SENAI NAMI JAFET

TÉCNICO EM ELETROELETRÔNICA

CONTROLE DE UM ROBÔ QUADRÚPEDE

MOGI DAS CRUZES

2020

SENAI NAMI JAFET

TÉCNICO EM ELETROELETRÔNICA

ANDRESSA BOCZ

BRENO LISBOA

ERIC NOÉ

FELIPE SERRA

JASMIN MIANI

CONTROLE DE UM ROBÔ QUADRÚPEDE

Trabalho de Conclusão de Curso apresentado ao SENAI Nami Jafet como um pré-requisito para a obtenção do certificado de Técnico em Eletroeletrônica, sob orientação do Professor Carlos.

MOGI DAS CRUZES

2020

SENAI NAMI JAFET

TÉCNICO EM ELETROELETRÔNICA

ANDRESSA BOCZ

BRENO LISBOA

ERIC NOÉ

FELIPE SERRA

JASMIN MIANI

CONTROLE DE UM ROBÔ QUADRÚPEDE

Este trabalho foi considerado adequado para a obtenção de aprovação na disciplina de Projetos do Curso Técnico em Eletroeletrônica da escola SENAI Nami Jafet.

_____

Professor Orientador

_____

Professor

_____

Professor

MOGI DAS CRUZES

2020

Dedicamos este trabalho primeiramente aos alunos membros da equipe que juntos trabalharam tanto por este projeto e aos professores que tanto nos ensinaram com muito esforço e dedicação. Também aos nossos amigos e colegas de classe que aos poucos conhecemos e trocamos experiências escolares e opiniões sobre o conteúdo por todos aprendido.

Agradecemos a todos os professores que se dedicaram a nos transmitir conhecimento e todo o conteúdo a qual usaríamos para este projeto, e a todos os companheiros que sempre nos apoiaram ajudando no possível. Assim como aos familiares dos membros da equipe que nos deram apoio moral nos incentivando sempre, mesmo sem terem envolvimento direto.

EPÍGRAFE

“Conhecimento não é aquilo que você sabe, mas o que você faz com aquilo que você sabe.”

Aldous Huxley

RESUMO

O robô quadrúpede tem como objetivo, ser um objeto para lazer e, principalmente, servir de auxílio para estudos sobre microcontroladores, programação e eletrônica. O robô quadrúpede, aprimorado pelos membros da equipe, deve oferecer funções e ações diversas, graças a programação em linguagem C que foi adicionada ao microcontrolador e ao aplicativo desenvolvido, um baixo custo, alto rendimento, por conta do banco de três baterias de lítio, e facilidade de manipulação, de modo a oferecer uma completa e divertida experiência, por conta do design intuitivo e colorido.

Palavras-chave: Quadrúpede, robô, micro servo, Arduino, microcontrolador.

ABSTRACT

The quadruped robot aims to be an object for leisure and, mainly, to serve as an aid for studies on microcontrollers, programming and electronics. The quadruped robot, enhanced by team members, should offer various functions and actions, thanks to c-language programming that has been added to the microcontroller and the developed application, a low cost, high yield, on account of the bank of three lithium batteries, and ease of handling, to offer a complete and fun experience, due to the intuitive and colorful design.

Keywords: Quadruped, robot, micro-servo, Arduino, microcontroller.

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

A área de eletrônica e programação é bem extensa e sempre em constante evolução, com novidades a cada dia. Visando essa necessidade de estar sempre inteirado no assunto, o robô quadrúpede, também chamado de robô aranha, pode nos ajudar a ensinar e aprender sobre o assunto, de forma didática e divertida. Como o robô é leve e de fácil transporte, o seu uso diário acaba sendo acessível a todo tipo de público, professores, alunos e até para quem só quer utilizá-lo para lazer sem fins estudantis.

Esse robô busca melhorar o entendimento estudantil em eletrônica, tanto na parte teórica como na parte prática, o projeto é de fácil manipulação para customização e edição de funções em sua programação e estrutura.

1. DESENVOLVIMENTO

Para o desenvolvimento desse projeto foram utilizados: um conjunto pronto para a montagem da estrutura do robô, um Arduino UNO, um sensor shield, oito micro servos, um banco de baterias, uma placa controladora de Power Bank, uma alavanca de 3 estados e um módulo bluetooth utilizado em conjunto com um aplicativo para celular desenvolvido com o fim de controlar os movimentos do robô.

1.1 Componentes

1.1.1 Arduino UNO

O Arduino é uma placa de prototipagem que torna possível o desenvolvimento de projetos, atuando como o “cérebro” eletrônico programável. É projetado através de um micro controlador de programação específico, com pinos de entrada e de saída, digitais e analógicas, além de pinos próprios para alimentação e comunicação. O Arduino funciona a partir de códigos de programação, onde pode ser destinado a diversos tipos de funções. A programação é em linguagem C/C++ e pode ser feita por meio do programa “IDE Arduino” e a conexão com o computador é feita via cabo USB, permitindo que os comandos definidos no programa sejam transferidos até a placa. Após gravar os códigos, ele pode ser instalado em diversos locais com o uso de fontes para alimentação ou até mesmo baterias se precisar ficar em local isolado. A conexão com os sensores pode ser feita de forma direta em suas portas de comunicação com o uso de jumpers ou em protoboards (placas de ensaio).

A placa Arduino Uno, que foi a utilizada neste projeto, possui várias características interessantes em seu hardware. A alimentação da placa pode ser feita por uma fonte externa ou por uma conexão USB (como por exemplo um notebook), a alimentação por fonte externa é feita por um conector “Jack”, onde a tensão externa deve estar entre 6V a 20V, porém se for alimentada por uma tensão abaixo de 7V, a tensão de funcionamento do Arduino UNO, que é 5V, pode ficar instável e quando

alimentada acima de 12V, o regulador pode sobreaquecer e acabar danificando a placa. Por isso é recomendado usar valores de 7V a 12V para fontes externas. Quando a alimentação é dada por um cabo USB, a tensão não precisa ser estabilizada pelo regulador de tensão por que o circuito da USB já possui componentes que protegem a porta USB do computador em caso de alguma anormalidade.

Os conectores para alimentação são: IOREF (fornece tensão para que Shields possam selecionar o tipo de interface apropriada, dessa forma, Shields que funcionam com placas que precisam de 3,3V, podem se adaptar e serem utilizados em 5V e vice-versa), 3,3V (alimentação de 3,3V para shields e módulos externos), 5V (alimentação de 5V para circuitos externos e shields), GND (ground ou terra) e Vin (pino que alimenta a placa através de uma bateria externa ou um shield, quando a alimentação vem através do conector “Jack” a tensão da fonte se encontra nesse pino).

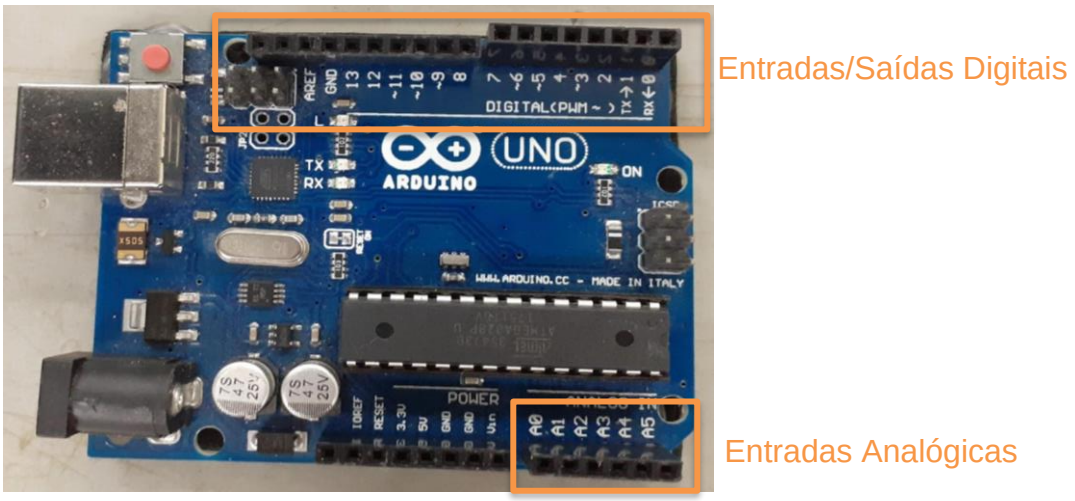


Figura 1 - Arduino UNO com indicação das entradas - Fonte: Própria

Como interface de comunicação com o computador, existe na placa, um microcrontrolador ATMEL ATMEGA16U2, com 32KB de Flash (com 512 Bytes usados para o bootloader), 2KB de RAM e 1KB de EEPROM, e é o responsável pela forma como funciona a placa do Arduino UNO, tornando possível o upload do código em binário gerado depois da compilação do programa feito pelo usuário. Neste microcontrolador também existem dois LEDS, TX e RX, que são responsáveis por indicar o envio e a recepção dos dados da placa para o computador. Ele possui 28

