usuários.

Ao analisar o resultado com o que foi programado no primeiro semestre as principais diferenças ocorreram devido às mudanças realizadas no hardware. Não foi prevista a necessidade de alterar os sensores de distância e o módulo bluetooth. A segunda mudança principalmente acabou necessitando de um tempo maior de programação e adaptação do código anterior para utilizar o módulo.

A linha do tempo a seguir mostra como foi o andamento do projeto nos dois semestres:

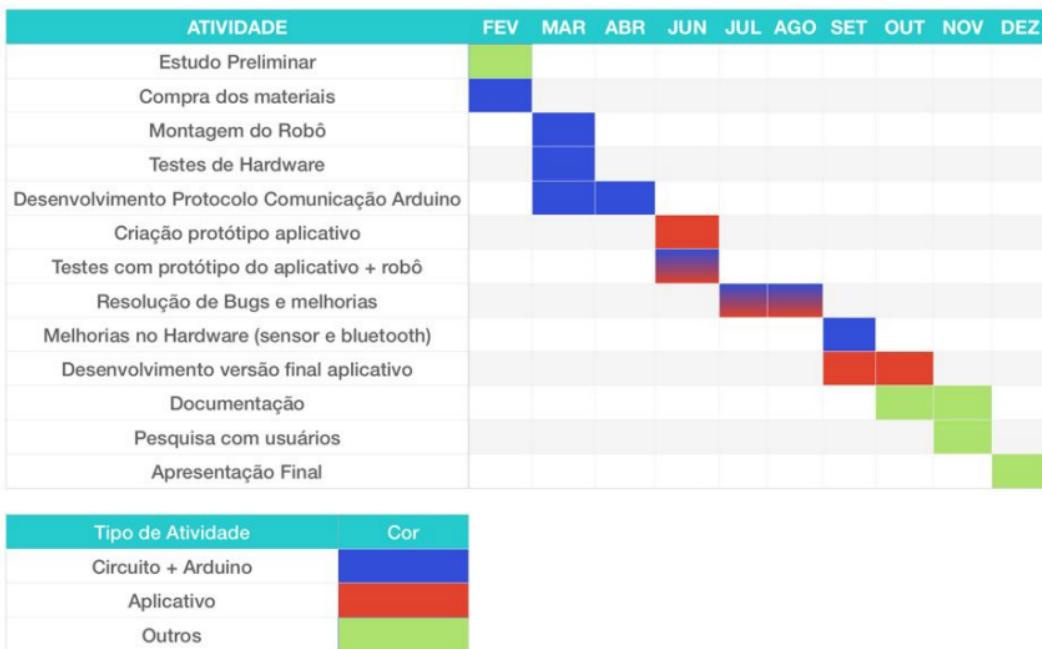


Figura 10: Cronograma

5

Projeto e especificação

O projeto tem como objetivo a criação de um robô que pode ser controlado por um aplicativo que possui uma interface simples e divertida para crianças e jovens.

O modelo do projeto está descrito no esquemático abaixo.

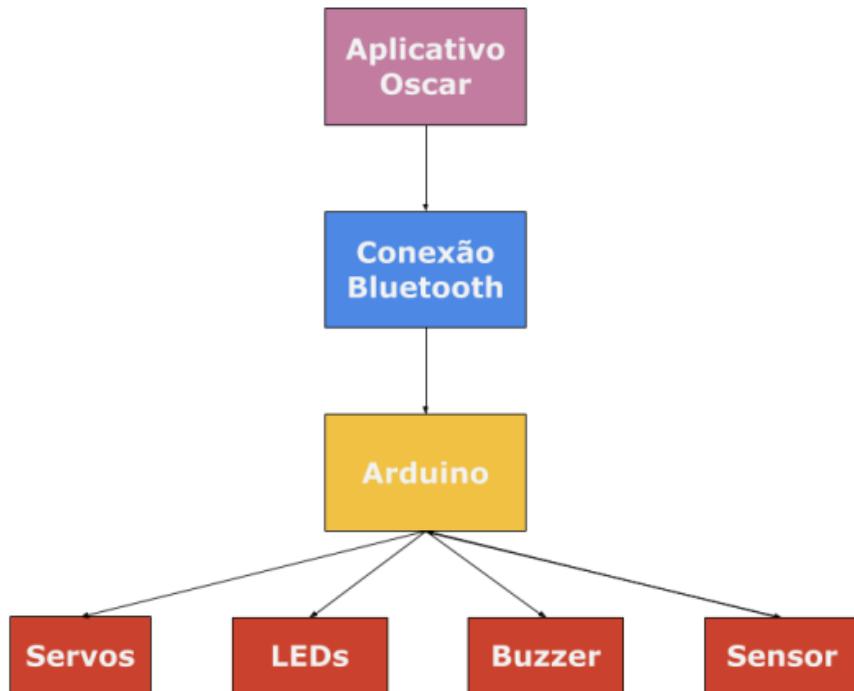


Figura 11: Arquitetura do projeto

Para um melhor entendimento do projeto ele vai ser apresentado separado em três partes: robô, protocolo de comunicação e aplicativo.

5.1 Robô

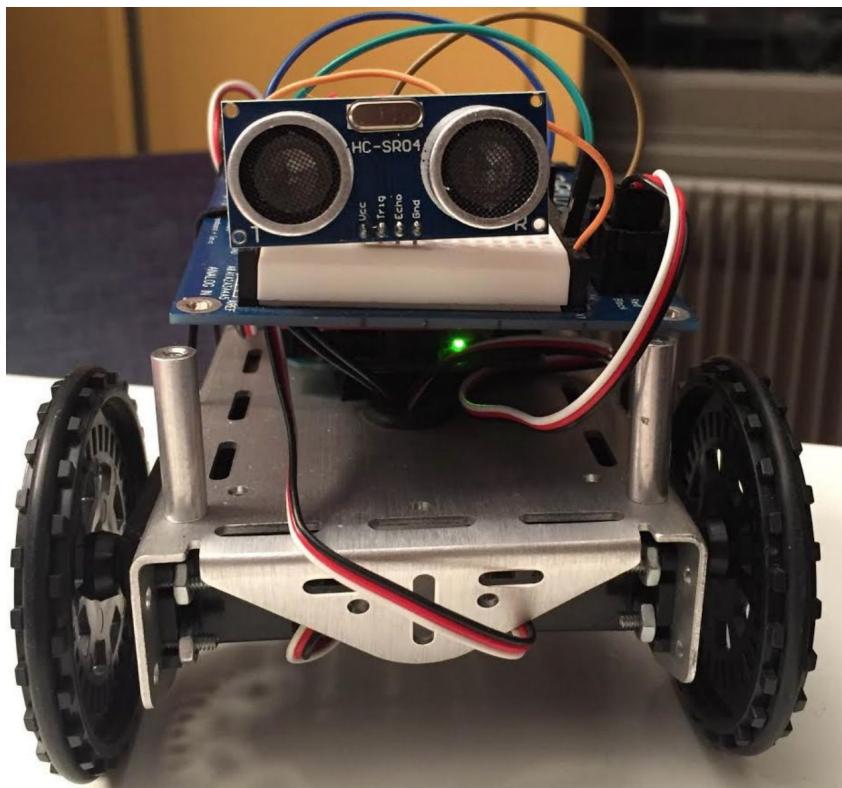


Figura 12: Robô

O robô foi montado com os seguintes componentes:

- 1 Arduino UNO (19)
- 1 Chassi Parallax (20)
- 2 LEDs
- 1 Buzzer
- 2 Resistores - $220\ \Omega$
- 1 BLE Shield Red Bear Lab (21)
- 1 Sensor Ultrassônico HC-SR04 (22)
- 5 Pilhas
- 2 Servos Parallax de rotação contínua (23)

A escolha do microcontrolador Arduino foi feita devido a sua grande popularidade e facilidade de uso. Devido ao seu sucesso é possível encontrar de forma online diversos tutoriais e referências que podem ser utilizados ao criar um novo projeto. Além disso, existem várias bibliotecas online que facilitam a programação

do microcontrolador. Por exemplo, para esse projeto foi utilizada a biblioteca disponibilizada pela Red Bear Lab que ajudou na comunicação Bluetooth entre shield e o aplicativo mobile.

O esquemático do circuito, mostrado na figura 11, foi feito na aplicação Circuitos Lab do Autodesk Circuits (25).

