MAC0499 - Trabalho de Formatura
Besourino: Robô Aberto para o Ensino de Programação e Robótica

Daniel Jorge Renjiffo
Hardware Livre USP

Supervisor: José Coelho de Pina

27 de fevereiro de 2018

Resumo

Neste trabalho descrevemos o passo a passo do desenvolvimento do robô Besourino e de seu ambiente de desenvolvimento integrado. O projeto do robô é inteiramente aberto, pode ser realizado com baixo custo e se baseia na placa `Arduino Nano`. A estrutura do Besourino foi produzida em impressoras 3D e criada no programa aberto de modelagem 3D `Blender 3D`. O ambiente de desenvolvimento foi criado com a biblioteca aberta `Kivy` e foi pensando para ser utilizado em dispositivos móveis ou mesmo microcomputadores. Todo o projeto está disponível no GitHub.

Sumário

Capítulo 1

Introdução

Em uma época rodeada cada vez mais por dados e informações, existe a tendência de um futuro próximo em que seja indispensável qualquer profissional de área diversa ter conhecimentos básicos de programação. Países como Suíça e Estados Unidos já adotam no currículo da escola fundamental aulas de programação usando ferramentas e kits voltados para este intuito. A partir de 2018, a rede paulista das escolas públicas irá inserir aulas de programação e ética na internet na grade do ensino fundamental e médio .

A placa aberta de desenvolvimento `Arduino` foi criada em meados de 2008 no *Interaction Design Institute*, Ivrea, Itália. Desde então, sensores e microcontroladores que antes, eram restritos apenas para empresas ou fábricas se tornaram de fácil acesso em lojas ou no comércio virtual. Essa facilidade em conjunto com o avanço tecnológico possibilitou o surgimento de kits e produtos voltados para o ensino de programação e robótica para crianças e adolescentes. Dentre esses conjunto de ferramentas destacam-se os produtos da linha `Lego Mindstorm` e a recém nova linha de brinquedos educativos `Lego Boost`. O Lego Mindstorm teve seu surgimento em 1998 e desde então teve três versões: RCX, NXT e EV3. Ele consiste de um bloco programável chamado *Brick* (figura 1.1) em que é possível conectar motores e sensores feitos especialmente para funcionar com ele. Em conjunto com o robô o Lego Mindstorm possui uma interface gráfica de programação com blocos lógicos. No caso do EV3 existe uma versão um pouco mais simples para tablets com *Bluetooth Low Energy* (BLE).

Fonte: https://www.fspartner.no

Figura 1.1: Bloco lógico EV3

Esse kit é bastante eficaz emincentivar o aprendizado de programação e robótica, além de ser usado também ambientes universitários para estudos de robótica móvel. Porém dado seu alto custo, cerca de 350 euros ou em torno de 2000 reais em lojas brasileiras ele se torna pouco acessível. Fora o custo, tanto o Brick como a interface gráfica são projetos fechados ou seja a comunidade de fazedores ou profissionais do ensino não podem modificá-los ou aperfeiçoá-los.

O novo produto educativo Lego Boost (figura 1.2) apresentado na Consumer Electronics Show de 2016 mostra, diferente do Lego Mindstorm, ser especificamente voltado para crianças e possui um custo mais acessível de cerca de 200 euros. Porém, da mesma maneira possui sua ferramenta de programação fechada e restrita apenas para tablets com BLE.

Fonte: http:www.lego.com

Figura 1.2: Lego boost

Diferente da linha da Lego, o Primo Toys e Makeblock usam a plataforma aberta Arduino. Makeblock (figura 1.3) possui sua própria interface de programação através do Bluetooth Classic usando blocos muito parecidos com os da ferramenta Scratch, o CodingPanda. Apesar de usarem hardware aberto, estes kits possuem alto preço. Cubeto da PrimoToys custa 225 dólares e o robô de entrada da Makeblock custa 150 dólares. Outro empecilho seria que esses kits possuem apenas uma parte do projeto aberto.

Fonte: http://store.makeblock.com/en

Figura 1.3: Kit da MakerBlock

Este trabalho de conclusão de curso apresenta uma ferramenta (figura 1.4) que pode ser facilmente estendida e modificada voltada para ensino de programação e robótica para crianças e adolescentes. Ela se divide em duas partes: o robô *Besourino* e a interface gráfica para lhe enviar comandos. Diferente dos kits mostrados anteriormente, tanto o software como o hardware estarão abertos. Os códigos fontes ficarão disponíveis para serem modificados assim como a lista de materiais e os esquemáticos necessários para a construção da placa. Também estará disponível os arquivos que descrevem as peças físicas do Besourino.

1.1 Anatomia do projeto

O Besourino é composto por atuadores como motores e buzzer e sensores de distância e infravermelho. A interface gráfica é usada para construir uma sequência de comandos

através de blocos lógicos disponíveis na caixa de ferramentas da interface. É possível enviar os comandos através de um rádio WiFi. Por sua vez, o Besourino recebe esses comandos e os executa.

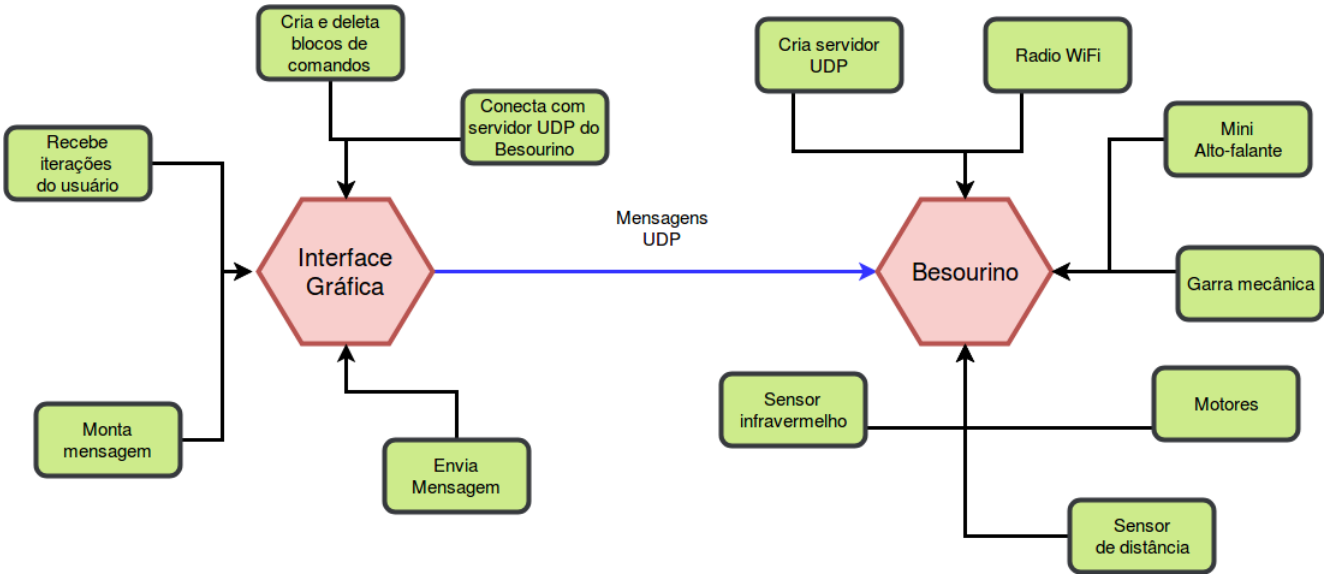


Figura 1.4: Diagrama de projeto

Os próximos capítulos irão apresentar de forma detalhada o desenvolvimento e o funcionamento dessas duas partes do projeto. O capítulo 2 irá apresentar o funcionamento da placa, componentes, e o solftware do Besourino. Já no capítulo 3 será apresentado os modelos do chassi. O capítulo 4 está reservado para explicar e mostrar o desenvolvimento da interface gráfica. Finalmente, terminamos este texto no capítulo 5 descrevendo o Besourino em algumas exposições.

