UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

“Implementação de uma Plataforma Robótica controlada remotamente utilizando o Arduino”

ANDRÉ CREPALDI GEIGER SMIDT

São Carlos

Junho/2013

ANDRÉ CREPALDI GEIGER SMIDT

IMPLEMENTAÇÃO DE UMA PLATAFORMA ROBÓTICA CONTROLADA REMOTAMENTE UTILIZANDO O ARDUINO

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia de Computação

Orientador: Maximilliam Luppe

São Carlos
2013

Smidt, Andre Crepaldi Geiger
S642i Implementação de uma plataforma robótica controlada remotamente utilizando o Arduino / Andre Crepaldi Geiger Smidt; orientador Maximiliam Luppe. São Carlos, 2013.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2013.

1. Arduino. 2. ATMega328. 3. robótica. I. Título.

FOLHA DE APROVAÇÃO

Nome: André Crepaldi Geiger Smidt

Título: "Implementação de uma plataforma robótica controlada remotamente utilizando o microcontrolador ATMega328"

Trabalho de Conclusão de Curso defendido em 20/06/2013.

Comissão Julgadora:	**Resultado:**
Prof. Dr. Maximiliam Luppe - Orientador SEL/EESC/USP	Aprovado
Prof. Dr. Marcelo Andrade da Costa Vieira SEL/EESC/USP	APROVADO
Prof. Dr. Valdir Grassi Júnior SEL/EESC/USP	APROVADO

Coordenador pela EESC/USP do Curso de Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

Sumário

Resumo

Com a popularização dos microcontroladores e do *open hardware* (“*hardware* livre”), a construção de unidades robóticas tornou-se economicamente viável. A construção de um robô envolve o uso de conceitos relacionados com *hardware*, *software* e mecânica, que necessitam estarem integrados através de um microcontrolador, para coordenar os periféricos do robô. Neste trabalho, propomos o projeto e a montagem de um robô móvel de pequeno porte controlado remotamente, utilizando a plataforma Arduino e o microcontrolador ATMega328.

Palavras-chaves: Arduino, ATMega328, robótica.

Abstract

Along with the popularization of microcontrollers and open *hardware*, the construction of robotic units has become economically viable. The construction of a robot involves the use of concepts related to *hardware*, *software* and mechanics that need to be integrated trough a microcontroller to coordinate the peripherals of the robot. In this work we propose the project and the assembly of a remote controlled small scale mobile robot using the Arduino platform and the ATMega328 microcontroller.

Keywords: Arduino, ATMega328, robotic.

Capítulo 1: Introdução

A popularização dos microcontroladores, junto com a queda dos custos de componentes eletrônicos, baterias e motores, possibilitaram o desenvolvimento de pequenas plataformas robóticas de custo acessível.

A implementação de uma plataforma robótica envolve conceitos de eletrônica, computação e mecânica que, integrados de forma sincronizada, possibilitam a construção de um sistema embarcado robótico que percebe o ambiente através de seus sensores e interage com o meio através de seus atuadores.

Para realizar essa integração entre sensores e atuadores é utilizado um microcontrolador para gerenciar o funcionamento de todos os dispositivos ligados a ele, e garantir a sincronia necessária para o funcionamento de todos os elementos dentro de, por exemplo, uma plataforma robótica.

Existem diversas plataformas robóticas e outros tipos de projetos, utilizando microcontroladores, que podem ser vistos em sites como [1], [2], [3] e no livro [4]. Essas literaturas mostram muitas das diversas possibilidades de plataformas robóticas, e projetos, que podem ser construídas.

Nas Figuras 1 e 2, temos alguns exemplos de plataformas robóticas construídas utilizando como cérebro da plataforma um microcontrolador. A Figura 1 mostra uma plataforma robótica capaz de andar pelo ambiente, de forma autônoma, evitando colisões frontais utilizando um sonar fixo para realizar essa tarefa.

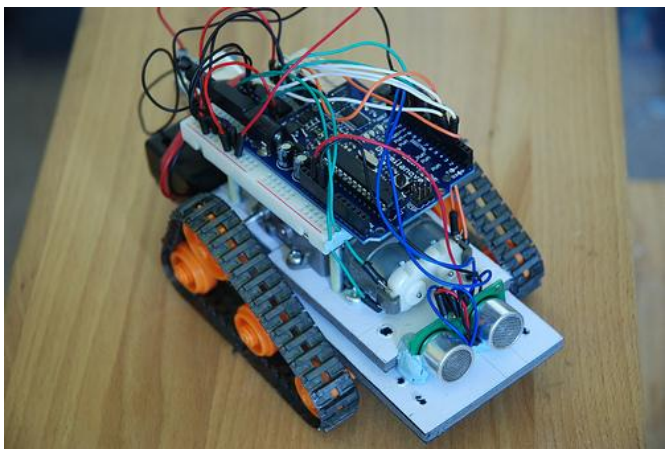


Figura 1 - Exemplo de plataforma robótica [3].

Na Figura 2, podemos ver uma plataforma desenvolvida para andar autonomamente pelo ambiente, utilizando uma heurística baseada em sensores de toque na sua parte frontal, possibilitando à mesma detectar quando uma colisão ocorreu e assim mudar sua direção.

Figura 2 - Exemplo de plataforma robótica [2].

Esses projetos ilustram uma pequena parte da gama de possibilidades existentes para as possibilidades e funcionalidades de uma plataforma robótica. Podemos construir robôs autônomos, remotamente controlados, com sensores de distância, temperatura, utilizando motores e servos entre outras milhares de opções, utilizando um microcontrolador para integrar e coordenadas todos os periféricos ligados a plataforma.

A construção de uma plataforma robótica envolve a integração de diversos sistemas eletrônicos e mecânicos, coordenados via *software*, realizar esse processo de integração é parte do papel que o engenheiro de computação pode desenvolver ao longo de sua carreira, e demostra uma área de perfeita atuação desse profissional no mercado crescente da robótica.

Esse processo de integração pode ser visto em literaturas como [5], [6] e [7] onde são demonstradas outras possibilidades de plataformas robóticas, diferentes das apresentadas acima, nesses artigos são apresentados alguns dos conhecimentos necessários para a construção de uma unidade robótica de médio porte, bem como a integração necessária para conectar todos os periféricos do robô ao seu microcontrolador e sistemas de controle implementados.

Assim este trabalho de conclusão de curso apresenta o desenvolvimento de uma plataforma robótica utilizando o microcontrolador ATMega328, através do Arduino [1]. A plataforma apresentada é composta de dois módulos, o robô e a plataforma de controle, que após o desenvolvimento ficaram como mostrada na Figura 3 e Figura 4 respectivamente.

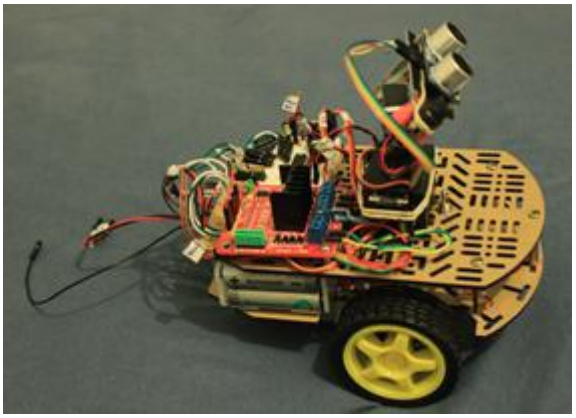


Figura 3 – Robô.

Figura 4 – Plataforma de Controle

A escolha desse microcontrolador se deve principalmente às seguintes características: baixo custo, facilidade de programação, confiabilidade e extensa gama de informações sobre seu uso no próprio fórum da comunidade Arduino [1].

A disponibilidade de diversas bibliotecas com várias funções já implementadas para esse microcontrolador facilita o desenvolvimento de projetos maiores e com maior rapidez.

Optou-se por construir uma plataforma remotamente controlada e utilizando sensores como o acelerômetro e o sonar. O módulo sonar está ajustado dentro de uma estrutura em alumínio que possibilita uma movimentação do tipo *pan & tilt*[1], o que permite apontar o dispositivo para qualquer direção.

[1] Estrutura em alumínio servo controlada com dois graus de liberdade, ver Figura 24.

O *link* de comunicação que permite controlar a plataforma remotamente pode ser de dois tipos, utilizando radio frequência, RF, ou Bluetooth, possibilitando usar tecnologias de comunicação diferentes para realizar o controle da plataforma.

Neste documento, iremos apresentar todos os detalhes envolvidos na construção do robô, desde o microcontrolador escolhido até as ações implementadas. O capítulo 2 descreve os materiais e a metodologia utilizada nesse projeto mostrando todos os conceitos por trás da criação do robô. O capítulo 3 descreve a implementação dos *softwares* de controle, no capítulo 4 é apresentado os resultados obtidos, no capítulo 5 é proposto possíveis trabalhos futuros e por fim o capítulo 6 traz as conclusões acerca do desenvolvimento. Os anexos I, II, III, IV contém respectivamente o esquemático do robô, o esquema da plataforma de controle, o código fonte do robô e o código fonte da plataforma de controle.

Capítulo 2: Metodologia

2.1 Materiais e métodos

A plataforma controlável é formada por um carro robótico de dois andares com dois motores DC e duas caixas de redução, um Arduino Uno com ATMega328, uma ponte H que utiliza o CI L298N, um módulo sonar HC-SR04, dois servos-motores, um módulo de comunicação RF (radiofrequência), um módulo de comunicação Bluetooth e as baterias necessárias.

A plataforma de controle é formada por um Arduino Uno com ATMega328, um controle do videogame Nintendo Wii, um controle do videogame Sony Playstation, um módulo de comunicação RF e as baterias necessárias.

A seguir são apresentadas e descritas com mais detalhes as funcionalidades de cada componente citado anteriormente e utilizado na construção da plataforma robótica.

2.2. Arduino

Arduino [1] é uma plataforma de prototipação eletrônica *open-source* flexível que utiliza o microcontrolador ATMega328. O Arduino pode receber sinais de vários sensores eletrônicos e lidar com essas informações para controlar motores, luzes, servos e qualquer outro atuador. O modelo utilizado no robô é o Arduino UNO (figura 5).

Figura 5 – Arduino

Na tabela 1 estão os dados técnicos do Arduino fornecidos pelo fabricante.

Tensão de operação	5 V
Tensão de entrada (recomendada)	7 – 12 V
Tensão de entrada (limites)	6 – 20 V
Pinos digitais	14 (6 com saída PWM)
Pinos analógicos	6
Clock	16 MHz

Tabela 1 – Dados técnicos do Arduino [1].