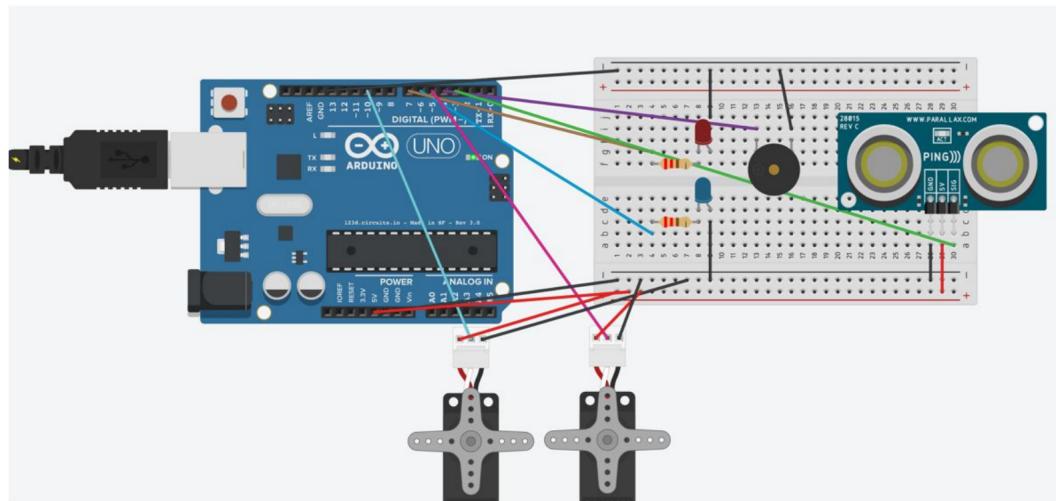


Figura 13: Esquemático Circuito

Tabela 1: Entradas digitais Arduino

PIN Digital	Componente
0	Rx
1	Tx
2	TRIG Sensor Ultrassônico
3	ECHO Sensor Ultrassônico
4	Buzzer
5	Servo Roda Direita
6	LED azul
7	LED vermelho
8	-
9	-
10	Servo Roda Esquerda
11	BLE Shield
12	BLE Shield
13	BLE Shield

5.2

Protocolo de Comunicação

Foi criado um protocolo de comunicação para que seja possível trocar informações entre o aplicativo e o robô. Essa comunicação é unidirecional, os comandos são enviados a partir do aplicativo e posteriormente recebidos, interpretados e executados pelo robô.

Os comandos são enviados como sequências de caracteres, essa sequência é quebrada e interpretada pelo código presente no Arduino. Cada comando é representado por uma sequência de caracteres mostrada a seguir:

- Led Vermelho: **ledr1** para ser aceso e **ledr0** para ser apagado.
- Led Azul: **ledb1** para ser aceso e **ledb0** para ser apagado
- Buzzer: **buzz** + microsegundos que ele irá tocar. Por exemplo: **buzz500**.
- Tempo de Espera: **wait** + tempo em microsegundos . Por exemplo: **wait500**.
- Movimento para frente e para trás: **move1** para frente e **move0** para trás.
- Vira para direita e para frente: **turn1** para a direita e **turn0** para a esquerda.
- Loop: **loop** coloca lista de comandos para ser repetida, irá para apenas se alguma reação ocorrer.

Além dos comandos foi necessário criar uma maneira de representar as reações. Existem dois tipos possíveis de reações: tempo de espera e sensor detectou alguma obstáculo a frente. Dessa maneira esse protocolo permite que sejam enviadas quantas trilhas de comandos o usuário quiser, por exemplo podem ser criada uma trilha de comandos que vai ocorrer após passar 2 segundos, uma trilha que irá ocorrer após 1 minuto e uma trilha que irá ocorrer caso o robô detecte um obstáculo.

As trilhas são representadas por um número sendo 0 para a trilha principal, 1 para reações de tempo e 2 para reação do sensor de distância. E elas são separadas por uma barra vertical. E o fim do envio de todos as trilhas é representado por uma barra comum.

Para ser compreendido como o protocolo funciona são apresentados a seguir 3 exemplos com suas respectivas traduções:

- *0loopmove1/2stopwait500turn1/*

Tradução: Robô move para frente até detectar algum obstáculo e parar, esperar 500 microsegundos e virar para a direita.

- *0loopmove1/1wait2000turn0/*

Tradução: Robô move para frente e quando passar 2 segundos ele vira para a esquerda.

– 0loopmove1/1wait2000turn0/1wait4000buzz1000ledr1/

Tradução: Robô move para frente, quando passar 2 segundos ele vira para a esquerda e quando passarem 4 segundos ele buzina por 1 segundo e acende o led vermelho.

5.3 Aplicativo

Para a criação do aplicativo foi utilizado o ambiente de desenvolvimento Xcode 7.3. Foi dado o nome de Oscar para o aplicativo e para o robô que é controlado por ele. A primeira tela do aplicativo consiste na conexão Bluetooth com o robô, ao clicar em “connect to Oscar” a conexão será estabelecida e caso obtenha sucesso o aplicativo irá para a tela do menu principal. Caso não tenha sucesso na conexão uma mensagem de erro é mostrada e o aplicativo continua na mesma tela, sendo possível tentar novamente a conexão. No menu principal encontramos as três partes no qual o aplicativo foi separado: Drive, Learn e Play.

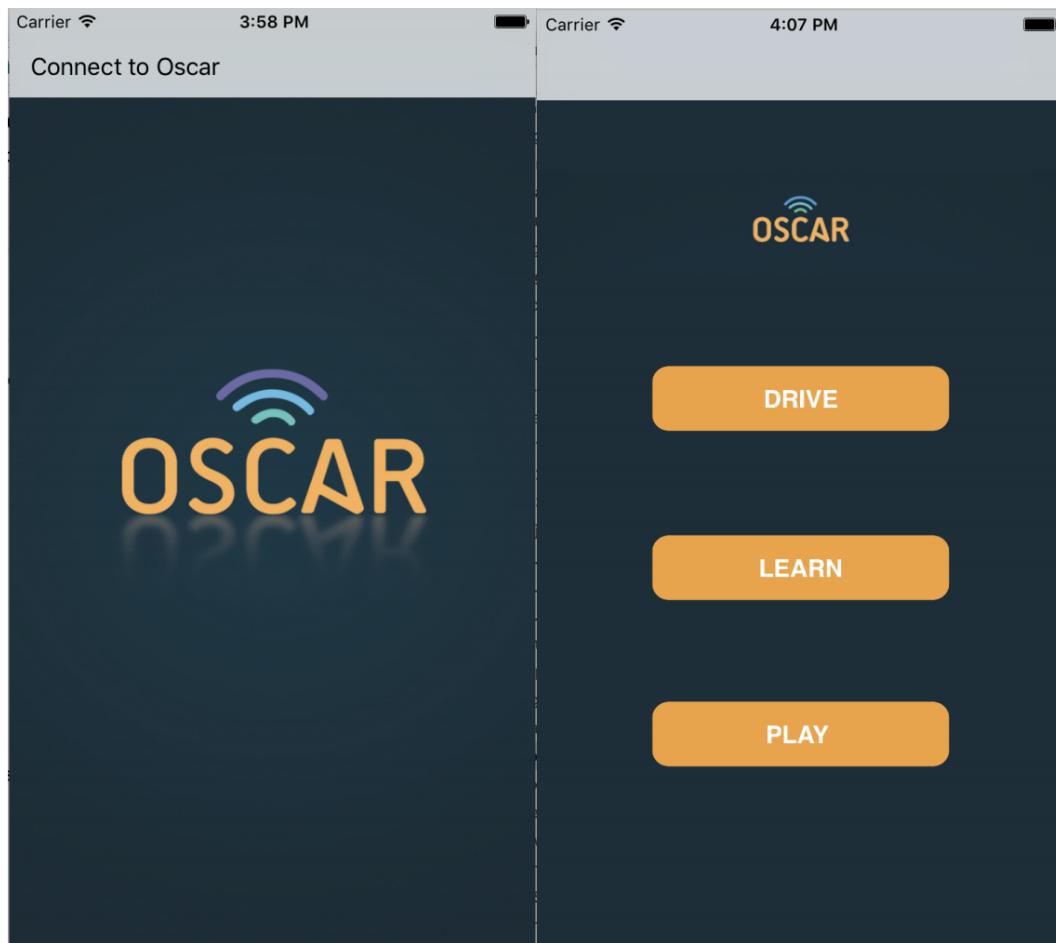


Figura 14: Esquerda: Primeira tela do aplicativo (conexão com o robô). Direita: Menu principal.