pinos, sendo que 23 desses podem ser utilizados com I/O. Esse componente possui como periféricos uma USART (Universal Serial Asynchronous Receiver Transmitter, é uma porta serial para comunicação entre a placa Arduino e um computador ou outros dispositivos), uma SPI (Serial Peripheral Interface, um protocolo de dados seriais síncronos para comunicação entre o microcontrolador e outros periféricos) e uma I2C (Inter-Integrated Circuit, usa dois fios, DAS: Dados e SCL: Clock, para transmitir e receber informações, não ao mesmo tempo, apenas um sentido por vez), além disso possui também um comparador analógico interno ao CI e 6 PWM (Pulse Width Modulation, é o conceito de pulsar rapidamente um sinal digital em um condutor).

1.1.2 Sensor Shield

Os shields são placas de circuito para conectar ao Arduino, servem para expandir e facilitar as conexões. Eles possuem diversos tipos para qualquer usuário e podem conter displays LCD, módulos de comunicação, sensores ou relés, por exemplo. O que foi utilizado nesse projeto foi o “Arduino Sensor Shield V5.0”, que é um shield expansor que facilita a conexão de outros módulos, podendo ser um Arduino UNO ou Arduino Mega. Esse módulo suporta interface I2C, que são utilizadas apenas duas linhas de dados para comunicação de dispositivos entre si, módulos APC220, que são módulos de rádio wireless, utilizados para controlar o Arduino através do computador e também suporta LCD Serial e Paralelo, Servo Motor, módulos bluetooth, SD Card e ainda possui entradas para alimentação ganhando muito espaço para trabalhar de forma livre.

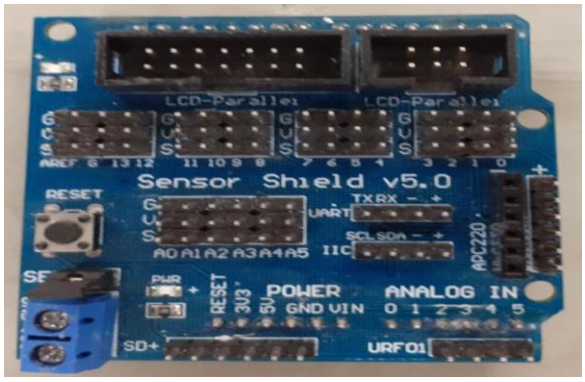


Figura 2 - Sensor Shield - Fonte: Própria

1.1.3 Micro-servo SG90

O Micro Servo Motor SG90 é um motor muito utilizado em aplicações para robótica, nos sistemas microcontroladores, como, por exemplo, Arduino, PIC e AVR. Ele é um módulo que apresenta movimentos proporcionais aos comandos indicados, controlando o giro e a posição, diferente da maioria dos motores. Em específico trabalha com uma margem de tensão de 3 a 6 volts e uma corrente de 500mA. Servo motores são dispositivos de malha fechada, seu funcionamento dá-se por meio do recebimento de um sinal de controle chamado de PWM, onde passam a verificar a posição atual, e atuam no sistema indo para a posição desejada.

O sinal PWM (Pulse Width Modulation) ou Modulação de Largura de Pulso, é utilizado em sistemas digitais, consiste em uma técnica para variar o valor médio de uma onda de forma periódica. Assim, é possível controlar em uma onda quadrada o tempo em que ela fica em nível lógico alto, mantendo sua frequência, mas alterando o seu valor médio de tensão ao longo do tempo. O tempo em que o sinal está em nível lógico alto, é chamado de Duty Cycle, ou, Ciclo Ativo.

Figura 3 - Micro-servo SG90 - Fonte: Própria

1.1.4 Banco de baterias (LGAAS31865)

Na construção do banco de baterias foram utilizados três módulos de uma bateria de lítio, LGAAS31865, em paralelo, pois assim a capacidade das células é somada. Cada módulo possui uma tensão nominal de 3,6V e 2200mAh de capacidade, o que significa que se ligado a um circuito que exija 2,2A ele tem capacidade para alimentá-lo por uma hora ininterrupta, e uma tensão de recarga de até 4,2V e 1A.

O motivo da utilização de um banco de baterias e não um único módulo se dá por uma única célula não possuir corrente de descarga máxima (2700mA) suficiente para todas as funções programadas, sendo que a corrente máxima exigida, a corrente de pico na partida dos Microservos, é de 3200mA, ou seja, colocar duas baterias em paralelo dá conta de fornecer a corrente necessária para o pleno funcionamento do projeto, porém, sobraria espaço útil no local onde o banco de baterias se encontra, então foi optado por colocar um terceiro módulo no conjunto, o que aumenta ainda mais a capacidade, totalizando 6600mAh, que faz a duração da utilização do projeto aumentar, de quase 5 horas, com dois módulos, para 7 horas e 30 minutos, com um consumo máximo e quase constante de 900mA, sendo que este valor de consumo é originário da opção de movimento em que há um maior consumo, então ela ficará ativa por no mínimo 7h e 30 min. Sendo assim, o tempo de recarga é, por volta, de 5h. E é necessário estar ciente de que as baterias utilizadas não são novas, ou seja, não estão em seu pleno potencial.

Figura 4 - Baterias de lítio LGAAS31865 - Fonte: Própria

1.1.5 Módulo carregador de baterias BMS TP 4056

Também conhecido como placa controladora, possibilita carregar uma bateria por meio de um carregador de celular, conectando como se fosse um celular comum, e utilizar essa carga por meio da porta Full USB. Sendo que, o valor máximo de entrada é 5V e de saída 4,2V, que é o máximo que a nossa bateria pode ser alimentada, possuindo um LED que sinaliza quando está carregando, em vermelho piscante, e quando está totalmente carregado, em vermelho contínuo. Foi utilizada em conjunto com uma alavanca de 3 estados para acionamento da bateria.

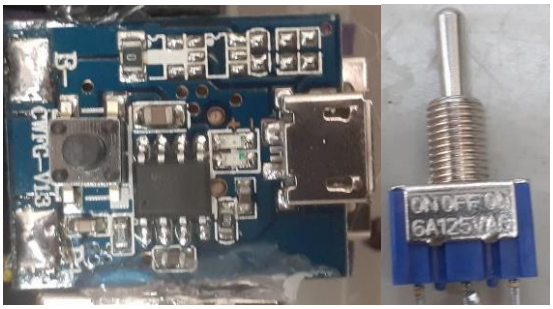


Figura 5 - Placa controladora e alavanca de 3 estados - Fonte: Própria

1.1.6 Módulo Bluetooth (HC-06)

O Módulo Bluetooth HC-06 trabalha com uma baixa voltagem (3.1V~4.2V) e corrente em emparelhamento está na faixa de 30~40mA. Ele possibilita transmitir e receber dados através de comunicação sem fio (comunicação wireless) pela porta serial do arduino (TX e RX), todas as placas Arduino possuem pelo menos uma porta serial (também conhecida como UART ou USART) simplificando é um Transmissor/Receptor Universal Síncrono e Assíncrono sendo assim um formato padrão para comunicação de dados de forma serial.