O microcontrolador utilizado no Arduino é o ATMega328 da Atmel, que utiliza a arquitetura de Harvard e é de 8 bits, RISC[2], com 32KB de memória *flash*, 1KB de EEPROM[3], 2KB de SRAM[4], 32 registradores de uso geral, 3 temporizadores/contadores, uma USART[5], portas para comunicação SPI[6], 6 conversores AD[7] de 10bits e um *watchdog timer*[8], entre outras características. A tensão de operação dele é entre 1,8 e 5,5 V. [8]

O *software* que controla os pinos e as ações do Arduino pode ser desenvolvido no programa chamado Arduino IDE (disponibilizado para *download* em [1]), que conta com uma interface gráfica simples e intuitiva, e aceita códigos em C/C++.

O código a ser embarcado deve estar no formato ".ino", e contar com as seguintes chamadas de procedimento: *setup()*, que ajusta as configurações das portas de entrada e saída e a comunicação serial, e *loop()*, que é um laço infinito onde o usuário cria a rotina do seu programa.

[2] RISC: "***R**educed **I**nstruction **S**et **C**omputer"* ou "Computador com conjunto reduzido de instruções"
[3] EEPROM: "***E**lectrically-**E**rasable **P**rogrammable **R**ead-**O**nly **M**emory"* é um tipo de memória não volátil e que pode ser reescrita um número limite de vezes.
[4] SRAM: "***S**tatic **R**andom **A**ccess **M**emory"* é um tipo de memória volátil de acesso randômico.
[5] USART: "**U**niversal **S**ynchronous/**A**synchronous **R**eceiver/**T**ransmitter" é utilizado para realizar a comunicação serial com o microcontrolador.
[6] SPI: "**S**erial **P**eripheral **I**nterface**"** é utilizado para implementar comunicação serial com o microcontrolador.
[7] AD: Analógico Digital
[8] Utilizado para forçar um reset no microcontrolar, reiniciando seu programa.

2.3. Plataforma Robótica

2.3.1. Robô

O Robô foi construído utilizando o microcontrolador ATMega328, quem coordena todos os eventos que ocorrem com o ele. Na Figura 6 temos a estrutura desenvolvida em torno do Arduino.

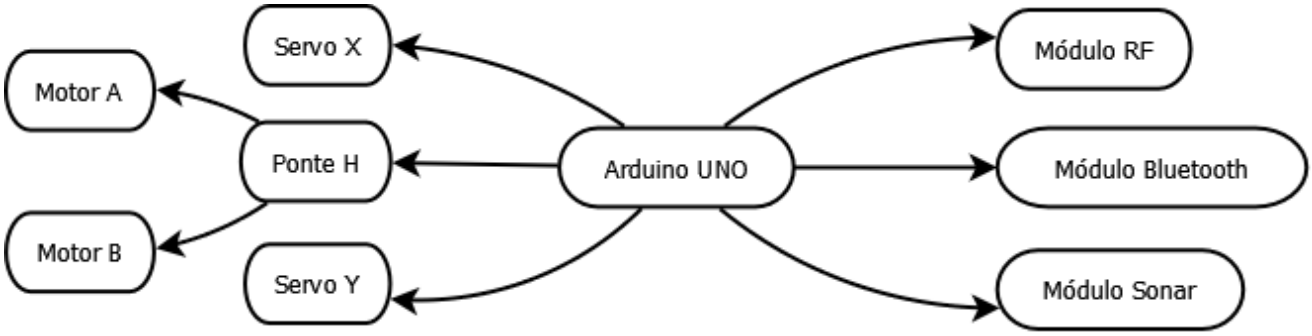


Figura 6 – Organização do *hardware* da plataforma móvel

Na Figura 7 é mostrada a localização dos componentes dentro do carro do robô.

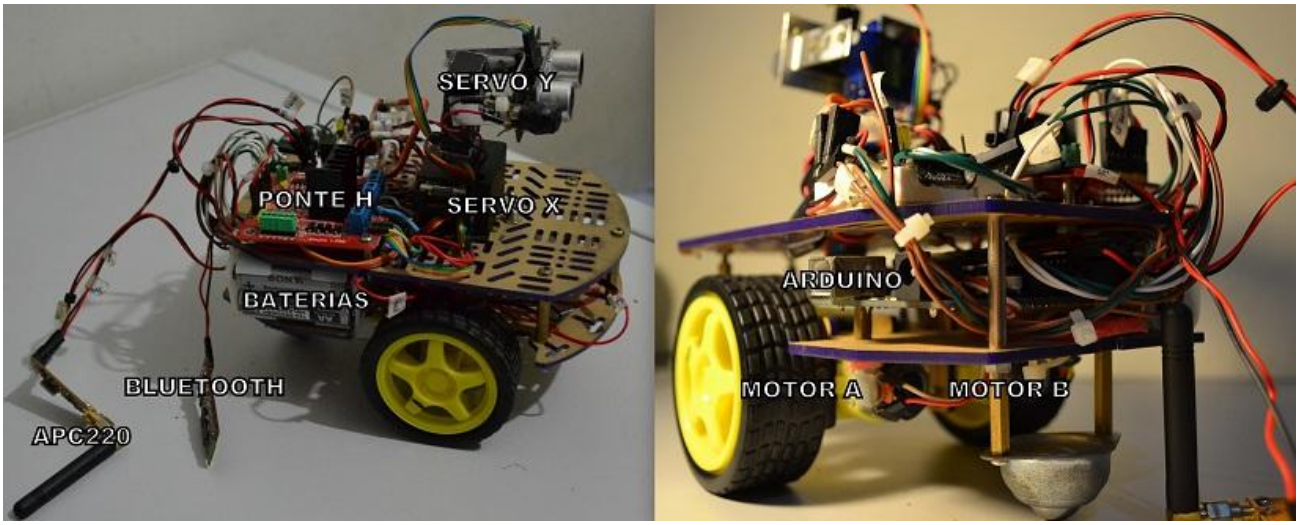


Figura 7: Robô e a localização de seus componentes.

O cérebro da plataforma criada é o microcontrolador ATMega328 que coordena e gerencia todos os periféricos ligados a ele, possibilitando o robô de realizar as seguintes funções:

- Andar para frente/trás;
- Girar no eixo para direita/esquerda;
- Controle com dois graus de liberdade do sonar;
- Evitar colisões quando algo está na frente do sonar;
- Comunicar-se via Radio Frequência ou via Bluetooth;

O código estruturado para esse robô, em sua essência, é um *loop* infinito onde é realizada uma leitura dos sensores (ver seção 2.4) e dos periféricos de comunicação (ver seção 2.5), e, com base nessas informações, são acionados seus atuadores (ver seção 2.6).

2.3.2. Plataforma de Controle

A plataforma de controle possui a função de enviar os comandos desejados para o robô. Os comandos são gerados a partir do controle de Playstation ou um controle de Nintendo Wii. A figura 8 apresenta os elementos da plataforma de controle.

O controle de Playstation foi escolhido por apresentar diversos botões e um par de *joysticks*, Figura 26, possibilitando uma grande combinação de comandos, para realizar as diversas operações do robô.

A escolha de se utilizar um controle de Nintendo Wii, Figura 26, se deve ao fato de o mesmo apresentar um pequeno acelerômetro digital, o que permite utilizar esse dispositivo para controlar o robô utilizando o movimento das mãos.

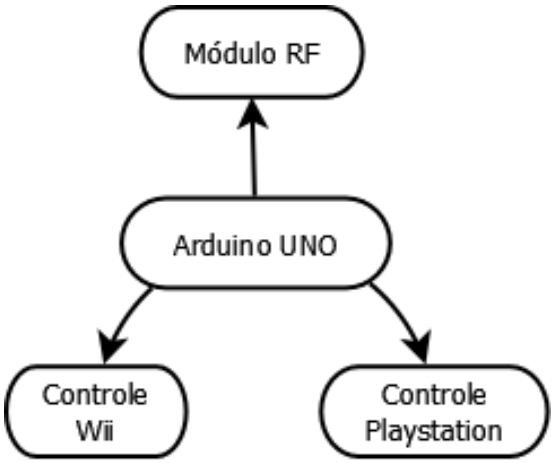


Figura 8 – Organização da plataforma de controle.

Na Figura 9, é indicada a localização dos elementos que compõem essa parte do projeto. O módulo RF indicando na Figura 8 é o dispositivo identificado como APC220 na Figura 9.

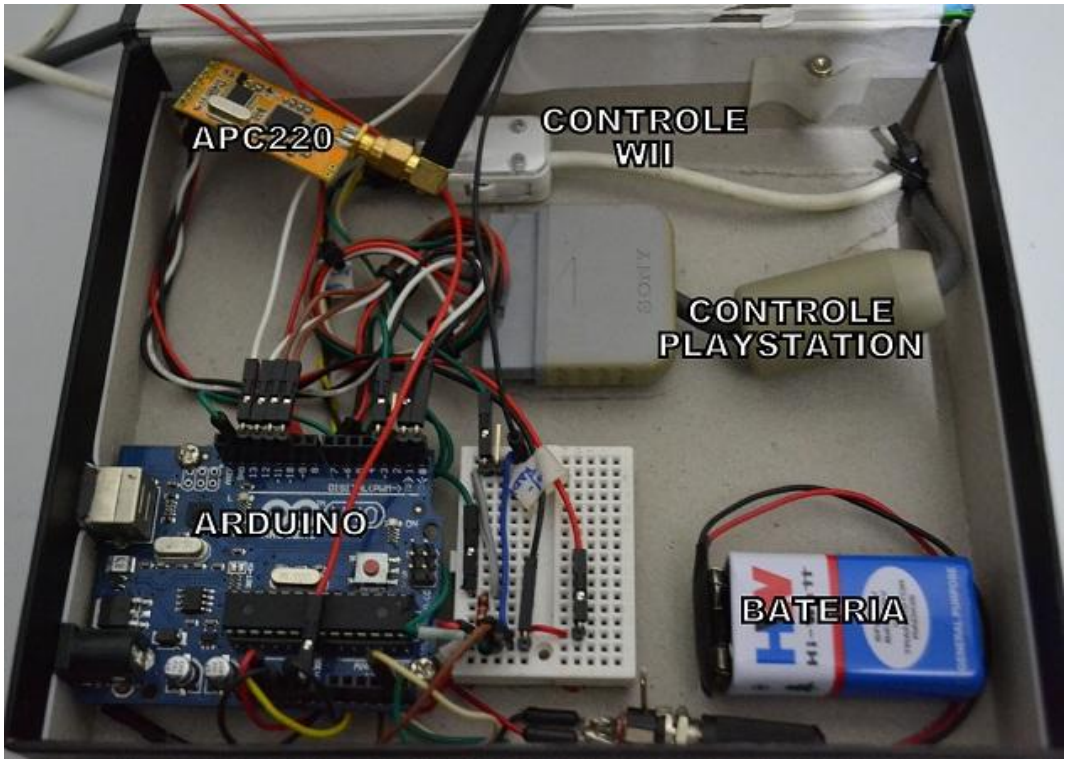


Figura 9 – Plataforma de controle e seus componentes

Assim como no robô, o cérebro da estação base é um Arduino executando um código onde se realiza uma leitura periódica dos controles conectados e obtém-se o código do botão pressionado ou a leitura do acelerômetro presente no controle do Nintendo Wii. Essa informação é codificada e enviada ao robô, que executa os comandos correspondentes.