Figura 1.5: Besourino

Capítulo 2

Robô

2.1 Introdução

Os componentes do robô (figura 2.1) foram escolhidos pensando na facilidade de encontrá-los, preço e facilidade para construí-lo. Outra decisão de projeto foi a escolha de sensores e atuadores comuns em competições de robótica como a Olimpíada Brasileira de Robótica (OBR) com a intenção de familiarizar os alunos a esse tipo de evento. Neste capítulo serão descritos os componentes eletrônicos assim como o programa que controla seu funcionamento.

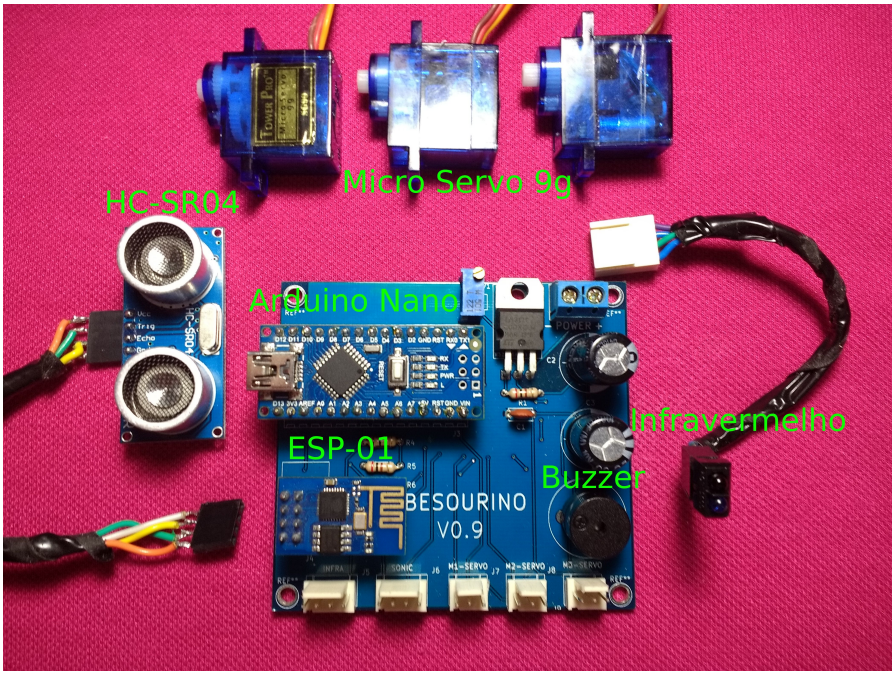


Figura 2.1: Componentes do Besourino

2.2 Hardware

Besourino é composto por quatro atuadores e dois sensores. Atuadores são componentes capazes de produzir movimento a partir de alguma forma de energia. Um exemplo seria o motor de corrente contínua (motor DC). Sensores por sua vez são aparelhos que

produzem sinais elétricos a partir da variação do ambiente onde são instalados. Um exemplo seria o sensor infravermelho usado no Besourino. Ele produz sinais elétricos quando é atingido por ondas infravermelhas. Entre os atuadores: três micro servos 9g e um buzzer. Já os sensores são ultrassônico e o sensor infravermelho. Para comunicação sem fio foi utilizado o rádio WiFi ESP-01. A placa Arduino Nano composta do microcontrolador Atmega 328P serve como central de processamento e controle do robô.

No primeiro protótipo do projeto foram utilizados dois motores DC 6 V (figura 2.2) com caixa de redução. Posteruimente para reduzir o custo do projeto, eles foram trocados por dois motores micro servo-9g. Essa decisão foi tomada pois, para usar os motores DC 6V com o Arduino Nano é necessário usar o circuito integrado (CI) ponte-H (L293D).

Fonte: https://alexnld.com

Figura 2.2: Motor DC 6V

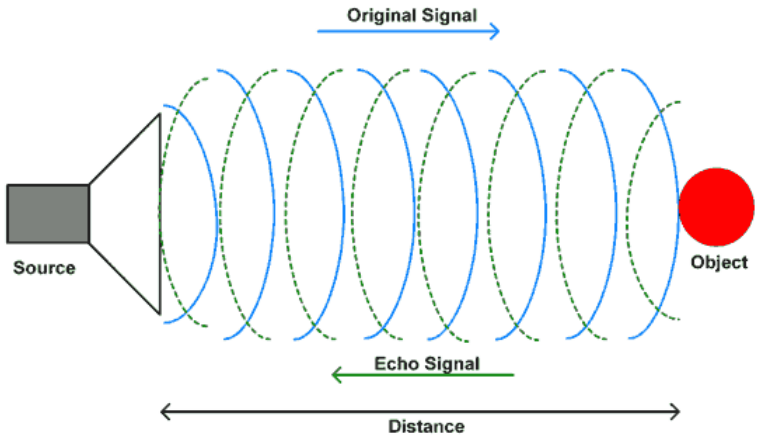
Este CI serve como interface para controlar os motores que consumem cerca de 250 ma (miliamperes), ultrapassando os 40 ma suportados pelas portas do microcontrolador. Por sua vez os motores micro servos 9g possuem uma interface de controle embutida e assim não precisam de um CI adicional para serem controlados. Porém, os micro servos 9g conseguem girar em torno do seu eixo entre 0 a 180 graus. Logo, os motores traseiros do robô foram modificados de maneira que seus eixos sejam capazes de fazer uma rotação completa. Para isso, uma trava mecânica foi cortada e o potenciômetro de seu circuito interno foi trocado por um resistor fixo. Este potenciômetro interno seria um resistor cujo valor varia entre 0 ohm e um valor máximo. Ele serve para calcular o ângulo atual do eixo. Por sua vez, trocando este potenciômetro por um resistor comum possibilita que o eixo do servo motor gire de forma contínua.

Neste projeto foi utilizado o HC-SR04 para medir a distância entre um objeto qualquer e a frente do robô. Seu funcionamento (figura 2.3) se dá através de um alto-falante e um microfone. Quando o pino trigger é ativado por sinal alto (5 V) por 10 ms, um sinal sonoro é emitido pelo alto-falante enquanto o microfone aguarda a onda de eco. Dessa maneira, a distância é calculada através de um contador dentro do microcontrolador que marca o tempo de saída e de chegada do sinal de eco. Usando a velocidade do som no ar e considerando a metade do tempo da chegada do sinal, é possível calcular a distância de um objeto. De acordo com datasheet (verifique a bibliografia), o HC-SR04 consegue medir distâncias entre 2cm a 4m com ângulo de efeito 15 graus.

Sensores infravermelho são utilizados para detecção de materiais reflexivos como papeis, fitas refletivas, cartões e etc. Neste trabalho ele foi utilizado para detecção de fitas adesivas para criação de robôs seguidores de linha. O sensor escolhido o TCRT5000 é composto por um emissor e um receptor de infravermelho. Um pino analógico do Arduino é reservado para efetuar suas medidas. Em nossos testes verificarmos uma forte influência da luz ambiente da leitura do mesmo. Para contornar este problema ele foi posicionado de baixo do robô.

O Buzzer é um alto-falante simples. Eles são utilizados para indicar emitir sons em brinquedos ou em robôs. Seu funcionamento consiste em desligar e ligar um eletroímã. Realizando esta sequência de liga e desliga na frequência audível é possível efetuar notas sonoras audíveis.