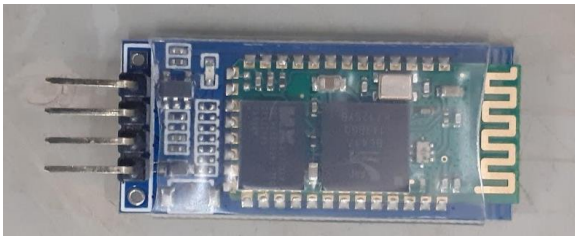


Figura 6 - Módulo bluetooth HC-06 - Fonte: Própria

1.2 Movimentação

1.2.1 Gravidade

Ao efetuar os estudos sobre a forma como o centro de gravidade do robô se comporta aos seus movimentos, foi necessário analisar o grau em que cada servo-motor deve se ajustar, para então poder transcrever esse movimento em linguagem C, pois essa é a linguagem aceita pelo software do Arduino, que foi o componente responsável por todos os acionamentos do robô.

1.2.2 Movimentos

Para a movimentação foi necessário elaborar uma sequência de acionamentos nos servo-motores para que as pernas do robô fizessem os movimentos de ir para a frente, traz, direita e esquerda, além de girar no mesmo lugar, acenar e dançar. Partindo desse ponto, constantes implementações foram feitas no código da programação, para que o sinal fosse recebido corretamente pelo microcontrolador, fazendo assim uma movimentação bem sincronizada entre as pernas do robô.

1.2.3 Programação do Arduino

1.2.3.1 Arduino IDE

Para escrever um programa no Arduino, só é preciso conectá-lo à um computador por meio de um cabo USB e utilizar o IDE, que é um ambiente de programação próprio do Arduino, onde se digita o programa e se faz testes para encontrar erros antes de compilar e transferir para o dispositivo. Após a transferência, o Arduino já está pronto para uso. A linguagem utilizada é a linguagem C.

Pode-se começar um programa utilizando a estrutura básica do Arduino, que é formada por dois blocos, ou duas partes: o primeiro bloco é o setup() e nessa parte são configuradas as opções iniciais da programação, sendo elas valores de uma variável, o que será entrada e saída, entre outras funções. O segundo bloco é o loop() e aqui é onde se repetem os comandos de forma contínua ou até que algum comando de “stop” seja enviado para o dispositivo.

Para o robô quadrúpede foram utilizadas além de estruturas básicas de decisão, uma biblioteca JSON para armazenamento das variáveis e uma biblioteca específica para integração do bluetooth com o arduino. O algoritmo desenvolvido em C está disponível no Apêndice A, página 27.

1.2.3.2 Blockbench

Para simulação dos movimentos e forma de facilitar a programação em C foi utilizado brevemente o software de simulação Blockbench, que é um programa de modelagem em 3D muito utilizado para animações e simulações de projetos que necessitam de movimentação.

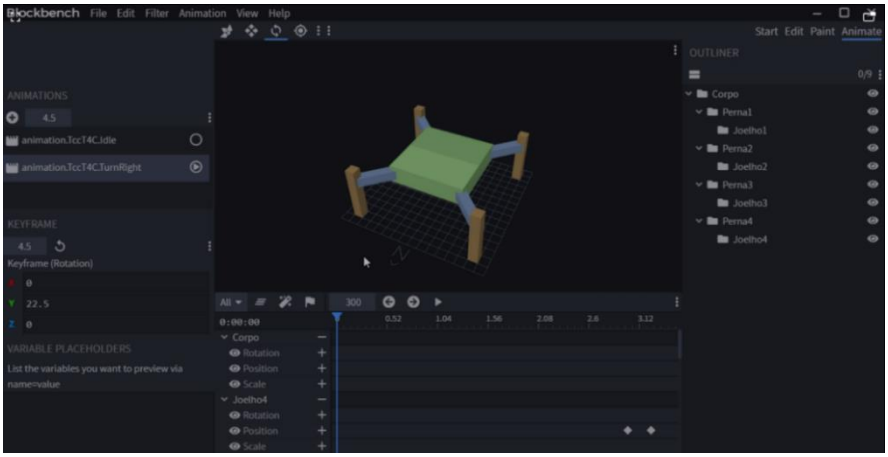


Figura 7 – Simulação - Fonte: Própria

1.2.4 Programação do Aplicativo

1.2.4.1 App Inventor

O App Inventor é uma ferramenta online e gratuita mantida pelo MIT (Massachusetts Institute of Technology) para iniciantes na programação de aplicativos para celular. Com ela é possível a criação de aplicativos a partir de blocos de programação.

Neste projeto ela foi utilizada para desenvolver um aplicativo capaz de controlar os microservos pelo celular utilizando sinal bluetooth. Sua praticidade também possibilitou a criação de uma tela inicial e de pareamento de simples utilização.

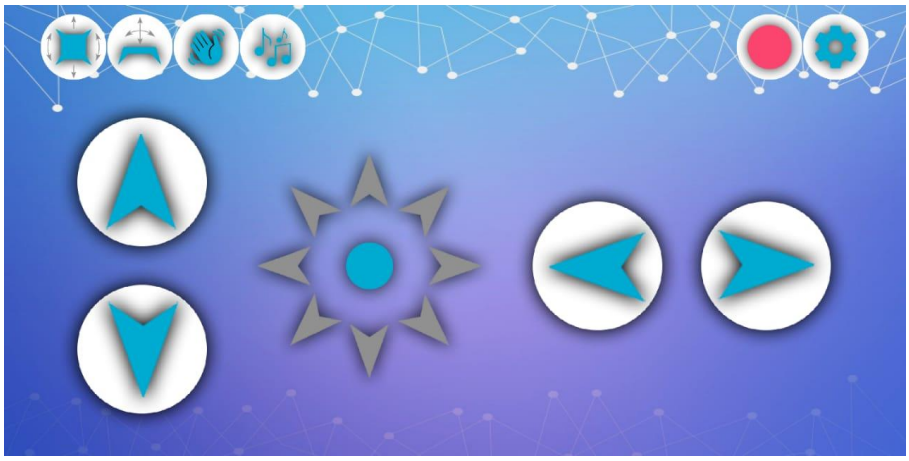


Figura 8 - Tela inicial do aplicativo - Fonte: Própria

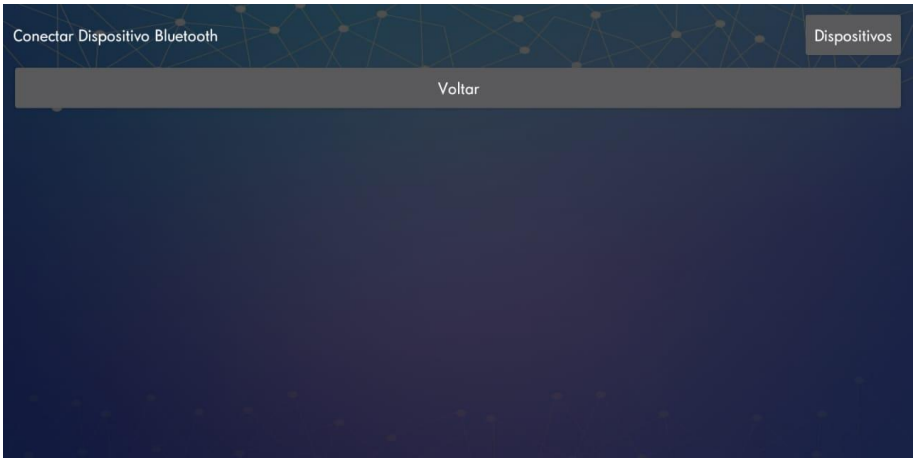


Figura 9 - Tela de pareamento - Fonte: Própria

O algoritmo desenvolvido em blocos está disponível no Apêndice B, página 44.