5.3.1 Drive

O modo Drive permite que o usuário controle o robô diretamente sem precisar criar uma lista de comandos. Essa opção faz com que a criança possa ter um contato inicial com o robô e ver seu funcionamento básico, para depois avançar para os modos mais complicados do aplicativo. Esse modo apresenta as seguintes opções de comandos:

- Pare: representado por uma placa escrita “stop”. O robô para qualquer movimento que esteja realizando.
- Buzina: representado pela imagem de um alto falante. O robô emite um som para representar a buzina do carro, esse som é realizado pelo buzzer.
- Direção de movimento: representado por setas ao redor do volante. É possível adicionar comandos de ir para frente, para trás, para a esquerda e para a direita.

Quando um botão for apertado pelo usuário, o aplicativo envia diretamente o comando escolhido para o robô e ele é executado.



Figura 15: Tela Drive.

5.3.2

Learn

A opção Learn apresenta diferentes exemplos de como o modo Play pode ser aplicado. O objetivo é apresentar para o usuário diferentes aplicações interessantes e divertidas que podem ser criadas no aplicativo, para isso o código é mostrado e é possível enviá-lo para o robô para ver o resultado. Nesse modo o usuário aprende o que é possível ser realizado pelo aplicativo e começa a ter uma ideia do que ele deseja desenvolver posteriormente.

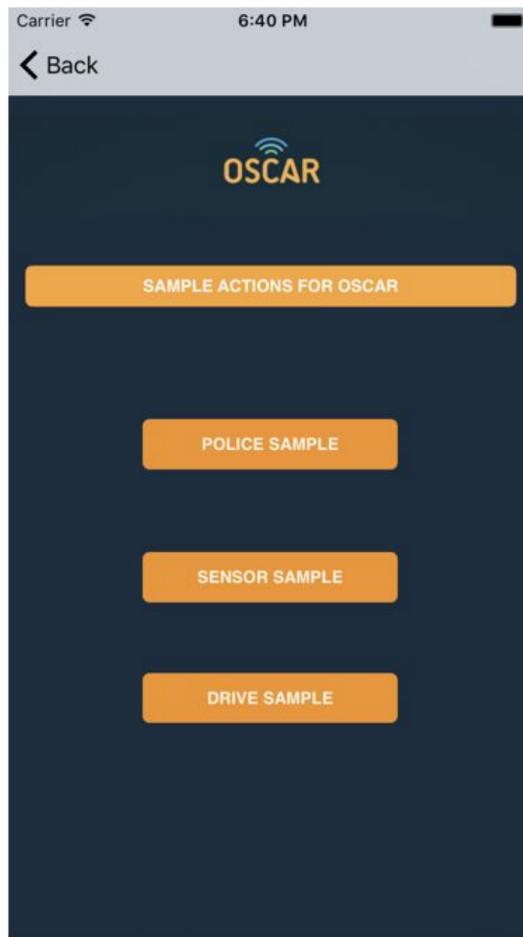


Figura 16: Tela Learn.

- *Police Sample*: o robô acende e apaga os leds azul e vermelho (similar ao funcionamento de uma sirene) e em toda mudança de led o buzz apita. Código em formato explicado na seção 5.2: *0ledr1ledb0buzz500wait1000ledr0ledb1buzz500wait1000/*
- *Sensor Sample*: esse exemplo apresenta o funcionamento do sensor de presença. O robô inicia com o LED vermelho aceso, caso detecte algo apaga o LED vermelho, acende o LED azul e buzina. Quando parar de detectar algo, ele volta para a trilha principal com o LED vermelho aceso e

o LED azul apagado. Código em formato explicado na seção 5.2: *0loop-pledr1ledb0/1ledr0ledb1buzz500wait1000/*

- *Sensor Sample*: o robô anda para frente até que o sensor detecte algo. Quando isso ocorrer, ele para de se mover, espera 1 segundo e vira para a direita. Caso o sensor não detecte algo novamente, ele volta a se mover para frente. Código em formato explicado na seção 5.2: *0loop-move1/1stopwait1000turn1/*

5.3.3

Play

O modo Play tem como objetivo a criação de sequência de comandos que serão executados pelo robô. O usuário clica em “new move for oscar” para adicionar um novo comando, e eles são adicionados sequencialmente em uma lista. Ao terminar de escolher os comandos, o usuário deve clicar em “send” para que eles sejam enviados para o robô e executados. Essa tela possui um botão stop ao lado do botão de envio, ele serve para que o usuário possa parar o robô caso ele esteja executando algum código antigo enquanto um novo está sendo criado.

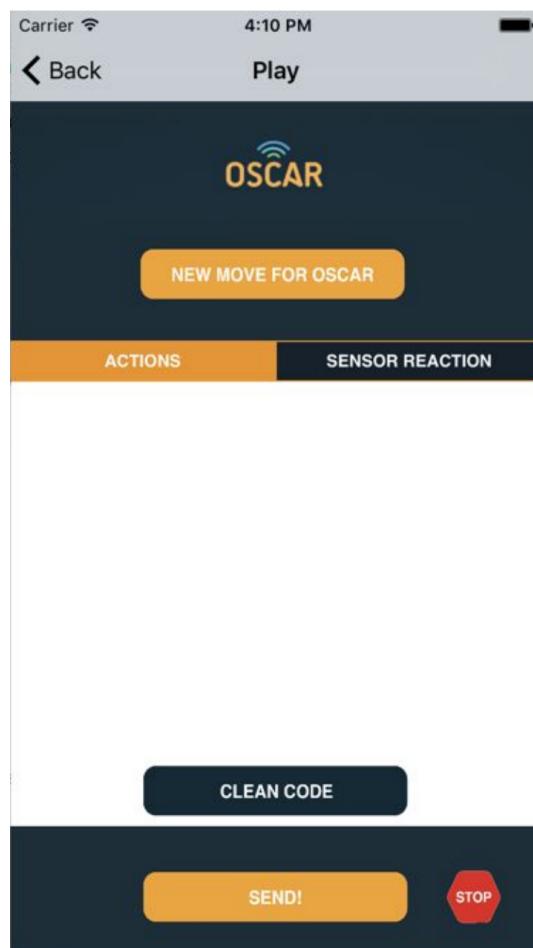


Figura 17: Tela Play.

Além da trilha principal de comandos, chamadas no aplicativo de “*ACTIONS*”, também é possível criar uma lista de comandos que serão executados caso o robô detecte algum obstáculo na sua frente. Para isso o usuário precisa mudar de aba, clicando em “*SENSOR REACTIONS*”, e escolher os comandos da mesma maneira que foi feita para a aba principal. Após terminar de criar as trilhas de comando, o usuário deve clicar em “send” para enviar as duas trilhas de comando para o robô.

A tela de escolha de comandos possui diferentes botões que possuem imagens representativas de seus comandos. As explicações da função de cada botão aparecem caso o usuário clique no botão “*HELP*”. Os possíveis comandos, além dos já apresentados na seção Drive, estão descritos a seguir:

- Tempo de espera: representado por um cronômetro. Esse comando permite adicionar um tempo de espera entre uma ação e outra. O tempo é informado pelo usuário após clicar no botão.
- *Loop*: representada por duas setas azuis. Após adicionar esse comando todos os comandos adicionados depois irão entrar em *loop* e apenas parar de serem executados caso ocorra uma reação do sensor ou uma nova trilha de comando seja enviada.



Figura 18: Esquerda: tela de adição de comando. Direita: tela “Help”.