O módulo ESP-01 faz parte da família de módulos desenvolvido pela Ai-Thinker composta pelo System on Chip (SoC) ESP8266, desenvolvido por sua vez pela Espressif. Este SoC possui um microcontrolador de 32 bits , operando a 80 Mhz com antena WiFi



Fonte:http://www.electronicwings.com/sensors-modules/ultrasonic-module-hc-sr04

Figura 2.3: HC-SR04 funcionamento

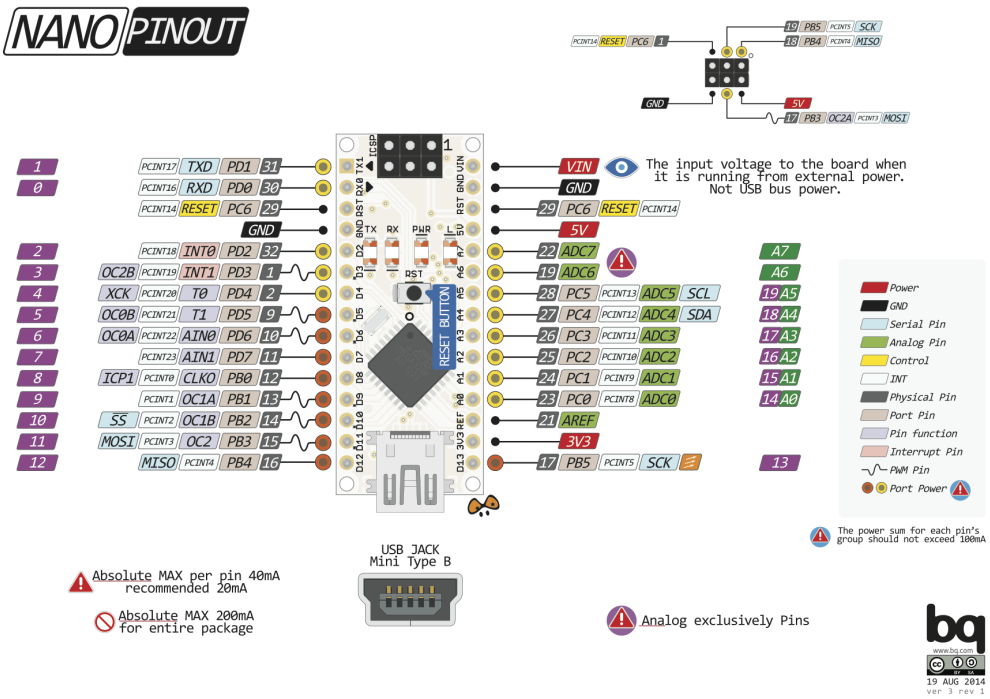
embutida na própria placa. Entre outras características é possível observar:

- Pilha de protocolo TCP/IP integrado, Wi-Fi 2.4 GHz com suporte para WPA/WPA2.
- Comunicação SPI, i2c, *Universal Asynchronous Receiver/Transmitter* (UART).
- Tamanho de 14.3 mm por 24.8 mm e 3 mm.
- Possibilidade de ser configurado e utilizado através de conjunto de comandos Hayes (AT) por um outro microcontrolador.

O Arduino Nano (figura 2.4) é composto pelo microcontrolador Atmega 328p. Nele existem 6 portas de leitura e escritas analógicas e 13 pinos para leitura e escrita de sinal digital. Dentre esses, 6 pinos possuem *Pulse Width Modulation* (PWM) podendo gerar sinais analógicos. Operando com clock de 16 MHz e com capacidade de armazenar códigos de até 32 Kbytes, este modelo foi escolhido dado seu baixo preço e seu pequeno tamanho de 3 cm de comprimento por 1.5 cm de largura.

Todos esses componentes anteriormente citados estão conectados na placa Besourino V0.9. Ela foi desenvolvida usando o programa aberto `Kicad`. Ela é composta por duas faces e seus respectivos arquivos para construção estão no repositório do projeto no GitHub. Conforme mostra o esquemático (figura 2.7), três pinos digitais foram reservados para controlar os motores micro servos. Sendo os pinos `D5` e `D6` reservados para motores da roda e o pino `D9` para controlar a garra. O pino `D3` está ligado para buzzer. O pino `A3` está ligado no sensor infravermelho. Os pinos `D12` e o `D11` estão conectados nos pinos `ECHO` e `TRIGGER` do `SR-HC04`.

Os pinos `TX` e `RX` por sua vez são usados para comunicar com módulo ESP-01. Para conectar o ESP-01 ao microcontrolador foi necessário tomar alguns cuidados devido a alimentação dos dois componentes. O ESP-01 possui uma alimentação 3.6 volts (V) já o Arduino Nano tem alimentação de 5 V, logo, o ESP-01 pode ser danificado pelo pino `TX` do Arduino Nano. Para contornar esse problema foi utilizado um circuito divisor de tensão (figura 2.5). Esse circuito é formado por dois resistores em séries em que o valor de tensão que passa por cada um é: $\texttt{T} \times (\texttt{R1}/(\texttt{R1} + \texttt{R2}))$. Onde o valor de `T` é de 5 V, e `R1` e `R2` os valores das resistências. Como é possível observar foram utilizados resistores de 33k e 22k Ohms. Logo, temos $5 \times 33/55 = 3$ V conectado no pino `RX` do ESP-01.



Fonte: www.arduino.cc

Figura 2.4: Arduino Nano pinagem.

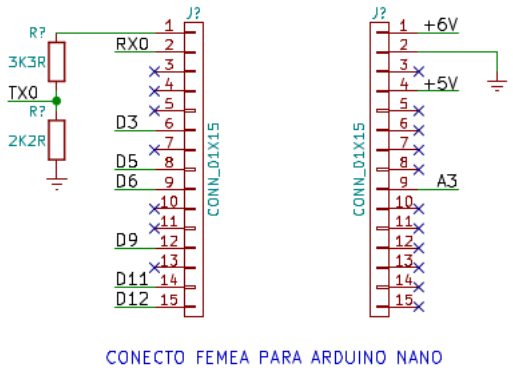


Figura 2.5: Arduino Nano com circuito divisor de tensão.

Outra medida importante relacionada ao ESP-01 é o consumo de corrente. Como mencionado anteriormente, o módulo WiFi precisa por volta de 3.6 V para funcionar. Na placa Arduino Nano temos o pino `VCC` que fornece 5 V para ligar sensores, módulos de comunicação e etc. Então bastaria utilizar a mesma técnica mencionada para reduzir a tensão de 5 V para 3.6 V. Porém devido ao consumo de corrente do ESP-01, não é possível usar o circuito divisor de tensão. O Arduino Nano possui um redutor de tensão que aceita em sua entrada voltagens entre 5 V e 15 V e fornece em sua saída 5 V, a entrada desse limitador de tensão é o pino `Vin` e sua saída é o pino `VCC`. Porém, esse redutor consegue fornecer corrente de até 50 ma, já o ESP-01 pode consumir corrente de até 220 ma. Com

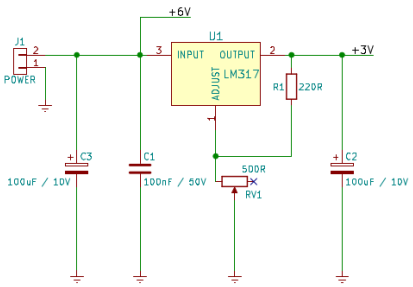


Figura 2.6: regulador de tensão

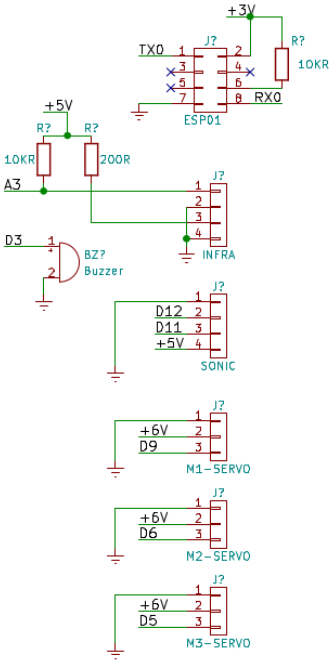


Figura 2.7: ligação dos componentes

isso, usar o pino `VCC` da placa Nano com apenas um divisor de tensão para alimentar o ESP-01 pode ocasionar quedas de energia. Para resolver esse problema foi utilizado o redutor de tensão ajustável `LM317` (figura 2.6) que pode fornecer corrente de até 3 A de saída. Para controlar a redução da tensão é preciso usar dois resistores. Como a redução pode variar de acordo com a alimentação de entrada, é preciso utilizar um trimpot de 5K ohm. O trimpot é uma resistência comum porém que pode ter seu valor ajustado através de um pequeno parafuso entre 0 e seu valor máximo.