1.3 Montagem e funcionamento

Começando com a montagem do “esqueleto” do projeto, foi necessário seguir um manual para evitar confusão e estragar as peças. Logo após colocar os microservos em seus devidos lugares, para evitar problemas posteriores, foi realizada a medição da posição inicial e final deles, com base na angulação padrão, 0 (zero) e 180 (cento e oitenta) graus.

Assim sendo, teve-se de posicionar o Arduino Uno junto do sensor shield provisoriamente em cima da placa superior, para assim ser possível analisar a melhor posição dos cabos que virão dos microservos, já que o Arduíno UNO possuí 6 entradas PWM que auxiliam em um melhor funcionamento dos motores, então demos a preferência das portas PWM para os quatro motores superiores, que são os principais para a movimentação, e as duas que sobraram foram colocadas nas duas patas da frente, mas sem preferência nesse caso.

Feito isso, ligou-se os conectores de sinal do Módulo Bluetooth HC-06, nas entradas 12 e 13 de sinal, as nomeando de RX e TX, que são entradas específicas de recebimento (RX) e transmissão (TX) de sinal, enquanto os outros dois terminais foram conectados nas entradas VCC e GND próprias para o módulo Bluetooth.

Por fim, o banco de baterias foi posicionado na parte inferior do projeto. Porém, não foi uma ligação direta entre o banco e a placa de Arduino, primeiro conectou-se o módulo BMS TP 4056 diretamente à bateria para permitir a recarga por cabo micro USB, e, paralelamente ao cabo de alimentação positiva, uma alavanca foi adicionada, para permitir que o projeto fosse ligado e desligado sem consumo de bateria, após isso foram feitas as conexões nos bornes de entrada da placa de Arduino UNO.

Terminada a montagem, iniciou-se a configuração dos movimentos e acionamentos do nosso robô. As configurações são dividias em 3 partes: Inicialização, conexão e movimentação.

- **Inicialização:** Faz com que o módulo bluetooth inicie, de modo a permitir a comunicação com o aplicativo de celular, e ao mesmo tempo faz com que os motores inferiores acionem em 70º, pois assim eles retiram o projeto do chão, permitindo ainda erguer as patas um pouco mais para desviar de algum obstáculo, assim como os motores superiores, que iniciarão em 90º.

- **Conexão:** Ao iniciar o aplicativo aparece um aviso “pop-up” na tela pedindo para ligar o bluetooth do celular. Feito isso, é necessário conectar com o projeto, bastando ir às configurações > dispositivos e selecionar o bluetooth “AranhaRobo”, e assim que a conexão for feita a luz vermelha no aplicativo ficará verde, e a luz piscante no módulo bluetooth ficará contínua.

- **Movimentação:** Neste tópico, além da movimentação padrão, que é ir para frente e para os lados, temos duas movimentações pré configuradas, o “aceno” e a “dança”, assim como dois modos de movimentação, o “tilt” ou “inclinar”, que ao invés de fazer o robô andar para frente ou em qualquer outra direção ele se inclina naquela direção, e o “rotacionar”, que faz o robô girar em seu eixo central.

1.4 Resultados

Não foram encontrados problemas na programação do robô, entretanto, logo no início do projeto foi necessário trocar a alimentação, que vinha de um Power Bank que não suportava os movimentos de dança, por um banco de baterias (descrito no item 1.1.4). Após este ajuste cogitamos a instalação de um módulo mp3, porém esta ideia foi descartada pois o microcontrolador utilizado não tem suporte para mais de um módulo que necessita das entradas RX-TX.

Já nos testes finais foi necessário colocarmos uma porca a mais em cada parafuso que faz a movimentação das patas, para que o robô conseguisse realizar a sequência de movimentos sem soltar nenhuma das partes de seu esqueleto.

Para efeito estético foram adicionados olhos de plástico e o robô foi nomeado pelos membros da equipe como “Aranha Cristal”, sendo utilizado também o pronome feminino para se referir ao projeto.

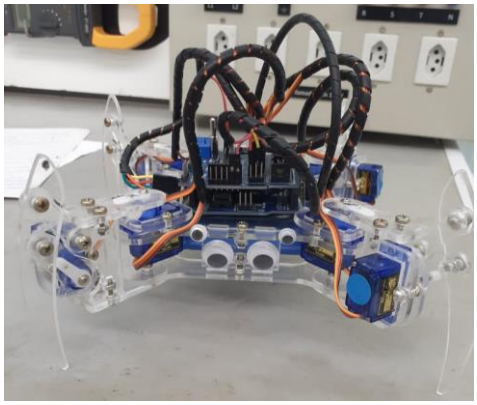


Figura 10 - Montagem Final - Fonte: Própria

2. ESTUDO DE VIABILIDADE

2.1 Custo

Componentes	Quantidade	Preço	Total	loja/site
Arduino UNO	1	R$ 33,50	R$ 33,50	Lojas Americanas
Sensor Shield	1	R$ 21,00	R$ 21,00	Mercado Livre
Micro Servo (SG90)	8	R$ 14,90	R$ 119,20	Robocore
Módulo Bluetooth (HC-06)	1	R$ 30,70	R$ 30,70	Lojas Americanas
Bateria LGAAS31865	3	R$ 25,00	R$ 75,00	Mercado Livre
Módulo carregador de litio (TP4056)	1	R$ 6,90	R$ 6,90	Filipe Flop
Kit de acrílico	1	R$ 89,99	R$ 89,99	Mercado Livre
		Total	R$ 376,29	

Figura 11 - Tabela de custos do projeto - Fonte: Própria

2.2 Comparativo

Com base em pesquisas feitas, foi possível analisar que o projeto tem um custo baixo em comparação com outros robôs quadrúpedes que são encontrados no mercado. A média de custo se encontra entre R$ 400 e R$500, sendo que muitas vezes a programação não vem inclusa, ou apenas vem um modelo, e com poucos movimentos disponíveis, enquanto este projeto possui grande diversidade de movimentos e programação completa, além de não haver necessidade da utilização de um controle remoto, bastando apenas o download do aplicativo.

2.3 Cronograma

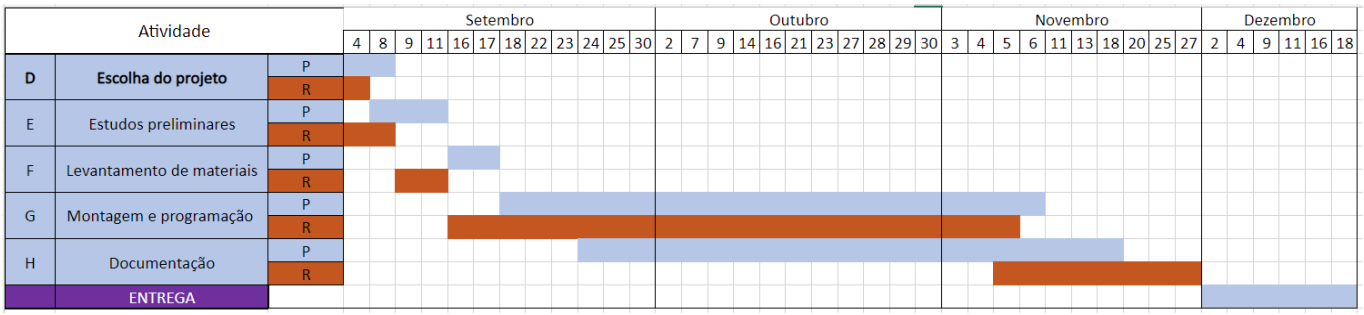


Figura 12 - Cronograma/Diagrama de Gantt - Fonte: Própria

As linhas coloridas em azul claro representam o tempo previsto (P) e as linhas alaranjadas representam o tempo real (R) utilizado pela equipe durante a resolução do projeto durante os meses de setembro, outubro, novembro e dezembro.