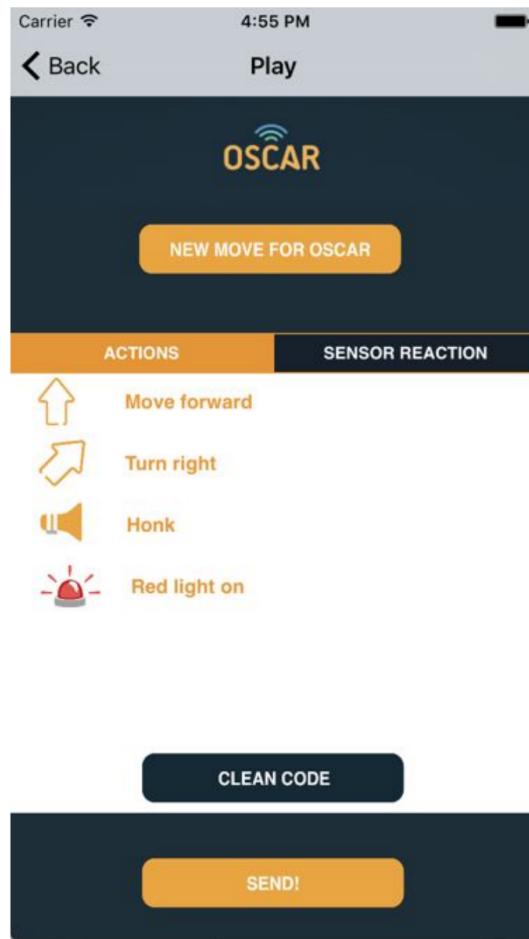


Figura 19: Exemplo de trilha de comando preenchida.

5.3.4 Aprendizado gerado pelo aplicativo

Os primeiros módulos, Drive e Learn, tem como objetivo introduzir o aplicativo para o usuário, sendo possível compreender seu funcionamento de uma forma divertida. Enquanto o modo Play proporciona o aprendizado da lógica computacional para crianças. As funções presentes neste módulo podem ser facilmente traduzidas para comandos e funcionalidades da linguagem de programação.

A seguir são mostrados alguns dos aprendizados relacionados a conceitos de linguagem de programação:

- Estrutura Sequencial: no decorrer da criação das trilhas de comandos o usuário aprende o conceito de algoritmo, sendo este uma lista de procedimentos que são executados sequencialmente com o objetivo de resolver um problema.

- Estrutura Condicional: a trilha de comandos para a reação ao sensor de presença ("sensor reaction") apresenta o conceito de condição ao usuário. Se a condição "sensor detectou algo" acontecer, então a trilha de reação será executada.
- Estrutura de Repetição: o comando *loop* presente na tela de escolha de comandos, permite que o usuário implemente estruturas que se repetem infinitamente, ou até que a reação do sensor ocorra. Por exemplo, caso o usuário coloque na trilha principal apenas um comando, como acender o LED, e crie uma sequência de comandos para a reação do sensor, ele irá perceber que provavelmente os comandos da reação não irão ser executados. Neste momento ele aprenderá a necessidade de adicionar um comando de repetição na trilha principal, pois dessa forma ela não irá finalizar antes que uma reação ocorra.
- *Wait*: o comando Tempo de Espera pode ser traduzido para o comando de linguagem de programação *wait*. Uma vez que ele suspende a execução de comandos do programa por um tempo determinado.

6

Implementação

6.1 Testes funcionais

Os testes funcionais foram feitos ao longo do desenvolvimento do projeto. Os principais testes relacionados ao robô estão descritos na subseção 4.2 do relatório. A partir dos testes é possível ver o andamento gradual do projeto:

- Teste dos servomecanismos: foi utilizada a biblioteca Servo (24) disponibilizada pelo Arduino para testar o controle das duas rodas do robô. Dessa maneira foi definida a função adicionada ao projeto final, onde é enviado se desejamos que o robô se movimente para frente, para trás, para esquerda ou para a direita. Para realizar o movimento de virar para um dos dois lados uma das rodas gira para a frente enquanto a outra gira para trás.
- Teste dos sensores: como mencionado anteriormente, no início do projeto foram usados sensores infravermelhos e após serem realizados testes eles foram trocados por sensores ultrassônicos. Para esses testes foram desenvolvidas funções no arduino de forma que, ao detectar um obstáculo um dos LEDs é aceso. Dessa maneira foi criada a função de detecção de obstáculos adicionada ao projeto final, modificando apenas que ao invés de acender o LED, a trilha de reação ao sensor é executada.
- Teste do shield Bluetooth: para testar o bluetooth utilizamos a biblioteca disponibilizada pelo Red Bear Lab. Para isso criado um simples chat, onde uma mensagem era enviada do celular para o Arduino. Foi possível verificar seu funcionamento conectando o arduino ao computador e utilizando o Serial Monitor disponibilizado pela aplicação de desenvolvimento Arduino. A função de recepção de mensagens foi posteriormente adicionada ao projeto final.
- Teste do aplicativo: com o protótipo do aplicativo mencionado na subseção 4.2 foi possível testar o recebimento dos comandos pelo robô e verificar a sua

execução. Na versão final do aplicativo diferentes exemplos foram utilizados para se verificar o seu funcionamento. Alguns desses exemplos acabaram se tornando os exemplos apresentados no modo Learn para a compreensão do funcionamento do aplicativo.

6.2

Testes com usuários

Os testes com o público alvo foram realizados após o desenvolvimento da versão iOS do aplicativo. Os testes foram feitos com 4 crianças, duas de 7 anos, uma de 8 anos e uma de 11 anos.

Nos primeiros testes foi dada liberdade ao usuário para conhecer o aplicativo, sem definir o que devia ser feito e foi observada a maneira de uso. Dessa forma foram facilmente notados os problemas de uso e mudanças foram feitas. Segue a seguir a explicação das principais mudanças:

- Ordem de execução do aplicativo: inicialmente o menu do aplicativo possuía a seguinte ordem: Play, Drive e Learn. Porém após os testes com usuários foi observado que a melhor maneira de iniciar era primeiro no modo Drive, onde a criança se familiariza com o robô e seus comandos. Em seguida, o modo Learn mostra para o usuário exemplos dos que será possível criar no modo seguinte. Por fim, o usuário chega no modo Play entendendo o que o aplicativo oferece e podendo a partir desse momento criar suas próprias linhas de comandos para o robô.
- Criação do botão stop na tela Play: esse botão foi adicionado após notar usuários querendo que o robô pare de executar seus comandos ou porque foi enviado um código errado para ele, ou simplesmente porque ainda estão criando um novo código e não desejam que ele fique executando o anterior.
- Adição do modo Help na escolha de comandos: inicialmente foi pensado que os botões mostravam de forma clara suas funções, mas ao apresentar o aplicativo para crianças na maioria das vezes elas perguntavam sobre a funcionalidade. Por isso foi adicionado o botão Help que direciona para uma tela com a explicação de cada comando.
- Teclado numérico para o comando Tempo de Espera: ao observar a utilização do aplicativo pelos usuários foi notada a dificuldade de digitar o tempo com o teclado normal, por isso foi alterado para o teclado numérico ao clicar no comando tempo de espera.

Após essas mudanças foram realizados novos testes, que mostraram claramente a melhoria na facilidade de uso. Durante o modo Drive as crianças aprenderam os principais comandos do robô. Elas se divertiram movendo ele pelo local, ligando e apagando os LEDs e usando a buzina. Após isso, elas seguiram para o modo Learn e se surpreenderam com o que o robô poderia realizar. Durante esse momento elas começaram a imaginar o que elas queriam que o robô fizesse, então ao chegar no modo Play elas não encontraram muitas dificuldades para a criação das trilhas de código.

Foi interessante notar o interesse pelo robô e pelo aplicativo durante a realização dos testes. Os usuários fizeram perguntas como: “como o robô consegue detectar algo?” e “como o robô recebe os comandos se não está conectado ao celular?”. Isso mostra como o aplicativo cumpriu o objetivo de além de estimular o raciocínio lógico, estimular também a curiosidade em relação à tecnologia

7

Considerações finais

Nesse projeto foi desenvolvida uma ferramenta de aprendizado de pensamento computacional para crianças a partir de 6 anos. Nele foi possível realizar um estudo de diferentes áreas como desenvolvimento de aplicativos, programação Arduino, sistemas reativos e educação infantil.

Os principais desafios do projeto foram a falta de conhecimento de desenvolvimento mobile e a instabilidade existente no hardware. Em relação ao primeiro, foi necessário estudar diferentes tutoriais e aprender ao longo do projeto a linguagem necessária para o desenvolvimento. Quanto à instabilidade do hardware, foi preciso contorná-la no código implementado no Arduino ou até mesmo alterar o equipamento utilizado, como foi o caso dos sensores e o módulo bluetooth.

Os resultados obtidos através dos testes foram satisfatórios, mostrando que projeto funciona de maneira estável e que satisfaz os objetivos definidos no planejamento. O aplicativo cumpre o propósito de apresentar de forma divertida conceitos de programação para crianças, como também incentiva a busca por mais conhecimento em relação à computação e a robótica.