2.3 Firmware

O código do Besourino foi desenvolvido em `C++` modificado para o ambiente arduino. Ele consiste de um módulo principal `Main.ino` com a rotina principal que é executada de acordo o diagrama de estado (figura 2.8). `Actuator.ino` é o módulo onde estão definidos os comandos para controlar os servos motores. O `WifiUtil.ino` é o módulo onde estão definidos rotinas para comunicação entre Arduino e o ESP-01, nele estão definidos funções para teste de comunicação, configuração e recebimentos de pacotes datagramas UDP. `Sensor.ino`, consiste em rotinas voltadas para ler a entrada dos sensores ultrassônico e infravermelho. Por último o módulo `Audio.ino`, voltado para os sons que o Besourino emite pelo buzzer.

Todo programa em arduino é dividido em dois blocos setup e loop. Quando o microcontrolador é inicializado o bloco setup é executado uma vez depois o bloco loop será repetidamente executado enquanto placa estiver com energia ou até o botão reset for pressionado.

Os pinos do microcontrolador são do tipo *General-Purpose Input/Output* (GPIO) ou seja é possível em cada pino ler ou gerar sinais elétricos. Porém, é preciso definir qual

ação o pino assumirá antes de usá-lo. Este processo é executado em Portas Init onde para todos os pinos, é definido se ele será de leitura ou saída.

Para comunicar com o módulo WiFi ESP-01 a placa usa os pinos digitais 0 e 1 pré-definidos como `tx` e `rx` da comunicação UART, respectivamente. Porém, é preciso inicializar estes pinos para estabelecer uma comunicação serial e definir uma velocidade de bits por segundo, neste projeto foi escolhido o padrão 9600 para garantir melhor estabilidade.

Depois das portas inicializadas e a comunicação serial pronta, é preciso configurar o módulo WiFi para atuar como um servidor. Antes é feito uma validação para verificar se módulo está funcionando ou até mesmo se ele está presente na placa. Para isso, é enviado o comando `"AT\r\n"` e se espera a resposta `"OK\r\n"`. Esse teste é feito 10 vezes, caso não haja reposta do módulo o Besourino emitirá um sinal sonoro e executará um reset. Depois da validação bem sucedida é enviado o comando `"AT+CIPMUX=1\r\n"`, que configura o módulo para receber múltiplas conexões se configurado como servidor. Depois o comando `"AT+CWMODE=2\r\n"` configura o ESP-01 como servidor. Seguido por `"AT+CWSAP="Besourino01","1234",5,0\r\n"` que configura nome da rede e senha, tipo de chave criptográfica. No caso Besourino é configurado para aceitar conexões sem pedido de senha porém ainda é necessário colocar algum valor no argumento, "1234". Por fim, o comando `"AT+CIPSERVER=1\r\n"` cria o servidor de fato que por padrão é configurado para receber datagramas na porta 333. Quando um pacote for enviado para o módulo o mesmo irá enviar seu conteúdo para o Arduino Nano que o armazenado em seu buffer interno.

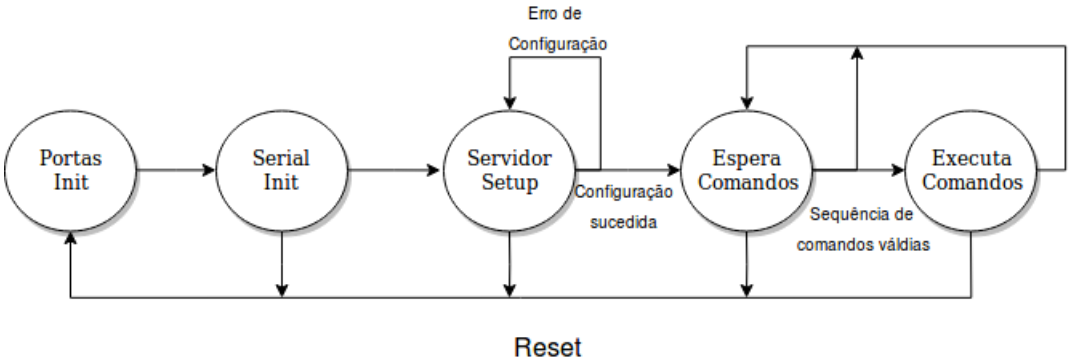


Figura 2.8: Diagrama de estado.

Com o servidor criado o código passa para o loop principal onde a cada 100 milissegundos verifica se existem dados armazenados no buffer interno. Caso ele não esteja vazio, seus dados serão lidos e interpretados, se os dados não seguirem o formato de um comando válido (figura 2.9) eles são excluídos do buffer.

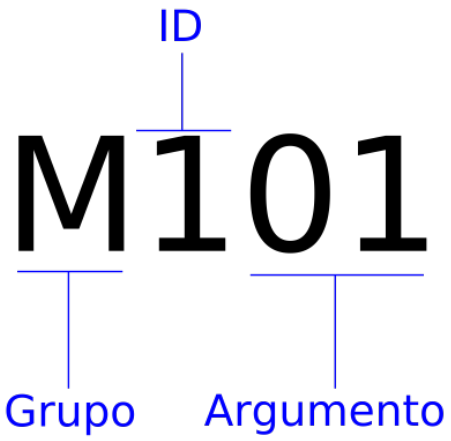


Figura 2.9: Formato de comando válido

Um comando válido é formado por até 4 bytes, sendo o primeiro byte um identificador do grupo do comando, atualmente podendo ser Movimento, Sensor, Áudio; um byte com o ID do comando e dois bytes de parâmetros se preciso.

Capítulo 3

Chassi

Toda estrutura do Besourino foi criada usando a ferramenta de modelagem 3D aberta Blender 3D. Ao longo do desenvolvimento do projeto foram três versões de estruturas.

No primeiro protótipo foi utilizado uma caixa de papelão (figura 3.1) com dois motores DC 6V e uma roda livre. Esta estrutura serviu apenas como protótipo inicial para testar a comunicação WiFi do ESP-01 com a interface gráfica.

Figura 3.1: Primeiro protótipo

No segundo protótipo (figura 3.2) foi desenvolvido uma estrutura com impressão 3D usando plástico *Acrylonitrile Butadiene Styrene* (ABS). Esse modelo serviu para testar a mudança do motor DC 6 V para o motor micro servo 9g e também foi inserido a garra.

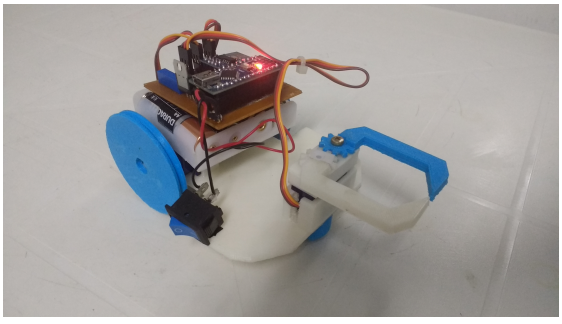


Figura 3.2: Segundo protótipo

No terceiro protótipo (figura 3.3) foi desenvolvido um novo modelo de estrutura onde tanto a placa e bateria encaixam através de um jogo de porcas e parafuso. Também foi feito uma tampa para proteger a placa em caso de queda. Em relação a versão anterior, neste modelo foram inseridos os sensores de distância e o sensor infravermelho.