2.4. Sensores

Sensores são dispositivos que possibilitam coletar informações a cerca de algum fenômeno, transformando elas em dados que possam ser processados por um microcontrolador.

2.4.1. Sensores no Robô

A plataforma criada conta com um sonar, o módulo HC-SR04 [9]. Com esse dispositivo é possível medir a distância de um objeto que se encontra em sua frente. Um

pulso ultrassônico é emitido pelo dispositivo, se reflete sobre o objeto e retorna para o sonar e, com base no tempo decorrido entre a emissão e a recepção do pulso, é possível determinar a distância do objeto.

O pulso é emitido de forma automática pelo sonar a uma frequência de 40 KHz e, quando ele é recebido de volta, o tempo passado durante esse processo é utilizado para resolver a seguinte equação[9]:

$$\Delta S = \frac{HLT * vSom}{2}$$

Na Figura 10 temos o módulo HC-SR04. Suas características operacionais estão descritas na tabela 2.

Figura 10 – Módulo HC-SR04 [9].

Voltagem de Operação	DC 5 V	Alcance Máximo	4 m
Corrente de Operação	15 mA	Alcance Mínimo	2 cm
Frequência de Operação	40 Hz	Ângulo de Medição	15°
Dimensão	45 x 20 x 15 mm		

Tabela 2 – Características Operacionais do módulo HC-SR04 [9].

O sonar é utilizado na plataforma desenvolvida para evitar que o robô colida com objetos em sua frente. O mesmo se encontra estruturado sobre uma estrutura de alumínio presa a dois servos, formando um sistema do tipo *pan & tilt* (ver seção 2.6.2), possibilitando que o mesmo seja apontado em diversas posições.

[9] HLT significa "*High-Level Time"*, tempo em que o pino ECHO está no nível lógico um. O fabricante considera a velocidade do som constante em 340 m/s.

2.4.2. Sensores na plataforma de controle

2.4.2.1. Controle do Playstation

O controle do Playstation é o principal controle da plataforma criada. Seus botões e *joysticks* são usados para enviar comandos ao robô.

A cada intervalo de 70 ms, o Arduino da estação de controle realiza a leitura do *status* dos botões do controle e, com base nesses dados e na tabela 8 (ver seção 2.7.1) é determinado o comando enviado para robô.

A empresa Sony ao criar esse controle para a sua plataforma de videogames utilizou um protocolo de comunicação próprio, criado para aquele console, assim, para realiza a comunicação entre o Arduino e o controle e necessário utilizar uma biblioteca denominada "*PS2X_lib_v1.0",* presente em [10], onde é implementado os protocolos de comunicação para conseguir ler os dados do controle.

Os pinos do conector do controle são mostrados na figura 11 e descritos em seguida.

Figura 11 – Pinos do conector do controle do Playstation [10].

Pino 1 – data

Transmite um sinal síncrono de 8 bits na borda de descida do *clock* com os estados dos botões e *joysticks* do controle.

Pino 2 – command

Recebe um sinal de 8 bits na borda de descida do *clock*. Contador do DATA.

Pino 5 – VCC

A tensão de operação do controle é 5 V, suportando correntes de até 750 mA.

Pino 6 – ATT

Usado para ativar o controle, como um "*chip select"*.

Pino 7 – CLK

Gera o sinal que mantém o sincronismo entre o controle e o microcontrolador.

Pino 9 - ACK

Recebe o sinal que o microcontrolador envia quando termina de receber os dados.

2.4.2.2. Controle Nintendo Wii

A plataforma de controle criada tem também ligada a ela um controle do videogame Nintendo Wii que internamente apresenta um pequeno acelerômetro digital. Assim, se torna possível detectar, de modo simples, movimentos sobre os eixos Y e Z indicados na Figura 12.

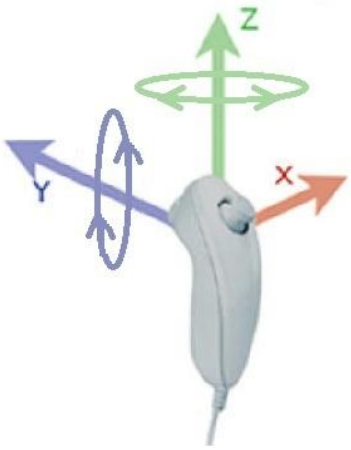


Figura 12 – Eixos de movimentos possíveis (Y e Z)

O valor indicado pelo acelerômetro é passado para o robô, que traduz essas informações em comandos para manipular os motores presentes na plataforma (servomecanismo e motores das rodas).

O controle do Nintendo Wii apresenta uma interface de comunicação do tipo I2C[10] com o Arduino, utilizando apenas dois fios para realizar a comunicação, um denominado de SDA (fluxo de dados) e o outro de CLK (*clock*) e funciona sobre a tensão 3,3V.

Na biblioteca "*wiinunchuk.h"* disponível em [11], já existem implementadas todas as funções necessárias para realizar a comunicação entre o Arduino e o controle. Essa biblioteca é baseada na biblioteca "*Wire*.h", padrão do Arduino.

De forma semelhante ao controle de Playstation, a cada 70 ms o *software* da estação de controle faz uma leitura dos estados dos elementos do controle e essa informação é enviada para o microcontrolador no formato de uma *string,* que é analisada para determinar quais botões foram pressionados ou para que lado o controle foi virado.

2.4.2.3. Controle através do Android

A plataforma criada pode ser controlada também através da utilização de um Smartphone utilizando o sistema operacional Android.

Para isso, foi criado um pequeno programa possibilitando o controle do robô utilizando uma interface do tipo *touchscreen* (pressionando botões na tela do celular) ou utilizando os acelerômetros existentes dentro do aparelho.

Para estabelecer conexão entre o Arduino e o *smarthphone*, fez-se necessária a instalação de um módulo Bluetooth [12] no sistema (vide Figura 13). Este módulo é independente, e tem como função receber os comandos do aplicativo Android através da interface Bluetooth e repassá-los via comunicação serial para o Arduino, que, por sua vez, recebe os comandos do módulo, interpreta-os e controla as peças do robô (servos, motor e *buzzer*).

Figura 13 – Módulo Bluetooth da Zuchi [12].

[10] Tipo de barramento de comunicação, permite a conexão de diversos dispositivos utilizando apenas dois fios para realizar essa integração.

O módulo Bluetooth utilizado é fabricado pela empresa nacional Zuchi e apresenta as seguintes características descritas pelo fabricante na Tabela 3 [12]:

Versão do módulo	ZT-05
Versão Bluetooth	V2.0+EDR
Classe de Potência	Classe II (até 10 m)
Tensão de Operação	5V
Interface	UART

Tabela 3 – Características do Módulo Bluetooth [12].

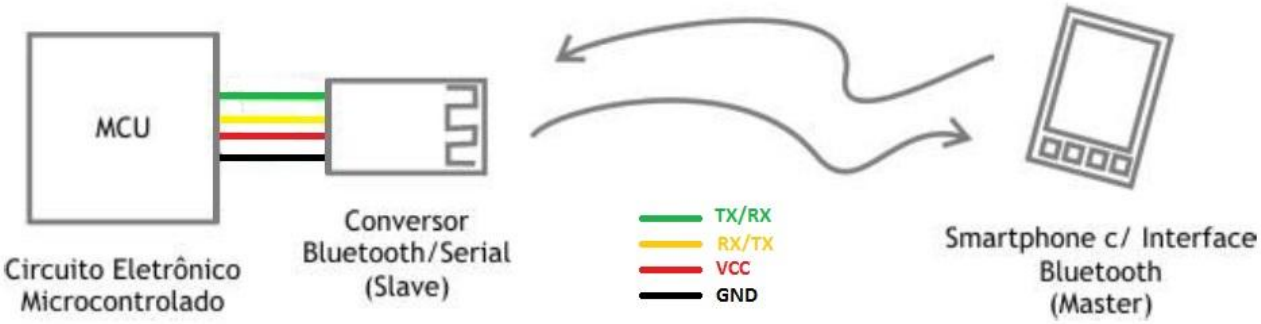


Figura 14 – Comunicação utilizando o módulo Bluetooth [12].

A Figura 14 mostra as ligações necessárias para realizar a conexão do módulo Bluetooth ao microcontrolador Arduino de modo correto e também o esquema de comunicação entre os dispositivos.

O *smartphone* com Android rodando o aplicativo se comunica exclusivamente com o módulo Bluetooth, o Arduino também se comunica unicamente com o módulo Bluetooth, e este é o único que possui comunicação com ambos os lados. Ou seja, não existe comunicação direta entre o Arduino e o Android.

Estabelecida a comunicação entre os dispositivos foi desenvolvido o *software* mostrado na Figura 15, nomeado de Dcontrol.

O Dcontrol apresenta um botão “Bluetooth” que é utilizado para estabelecer a conexão com o módulo Bluetooth independente do Arduino. Ao lado deste botão, há um botão “Sync” (ver seção 2.7.2).

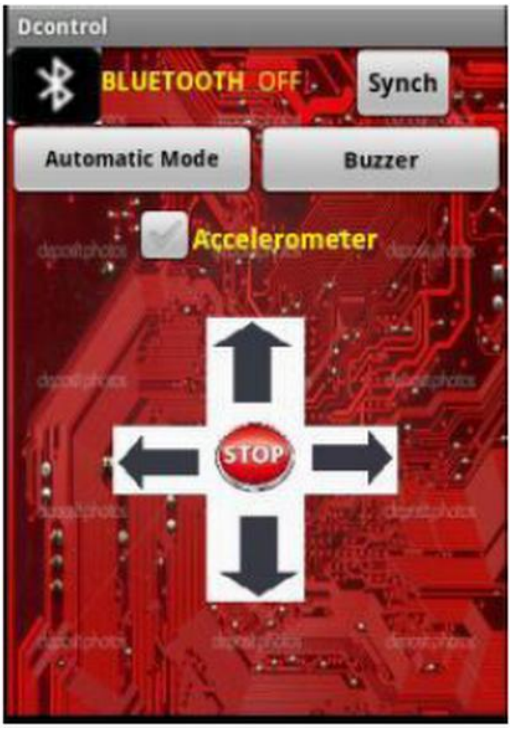


Figura 15: Dcontrol

Debaixo dos botões mencionados há mais dois botões. Quando o primeiro deles (“Automatic Mode”) é pressionado, é enviado para o robô o código que indica a chamada da rotina que possibilita o robô andar de forma autônoma sem colidir com objetos próximos, o que é possível com o uso do módulo sonar. Para interromper o modo autônomo, basta clicar em algum dos outros comandos presentes na interface.