Para trabalhos futuros, melhorias podem ser feitas tanto no robô quanto no aplicativo. No robô pode-se adicionar novas funcionalidades, como diferentes sensores e alto-falante. A adição de sensores permitiria a apresentação de diferentes condições nos programas criados, aumentando a complexidade do código. Com os alto-falantes seria possível a criação musical, como existente no robô Dash (5). Em relação ao aplicativo, seria interessante implementar uma versão com desenvolvimento por blocos, onde seria possível programar juntando blocos de comandos, condições e loops.

Referências Bibliográficas

- [1] Scratch - <https://scratch.mit.edu/> 1a
- [2] Tynker - <https://www.tynker.com/> 1b
- [3] Move The Turtle - <http://movetheturtle.com/> 1c
- [4] Lego Mindstorm EV3 - <https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313> 2a
- [5] Dash - <https://www.makewonder.com/dash> 2b, 7
- [6] Cubetto - <https://www.primotoys.com/> 2c
- [7] MIT Media Lab - <https://www.media.mit.edu/> 2.2.4
- [8] Preço Scratch - <https://scratch.mit.edu/info/faq/> 2.2.1
- [9] Preço Tynker - <https://www.tynker.com/school/pricing> 2.2.2
- [10] Preço Move The Turtle - <https://itunes.apple.com/us/app/move-turtle.-programming-for/id509013878?mt=8> 2.2.3
- [11] Preço Lego Mindstorm EV3 - <https://shop.lego.com/en-US/LEGO-MINDSTORMS-EV3-31313> 2.2.4
- [12] Wonder Workshop - <https://www.makewonder.com/> 2.2.5
- [13] Preço Dash - <https://store.makewonder.com/robots> 2.2.5
- [14] Primo Toys - <https://www.primotoys.com/> 2.2.6
- [15] Kickstarter - <https://www.kickstarter.com/> 2.2.6
- [16] Preço Cubetto - <https://www.primotoys.com/shop/> 2.2.6
- [17] Red Bear Lab - <http://redbearlab.com/> 4.1
- [18] Documentação Bluetooth Apple - <https://developer.apple.com/bluetooth/> 4.2.3

- [19] Arduino UNO - <https://www.arduino.cc/en/Main/ArduinoBoardUno> 5.1
- [20] Chassi Parallax - <https://www.parallax.com/product/130-35000> 5.1
- [21] BLE Shield Red Bear Lab - <http://redbearlab.com/bleshield/> 4.2.3, 5.1
- [22] Sensor Ultrassônico HC-SR04 - <https://www.amazon.fr/dp/B01C609AH6> 5.1
- [23] Servo Parallax de rotação contínua- <https://www.parallax.com/product/900-00008> 5.1
- [24] Biblioteca Servo - <https://www.arduino.cc/en/Reference/Servo> 6.1
- [25] Circuits Lab - <https://circuits.io/> 5.1

Appendices

A Código Arduino

```
#include <SoftwareSerial.h>
#include <SPI.h>
#include <boards.h>
#include <RBL_nRF8001.h>
#include "Servo.h"

#define MAX_CODE      100
#define SERVO_LEFT    10
#define SERVO_RIGHT   5
#define LED_BLUE      6
#define LED_RED       7
#define BUZZER        4
#define BLOCK_THRESH  20
#define TRIG          2
#define ECHO          3

typedef enum ReactionType {
    GENERIC      = 0,
    SENSOR       = 1,
    TIMER        = 2,
} ReactionType;

typedef struct Reaction {
    ReactionType type;
    char code[MAX_CODE];
    int codeLen;
    int codePos;
    int loopPos;
    long wait;
    boolean isTurning;
} Reaction;
```

```
struct Reaction newReaction() {
    Reaction r;
    r.type      = GENERIC;
    r.codeLen   = 0;
    r.codePos   = 0;
    r.loopPos   = -1;
    r.wait      = 0;
    r.isTurning = false;
    return r;
}

boolean receivingCode = true;
boolean recvReact = false;
int blockCount = 0;
Reaction reactions[2];
int nReactions = 1;
int curReaction = 0;
Servo servoLeft;
Servo servoRight;

void setServo(int speedLeft, int speedRight)
{
    servoLeft.writeMicroseconds(1500 + speedLeft);
    servoRight.writeMicroseconds(1500 - speedRight);
}

void servo_setup()
{
    servoLeft.attach(SERVO_LEFT);
    servoRight.attach(SERVO_RIGHT);
}

void setWait(Reaction* r, unsigned long now,
            unsigned long interval) {
    r->wait = now + interval;
}

boolean runCode(Reaction* r) {
    int i;
    long now = millis();
```

```
if (r->wait > now)
    return true;
else {
    r->wait = 0;
    if (r->isTurning) {
        setServo(0,0);
        r->isTurning = false;
    }
}

if (r->codePos < r->codeLen) {
    char command[5];
    for (i = 0; i < 4; i++)
        command[i] = r->code[(r->codePos)++];
    command[i] = '\0';

    if (strcmp(command, "loop") == 0) { // loop
        r->loopPos = r->codePos;
    }
    else if (strcmp(command, "ledr") == 0) { // red led
        digitalWrite(LED_RED,
            r->code[(r->codePos)++] == '1' ? HIGH : LOW);
    }
    else if (strcmp(command, "ledb") == 0) { // blue led
        digitalWrite(LED_BLUE,
            r->code[(r->codePos)++] == '1' ? HIGH : LOW);
    }
    else if(strcmp(command,"buzz") == 0) { // buzzer
        int timer;
        sscanf(r->code + r->codePos, "%d", &timer);
        r->codePos += floor(log10(abs(timer))) + 1;
        tone(BUZZER, 1500, timer);
    }
    else if(strcmp(command,"wait") == 0) { // wait
        int timer;
        sscanf(r->code + r->codePos, "%d", &timer);
        r->codePos += floor(log10(abs(timer))) + 1;
        setWait(r, now, timer);
    }
}
```

```
    else if(strcmp(command,"move") == 0) { // move
        if(r->code[(r->codePos)++] == '1')
            setServo(-200, -200);
        else
            setServo(200, 200);
        setWait(r, now, 1500);
        r->isTurning = true;
    }

    else if(strcmp(command,"turn") == 0) { // turn
        if(r->code[(r->codePos)++] == '0') {
            setServo(-200, 200);
            setWait(r, now, 1500);
        }
        else {
            setServo(200, -200);
            setWait(r, now, 1500);
        }
        r->isTurning = true;
    }

    else if(strcmp(command,"stop") == 0) { // stop
        setServo(0,0);
        setWait(r, now, 20);
    }

    return true;
}

else if (r->loopPos >= 0) {
    r->codePos = r->loopPos;
    return true;
}

return false; // program ended
}

void setup() {
    pinMode(LED_RED, OUTPUT);
    pinMode(LED_BLUE, OUTPUT);
```

```
pinMode(TRIG, OUTPUT);
pinMode(ECHO, INPUT);
servo_setup();
ble_begin();
Serial.begin(9600);
resetState();
}

void resetState() {
    receivingCode = true;
    recvReact     = true;
    curReaction   = 0;
    nReactions    = 1;
    reactions[0]  = newReaction();
    blockCount = 0;
    for(int i = 0; i < MAX_CODE; i++)
        reactions[0].code[i] = '\0';
}

int checkReactions() {
    int i, reactionIdx = 0;

    for (i = 1; i < nReactions; i++) {
        // Check if car is blocked
        ReactionType type = reactions[i].type;
        if (type == SENSOR && blockCount >= BLOCK_THRESH){
            reactionIdx = i;
            blockCount = 0;
        }
        // Check time reaction
        else if (type = TIMER) {
            if (reactions[i].wait == -1 || reactions[i].wait != 0
                && millis() > reactions[i].wait)
                reactionIdx = i;
        }
    }

    // If a reaction was found, leave
    if (reactionIdx)
        break;
}
```

```
}

    return reactionIdx;
}

void loop() {
    char codeChar;

    // Receive code from app, if any available
    if (ble_available()) {
        if (!receivingCode)
            resetState();

        Reaction* recv = &(reactions[nReactions - 1]);
        codeChar = (char)ble_read();

        // End of the receiving code
        if (codeChar == '/') {
            receivingCode = false;
            // New reaction
        } else if (codeChar == '|') {
            nReactions++;
            reactions[nReactions - 1] = newReaction();
            for(int i = 0; i < MAX_CODE; i++)
                reactions[nReactions - 1].code[i] = '\0';
            recvReact = true;
        }
        else {
            // The first char should set the type of the reaction
            if (recvReact) {
                switch (codeChar) {
                    case '1': {
                        Serial.println("Received sensor");
                        recv->type = SENSOR;
                        break;
                    }
                    case '2': {
                        Serial.println("Received timer");
                        recv->type = TIMER;
                    }
                }
            }
        }
    }
}
```

```
    recv->wait = -1;
    break;
}
default:  recv->type = GENERIC;
}
recvReact = false;
}
// Part of a command
else
    recv->code[(recv->codeLen)++] = codeChar;
}

Serial.println(recv->code);
}

// Check for blocking
digitalWrite(TRIG, HIGH);
delayMicroseconds(1000);
digitalWrite(TRIG, LOW);
int duration = pulseIn(ECHO, HIGH);
int distance = (duration/2) / 29.1;
if(map(distance, 0, 45, 8, 0) >= 8)
    blockCount +=1;

boolean isAlive, timerFirstRun = false;

// All the code received
if (!receivingCode) {

    // Check if there is a reaction to execute
    if (curReaction == 0)
        curReaction = checkReactions();

    // If reaction is of the type Timer and is being run by the
    // first time, set the control flag
    if (reactions[curReaction].type == TIMER &&
reactions[curReaction].codePos == 0)
        timerFirstRun = true;
```

```
    isAlive = runCode(&(reactions[curReaction]));

}

if (timerFirstRun) {
    curReaction = 0;
}

// Code has finished, so clean code
else if (!isAlive) {
    if (curReaction) {
        reactions[curReaction].codePos = 0;
        reactions[curReaction].wait = 0;
        reactions[curReaction].isTurning = false;
        curReaction = 0; // return to main trail
    }
    else
        resetState();
}

ble_do_events();

}
```