Figura 3.3: Terceiro protótipo.

Capítulo 4

Interface Gráfica

4.1 Introdução

Dado o fato do projeto ser aberto para que possa ser expandido e modificado pela comunidade, visou-se o uso de ferramentas acessíveis e que possam ser utilizadas em diferentes ambientes. Sendo assim foi escolhida a linguagem de programação Python por ter um desenvolvimento rápido, possuir uma vasta quantidade de bibliotecas e arcabouço e funcionar em diversos ambientes.

No começo do projeto foi feito uma pesquisa e testes com alguns arcabouços para desenvolvimento de interface gráfica, entre eles o *wx*, *pyQT* e o *Kivy*. Pensando na portabilidade foi escolhido o *Kivy*.

4.2 Kivy

Kivy é um arcabouço para linguagem Python aberto com licença MIT, lançado em fevereiro de 2011. Sendo capaz de desenvolver programas para Android, IOS, Linux, Mac OS e Windows. Ele possui uma vasta biblioteca para desenvolvimento de aplicativos iterativos com muitas qualidades como:

- Suporte para entrada de mouse, teclado, toque múltiplo e Tangible User Interface (TUI).
- Biblioteca gráfica usando apenas o OpenGL ES 2, baseando também no Vertex Buffer Object e shaders.
- Vasta quantidade de Widget que aceita toque múltiplo.
- Linguagem intermediária Kv language para criação de widget customizado.

4.3 Escolha de Design

Dado o fato de o público alvo deste projeto ser crianças e adolescentes, foi necessário atender alguns detalhes de Design. A tela do programa é simples ou seja com poucos menus. Existe apenas duas divisões: Menu de blocos e a área de montagem dos blocos.

Como é possível observar na figura 4.1, não existe textos ou palavras indicando a função de cada bloco mas sim símbolos que representam sua função. Esta escolha permite

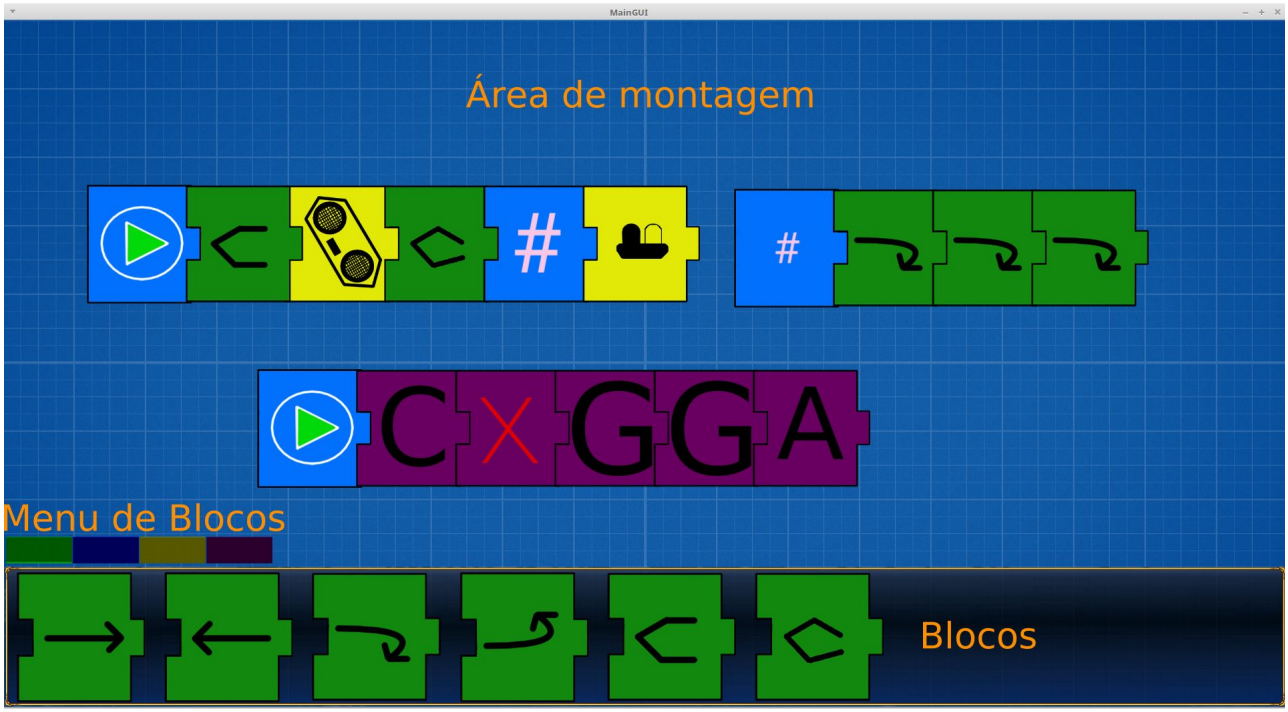


Figura 4.1: Tela do aplicativo.

que uma criança que ainda está no período de alfabetização possa usar a interface. Outro ponto importante é o formato dos blocos que levam a entender a possibilidade de encaixar um bloco um do lado do outro sempre da esquerda para direita. Como mencionado anteriormente no capítulo 1, os comandos são divididos em grupos, cada grupo representa um conjunto de comando específicos para uma parte do Besourino como movimento, sensores, som e etc. Para diferenciar esses grupos foi escolhido uma cor representando o grupo, na figura acima o grupo relacionado aos movimentos é representado pela cor verde.

4.4 Funcionamento

Para enviar comandos para o Besourino é preciso antes 'montar' o código. Para isso é necessário arrastar os blocos do menu para Área de montagem. Depois, é preciso encaixar os blocos na ordem desejada, sendo que os comandos são executados da esquerda para direita. Depois de montado o código deve ser ligado no bloco play, bloco azul com a seta de fundo verde e por último o usuário deve clicar na região da seta para enviar os comandos para o Besourino. A seguir a descrição dos comandos disponíveis.

Figura 4.2: Grupo Movimento.

O grupo movimento (figura 4.2) com a cor verde possui 6 blocos que controlam os motores do Besourino. O bloco com a seta apontada para direita faz o Besourino avançar para frente por um tempo que representa metade de seu comprimento, o bloco da seta

para esquerda faz o oposto. Já os blocos com setas curvadas para direita e para esquerda faz Besourino rotacionar 45 graus para direita ou para esquerda. Os dois últimos blocos desse grupo estão reservados para controlar a garra, um para abrir e outro para fechar. Colocando blocos iguais um do lado do outro estendem o tempo de duração do comando menos para os blocos que controlam a garra.

Figura 4.3: Grupo Lógico.

O grupo Lógico (figura 4.3) com a cor azul representa comandos não para controlar o Besourino mas sim voltado para semântica da ligação dos blocos. Ele é composto por três blocos. Como mencionado anteriormente o bloco play serve como bloco e botão pois ao ser clicado a seta do meio a interface irá enviar todos os comandos conectados a ele. É possível ter múltiplas instâncias de blocos play mas apenas um conjunto de comandos é executado por vez. Os dois últimos blocos são blocos macro. O primeiro parecido com o bloco play ,porém sem a seta, serve para armazenar um conjunto de operações. O segundo bloco macro quando associado com outro bloco macro irá armazenar as mensagens de comandos associadas a ele. Assim na criação da mensagem que será enviada para o Besourino ele será substituído pela série de comandos. A figura 4.4 demonstra seu uso, tanto o botão da esquerda e o da direita vão enviar a mesma mensagem.