Quando o botão “Buzzer” é pressionado, o robô emite um pequeno som do tipo “bip".

Na interface criada, está presente uma caixa do tipo *checkbox*, e, quando a mesma é ativada, comandos referentes à movimentação do robô são enviados para o mesmo, usando como base os dados retornados pelos acelerômetros do celular, ou seja, ativando essa *checkbox*, passa-se a controlar o robô com base na posição do *smartphone*. A seguir as possíveis posições são mostradas:

- *Smartphone* inclinado para frente: andar para frente;
- *Smartphone* inclinado para trás: andar para trás;
- *Smartphone* inclinado para direita: virar à direita;
- *Smartphone* inclinado para esquerda: virar à esquerda.

Para uma boa leitura dos dados do acelerômetro e simplificação do *software* criado, é necessário que não se misture dois comandos ao mesmo tempo, ou seja, inclinar o *smartphone* para frente e para direita ao mesmo tempo, irá gerar valores inválidos, com

muito erro agregado por parte dos acelerômetros. Para um controle preciso da plataforma dessa maneira, é necessário realizar um movimento de cada vez.

Na parte inferior da interface, têm-se os botões das setas direcionais e um botão “Stop”. Quando o usuário pressiona uma das setas, o comando correspondente a essa seta é enviado. Os comandos de seta somente são enviados caso a *checkbox* esteja desmarcada.

A função “Sync” foi implementada para retomar a conexão entre o Arduino e o módulo Bluetooth caso o Arduino sofra um processo de *reset*. Nessa situação teríamos o módulo Bluetooth conectado ao *smarthphone*, mas o microcontrolador não saberia dessa conexão.

Ao pressionar o botão “Sync”, é informado ao robô que ele está sobre o controle do *smarthphone* e então o mesmo ajusta o seus parâmetros para isso. O processo de “Sync” é realizado automaticamente durante a primeira conexão bem sucedida do sistema (Arduino, Bluetooth e Android).

Na figura 16 é mostrado o fluxograma do *software* Dcontrol.

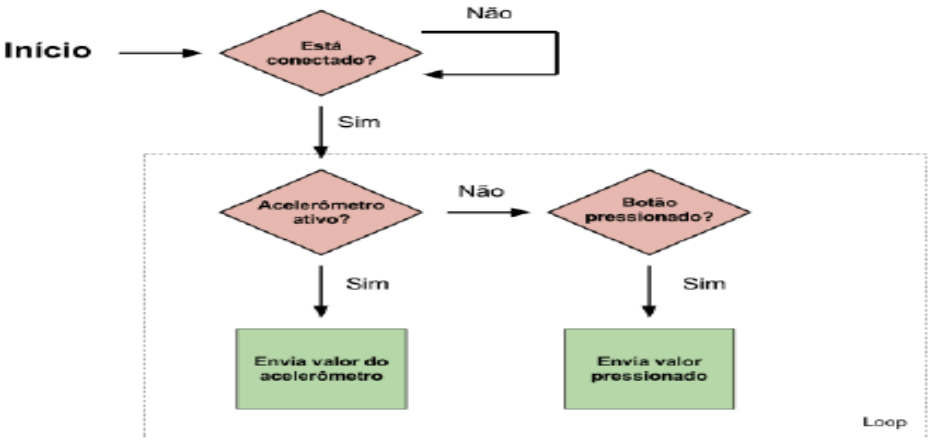


Figura 16 – Fluxograma do *software* Dcontrol.

2.5. Periféricos de Comunicação

O enlace de comunicação entre os controles de Playstation e Nintendo Wii com o robô é feito utilizando-se radiofrequência (RF), e para isso é usado o módulo APC220 [13].

O módulo realiza a interface entre a comunicação serial do Arduino e a comunicação RF com outro módulo igual operando na mesma frequência. (vide Figura 17).

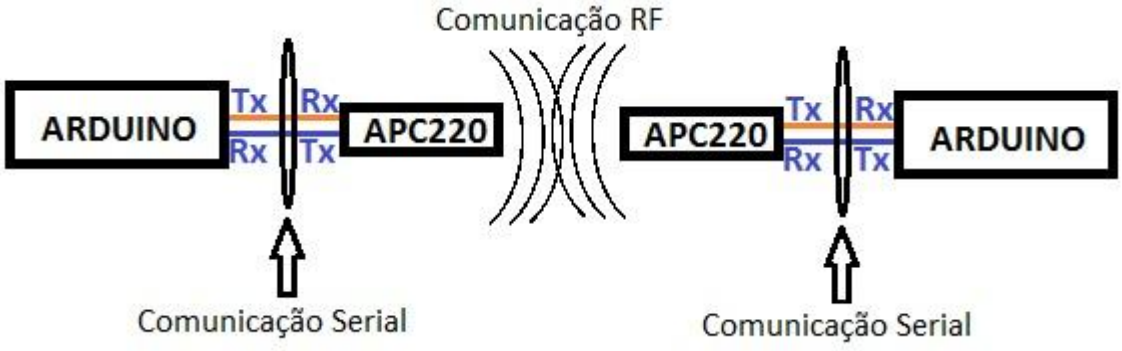


Figura 17 – Comunicação com o módulo APC220.

Na tabela 4, temos as características operacionais do módulo.

Frequência de operação	418-455 MHz
Modulação	GFSK
Intervalo de frequência	200 kHz
Potência transmitida	20 mW
Sensibilidade recebida	-113 dBm
Umidade de operação	10-90%
Temperatura de operação	-30°-85°C
Tensão de alimentação	3,5-5,5 V
Corrente transmitida	< 42 mA
Corrente recebida	< 28 mA
Distância de transmissão	< 5 µA
Dimensão	37,5 x 18,3 x 7 mm

Tabela 4 – Características Operacionais do APC220 [13].

Para o funcionamento correto dos módulos é necessário configurar ambos para que operem sobre a mesma frequência e mesmo *baudrate*[11]. O procedimento de configuração ocorre utilizando o *software* RF-Magic fornecido pelo fabricante (Figura 18) e o adaptador USB necessário para ligar os módulos ao computador (Figura 19). Os parâmetros utilizados para configuração são mostrados na Figura 18, a única exceção é o parâmetro de *RF frequency* onde foi utilizando a frequência 433Mhz.

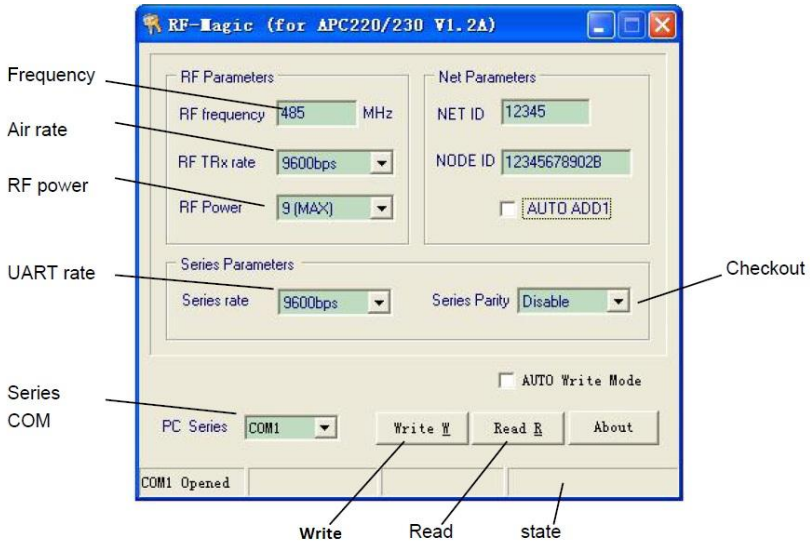


Figura 18 – Interface gráfica RF-Magic [13].

Figura 19 – APC220 conectado ao computador

[11] "*baudrate"* significa "taxa de transmissão"

2.6. Atuadores

2.6.1. Motores DC

A plataforma robótica apresenta dois motores DC com caixas de redução acopladas aos eixos para realizar o movimento das rodas do robô (Figura 20).

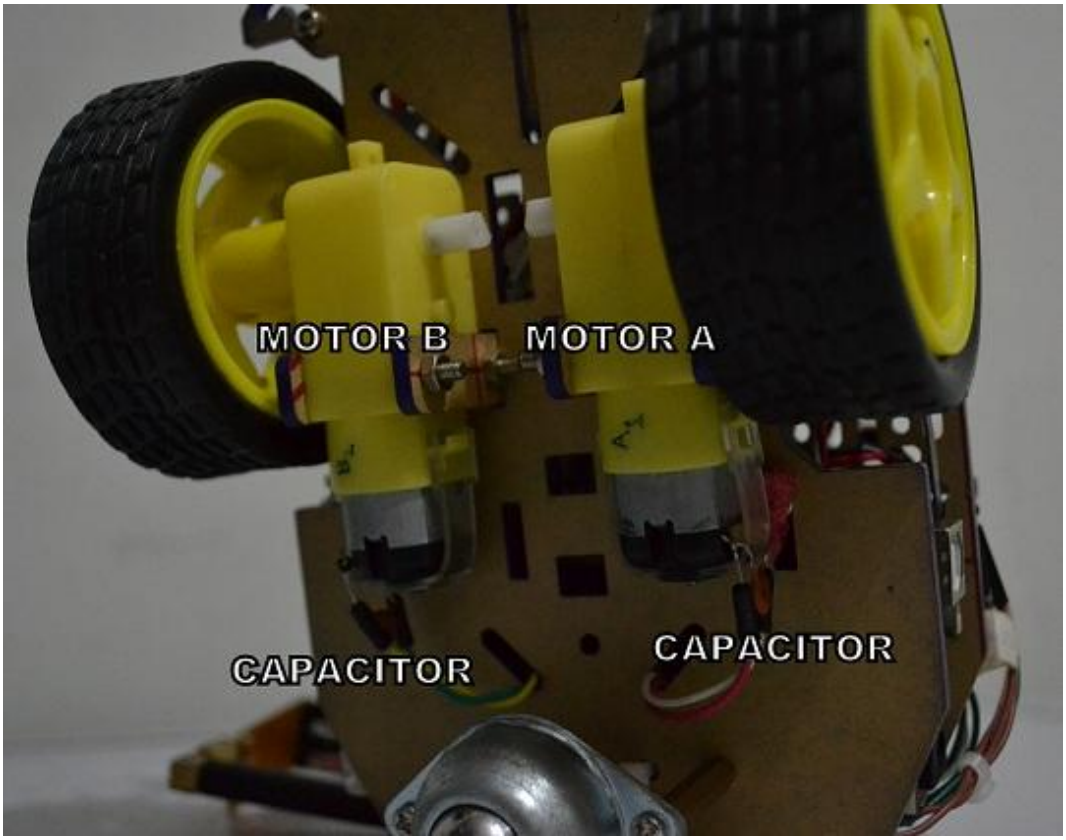


Figura 20 – Detalhes dos motores DC

Alguns dados técnicos dos motores são descritos na tabela 5.