3. CONCLUSÃO

Neste trabalho, pôde-se concluir que, o projeto de conclusão de curso pode reunir todos os módulos aprendidos durante o curso para o desenvolvimento de um único projeto, que visa oferecer suporte a uma empresa real, e dessa forma nos dar início no meio profissional. Também a inicialização desse projeto tornou a equipe mais unida juntamente com todos os integrantes de outros grupos, e dessa forma abrindo interatividade no meio acadêmico e profissional, gerando assim uma forma maior de conhecimentos.

REFERÊNCIAS BIBLIOGRÁFICAS

A Spierce Technologies. (s.d.). Fonte: mePed: http://meped.io/mepedv2

Buckley, I. (s.d.). *How to Control Robots With a Game Controller and Arduino*. Fonte: Make use of: https://www.makeuseof.com/tag/arduino-robot-game-controller/

Girish, R. (s.d.). *Circuito indicador de nível de bateria do Arduino*. Fonte: Axtudo: https://www.axtudo.com/circuito-indicador-de-nivel-de-bateria-do-arduino/

LG Chem. (28 de maio de 2007). *Rechargeable Lithium Ion Battery ICR18650 S3 2200mAh.* Fonte: https://www.tme.eu/Document/851884e7c9339fa73b5bf66663baac51/ACCU-ICR18650-2.2.pdf

Oliveira, E. (s.d.). Fonte: Master Walker Electronic Shop: https://blogmasterwalkershop.com.br/arduino/como-usar-com-arduino-micro-servo-motor-sg90-9g/

Robocore. (s.d.). *Modulo I2C - Primeiros passos*. Fonte: Robocore: https://www.robocore.net/tutoriais/primeiros-passos-com-modulo-i2c

Rosa, D. L. (2017). *O que é Arduino?* Fonte: Usinainfo: https://www.usinainfo.com.br/blog/o-que-e-arduino/

Shields e incrementos para Arduino. (s.d.). Fonte: Fazedores: https://blog.fazedores.com/conheca-os-shields-e-incremente-seu-arduino-com-eles/

Silveira, C. B. (s.d.). *Oque é PWM?* Fonte: CitiSystems: https://www.citisystems.com.br/pwm/

Souza, F. (s.d.). *Arduino UNO*. Fonte: Embarcados: https://www.embarcados.com.br/arduino-uno/

Souza, F. (s.d.). *Usando as saídas PWM do Arduino*. Fonte: Embarcados: https://www.embarcados.com.br/pwm-do-arduino/

USART in Arduino Uno. (s.d.). Fonte: Electronic Wings: https://www.electronicwings.com/arduino/usart-in-arduino-uno

Xukyo. (s.d.). *Arduino and Bluetooth module HC-06*. Fonte: Aranacorp: https://www.aranacorp.com/en/arduino-and-bluetooth-module-hc-06/