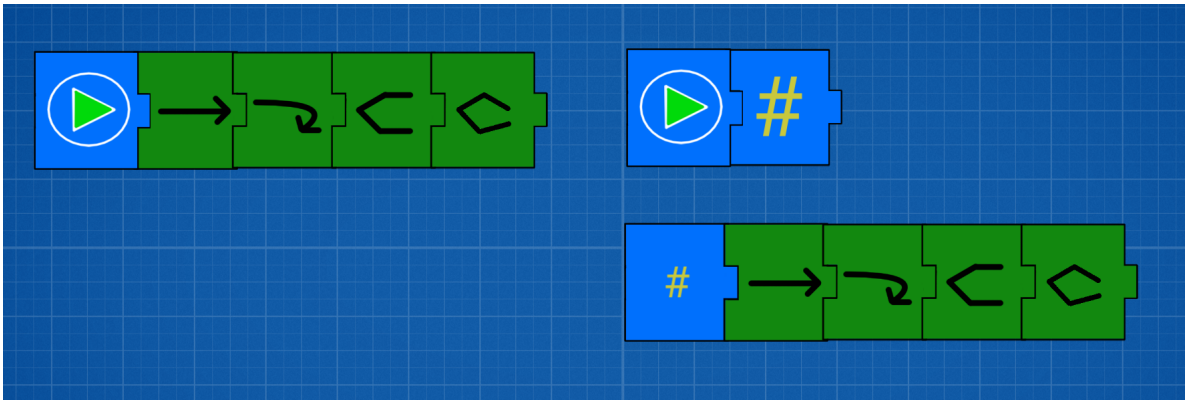


Figura 4.4: Exemplo de uso de bloco macro.

Figura 4.5: Grupo Sensor.

O grupo Sensor (figura 4.5) com a cor amarela possui apenas dois blocos de comando um relacionado ao sensor de distância e um ao sensor infravermelho. No caso o bloco sensor de distância, o robô seguirá em frente até que um objeto esteja na distância do

dente da garra. De forma semelhante o bloco sensor infravermelho fará o Besourino ir em frente até encontrar uma faixa preta.

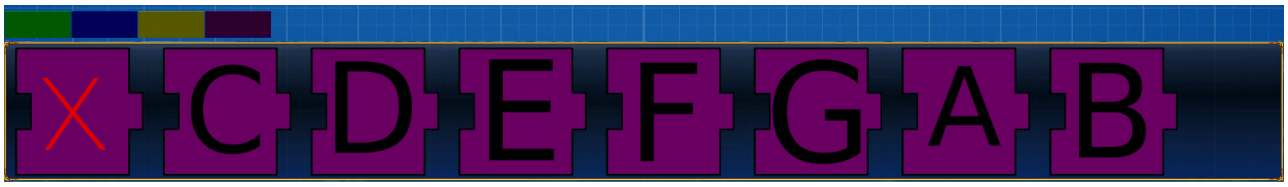


Figura 4.6: Grupo Áudio

O grupo Áudio (figura 4.6) com a cor roxa possui blocos relacionados ao buzzer, cada bloco representa uma frequência sonora diferente. Seguindo a nomenclatura de cifras temos `C` (dó), `D` (ré), `E` (mi), `F` (fá), `G` (sol), `A` (lá), `B` (si), sendo que a duração de cada nota é de 125 ms. O bloco com `x` representa uma pausa, na qual por um período de tempo de 125 ms o Besourino não fará nada. Agrupando notas iguais faz o tempo de duração de nota aumenta por `b` × `125ms`, sendo `b` quantidade de blocos iguais um ligado no outro.

4.5 Código

O código (figura 4.7) foi escrito em módulos em Python e arquivos em linguagem Kv. O módulo *main.py* contêm a classe MainGuiApp que é filha de App que contém o loop principal que por sua vez cuida de eventos e o estado da janela do aplicativo. Em kivy widget é a unidade que representa um elemento de uma interface gráfica podendo ser um botão, entrada de texto, um conjunto de widgets (layout) e etc. A classe *MainGUI* possui apenas um widget, no caso um layout MainGUI que herda da classe BoxLayout. A descrição do widget MainGUI está no arquivo *maingui.kv*, por padrão o Kivy relaciona classes e arquivos .kv que possuem o mesmo nome sem considerar letras maiúsculas. Além de *maingui.kv* os arquivos *draggingarea.kv* e *menublocks.kv* são carregados manualmente usando o método Builder.load.file do módulo Builder. Esses arquivos definem widgets que serão usados no aplicativo.

No arquivo *maingui.kv*, estão definidos os parâmetros do layout boxlayout e quais widgets ele contêm. BoxLayout posiciona todos os widgets da esquerda para direita (horizontal) ou de cima para baixo (vertical), para isso é preciso definir o parâmetro orientation. Ele possui dois widgets, o BlocksMenu e DrawingArea, que estão definidos respectivamente pelos arquivos *menublocks.kv* e *draggingarea.kv*. BlocksMenu é filho da classe RelativeLayout, layout que posiciona todos os objetos dependendo do tamanho pré-definido. BlocksMenu possui apenas o TabbedPanel. Este widget representa um menu de seleção de painéis. Ele é definido dinamicamente no arquivo *menublocks.py* a partir do arquivo *config.json*. O formato .json é um formato de dados independente de linguagem de programação semelhante ao Extensible Markup Language (XML), com a vantagem de ser mais compacto e simples de ler. Neste arquivo, estão definidos os menus e os blocos de comandos assim como a imagem que representa-o. Dessa maneira é possível adicionar novos grupos de blocos ou blocos apenas modificando o arquivo *config.json*, cada item em BlocksMenu é widget do tipo Block cuja as dimensões estão definidas no arquivo *menublocks.kv*.

O arquivo *config.json* (figura 4.8) é composto por um objeto lista *tab* que por sua vez é composto por outros objetos tipo listas representado cada grupo de comandos. Por último, cada grupo possui os atributos id, rgba e blocks. O atributo blocks é um lista

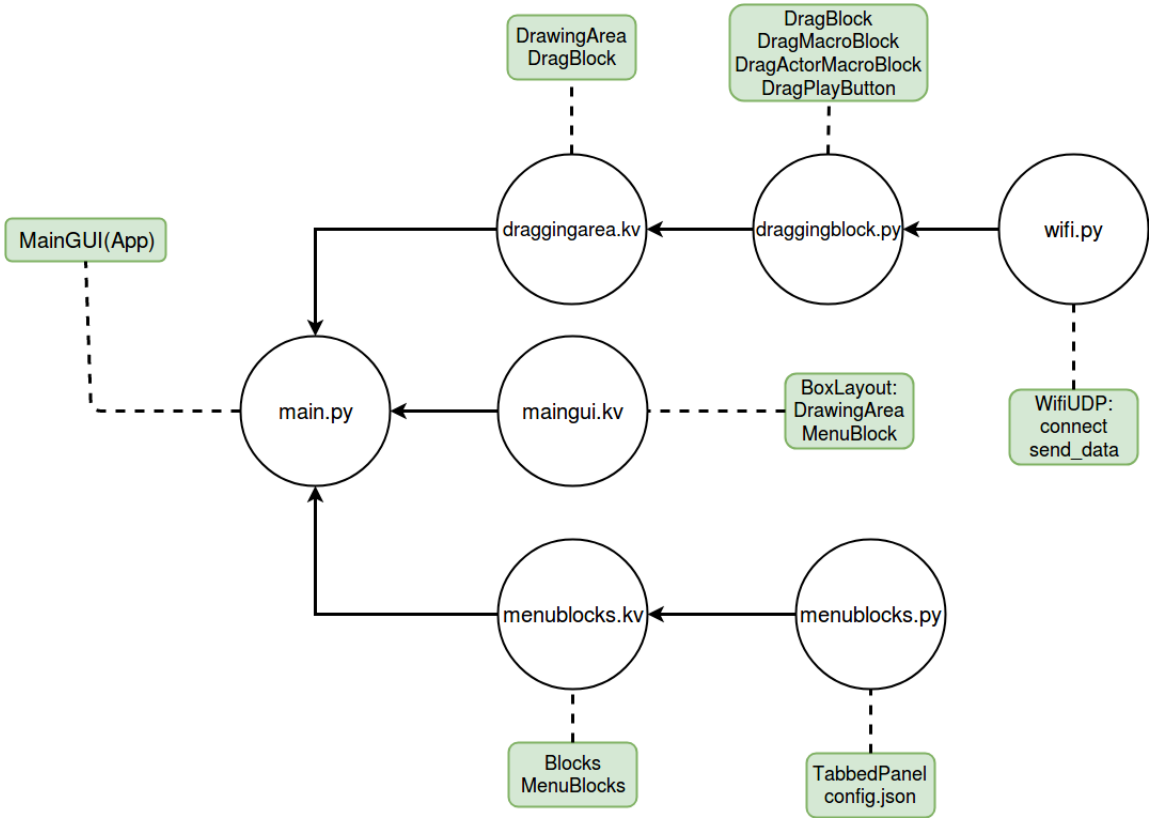


Figura 4.7: Relação dos arquivos.