B Código Aplicativo iOS

Arquivo: Command.swift

```
import UIKit

class Command: NSObject {

    var children = [Command]()
    var type = String()
    var param = Int()

    // Constructors
    init(type:String, param: Int) {
        self.type = type
        self.param = param
    }
    convenience init(type:String) {
        self.init(type: type, param: -1)
    }

    // New command
    func addCommand(c : Command) -> Bool {
        if (type == "loop" && !(c.type == "loop")) {
            children.append(c)
            return true
        }
        return false;
    }

    // Delete command
    func removeCommand(index:Int) -> Bool {
        if (index > children.count){
            return false;
        }
        let child = children.remove(at: index)
        if (child.type == "loop") {
            for c in child.children {
                removeCommand(index: 0)
            }
        }
        return true;
    }
}
```

```
        }

        children.removeAtIndex(index);
        return true;
    }

    // Format to send the command to robot
    func serialize() -> String{
        var str = self.type
        if(type == "loop")
        {
            for (_,c) in children.enumerate() {
                str += c.toString()
            }
        }
        else if (param != -1)
        {
            str += String(param)
        }

        return str
    }

    // Format to print the command in the list of commands
    func toString() -> String {
        var str = ""

        if(type == "move"){
            str += "      Move "
            if(param == 1)
            {
                str += "forward"
            }
            else if(param == 0)
            {
                str += "backward"
            }
        }
        else if(type == "turn")
        {
```

```
        str += "    Turn "
        if(param == 1)
        {
            str += "right"
        }
        else if(param == 0)
        {
            str += "left"
        }
    }

    else if(type.lowercaseString.rangeOfString("led") != nil)
    {
        if(type == "ledr")
        {
            str = "Red light"
        }
        else if(type == "ledb")
        {
            str = "Blue light"
        }
    }

    str += "  "

    if(param == 1)
    {
        str += "on"
    }
    else if(param == 0)
    {
        str += "off"
    }
}

else if(type == "wait")
{
    str += "    Wait "
    str += String(param/1000) + " s"
}

else if(type == "buzz")
{
```

```
        str += " Honk "
    }
    else if(type == "stop")
    {
        str += " Stop "
    }
    else if(type == "loop")
    {
        str += "Repeat "
    }

    return str
}

}
```

Arquivo: MenuViewController.swift

```
import Foundation
import UIKit

class MenuViewController: UIViewController {

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

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

Arquivo: PlayViewController.swift

```
import UIKit

class PlayViewController: UIViewController {

    @IBOutlet weak var CleanCodeButton: UIButton!
    @IBOutlet weak var SendButton: UIButton!
    @IBOutlet weak var TabButtons: UISegmentedControl!
    var commandList = Dictionary<String, Array<Command>>()
    let appDelegate = UIApplication.sharedApplication().delegate
        as! RBLAppDelegate

    var currentTab = "Main"
    var typeCommand = ""
    var typeParam = ""

    // Update table of commands
    func reloadTable()
    {
        let tv : UITableViewController = self.childViewControllers[0]
            as! UITableViewController
        tv.commandList = commandList
        tv.tableView.reloadData()
        tv.viewWillAppear(true)
    }

    // Clean all the table
    @IBAction func CleanCodeAction(sender: AnyObject) {
        commandList[currentTab]!.removeAll()
        reloadTable()
    }

    // Save new command to the table
    func saveCommand()
    {
        let type = appDelegate.command
        let param = appDelegate.param
        let newCommand = Command(type: type!, param: Int(param!)!)
        commandList[currentTab]?.append(newCommand)
    }
}
```

```
        reloadTable()
    }

    func saveCommand(action: UIAlertAction, _ alert:UIAlertController,
                     param:String) {
        let newCommand = Command(type: typeCommand, param: Int(param)!)
        commandList[currentTab]!.append(newCommand)
        reloadTable()
    }

}

// Come back from previous screen
override func viewDidAppear(animated: Bool) {
    if(appDelegate.reload == "1" )
    {
        saveCommand()
    }
}

override func viewDidLoad() {
    super.viewDidLoad()

    let attributes = [ NSForegroundColorAttributeName : UIColor.whiteColor(),
                      NSFontAttributeName : UIFont(name: "Helvetica-Bold",
                                                    size: 12.0)!];
    let attributesSelected = [ NSForegroundColorAttributeName :
                                UIColor.whiteColor(),
                               NSFontAttributeName :
                                UIFont(name: "Helvetica-Bold", size: 12.0)!];
    self.TabButtons.setTitleTextAttributes(attributes,
                                            forState: UIControlState.Normal)
    self.TabButtons.setTitleTextAttributes(attributesSelected,
                                            forState: UIControlState.Selected)

    self.TabButtons.layer.cornerRadius = 0.0;
    self.TabButtons.layer.borderColor = UIColor(red:0.916,
                                              green:0.600 ,
                                              blue:0.168 ,
                                              alpha:1.00).CGColor
    self.TabButtons.layer.borderWidth = 1;

    if(appDelegate.reload == "1")
```

```
{  
    saveCommand()  
    appDelegate.reload = "0"  
}  
  
else{  
    commandList["Main"] = []  
    commandList["Timer"] = []  
    commandList["Sensor"] = []  
}  
}  
  
override func didReceiveMemoryWarning() {  
    super.didReceiveMemoryWarning()  
}  
  
// Prepare to go to next screen  
override func prepareForSegue(segue: UIStoryboardSegue,  
                             sender: AnyObject?) {  
    if (segue.identifier == "TransitionToTableView") {  
        let childViewController = segue.destinationViewController  
            as! UITableViewController  
        childViewController.commandList = commandList  
    }  
}  
  
// Send Code to Robot  
@IBAction func SendCode(sender: AnyObject) {  
  
    let sendCode: RBLSendCode = RBLSendCode()  
    sendCode.text = "0";  
    for command in commandList["Main"]!{  
        sendCode.text = sendCode.text + command.serialize()  
    }  
    if(commandList["Sensor"]?.count > 0)  
    {  
        sendCode.text = sendCode.text + "|1";  
        for command in commandList["Sensor"]!{  
            sendCode.text = sendCode.text + command.serialize()  
        }  
    }  
}
```

```
        }

    }

    sendCode.text = sendCode.text + "/"

    // Break the code for the bluetooth
    if (sendCode.text.characters.count > 16){
        let array_code = Array( sendCode.text.characters)
        for index in 0.stride(to: array_code.count, by: 16) {
            sendCode.text  = String(array_code[index..
```

```
        let newCommand = Command(type: "stop");
        let sendCode: RBLSendCode = RBLSendCode()
        sendCode.text = "0" + newCommand.serialize() + "/";
        sendCode.sendCodeArduino()
    }