APÊNDICE A – ALGORITMO DESENVOLVIDO PARA O ARDUINO

```
// Inclusão das Bibliotecas necessárias:
#include <SoftwareSerial.h>
#include <Servo.h>
#include <ArduinoJson.h>
SoftwareSerial btserial(12, 13); // Define o Serial Bluetooth nos pinos 12 e 13 como RX e TX
// Declara a pinagem dos servo motores:
    int pinleg1 = 10;
    int pinleg2 = 11;
    int pinleg3 = 6;
    int pinleg4 = 9;
    int pinfeet1 = 7;
    int pinfeet2 = 8;
    int pinfeet3 = 3;
    int pinfeet4 = 5;
StaticJsonDocument<256> jsondata; // Cria dados para armazenar as informações da Cristal em um JSON
// Declara as variáveis padrão para o programa:
String btdata = "";
String jsoninfo = "";
String response = "";
bool is_iddle = false;
// Cria as variáveis para controlar os servo motores:
    Servo leg1;
    Servo leg2;
    Servo leg3;
```

```
        Servo leg4;
        Servo feet1;
        Servo feet2;
        Servo feet3;
        Servo feet4;

void setup() { // Aplica as configurações e inicia o Arduino
  // Inicia os monitores seriais do Módulo Bluetooth e USB:
  Serial.begin(9600);
  btserial.begin(9600);
  // Envia um sinal pelo Serial Bluetooth para realizar a ação no Aplicativo:
  btserial.println("m");
  delay(100);
  btserial.println("q");
  // Cria as informações padrões das funções no documento JSON:
  jsondata["walk_mode"] = 0;
  jsondata["tilt"] = false;
  jsondata["dance"] = false;
  jsondata["hello"] = false;
  jsondata["connect"] = false;
  ResetStates();
  serializeJson(jsondata, jsoninfo); // Converte as informações do JSON para ser exibido de
maneira correta
        // Atribui cada servo motor ao seu respectivo pino do Arduino:
        leg1.attach(pinleg1);
        leg2.attach(pinleg2);
        leg3.attach(pinleg3);
```

```
    leg4.attach(pinleg4);
    feet1.attach(pinfeet1);
    feet2.attach(pinfeet2);
    feet3.attach(pinfeet3);
    feet4.attach(pinfeet4);

  Initialize(); // Executa a Função para iniciar a Cristal na posição padrão
}
void loop() { // Roda o programa principal que faz o gerenciamento do comportamento da Cristal
  while (btserial.available() > 0) { // Verifica se há texto pendente disponível no Monitor serial Bluetooth
    btdata = (char)btserial.read(); // Salva o texto pendente em uma variável para utilizar no resto do código
    //while (btserial.available() > 0) btserial.read(); // Verifica se possui textos a mais no serial e limpa caso houver
  }
  if (btdata != "") { // Verifica se o comando não está em branco e pula a etapa de definição de funções caso estiver
    ResetStates(); // Limpa todos os as funções anteriores salvas para receber comandos novos
    is_iddle = false; // Retira o estado "aguardando por comando" da Cristal
    Serial.println("Comando Recebido: \"" + btdata + "\""); // Exibe no Monitor Serial USB o comando recebido pelo Monitor Serial Bluetooth
    //Serial.println(jsoninfo); // Exibe no Monitor Serial USB o estado das variáveis JSON salvas
    // Faz a verificação dos comandos recebidos e o gerenciamento das funções habilitadas em suas variáveis:

    if (btdata == "C") {
      jsondata["connect"] = true;
```

```
    } else if (btdata == "S") {
      jsondata["iddle"] = true;
    } else if (btdata == "F") {
      if (jsondata["tilt"] == false) jsondata["walk_front"] = true;
      if (jsondata["tilt"] == true) jsondata["tilt_front"] = true;
    } else if (btdata == "B") {
      if (jsondata["tilt"] == false) jsondata["walk_back"] = true;
      if (jsondata["tilt"] == true) jsondata["tilt_back"] = true;
    } else if (btdata == "L") {
      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 0) jsondata["turn_left"] = true;
      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 1) jsondata["walk_left"] =
true;
      if (jsondata["tilt"] == true) jsondata["tilt_left"] = true;
    } else if (btdata == "R") {
      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 0) jsondata["turn_right"] =
true;
      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 1) jsondata["walk_right"] =
true;
      if (jsondata["tilt"] == true) jsondata["tilt_right"] = true;
    } else if (btdata == "G") {
      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 0) jsondata["walk_front_left"]
= true;
      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 1)
jsondata["walk_turn_front_left"] = true;
      if (jsondata["tilt"] == true) jsondata["tilt_front_left"] = true;
    } else if (btdata == "I") {
      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 0)
jsondata["walk_front_right"] = true;
      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 1)
jsondata["walk_turn_front_right"] = true;
```

```
      if (jsondata["tilt"] == true) jsondata["tilt_front_right"] = true;

    } else if (btdata == "H") {

      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 0) jsondata["walk_back_left"]
= true;

      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 1)
jsondata["walk_turn_back_left"] = true;

      if (jsondata["tilt"] == true) jsondata["tilt_back_left"] = true;

    } else if (btdata == "J") {

      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 0)
jsondata["walk_back_right"] = true;

      if (jsondata["tilt"] == false && jsondata["walk_mode"] == 1)
jsondata["walk_turn_back_right"] = true;

      if (jsondata["tilt"] == true) jsondata["tilt_back_right"] = true;

    } else if (btdata == "w") {

      jsondata["walk_mode"] = 0;

      response = "q";

    } else if (btdata == "W") {

      jsondata["walk_mode"] = 1;

      response = "Q";

    } else if (btdata == "u") {

      response = "m";

      jsondata["tilt"] = false;

    } else if (btdata == "U") {

      jsondata["tilt"] = true;

      response = "M";

    } else if (btdata == "X") {

      jsondata["dance"] = true;

    } else if (btdata == "V") {

      jsondata["hello"] = true;
```

```
    }


    btdata = ""; // Limpa a variável para poder armazenar um novo valor assim que receber.
  }
  if (is_iddle == false && response != "") {
    is_iddle = true;
    //Serial.println("Resposta: " + response); // Envia um sinal no Serial USB apenas para fins de teste e monitoramento
    btserial.println(response); // Envia um sinal pelo Serial Bluetooth para realizar a ação no Aplicativo
    response = "";
  }
  // Verifica as variáveis salvas e executa a(s) funções habilitadas:
  if (jsondata["connect"]) {
    ConnectResponse();
  }else if (jsondata["iddle"]) {
    Iddle();
  } else if (jsondata["turn_left"]) {
    TurnLeft();
  } else if (jsondata["turn_right"]) {
    TurnRight();
  } else if (jsondata["walk_front"]) {
    WalkFront();
  } else if (jsondata["walk_back"]) {
    WalkBack();
  } else if (jsondata["walk_left"]) {
    WalkLeft();
  } else if (jsondata["walk_right"]) {
```

```
    WalkRight();
  } else if (jsondata["tilt_front"]) {
    TiltFront();
  } else if (jsondata["tilt_back"]) {
    TiltBack();
  } else if (jsondata["tilt_left"]) {
    TiltLeft();
  } else if (jsondata["tilt_right"]) {
    TiltRight();
  } else if (jsondata["tilt_front_left"]) {
    TiltFrontLeft();
  } else if (jsondata["tilt_front_right"]) {
    TiltFrontRight();
  } else if (jsondata["tilt_back_left"]) {
    TiltBackLeft();
  } else if (jsondata["tilt_back_right"]) {
    TiltBackRight();
  } else if (jsondata["hello"]) {
    Hello();
  } else if (jsondata["dance"]) {
    Dance();
  }
}
// Função para mover todos os Eixos (Pernas e Joelhos) da Cristal ao mesmo tempo:
void MoveAll(int leg1val, int leg2val, int leg3val, int leg4val, int feet1val, int feet2val, int feet3val, int feet4val) {
  if (leg1val != -1) leg1.write(leg1val);
```

```
  if (leg2val != -1) leg2.write(leg2val);
  if (leg3val != -1) leg3.write(leg3val);
  if (leg4val != -1) leg4.write(leg4val);
  if (feet1val != -1) feet1.write(feet1val);
  if (feet2val != -1) feet2.write(feet2val);
  if (feet3val != -1) feet3.write(feet3val);
  if (feet4val != -1) feet4.write(feet4val);
}
// Função para mover todos as Pernas da Cristal ao mesmo tempo:
void MoveLegs(int leg1val, int leg2val, int leg3val, int leg4val) {
  if (leg1val != -1) leg1.write(leg1val);
  if (leg2val != -1) leg2.write(leg2val);
  if (leg3val != -1) leg3.write(leg3val);
  if (leg4val != -1) leg4.write(leg4val);
}
// Função para mover todos os Joelhos da Cristal ao mesmo tempo:
void MoveFeets(int feet1val, int feet2val, int feet3val, int feet4val) {
  if (feet1val != -1) feet1.write(feet1val);
  if (feet2val != -1) feet2.write(feet2val);
  if (feet3val != -1) feet3.write(feet3val);
  if (feet4val != -1) feet4.write(feet4val);
}
// Função executada ao inicializar o arduino, colocando a Cristal em sua posição inicial:
void Initialize() {
  MoveLegs(90, 90, 90, 90);
  MoveFeets(70, 70, 70, 70);
}
```

```
// Função executada para enviar um sinal pelo Serial Bluetooth e resetar o estado dos ícones no Aplicativo:
void ConnectResponse() {
  jsondata["walk_mode"] = 0;
  jsondata["tilt"] = false;
  jsondata["connect"] = false;
  btserial.println("m");
  delay(100);
  btserial.println("q");
}
// Função executada sempre que um novo comando é recebido para limpar as variáveis salvas anteriormente:
void ResetStates() {
  jsondata["iddle"] = false;
  jsondata["turn_left"] = false;
  jsondata["turn_right"] = false;

  jsondata["walk_front"] = false;
  jsondata["walk_back"] = false;
  jsondata["walk_left"] = false;
  jsondata["walk_right"] = false;

  jsondata["walk_front_left"] = false;
  jsondata["walk_front_right"] = false;
  jsondata["walk_back_left"] = false;
  jsondata["walk_back_right"] = false;

  jsondata["walk_turn_front_left"] = false;
```

```
  jsondata["walk_turn_front_right"] = false;
  jsondata["walk_turn_back_left"] = false;
  jsondata["walk_turn_back_right"] = false;

  jsondata["tilt_front"] = false;
  jsondata["tilt_back"] = false;
  jsondata["tilt_left"] = false;
  jsondata["tilt_right"] = false;

  jsondata["tilt_front_left"] = false;
  jsondata["tilt_front_right"] = false;
  jsondata["tilt_back_left"] = false;
  jsondata["tilt_back_right"] = false;
}
// Sequência executada para a Cristal ficar parada:
void Iddle() {
  MoveAll(90, 90, 90, 90, 70, 70, 70, 70);
  jsondata["iddle"] = false;
}
// Sequência executada para dar um passo para Frente:
void WalkFront() {
  MoveLegs(45, 135, 135, 45);
  delay(50);
  feet3.write(30); delay(50);
  leg3.write(45);  delay(50);
  feet3.write(70);  delay(50);
  feet4.write(30);  delay(50);
```

```
  leg4.write(135);  delay(50);
  feet4.write(70);  delay(50);
  feet2.write(30);  delay(50);
  leg2.write(90);  delay(50);
  feet2.write(70);  delay(50);
  feet1.write(30);  delay(50);
  leg1.write(90);  delay(50);
  feet1.write(70);  delay(50);
}
// Sequência executada para dar um passo para trás:
void WalkBack() {
  MoveLegs(135, 45, 45, 135);
  delay(50);
  feet1.write(30);  delay(50);
  leg1.write(45);  delay(50);
  feet1.write(70);  delay(50);
  feet2.write(30);  delay(50);
  leg2.write(135);  delay(50);
  feet2.write(70);  delay(50);
  feet4.write(30);  delay(50);
  leg4.write(90);  delay(50);
  feet4.write(70);  delay(50);
  feet3.write(30);  delay(50);
  leg3.write(90);  delay(50);
  feet3.write(70);  delay(50);
}
// Sequência executada para dar um passo para Esquerda:
```

```
void WalkLeft() {
  MoveLegs(45, 45, 135, 135);
  delay(50);
  feet4.write(30);  delay(50);
  leg4.write(45);  delay(50);
  feet4.write(70);  delay(50);
  feet1.write(30);  delay(50);
  leg1.write(135);  delay(50);
  feet1.write(70);  delay(50);
  feet3.write(30);  delay(50);
  leg3.write(90);  delay(50);
  feet3.write(70);  delay(50);
  feet2.write(30);  delay(50);
  leg2.write(90);  delay(50);
  feet2.write(70);  delay(50);
}
// Sequência executada para dar um passo para Direita:
void WalkRight() {
  MoveLegs(135, 135, 45, 45);
  delay(50);
  feet2.write(30);  delay(50);
  leg2.write(45);  delay(50);
  feet2.write(70);  delay(50);
  feet3.write(30);  delay(50);
  leg3.write(135);  delay(50);
  feet3.write(70);  delay(50);
  feet1.write(30);  delay(50);
```

```
  leg1.write(90);  delay(50);
  feet1.write(70);  delay(50);
  feet4.write(30);  delay(50);
  leg4.write(90);  delay(50);
  feet4.write(70);  delay(50);
}
// Sequência executada para Inclinar para Frente:
void TiltFront() {
  MoveFeets(0, 0, 90, 90);
}
// Sequência executada para Inclinar para Trás:
void TiltBack() {
  MoveFeets(90, 90, 0, 0);
}
// Sequência executada para Inclinar para Esquerda:
void TiltLeft() {
  MoveFeets(90, 0, 0, 90);
}
// Sequência executada para Inclinar para Direita:
void TiltRight() {
  MoveFeets(0, 90, 90, 0);
}
// Sequência executada para Inclinar para Diagonal Frente-Esquerda:
void TiltFrontLeft() {
  MoveFeets(45, 0, 45, 90);
}
// Sequência executada para Inclinar para Diagonal Frente-Direita:
```