que contêm um conjunto de objetos que representa os blocos vistos em cada menu. Em cada objeto temos três atributos como `name` (nome do comando), `id` (número usado na mensagem), `source` (caminho para uma imagem).

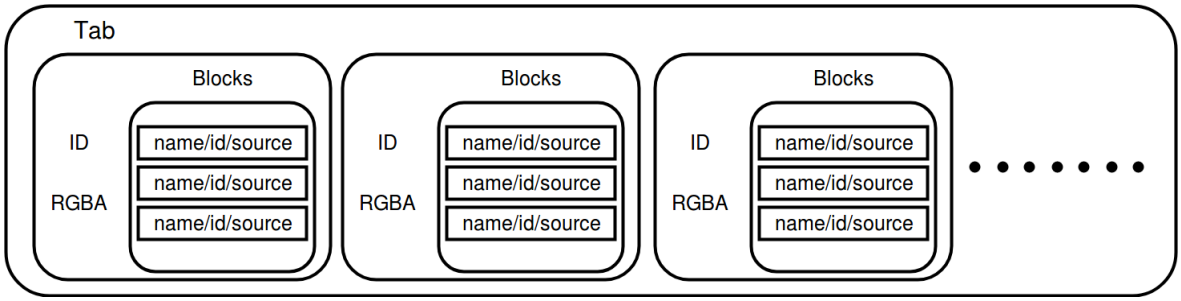


Figura 4.8: Arquivo de configuração.

DrawingArea definido em *draggingarea.kv* representa o espaço de montagem dos blocos, além desse espaço no arquivo *draggingarea.kv* está definido o widget DragBlock que por sua vez é definido de forma semelhando ao widget Block usado em menublocks.kv porêm, esse possui comportamento de arrastar e soltar e conectar com outros blocos definido em draggingblock.py. Neste módulo estão definidos comportamentos para os blocos. Ele possui quatro classes `DragBlock`, `DragMacroBlock`, `DragActorMacroBlock` e `DragPlayButton`. A classe `DragBlock` define comportamento de um bloco arrastável. Para isso foi utilizado eventos do Kivy. Todo widget pode gerar um evento e é necessário que o desenvolvedor define o que deve acontecer caso ele acontece. Para o `DragBlock` foram utilizados os eventos:

- `on_touch_down` que ocorre quando temos um clique ou um toque na tela;
- `on_touch_move` que ocorre quando o cursor é movido enquanto mantemos um clique ou um toque; e
- `on_touch_up` que ocorre quando liberamos um clique ou um toque.

Quando um bloco é solto é feito uma verificação se o mesmo está colidindo com outro bloco e se estiver próximo do encaixe os blocos se conectam.

`DragPlayButton` herda da classe DragBlock porém sem possibilidade de conectar com outro bloco pela esquerda. Outra diferença seria que este bloco funciona como um botão para enviar o grupo de comandos conectados pela direita. Usa o objeto criado em `wifi.py` para enviar os datagramas. `DragMacroBlock` e `DragActorMacroBlock` de forma semelhante a `DragPlayButton` herdam os mesmo comportamentos de DragBlock porém com algumas particularidades. No caso de `DragMacroBlock` existe um objeto para guardar uma lista de comandos que é atualizada toda vez que um bloco é conectado a ele, caso um bloco do tipo `DragActorMacroBlock` for conectado é feito um laço representado pelo símbolo # da mesma cor. Quando um `DragActorMacroBlock` está relacionado a um `DragMacroBlock` ele possui uma cópia da lista de comandos em DragMacroBlock.

4.6 Protocolo UDP

Como mencionado anteriormente `DragPlayButton` usa objeto my_socket definido em `wifi.py` para transmitir comandos para o Besourino. Para isso, `wifi.py` usa módulo padrão socket. Esse módulo possui métodos e objetos para realizar comunicação entre processos em máquinas diferentes através do protocolo internet. Nele, é possível utilizar o protocolo User Datagram Protocol (UDP).

O protocolo UDP utiliza datagramas para realizar comunicação entre máquinas. Datagramas são blocos de mensagem formado por cabeçalho e dados. No caso do UDP, um pacote é formado pelo endereço IP de origem e destino mais o datagrama, sendo assim composto por 7 campos:

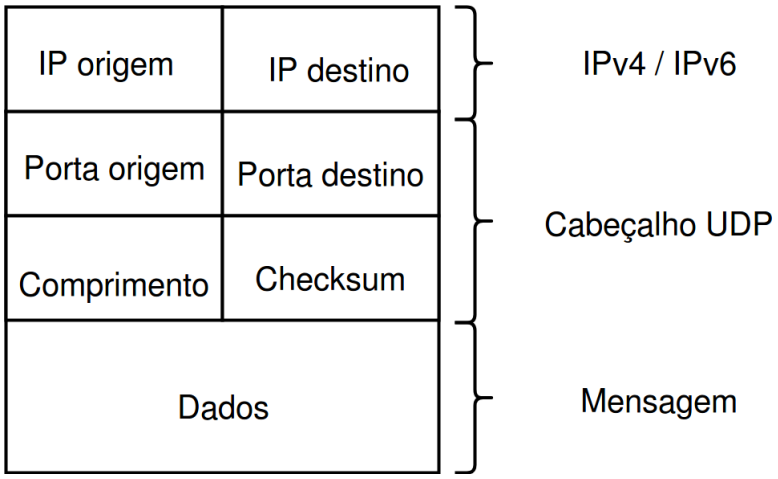


Figura 4.9: Pacote UDP

- **endereço de origem:** endereço IP da máquina que está enviando a mensagem, utiliza 4 bytes (32 bits);

- **endereço do destinatário:** endereço IP da máquina que irá receber a mensagem, utiliza 4 bytes (32 bits);
- **porta da origem:** número da porta (opcional) do processo da máquina que está enviando a mensagem, utiliza 2 bytes (16 bits);
- **porta do destinatário:** número da porta do processo da máquina que está recebendo a mensagem, utiliza 2 bytes (16 bits);
- **comprimento:** quantidade de bytes utilizado pela mensagem cabeçalho mais dados, utiliza 2 bytes (16 bits). Obs: um datagrama UDP deve ter no mínimo 8 bytes;
- **checksum:** única proteção contra erros de envios ou de perda de dados, sendo opcional também; e
- **dados:** Conteúdo da mensagem em si em bytes.

Neste protocolo diferente do *Transmission Control Protocol*(TCP), não há sincronização ou verificação dos pacotes. A máquina cliente envia o pacote para máquina destinatária, que por sua vez apenas aguarda o pacote chegar. Apesar de não ter garantia da recepção e da consistência dos dados (apenas o checksum, quando for utilizado), o protocolo UDP mostra-se ideal em casos onde precisa transmitir um número pequenos de bytes o mais rápido possível. Logo devido a curta distância entre máquina executando a interface gráfica e o Besourino e também pelo fato que o tamanho dos dados enviados estar na casa de dezenas bytes, foi escolhido o protocolo UDP como meio de comunicação.