Tensão de Operação	DC 3 V	
Parâmetros do Motor	Rotação	5000 RPM
	Corrente suportada	80-100 mA
Parâmetros da Caixa de Redução	Taxa de redução	1:220
	Velocidade (s/ carga)	20 RPM
	Velocidade (c/ carga)	14 RPM
	Torque de saída	1 kg.cm
	Corrente suportada	110-130 mA

Tabela 5 – Dados Técnicos dos motores DC [14]

Como podemos ver, a tensão de operação desses motores é de 3 V a 4,5 V. Para o sistema criado, foi escolhido para a alimentação dos motores utilizando um *pack* de cinco pilhas do tipo NiMh de 2300 mA com 1,2 V de tensão típica.

Na Figura 20 também podemos ver a presença de dois capacitores de 100 nF sobre os terminais de alimentação dos motores. Eles são colocados nessa posição para evitar possíveis *sparks*[12] produzidas pelo motor devido ao funcionamento dos motores DC.

O controle dos motores é feito utilizando o CI[13] L298N [15] (ponte H para dois motores de corrente continua) e, segundo o fabricante, suporta as operações de funcionamento descritas na tabela 6.

Tensão de Alimentação	Até 50 V
Tensão lógica	Até 7 V
Corrente de saída	2 A/canal
Potência dissipada (T = 75°C)	25 W
Temperatura da junção	-25 a 130 °C

Tabela 6 – Dados técnicos do CI L298N [15]

O CI L298N se encontra sobre a uma placa de circuito impresso (Figura 21). Podemos notar, na placa, os pinos de entrada e saída, bem como os pinos de controle.

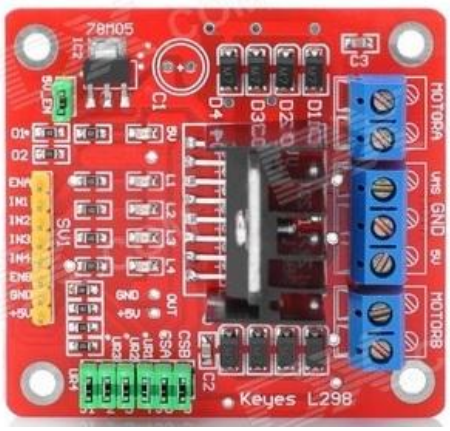
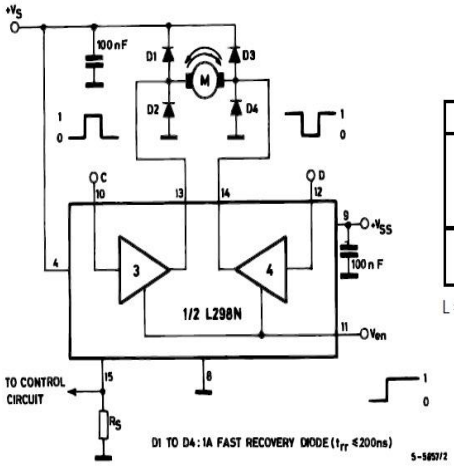


Figura 21 – Circuito impresso com o CI L298N [16].

[12] "Spark" (faísca, em inglês) pequenas faíscas que ocorrem dentro do motor, devido a sua dinâmica de funcionamento.
[13] Circuito Integrado

O circuito elétrico implementado sobre essa placa em sua essência é o mesmo mostrado pelo fabricante do CI em seu *datasheet* [15] e está mostrado na figura 22. A diferença é que na placa da figura 21 essa implementação ocorre duas vezes, uma para cada motor.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figura 22 – Esquemático de utilização do CI L298N [15].

O esquemático (Figura 34), Anexo I, demonstra as ligações necessárias para utilizar a Ponte H em conjunto com os motores e microcontrolador e como ela foi implementada nessa plataforma.

Os pinos IN1, IN2, IN3 e IN4 são usados para realizar o controle de direção de rotação dos motores e os pinos ENA e ENB recebem os sinais de PWM[14] emitidos pelo Arduino para girar os motores.

Podemos ver no esquemático que os pinos IN1 e IN2, assim como o IN3 e IN4, estão ligados ao mesmo sinal, porém um passa por um inversor antes de chegar ao microcontrolador e o outro não. Isso foi feito para reduzir (de quatros para dois) o número de pinos de controle necessários.

A combinação dos pinos de controle da ponte H e o sentido de rotação dos motores são descritos na tabela 7.

[14] "*Pulse-Width Modulation"* ou "Modulação por largura de pulso" de um sinal é o controle do valor da alimentação entregue à carga. No caso, aos motores.

IN1	IN2	IN3	IN4	Direção
0	1	0	1	Trás
0	1	1	0	Esquerda
1	0	0	1	Direita
1	0	1	0	Frente

Tabela 7 – Combinação dos pinos de direção do motor

2.6.2. Servos

A plataforma Arduino já apresenta bibliotecas para o controle de servos ligados ao ATMega328. Assim, todas as funções para realizar o controle de múltiplos servos já existem, sendo necessária somente a implementação de algumas rotinas.

A plataforma criada conta com dois servos de 9g da Tower Pro, como o da figura 23.

Figura 23 – Servo de 9g da Tower Pro [18]

Os servos são montados dentro de uma pequena estrutura de alumínio formando um sistema do tipo *pan & tilt*, onde é adicionado o módulo sonar, possibilitando a movimentação do mesmo sobre os eixos X e Y.

Detalhes sobre essa estrutura podem ser vistos no modelo feito no Google Sketchup na figura 23.

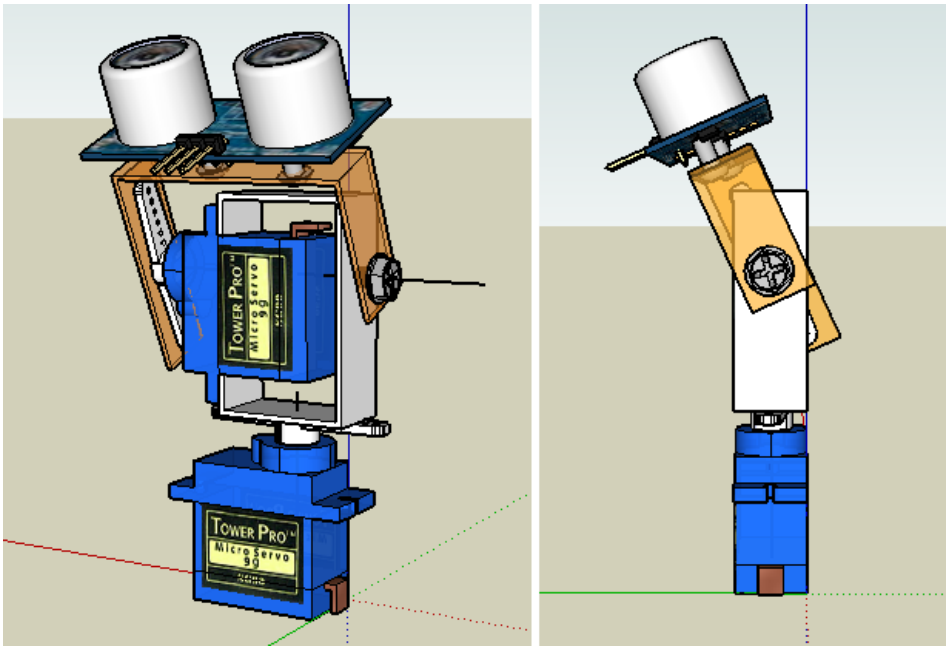


Figura 24 – Sistema *pan & tilt* e os servos

Junto à estrutura foi colocada também uma pequena ponta de caneta *laser*, que pode ser acionada para indicar, de forma aproximada, a direção para qual sentido o módulo sonar está posicionado (vide figura 25).

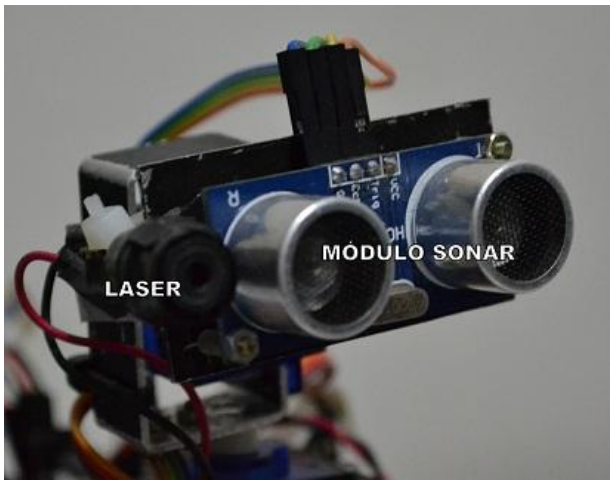


Figura 25 – Detalhe da parte superior da estrutura

2.7. Comandos e decisões

Existem dois modos de controlar o robô: utilizando a plataforma de controle e usando um *smarthphone*.

2.7.1. Comandos

Os comandos implementados para a plataforma são mostrados na tabela 8, e a localização dos botões referentes a cada comando é mostrada nas figuras 26 e 27.