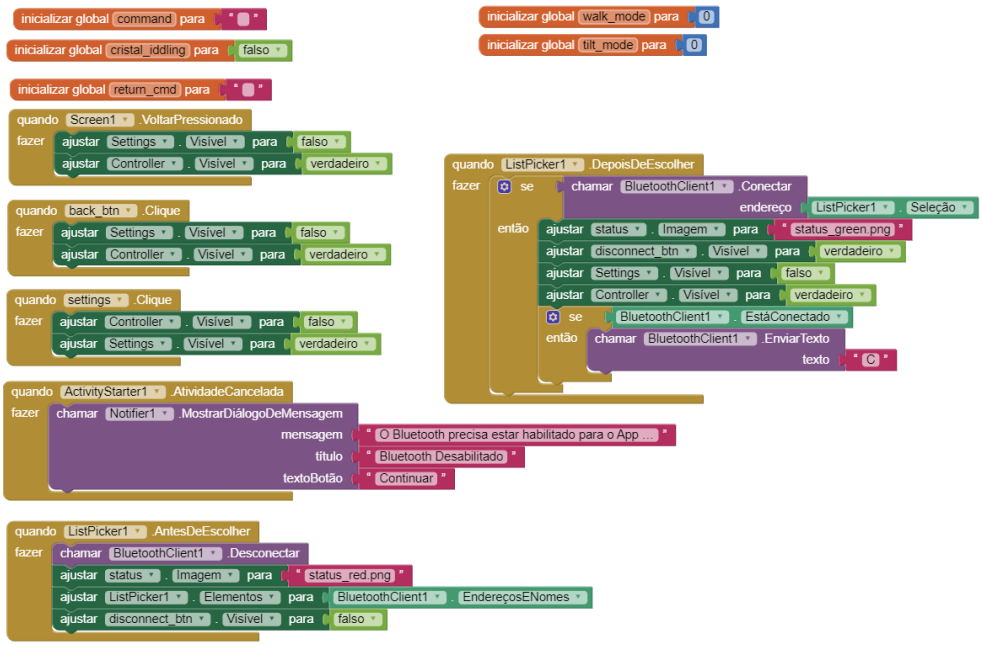
```
void TiltFrontRight() {
  MoveFeets(0, 45, 90, 45);
}
// Sequência executada para Inclinar para Diagonal Trás-Esquerda:
void TiltBackLeft() {
  MoveFeets(90, 45, 0, 45);
}
// Sequência executada para Inclinar para Diagonal Trás-Direita:
void TiltBackRight() {
  MoveFeets(45, 90, 45, 0);
}
// Sequência executada para girar em seu próprio Eixo para a Esquerda:
void TurnLeft() {
  feet1.write(0);  delay(50);
  leg1.write(135);  delay(50);
  feet1.write(70);  delay(100);
  feet2.write(0);  delay(50);
  leg2.write(135);  delay(50);
  feet2.write(70);  delay(100);
  feet3.write(0);  delay(50);
  leg3.write(135);  delay(50);
  feet3.write(70);  delay(100);
  feet4.write(0);  delay(50);
  leg4.write(135);  delay(50);
  feet4.write(70);  delay(100);
  MoveLegs(90, 90, 90, 90);
  delay(100);
```

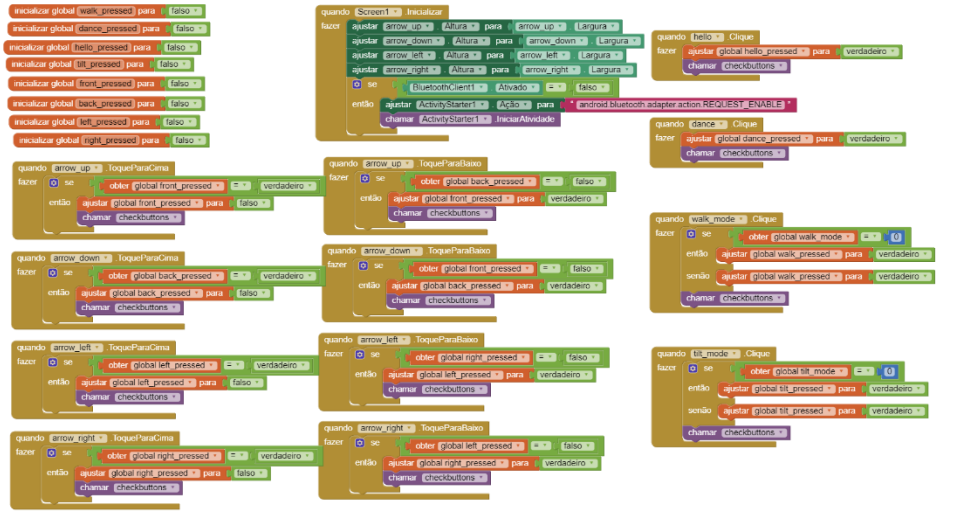
```
}
// Sequência para girar em seu próprio Eixo para a Direita:
void TurnRight() {
  feet2.write(0);  delay(50);
  leg2.write(45);  delay(50);
  feet2.write(70);  delay(100);
  feet1.write(0);  delay(50);
  leg1.write(45);  delay(50);
  feet1.write(70);  delay(100);
  feet4.write(0);  delay(50);
  leg4.write(45);  delay(50);
  feet4.write(70);  delay(100);
  feet3.write(0);  delay(50);
  leg3.write(45);  delay(50);
  feet3.write(70);  delay(100);
  MoveLegs(90, 90, 90, 90);
  delay(100);
}
// Sequência executada para realizar movimentos de dança:
void Dance() {
  MoveFeets(90, 90, 90, 90);
  MoveLegs(0, 0, 0, 0);  delay(250);
  MoveLegs(180, 180, 180, 180);  delay(250);
  MoveLegs(0, 0, 0, 0);  delay(250);
  MoveLegs(180, 180, 180, 180);  delay(250);
  MoveLegs(135, 45, 135, 45);  delay(250);
  MoveFeets(0, 0, -1, -1);  delay(250);
```