}
```

Arquivo: DriverController.swift

```
import UIKit

class DriverViewController: UIViewController {

    @IBOutlet weak var StopButton: UIButton!
    @IBOutlet weak var BuzzButton: UIButton!
    @IBOutlet weak var BlueLedButton: UIButton!
    @IBOutlet weak var RedLedButton: UIButton!
    @IBOutlet weak var ForwardButton: UIButton!
    @IBOutlet weak var LeftButton: UIButton!
    @IBOutlet weak var RightButton: UIButton!
    @IBOutlet weak var BackwardButton: UIButton!
    let defaults = UserDefaults.standardUserDefaults()

    // Send code to robot
    func SendCode(command: Command) {

        let sendCode: RBLSendCode = RBLSendCode()
        sendCode.text = "0" + command.serialize() + "/";
        sendCode.sendCodeArduino()
    }

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

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

    // Stop Button Pressed
}
```

```
©IBAction func StopButtonAction(sender: AnyObject) {
    let newCommand = Command(type: "stop");
    SendCode(newCommand);
}

// Move Forward Button Pressed
©IBAction func ForwardButtonAction(sender: AnyObject) {
    let newCommand = Command(type: "move", param: 1);
    SendCode(newCommand);
}

// Turn Left Button Pressed
©IBAction func LeftButtonAction(sender: AnyObject) {
    let newCommand = Command(type: "turn", param: 0);
    SendCode(newCommand);
}

// Move Backwards Button Pressed
©IBAction func BackwardButtonAction(sender: AnyObject) {
    let newCommand = Command(type: "move", param: 0);
    SendCode(newCommand);
}

// Turn Right Button Pressed
©IBAction func RightButtonAction(sender: AnyObject) {
    let newCommand = Command(type: "turn", param: 1);
    SendCode(newCommand);
}

// Blue LED Button Pressed
©IBAction func BlueLedButtonAction(sender: AnyObject) {
    let alert = UIAlertController(title: "Led command for Oscar",
                                message: "",
                                preferredStyle: UIAlertControllerStyle.Alert)

    alert.addAction(UIAlertAction(title: "ON", style: .Default,
                                 handler: { (action: UIAlertAction!) in
        let newCommand = Command(type: "ledb", param: 1);
        self.SendCode(newCommand);
    })
}
```

```
        }))

        alert.addAction(UIAlertAction(title: "OFF", style: .Default,
                                     handler: { (action: UIAlertAction!) in

            let newCommand = Command(type: "ledb", param: 0);
            self.SendCode(newCommand);

        }))

        presentViewController(alert, animated: true, completion: nil)
    }

// Red LED Button Pressed
@IBAction func RedLedButtonAction(sender: AnyObject) {

    let alert = UIAlertController(title: "Led command for Oscar",
                                 message: "",
                                 preferredStyle: UIAlertControllerStyle.Alert)

    alert.addAction(UIAlertAction(title: "ON", style: .Default,
                                handler: { (action: UIAlertAction!) in
        let newCommand = Command(type: "ledr", param: 1);
        self.SendCode(newCommand);

    }))

    alert.addAction(UIAlertAction(title: "OFF", style: .Default,
                                handler: { (action: UIAlertAction!) in

        let newCommand = Command(type: "ledr", param: 0);
        self.SendCode(newCommand);

    }))

    presentViewController(alert, animated: true, completion: nil)
}

// Time Wait Button Pressed
@IBAction func WaitButtonAction(sender: AnyObject) {
    let newCommand = Command(type: "wait", param: 500);
    SendCode(newCommand);
```

```
}

// Buzz Button Pressed
@IBAction func BuzzButtonAction(sender: AnyObject) {
    let newCommand = Command(type: "buzz", param: 500);
    SendCode(newCommand);

}

}

Arquivo: CommandViewController.swift
```

```
import UIKit

class CommandViewController: UIViewController {

    @IBOutlet weak var LoopButton: UIButton!
    @IBOutlet weak var StopButton: UIButton!
    @IBOutlet weak var BuzzButton: UIButton!
    @IBOutlet weak var BlueLedButton: UIButton!
    @IBOutlet weak var WaitButton: UIButton!
    @IBOutlet weak var RedLedButton: UIButton!
    @IBOutlet weak var ForwardButton: UIButton!
    @IBOutlet weak var LeftButton: UIButton!
    @IBOutlet weak var RightButton: UIButton!
    @IBOutlet weak var BackwardButton: UIButton!
    let appDelegate = UIApplication.sharedApplication().delegate
        as! RBLAppDelegate
    let defaults = NSUserDefaults.standardUserDefaults()

    // Return to previous screen
    @IBAction func ReturnMainMenu() {
        appDelegate.reload = "1"
        navigationController?.popViewControllerAnimated(true)
    }

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

    override func didReceiveMemoryWarning() {
```

```
super.didReceiveMemoryWarning()
}

// Stop Button Pressed
@IBAction func StopButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "stop"
    appDelegate.param = "-1"
    ReturnMainMenu()

}

// Move Forward Button Pressed
@IBAction func ForwardButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "move"
    appDelegate.param = "1"
    ReturnMainMenu()
}

// Turn Left Button Pressed
@IBAction func LeftButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "turn"
    appDelegate.param = "0"
    ReturnMainMenu()
}

@IBAction func BackwardButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "move"
    appDelegate.param = "0"
    ReturnMainMenu()
}

// Turn Right Button Pressed
@IBAction func RightButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "turn"
```

```
    appDelegate.param = "1"
    ReturnMainMenu()
}

// Loop Button Pressed
@IBAction func LoopButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "loop"
    appDelegate.param = "-1"
    ReturnMainMenu()
}

// Blue LED Button Pressed
@IBAction func BlueLedButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "ledb"
    let alert = UIAlertController(title: "Led command for Oscar",
                                 message: "Do you want to turn " +
                                         "on or turn off Oscar's blue light?",
                                 preferredStyle: UIAlertControllerStyle.Alert)

    alert.addAction(UIAlertAction(title: "Turn on", style: .Default,
                                handler: { (action: UIAlertAction!) in
        self.appDelegate.param = "1"
        self.ReturnMainMenu()
    }))

    alert.addAction(UIAlertAction(title: "Turn off", style: .Default,
                                handler: { (action: UIAlertAction!) in
        self.appDelegate.param = "0"
        self.ReturnMainMenu()
    }))

    presentViewController(alert, animated: true, completion: nil)
    ReturnMainMenu()
}

// Red LED Button Pressed
@IBAction func RedLedButtonAction(sender: AnyObject) {
```

```
 appDelegate.reload = "1"
 appDelegate.command = "ledr"

 let alert = UIAlertController(title: "Led command for Oscar",
                             message: "Do you want to turn " +
                             "on or turn off Oscar's red light?",
                             preferredStyle: UIAlertControllerStyle.Alert)

 alert.addAction(UIAlertAction(title: "Turn on", style: .Default,
                             handler: { (action: UIAlertAction!) in
                             self.appDelegate.param = "1"
                             self.ReturnMainMenu()
                         }))

 alert.addAction(UIAlertAction(title: "Turn off", style: .Default,
                             handler: { (action: UIAlertAction!) in
                             self.appDelegate.param = "0"
                             self.ReturnMainMenu()
                         }))

 presentViewController(alert, animated: true, completion: nil)

}

// Wait Button Pressed
@IBAction func WaitButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "wait"
    //1. Create the alert controller.
    let alert = UIAlertController(title: "Wait command for Oscar",
                                message: "Enter how many seconds" +
                                "you want oscar to wait",
                                preferredStyle: .Alert)

    alert.addTextFieldWithConfigurationHandler({ (textField) -> Void in
        textField.keyboardType = UIKeyboardType.NumberPad
    })

    alert.addAction(UIAlertAction(title: "OK", style: .Default,
```

```
        handler: { (action) -> Void in
            let textField = alert.textFields![0] as UITextField
            self.appDelegate.param = textField.text! + "000"
            self.ReturnMainMenu()
        })
    }

    self.presentViewController(alert, animated: true, completion: nil)