Comandos	Playstation	Nintendo Wii	Celular
Andar p/ frente	Cima	C + Z + inclinar controle para frente	Cima
Andar p/ trás	Baixo	C + Z + inclinar controle para trás	Baixo
Girar p/ direita	Direita	C + Z + rotacionar controle no sentido horário	Direita
Girar p/ esquerda	Esquerda	C + Z + rotacionar controle no sentido anti-horário	Esquerda
Parar	Quadrado	C + Z + controle na posição neutra	Botão *Stop*
Modo automático	Bola	N/I	*Checkbox*
Controle completo *pan & tilt*	L1 + Joystick2	N/I	N/I
Eixo X *pan & tilt*	L2 + Joystick1	C + rotacionar controle	N/I
Eixo Y *pan & tilt*	L2 + Joystick2	Z + rotacionar controle	N/I
Buzzer	R1	N/I	Botão *Buzzer*
Laser	R2	N/I	N/I
Synch	Liga estação de controle	N/I	Botão *Sync*
Conectar Bluetooth	N/I	N/I	Botão *Bluetooth*

Tabela 8 – Movimentos e controle do robô

Figura 26 - Controle Playstation

Figura 27 - Controle Wii

A hierarquia entre os três possíveis controle da plataforma (controle de Playstation, controle de Nintendo Wii e Android) é estruturada da seguinte forma: Quando a plataforma é controlada pelo celular, nenhum outro controle consegue enviar comandos para a mesma. Quando não é usado o celular, o controle Nintendo Wii está acima do controle de Playstation na hierarquia, ou seja, quando se utiliza o controle Wii, os comandos vindos do Playstation não são processados. Não é possível enviar comandos de dois controles distintos para a plataforma ao mesmo tempo.

2.7.2. Decisões

Esta seção descreve os principais eventos realizados pelo robô quando um novo comando é recebido. As entradas e como elas interagem com a dinâmica da plataforma são mostradas a seguir:

Andar para frente

Quando o robô recebe o comando de andar para frente, as seguintes ações internas são realizadas:

- Os pinos referentes ao sentido de rotação de ambos os motores da plataforma são ajustados para que os dois motores rodem em sentido horário, propulsionando o robô pra frente.

- O sinal de PWM dos motores é ajustado para o valor de 255 (8 bits) em seu *duty cicle*.

Essa condição é mantida durante 100 ms. Se antes disso um novo comando não chegar, a plataforma ajusta os PWM dos motores para zero, parando o robô.

Andar para trás

Quando o robô recebe o comando de andar para trás, as seguintes ações internas são realizadas:

- Os pinos referentes ao sentido de rotação de ambos os motores da plataforma são ajustados para que os dois motores rodem em sentido anti-horário, propulsionando o robô pra trás.

- O sinal de PWM dos motores é ajustado para o valor de 255 (8 bits) em seu *duty cicle*.

Essa condição é mantida durante 100 ms. Se antes disso um novo comando não chegar, a plataforma ajusta os PWM dos motores para zero, parando o robô.

Girar para esquerda

- Os pinos referentes ao sentido de rotação do motor A[15] é ajustado para que o mesmo rode no sentido horário, e o motor B no sentido anti-horário.

- O sinal de PWM dos motores é ajustado para 255 (8 bits) em seu *duty cicle*.

Essa condição é mantida durante 100 ms. Se antes disso um novo comando não chegar, a plataforma ajusta os PWM dos motores para zero, parando o robô.

Girar para direita

[15] A identificação dos motores pode ser vista na figura 16.

- Os pinos referentes ao sentido de rotação do motor A é ajustado para que o mesmo rode no sentido anti-horário, e o motor B no sentido horário.

- O sinal de PWM dos motores é ajustado para 255 (8 bits) em seu *duty cicle*.

Essa condição é mantida durante 100 ms. Se antes disso um novo comando não chegar, a plataforma ajusta os PWM dos motores para zero, parando o robô.

Parar

- Os pinos de PWM dos motores são ajustados para zero, independentemente do sentido dos motores.

Essa condição é mantida enquanto um novo comando não for recebido.

Modo autônomo

Uma pequena sub-rotina denominada "*auto_mode()"* foi criada para permitir que o robô ande pelo ambiente tentando evitar colisões com objetos que se encontram na sua frente.

Ao iniciar o método é realizado o ajuste da posição do sonar, deixando o mesmo olhando para frente da plataforma.

A heurística do método desenvolvido se utilizar de números pseudoaleatórios para as tomadas de decisões: Toda vez que o módulo sonar detectar algum objeto em uma distância mínima de 20 cm, o microcontrolador "sorteia" um número de 0 a 2. O número sorteado indica a nova posição para onde o robô deve tentar andar, de acordo com o diagrama da Figura 28.

O valor inicial da semente utilizada para gerar os números aleatórios é igual à leitura do pino analógico A2, que, como não se encontra conectado a nenhum dispositivo (ver Anexo I), contém ruído, e assim o código não gera sempre a mesma sequência.

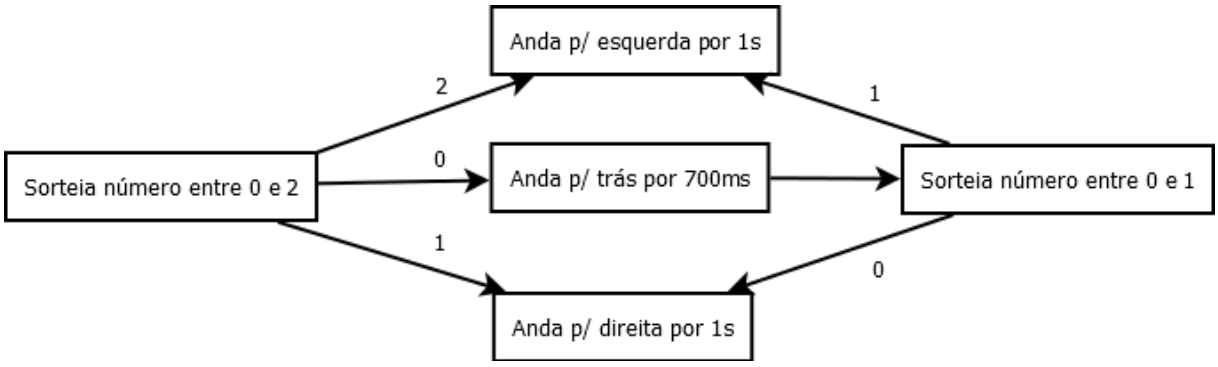


Figura 28 – Modo autônomo

Determinado o novo sentido do movimento, as saídas do microcontrolador são ajustadas e os pinos PWM dos motores são acionados.

A rotina não lida com situações de colisões laterais ou traseiras.

Controle *pan & tilt*

- As posições dos servos X e Y[16] são passadas para o robô através de uma *string*. A *string* é separada de modo a obter a posição de cada servo e posicionar os mesmos.

Buzzer

O pino D8 do Arduino fica no nível lógico zero por 10 ms, retornando para o nível lógico um em seguida. O pino está conectado a um *buzzer*, que gera um pequeno apito do tipo “bip”.

Laser

O pino D13 do Arduino fica no nível lógico um, fechando o circuito elétrico onde se encontra a ponta da caneta *laser*. Quando o pino está em nível lógico zero, o *laser* é desligado.

[16] A identificação dos servos pode ser vista na figura 5.

Sync

O “*Sync”* é um processo que ocorre automaticamente entre o robô e a plataforma de controle, quando são ligados nessa ordem.

O “*sync”* é um caractere de controle que é enviado para o robô para indicar quem está controlando-o (a plataforma de controle ou o celular com Android). Se nenhum caractere é recebido, o robô não é ativado e permanece aguardando esse sinal.

No *software* Dcontrol foi implementado um botão para essa ação, e na plataforma de controle ela ocorre automaticamente quando se liga a mesma.

A ativação do “*Sync”* realiza um *soft-reset* no Arduino, reiniciando a execução do seu código, caso ocorra depois de o robô já estar sincronizado.

Conectar Bluetooth

Função que realiza a conexão entre o módulo Bluetooth e o *smartphone.* Esse processo de conexão é totalmente independente do Arduino.

Quando o módulo Bluetooth não se encontra conectado com nenhum dispositivo, um pequeno *led* vermelho no módulo permanece piscando, e quando o mesmo encontra-se conectado, o *led* permanece aceso, indicando que há comunicação entre os dispositivos.

Capítulo 3: Implementação

3.1. *Software* do robô

O *software* criado foi desenvolvido na ferramenta Arduino IDE em linguagem C e conta com seis arquivos-fonte, com os métodos e operações necessários para coordenar todos os periféricos do robô.

Os arquivos são:
- *droid_v4*
- *manual_servo*
- *automatic_mode*
- *comandos_buzzer*
- *set_pin*
- *timer2_sonar*

3.1.1. droid_v4

Arquivo principal do sistema, contém as duas funções básicas do Arduino, *setup()* e *loop()*. Toda vez que o microcontrolador é ligado, as instruções do *setup()* são executadas e, em seguida, o programa fica em um laço infinito executando as instruções de *loop()*.

No método *setup()* são realizados os ajustes iniciais referentes a comunicação serial e pinos de entrada e saída, e o programa permanece inerte aguardando o sinal de "*sync"* (ver seção 2.7.2) a ser enviado por algum dos meios de comunicação.

Tanto para comunicação RF como Bluetooth, o *baudrate*[17] da comunicação serial é o mesmo, constante em 9600 bauds.

Após o primeiro sinal de "*sync"*, a função *setup*() é finalizada e dá-se início ao *loop* infinito, que consiste em cinco operações:

- Ler dados da serial;
- Codificar os dados lidos e determinar qual comando foi recebido;

[17] "baudrate" significa taxa de transmissão.

- Realizar o movimento correspondente ao comando recebido;
- Verificar a distância do sonar ao objeto em sua frente, para evitar colisões;
- Desligar os servos que controlam o sonar se não usados por mais de 1s, evitando o desperdício de bateria.

3.1.2. manual_servo

Contém a implementação dos métodos necessários para lidar com o controle dos servos no sistema de *pan & tilt* e verificar se os mesmos estão sendo usados.

3.1.3. automatic_mode

Contém o método descrito em 2.7.2 na parte referente ao funcionamento do modo automático implementado.

3.1.4. comandos_buzzer

Devido ao fato do *software* do robô estar na sua 4ª versão, algumas manipulações de baixo nível, como os ajustes de E/S e valores de PWM, foram encapsuladas em métodos prontos desde a 1ª versão, que foram reutilizados ao decorrer do desenvolvimento do projeto.

Os métodos implementados lidam com os seguintes eventos, referente ao controle dos motores:

- Ajuste de direção de cada motor;
- Ajuste do PWM aplicado a cada motor;
- Acionamento do *buzzer*.

Com exceção do acionamento do *buzzer,* todos os métodos realizam o acionamento do *Timer* 2 do Arduino, utilizado para parar a plataforma após 100 ms sem receber nenhum comando.

3.1.5. set_pin

Realiza todas as operações de ajuste dos pinos do Arduino, tanto os iniciais (entrada/saída) quanto os devido à dinâmica dos comandos de controle , chamando os métodos implementados em "*comandos_buzzer*".

3.1.6. timer2_sonar

Aqui é implementado o método chamado em *loop()* para o controle do sonar, verificando a distância dos objetos a seu alcance e informando esse valor ao microcontrolador. Se a distância detectada for menor do que 20 cm, o PWM dos motores é setado para zero e o robô para de andar.