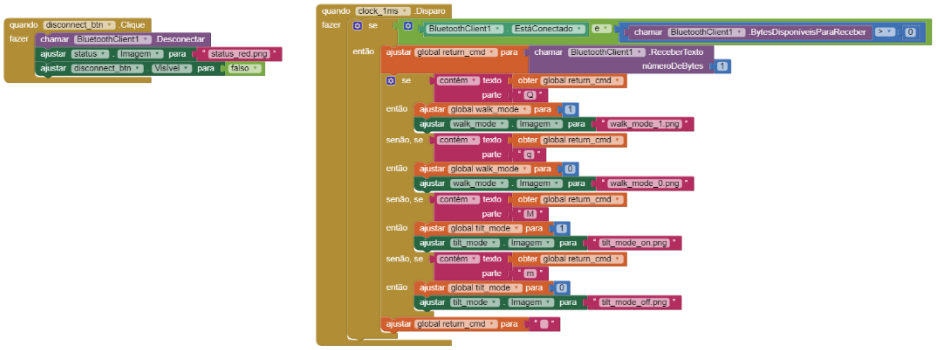
```
MoveFeets(90, 90, -1, -1);  delay(250);
MoveFeets(0, 0, -1, -1);  delay(250);
MoveFeets(90, 90, -1, -1);  delay(250);
MoveFeets(-1, -1, 90, 90);  delay(250);
MoveFeets(-1, -1, 0, 0);  delay(250);
MoveFeets(-1, -1, 90, 90);  delay(250);
MoveFeets(0, 0, 0, 0);  delay(250);
MoveFeets(90, 90, 90, 90);  delay(250);
MoveFeets(0, 0, 0, 0);  delay(250);
MoveFeets(90, 90, 90, 90);  delay(250);
MoveLegs(45, 135, 45, 135);  delay(250);
MoveFeets(-1, 0, 0, -1);  delay(250);
MoveFeets(-1, 90, 90, -1);  delay(250);
MoveFeets(-1, 0, 0, -1);  delay(250);
MoveFeets(-1, 90, 90, -1);  delay(250);
MoveFeets(0, -1, -1, 0);  delay(250);
MoveFeets(90, -1, -1, 90);  delay(250);
MoveFeets(0, -1, -1, 0);  delay(250);
MoveFeets(90, -1, -1, 90);  delay(250);
MoveFeets(0, 0, 0, 0);  delay(250);
MoveFeets(90, 90, 90, 90);  delay(250);
MoveFeets(0, 0, 0, 0);  delay(250);
MoveFeets(90, 90, 90, 90);  delay(250);
MoveFeets(0, 0, 0, 0);  delay(250);
MoveAll(90, 90, 90, 90, 70, 70, 70, 70);
delay(250);
jsondata["dance"] = false;
```

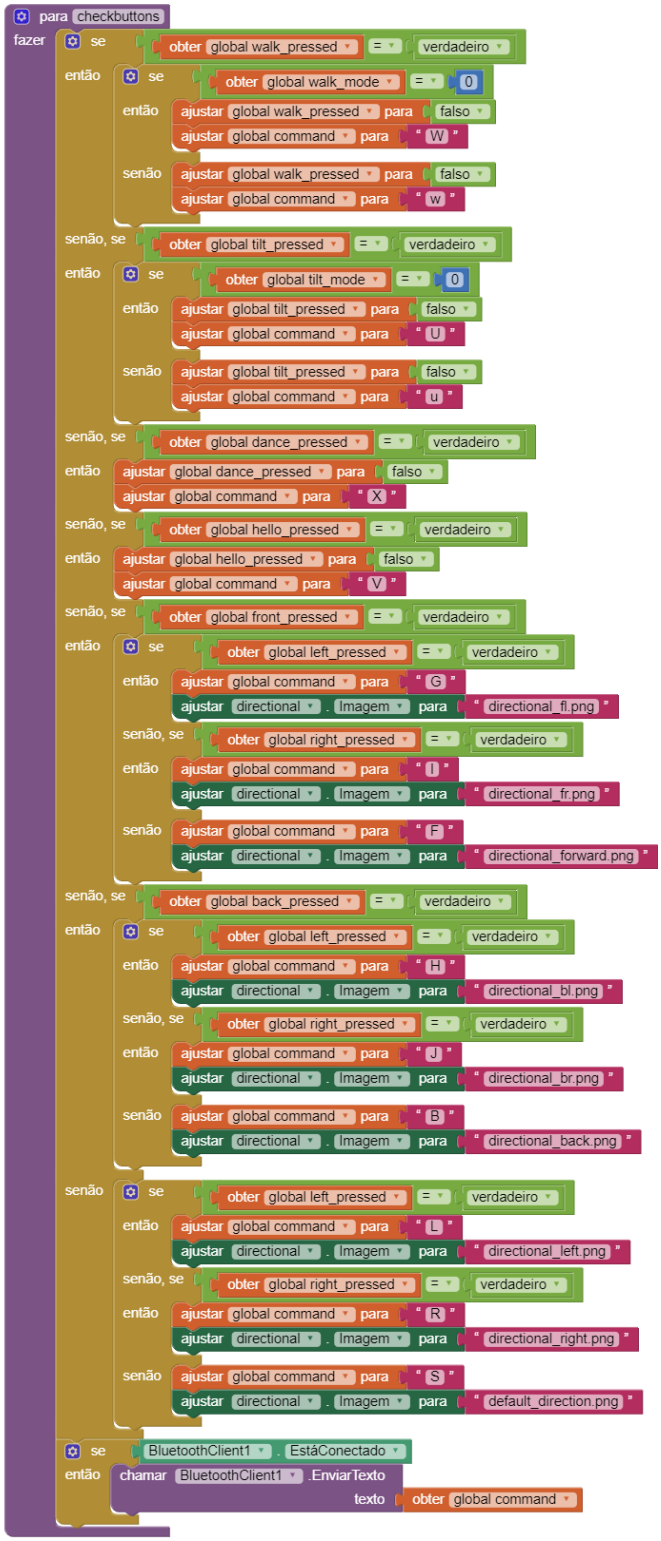
```
}
// Sequência executada para realizar o movimento de acenar:
void Hello() {
  MoveFeets(90, 90, 0, 0);
  delay(100);
  feet1.write(0);  delay(250);
  leg1.write(150);  delay(100);
  leg1.write(120);  delay(100);
  leg1.write(150);  delay(100);
  leg1.write(120);  delay(150);
  leg1.write(90);
  feet1.write(70);  delay(100);
  MoveFeets(70, 70, 70, 70);
  jsondata["hello"] = false;
}
```

APÊNDICE B – ALGORITMO DESENVOLVIDO PARA O APLICATIVO








para checkbuttons
fazer
se obter global walk_pressed = verdadeiro
então se obter global walk_mode = 0
então ajustar global walk_pressed para falso
ajustar global command para "W"
senão ajustar global walk_pressed para falso
ajustar global command para "w"
senão, se obter global tilt_pressed = verdadeiro
então se obter global tilt_mode = 0
então ajustar global tilt_pressed para falso
ajustar global command para "U"
senão ajustar global tilt_pressed para falso
ajustar global command para "u"
senão, se obter global dance_pressed = verdadeiro
então ajustar global dance_pressed para falso
ajustar global command para "X"
senão, se obter global hello_pressed = verdadeiro
então ajustar global hello_pressed para falso
ajustar global command para "V"
senão, se obter global front_pressed = verdadeiro
então se obter global left_pressed = verdadeiro
então ajustar global command para "G"
ajustar directional . Imagem para "directional_fl.png"
senão, se obter global right_pressed = verdadeiro
então ajustar global command para "I"
ajustar directional . Imagem para "directional_fr.png"
senão ajustar global command para "F"
ajustar directional . Imagem para "directional_forward.png"
senão, se obter global back_pressed = verdadeiro
então se obter global left_pressed = verdadeiro
então ajustar global command para "H"
ajustar directional . Imagem para "directional_bl.png"
senão, se obter global right_pressed = verdadeiro
então ajustar global command para "J"
ajustar directional . Imagem para "directional_br.png"
senão ajustar global command para "B"
ajustar directional . Imagem para "directional_back.png"
senão se obter global left_pressed = verdadeiro
então ajustar global command para "L"
ajustar directional . Imagem para "directional_left.png"
senão, se obter global right_pressed = verdadeiro
então ajustar global command para "R"
ajustar directional . Imagem para "directional_right.png"
senão ajustar global command para "S"
ajustar directional . Imagem para "default_direction.png"
se BluetoothClient1 . EstáConectado
então chamar BluetoothClient1 .EnviarTexto
texto obter global command