}

// Buzz Button Pressed
@IBAction func BuzzButtonAction(sender: AnyObject) {
    appDelegate.reload = "1"
    appDelegate.command = "buzz"
    let alert = UIAlertController(title: "Buzz command for Oscar",
                                 message: "Enter how many seconds" +
                                 "you want oscar to buzz",
                                 preferredStyle: .Alert)

    alert.addTextFieldWithConfigurationHandler({ (textField) -> Void in
        textField.text = "0"
    })

    alert.addAction(UIAlertAction(title: "OK", style: .Default,
                                handler: { (action) -> Void in
                                    let textField = alert.textFields![0] as UITextField
                                    self.appDelegate.param = textField.text! + "000"
                                    self.ReturnMainMenu()
                                }))
    appDelegate.param = "500"
    ReturnMainMenu()
}

}
```

SampleViewController.swift

```
import UIKit

class SampleViewController: UIViewController {

    @IBOutlet weak var CleanCodeButton: UIButton!
    @IBOutlet weak var SendButton: UIButton!
    @IBOutlet weak var TabButtons: UISegmentedControl!
    let defaults = NSUserDefaults.standardUserDefaults()
    var commandList = Dictionary<String, Array<Command>>()

    let appDelegate = UIApplication.sharedApplication().delegate
        as! RBLAppDelegate

    var currentTab = "Main"

    var typeCommand = ""
    var typeParam = ""

    func reloadTable()
    {
        let tv : UITableViewController =
            self.childViewControllers[0] as! UITableViewController
        tv.commandList = commandList
        tv.tableView.reloadData()
        tv.viewWillAppear(true)
    }

    @IBAction func CleanCodeAction(sender: AnyObject) {
        commandList[currentTab]!.removeAll()
        reloadTable()
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        let attributes = [ NSForegroundColorAttributeName :
            UIColor.whiteColor(),
            NSFontAttributeName :
```

```
        UIFont(name: "Helvetica-Bold",
             size: 12.0)!];
let attributesSelected = [ NSForegroundColorAttributeName :
                           UIColor.whiteColor(),
                           NSFontAttributeName :
                           UIFont(name: "Helvetica-Bold",
                                  size: 12.0)!];
self.TabButtons.setTitleTextAttributes(attributes,
                                       forState:
                                       UIControlState.Normal)
self.TabButtons.setTitleTextAttributes(attributesSelected,
                                       forState:
                                       UIControlState.Selected)

self.TabButtons.layer.cornerRadius = 0.0;
self.TabButtons.layer.borderColor = UIColor(red:0.916,
                                            green:0.600 ,
                                            blue:0.168 ,
                                            alpha:1.00).CGColor
self.TabButtons.layer.borderWidth = 1;

commandList["Main"] = []
commandList["Sensor"] = []

if(appDelegate.sample == 1)
{
    commandList["Main"]?.append(Command(type: "loop", param: -1))
    commandList["Main"]?.append(Command(type: "ledr", param: 1))
    commandList["Main"]?.append(Command(type: "ledb", param: 0))
    commandList["Main"]?.append(Command(type: "buzz", param: 500))
    commandList["Main"]?.append(Command(type: "wait", param: 1000))
    commandList["Main"]?.append(Command(type: "ledr", param: 0))
    commandList["Main"]?.append(Command(type: "ledb", param: 1))
    commandList["Main"]?.append(Command(type: "buzz", param: 500))
    commandList["Main"]?.append(Command(type: "wait", param: 1000))
    reloadTable()
}
if(appDelegate.sample == 2)
{
```

```
        commandList["Main"]?.append(Command(type: "loop", param: -1))
        commandList["Main"]?.append(Command(type: "ledr", param: 1))
        commandList["Main"]?.append(Command(type: "ledb", param: 0))
        commandList["Sensor"]?.append(Command(type: "ledr", param: 0))
        commandList["Sensor"]?.append(Command(type: "ledb", param: 1))
        commandList["Sensor"]?.append(Command(type: "buzz", param: 500))
        commandList["Sensor"]?.append(Command(type: "wait", param: 1000))
        reloadTable()
    }

    if(appDelegate.sample == 3)
    {
        commandList["Main"]?.append(Command(type: "loop", param: -1))
        commandList["Main"]?.append(Command(type: "move", param: 1))
        commandList["Sensor"]?.append(Command(type: "stop", param: -1))
        commandList["Sensor"]?.append(Command(type: "wait", param: 1000))
        commandList["Sensor"]?.append(Command(type: "turn", param: 1))

        reloadTable()
    }

}

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

override func prepareForSegue(segue: UIStoryboardSegue,
                             sender: AnyObject?) {
    if (segue.identifier == "TransitionToTableView") {
        let childViewController = segue.destinationViewController
            as! TableViewController
        childViewController.commandList = commandList
    }
}

func saveCommand(action: UIAlertAction, _ alert:UIAlertController,
                param:String) {

    let newCommand = Command(type: typeCommand,
```

```
        param: Int(param)!)

    commandList[currentTab]!.append(newCommand)

    reloadTable()
}

@IBAction func CleanCode(sender: AnyObject) {
    commandList[currentTab]!.removeAll()
    let tv : UITableViewController = self.childViewControllers[0]
        as! UITableViewController
    tv.tableView.reloadData()
    tv.viewWillAppear(true)
}

@IBAction func SendCode(sender: AnyObject) {

    let sendCode: RBLSendCode = RBLSendCode()
    sendCode.text = "0";
    for command in commandList["Main"]!{
        sendCode.text = sendCode.text + command.serialize()
    }
    if(commandList["Sensor"]?.count > 0)
    {
        sendCode.text = sendCode.text + "|1";
        for command in commandList["Sensor"]!{
            sendCode.text = sendCode.text + command.serialize()
        }
    }
    sendCode.text = sendCode.text + "/"

    if (sendCode.text.characters.count > 16){
        let array_code = Array( sendCode.text.characters)
        for index in 0.stride(to: array_code.count, by: 16) {
            sendCode.text = String(array_code[index..
```

```
        else{
            sendCode.sendCodeArduino()
        }

    }

@IBAction func ChangeTab(sender: AnyObject) {

    if(TabButtons.selectedIndex == 0){
        currentTab = "Main"
    }
    else{
        currentTab = "Sensor"
    }
    let tv : UITableViewController = self.childViewControllers[0]
        as! UITableViewController
    tv.commandList = commandList
    tv.currentTab = currentTab
    tv.tableView.reloadData()
    tv.viewWillAppear(true)
}

}
```

Arquivo: LearnViewController.swift

```
import Foundation
import UIKit

class LearnViewController: UIViewController {

    @IBOutlet weak var Sample1: UIButton!

    let appDelegate = UIApplication.sharedApplication().delegate
        as! RBLAppDelegate

    @IBAction func sample1Action(sender: AnyObject){
        appDelegate.sample = 1
    }
    @IBAction func sample2Action(sender: AnyObject) {
        appDelegate.sample = 2
    }
}
```

```
}

@IBAction func sample3Action(sender: AnyObject) {
    appDelegate.sample = 3
}

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

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

}
```

Arquivo: RBLSendCode.m

```
@synthesize text;

- (void) sendCodeArduino {
    RBLAppDelegate *appDelegate =
    (RBLAppDelegate *)
    [[UIApplication sharedApplication] delegate];
    BLE * bleShield = appDelegate->bleShield;

    if (bleShield.activePeripheral.state == CBPeripheralStateConnected) {
        NSLog(@"sendCodeArduino Arduino connected");

        NSString *s;
        NSData *d;

        if (text.length > 16)
            s = [text substringToIndex:16];
        else
            s = text;

        d = [s dataUsingEncoding:NSUTF8StringEncoding];
        [bleShield write:d];
    }
    NSLog(@"sendCodeArduino Ran");
}
```

```
@end
```

Arquivo: RBLAppDelegate.h

```
#import <UIKit/UIKit.h>
#import "BLE.h"

@interface TableViewController : UITableViewController

@end

@interface RBLAppDelegate : UIResponder <UIApplicationDelegate>
{
    NSString* reload;
    BLE *bleShield;
    NSString* command;
    NSString* param;
    int sample;

}

@property (retain) NSString* reload;
@property (retain) NSString* command;
@property (retain) NSString* param;
@property int sample;
@property (strong, nonatomic) UIWindow *window;
@end
```

Arquivo: RBLAppDelegate.m

```
#import "RBLAppDelegate.h"

@implementation RBLAppDelegate
@synthesize reload;
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    reload = @""0";
    param = @"";
    command = @"";
    sample = 0;
```

```
bleShield = [[BLE alloc] init];
return YES;
}
@end
```