A utilização do *Timer* 2 é feita utilizando a biblioteca "*MsTimer2.h"* encontrada em [17], e seu uso se deve ao fato de que a cada 100 ms sem um novo comando, o PWM dos motores seja setado para zero.

O sistema foi estruturado com essa condição para que, em caso de perda de comunicação, a plataforma não continue andando sem controle.

3.2. *Software* de controle

O *software* da estação de controle foi desenvolvido na ferramenta Arduino IDE em linguagem C e conta com quatro arquivos-fontes:

- *Droid_controle_v4*
- *playstation_search_cmd*
- *wiinunchuck_search_cmd*
- *command*

3.2.1. Droid_controle_v4

Contém os métodos *setup()* e *loop()*. Durante o método *setup()* são realizadas as chamadas dos métodos necessários para inicializar o controle da plataforma: Inicialização da comunicação I2C do controle Wii e inicialização da comunicação com o controle de Playstation. Se nenhum problema for detectado, o *led* amarelo na tampa da estação de

controle (ver Figura 4) acende. Caso contrário, o *led* permanece apagado indicando que não foi possível iniciar a comunicação.

O método *loop()* realiza a leitura dos dados do controle e envia o comando recebido para a plataforma robótica. Esse processo ocorre a cada 70 ms.

3.2.2. playstation_search_cmd

Contém os métodos que analisa a *string* obtida com a leitura dos dados do controle do Playstation e chama os métodos contidos no arquivo "*command*" para enviar o comando correspondente.

3.2.3. wiinunchuck_search_cmd

Análogo ao "*playstation_search_cmd*", mas os métodos são referentes a leitura dos dados do controle do Wii.

3.2.4. command

Envia pela comunicação serial (RF ou Bluetooth) um código referente ao comando que deve ser executado pelo robô. Os códigos utilizados encontram-se na tabela 10. A codificação é a mesma, tanto para a plataforma de controle, quanto para o robô, o que possibilita a consistência da comunicação entre as duas partes.

Comando	Código
Andar p/ frente	W
Andar p/ trás	S
Girar p/ direita	D
Girar p/ esquerda	A
Parar	P
Modo automático	U
Buzzer	B
Laser	L
Servos	(Posição servo X)(Posição servo Y)

Tabela 9 – Tabela de códigos

Toda vez que um comando é pressionado em algum controle, o mesmo é codificado em uma *string* de tamanho variável segundo a Tabela 10. A *string* então é enviada utilizando a comunicação serial do Arduino e o módulo RF ou Bluetooth se encarregam do resto do processo de codificação e decodificação da informação no recebimento.

3.3. Fluxograma dos *softwares*

O código fonte desenvolvido para a plataforma encontra-se no anexo III e anexo IV, na Figura 29, é mostrado o fluxograma do código de controle do robô, podemos observar a presença dos dois métodos principais do Arduino, *void setup()* e *void loop()*, representadas respectivamente pelos retângulos azul e vermelho e as principais operações realizadas dentro dessas funções.

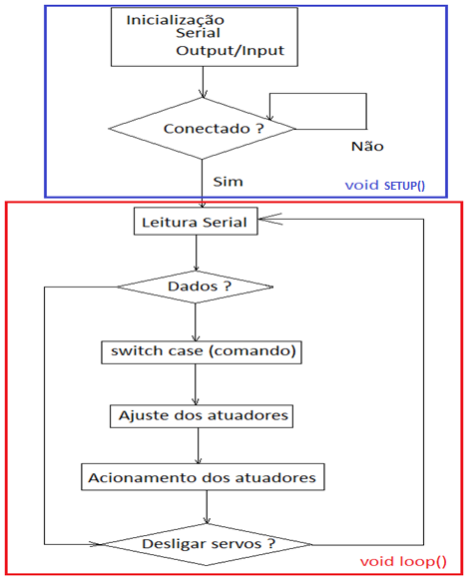


Figura 29: fluxograma do robô

Na figura 30 é apresentado o fluxograma do *software* implementado para a estação de controle, novamente é destacados os métodos *void setup()* e *void loop()*.

Tanto para o *software* do robô quanto para o *software* da plataforma de controle, na função *void setup()* é realizado a inicialização das variáveis e periféricos da plataforma e no método *void loop()* é realizado as operações de leitura de sensores, acionamento de atuadores e comunicação entre as partes da plataforma.

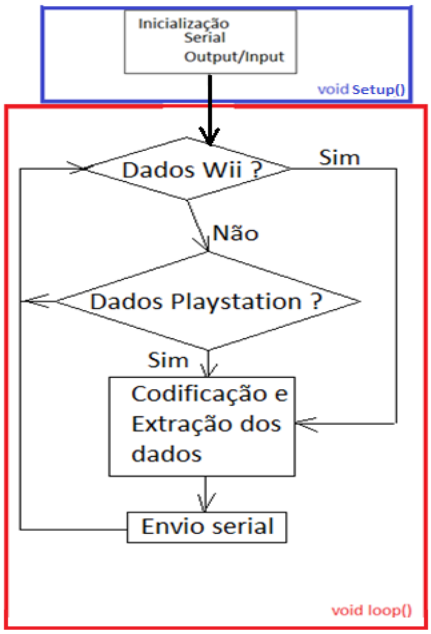


Figura 30: Fluxograma da plataforma de controle

Capítulo 4: Resultados

O desenvolvimento desse trabalho teve inicio com alguns testes em bancada para descobrir como se realizava o controle de motores DC utilizando o Arduino, Figura 31, e ao longo dos meses foi sendo adicionada novas funcionalidades a plataforma.

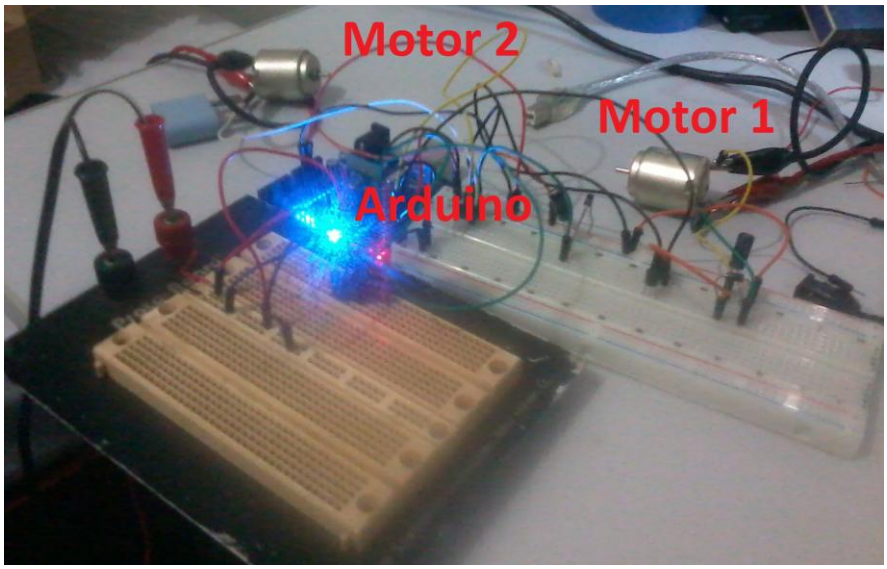


Figura 31: Experimentos.

O sistema com o sonar funciona bem para a tarefa de medir distancias, evitando colisões frontais da plataforma com objetos presente no meio em que a mesma esta andando. O sonar acoplado sobre o sistema de *pan & tilt*, possibilita aponta-lo em diversas direções, independentemente do sentido de deslocamento da plataforma, ou seja, é possível andar com a plataforma para um sentido e apontar o sonar para outro, isso permite uma maior flexibilidade para realizar medidas em relação a um sistema utilizando sonar fixo, por exemplo.

Podemos ver na tabela fornecida por [12], que o sonar apresenta um alcance mínimo de 2 cm e máximo de 4 m, com um ângulo de medida de 15º graus. Esse ângulo, na medida realizada pelo dispositivo, acaba introduzindo erros quando o objeto começa a se distanciar do sonar. Isso ocorre devido ao fato que área vista pelo sonar começa a aumentar (vide figura 32), permitindo que demais elementos presentes no mesmo ambiente venham a influenciar a medida realizada pelo sonar.

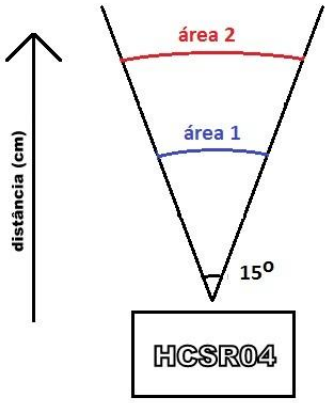


Figura 32 – Área detectável do sonar

Porém como o sonar é utilizado para realizar a detecção de distâncias próximas de 20 cm o erro introduzido na medida do dispositivo acaba sendo pequeno e, portanto o mesmo se torna viável para realizar esse tipo de operação.

O circuito de Ponte H atualizando o CI L298N funciona bem e permite o acionamento dos motores, permitindo a mobilidade do mesmo, sua utilização permite o controle dos motores DC.

Tanto o sistema de comunicação RF quanto de Bluetooth funcionam corretamente, porém utilizando a tecnologia Bluetooth a área de alcance do link de comunicação entre o robô e o celular é pequeno, restrito a dez metros, já a comunicação RF permite uma distância de aproximadamente duzentos metros entre a plataforma de controle e o robô. Devido ao tamanho reduzidos do robô distâncias maiores que vinte metros, já dificultam a visualização do mesmo, assim para as distancias onde o robô é visível a olho nu, ambas às comunicações permitem um link de comunicação seguro.

O controle de Playstation se apresentou como um ótimo dispositivo para controlar o robô, devido a sua grande diversidade de botões e joysticks foi possível implementar diversos comandos para serem enviados ao robô. O sistema de controle do *pan & tilt* utilizando os joysticks do controle é extremamente eficiente para controlar esse tipo de estrutura, apresentando boa resposta aos comandos enviados, possibilitando direcionar o *pan & tilt* para qualquer direção, dentro dos limites de comandos dos servos.

O uso dos acelerômetros, tanto do celular quanto do controle de Nintendo Wii, para controlar o robô, sofrem muito com efeitos de ruídos em suas medições, devido a sua

sensibilidade, para corrigir esse tipo de problema uma alternativa seria a implementação de filtros, ou isolar os movimentos do acelerômetro. Na plataforma não foi implementado um filtro para essa operação, por isso se realizou a isolação dos movimentos quando utilizando o acelerômetro.