Capítulo 5

Comentários finais

Ao longo do segundo semestre do ano de 2017 houve três exposições do Besourino em três eventos. Sendo o primeiro evento de cultura maker organizando pela prefeitura da cidade de São Paulo o SP Maker Week. Nele o grupo de extensão Hardware Livre USP participou realizando uma oficina de Arduino básico seguido de exposições de projetos. Neste evento Besourino foi apresentado para pessoas diversas, desde crianças até adultos.

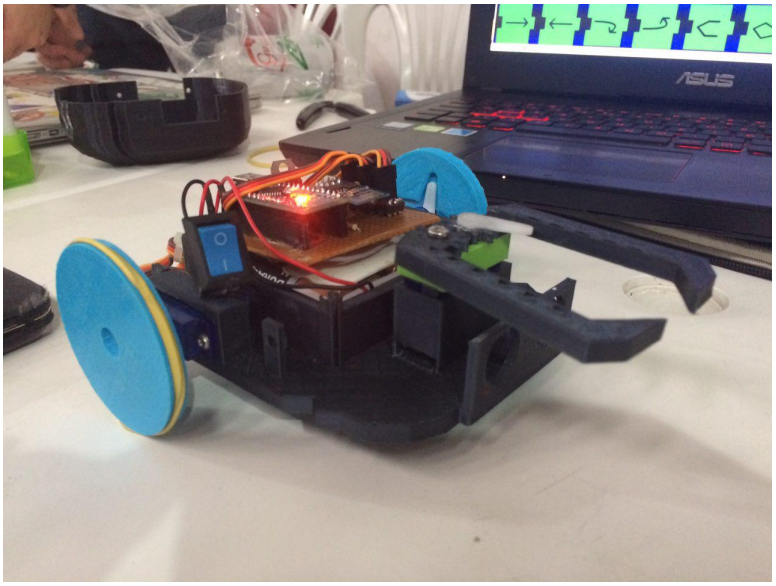


Figura 5.1: SP Maker Week

Em um segundo evento o Besourino foi mostrado para crianças e pais da escola Escola Municipal de Ensino Fundamental Desembargador Amorim Lima no evento Feira da Cultura. No local aconteceu uma feira de projetos feitos pelos alunos usando Scratch e Arduino. No caso, Besourino foi apresentado junto com a impressora 3D da escola. Neste evento, o projeto foi exposto e testado em sua maioria por crianças e adolescentes.

O terceiro e último evento foi a semana USP de tecnologia e ciência que ocorreu no Centro de Difusão Internacional (CDI -USP). Ao longo da semana escolas trouxeram alunos para participarem e observarem projetos de diversos grupos de pesquisa da universidade. No caso, o Hardware Livre USP apresentou seus projetos sendo Besourino um deles. Novamente o Besourino foi apresentado e testado por crianças e adolescentes porém de forma mais intensa. Neste evento foi possível confirmar a simplicidade de entender a interface gráfica e a curiosidade das crianças para ver a garra do robô em ação.

Figura 5.2: Feira da Cultura

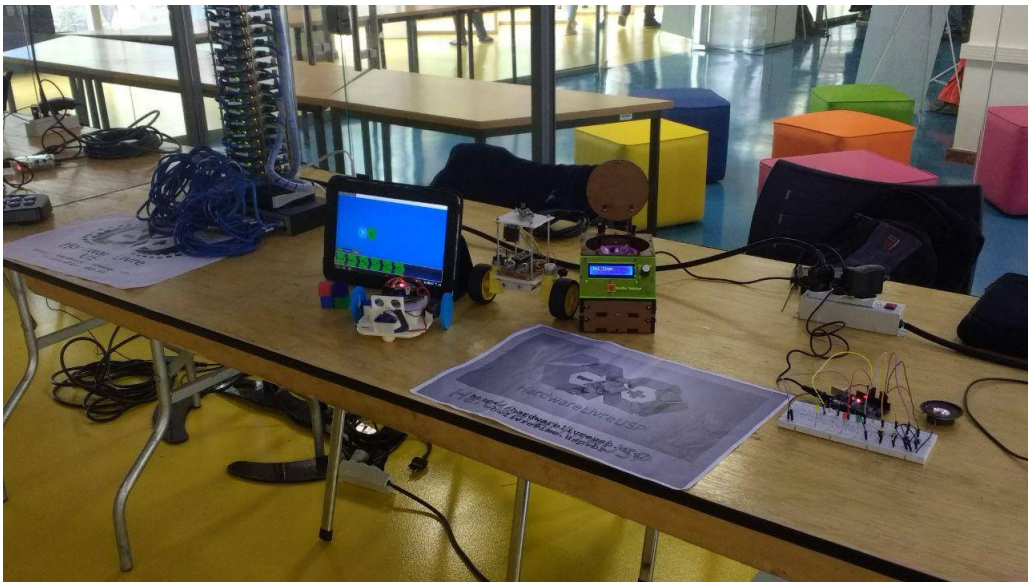


Figura 5.3: Semana USP ciência e tecnologia

Como vimos anteriormente existe uma tendência para que nos próximos anos os profissionais de áreas diversas precisarão ter um conhecimento básico de programação. Para atender essa necessidade, kits de robóticas surgiram ao longo dos anos, como os produtos da Lego e produtos que usam hardware abertos como a Maker Block. Dado seu alto custo e pouca flexibilidade, esses projetos são difíceis de se aplicar em escolas públicas. No caso do Besourino foi possível desenvolver um ferramenta que estimula lógica de programação com custo médio de duzentos reais. Usando ferramentas abertas e disponibilizando tanto o código e os esquemáticos, outros desenvolvedores podem contribuir ou iniciar novos projetos a partir dele. Para trabalhos futuros está planejado expandir o Besourino para interagir com a linguagem `Scratch`. No caso, pretende criar blocos personalizados voltados para controlar o Besourino. Para próxima versão pretende usar um microcontrolador com maior desempenho e com bluetooth e WiFi integrado, o ESP-32. Assim não seria mais preciso usar um módulo extra de comunicação sem fio. Pretende também fazer um quarto modelo de chassi que seja menor e que não precise usar porcas ou parafusos para encaixar suas partes, de modo a facilitar a produção de novos robôs.

Bibliografia

[1] Arduino Foundation. *Arduino*. URL: www.arduino.cc.

[2] ELEC Freaks. *Ultrasonic Ranging Module HC-SR04, consultado 2017*. [Online; acessado em agosto-2017]. URL: http://www.micropik.com/PDF/HCSR04.pdf.

[3] Michael Margolis. *Arduino Cookbook*. O'REILLY, 2011, p. 631. ISBN: 978-0-596-80247-9.

[4] Microchip. *ESP8266 Serial Esp-01 WIFI Wireless, consultado 2017*. [Online; acessado em julho-2017]. URL: http://www.microchip.ua/wireless/esp01.pdf.

[5] Kivy Organization. *Kivy API reference*. [Online; acessado em novembro-2017]. URL: https://kivy.org/docs/api-kivy.html.

[6] Brando Rhodes e John Goerzen. *Foundations of Python Network Programming*. Apress, 2015, p. 551. ISBN: 978-85-7522-437-3.

[7] Jenifer Tidwell. *Designing Interfaces*. O'REILLY, 2016, p. 547. ISBN: 978-1-449-37970-4.

[8] Roberto Ulloa. *Kivy - Interactive Applications and Games in Python*. PACKT, 2015, p. 183. ISBN: 978-1-78528-692-6.

[9] Vishay. *Reflective Optical Sensor with Transistor Output, consultado 2017*. [Online; acessado em agosto-2017]. URL: https://www.vishay.com/docs/83760/tcrt5000.pdf.