Essa alternativa não permite o uso de movimento combinados, por exemplo, andar para frente e para direita ao mesmo tempo. Ao realizando a divisão de movimento o ruído agregado na leitura de cada eixo do sensor é menor e possibilita o seu uso com certa precisão.

O método automático implementado, se apresentou como uma ótima solução de baixo custo computacional, para o problema de evitar colisões frontais, e devido a sua natureza randômica, permite uma movimentação mais natural do robô pelo meio, pois para o mesmo objeto em sua frente uma nova ação aleatória é gerada para desviar o robô do obstáculo.

A plataforma desenvolvida não possui uma malha de controle fechada para o controle dos motores, assim não é possível indicar uma quantidade fixa de graus para o robô rotacionar sobre o próprio eixo, ou garantir que ele fique paralelo a uma reta enquanto anda para frente. Tentou-se realizar a implementação de um sistema com *shaft encoder*, retirado de dois mouses (Figura 33), porém a resolução dos dispositivos é muito alta para a dinâmica dos motores, impossibilitando o uso do mesmo.

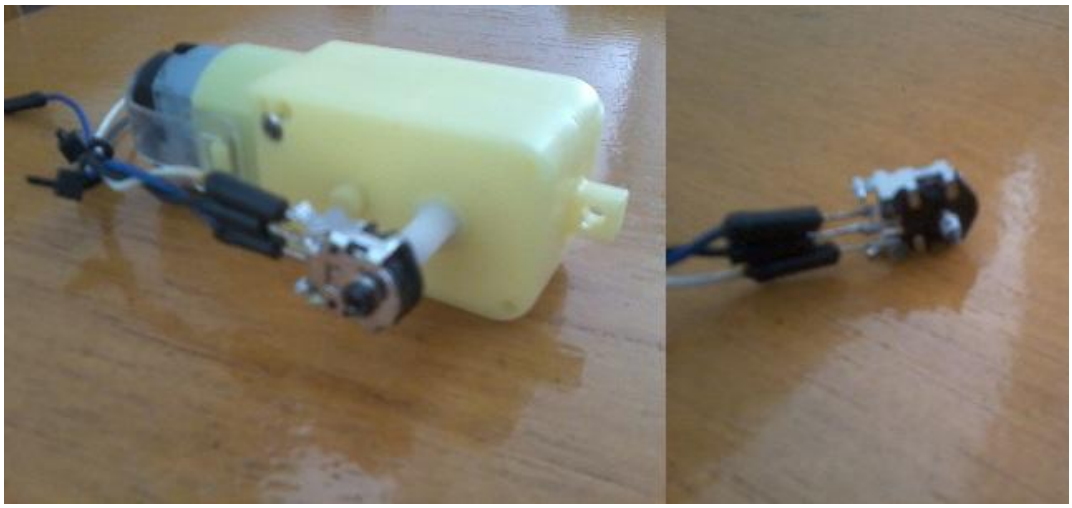


Figura 33 – Motor com *shaft encoder* à direita e o *encoder* em detalhe à esquerda.

Capítulo 5: Trabalhos futuros

Como trabalhos futuros sobre essa plataforma robótica podemos citar que um sistema de controle com malha fechada nos motores DC permitiria um controle da movimentação do robô com extrema precisão, o que seria o primeiro passo para a implementação de um conjunto de comando mais complexos como andar sobre uma velocidade constante, andar em linha reta ou andar sobre uma circunferência entre outros.

A implementação de filtros para o uso do acelerômetro, junto de um sistema de malha fechada para os motores DC permitiria a criação de movimentos combinados, o que tornaria o uso do controle de Nintendo wii um método ainda mais interessante e intuitivo de controle do robô, além de aumentar a gama de movimento possíveis realizado pela plataforma.

A implementação de uma rotina no *software* Dcontrol para lidar com a situações onde se esta controlando o robô e alguém realiza uma ligação para o celular que se esta utilizando seria interessante, pois esse tipo de situação pode vir ocorrer, com baixa frequência, durante a utilização da plataforma, atrapalhando o controle da mesma.

O método autônomo pode ser expandido para lidar com colisões laterias, utilizando a movimentação que o sistema de *pan & tilt* possibilita para o sonar. Métodos utilizando outras heurísticas podem ser implementando, fazendo com que o robô tome uma decisão sobre para qual lado ele deve andar, quando um objeto se encontra na frente do sonar, e não sortear um lado para andar como foi implementado.

A plataforma utiliza uma bateria de 9V para alimentar o Arduino e pilhas, recarregáveis de Nimh, totalizando 6V para alimentar os servos e os motores DC o conjunto de baterias da plataforma pode ser trocada por uma bateria do tipo Lipo com tensão de 7,2V, esse tipo de bateria apresenta melhores rendimentos, maiores capacidades de armazenamento de energia, maior capacidade de descarga, e menor peso em relação as baterias utilizadas. Menos peso sobre o robô significa menos energia gasta durante a movimentação do mesmo e, portanto um melhor rendimento da bateria.

Capítulo 6: Conclusões

Para realizar a construção de uma pequena plataforma robótica é necessário solucionar diversos pequenos problemas de *hardware*, *software* e mecânica, para que o conjunto final realize as tarefas desejadas, essa integração ocorre com o uso de um microcontrolador.

A integração dessas áreas, utilizando um microcontrolador, para a construção de um robô é um interessante desafio que mostra uma aplicação, em pequena escala, do possível papel que o engenheiro de computação pode assumir no mercado de trabalho.

A plataforma criada funcionou corretamente, dentro de suas limitações. As diversas possibilidades de controlar o robô demonstram maneiras bem distintas de controle para uma plataforma robótica. O controle Playstation apresenta uma grande diversidade de comandos possibilitando a combinação de diversões botões para enviar ações ao robô. O controle Nintendo Wii mostra uma possibilidade diferente de controlar a plataforma, utilizando a movimentação das mãos. Utilizando o *Smarthphone* para controlar a plataforma temos os mesmos comandos implementados quando usados o acelerômetro do controle de Nintendo Wii, porém é utilizado a tecnologia bluetooth para isso e o *software* desenvolvido permite ser utilizado em qualquer aparelho celular com suporte ao sistema operacional Android.

A construção dessa unidade robótica foi um grande desafio e com bons resultados, após a sua implementação. O controle mais eficaz para se utilizar o robô é o de Playstation, consequentemente o melhor sistema de comunicação, dentro dessa implementação, é o sistema de RF. O controle utilizando o *Smartphone* é eficiente por não depender da estação de controle, sendo assim um controle mais universal em relação aos outros.

Assim, a construção de um robô envolve diferentes áreas do conhecimento, o que o torna uma boa plataforma de aprendizagem e introdução para o mundo da robótica e suas possibilidades. A estação de controle, o robô, e os *softwares* criados para realizar a integração de todo o sistema, demonstra algumas das diversas possibilidade de integração de diversos componentes, disponíveis no mercado, através do uso de um microcontrolador. Responsável por coordenar e comandos todos os periféricos ligados no mesmo.

Realizar essa integração de tecnologias diversas e ajudar o desenvolvimento de novas e melhores tecnologias é parte do papel de engenheiro de computação, e reflete o

conhecimento obtido durante o curso de graduação. A robótica acaba sendo uma ótima maneira de introduzir esse universo de conhecimento de uma maneira prática funcional e didática. Agregando assim grande valor técnico, teórico e cientifico obtido com o desenvolvimento dessa plataforma robótica.

Anexo I – Esquemático do robô

O esquemático da plataforma robótica foi desenvolvido na ferramenta Eagle e pode ser visto na Figura 34. Nele podemos ver o Arduino e todos os periféricos já citados anteriormente: Ponte H, Motores DC (Motor A, Motor B), Servo-motores (Servo X, Servo Y), *uzzer, Laser,* módulo sonar (HC-SR04) e os periféricos de comunicação (APC220/Bluetooth).

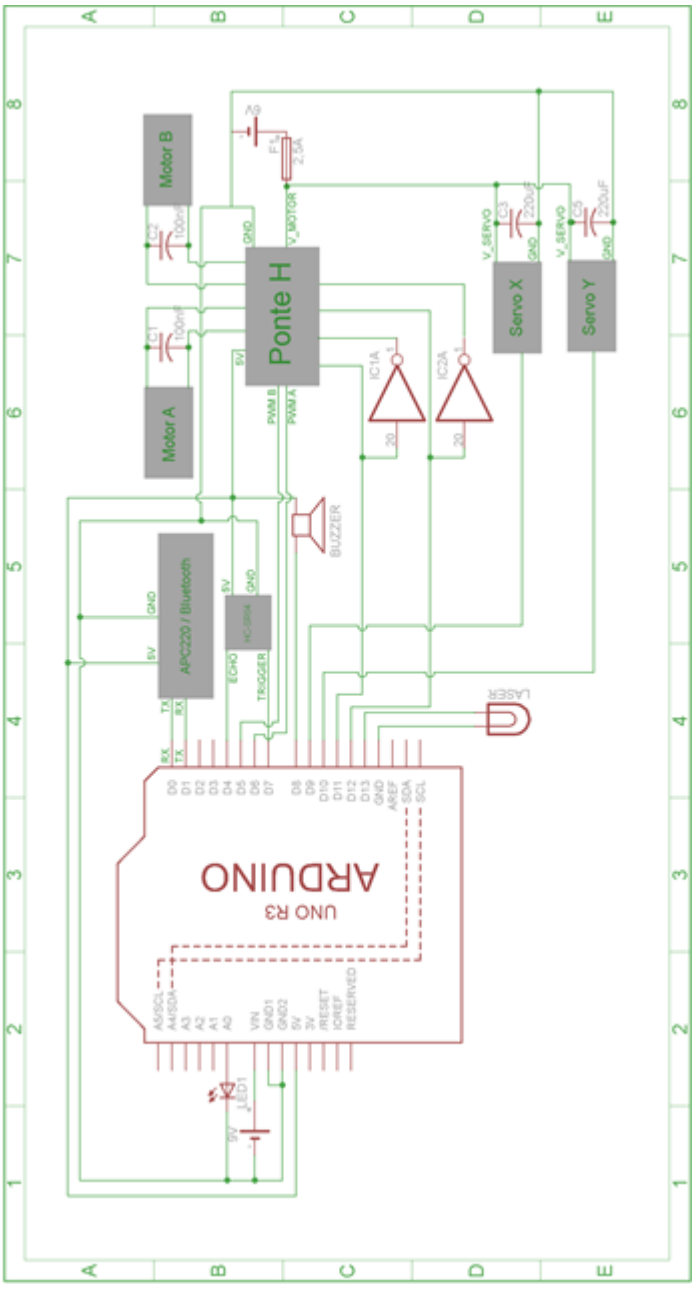


Figura 34 – Esquemático da plataforma robótica

Anexo II – Esquemático da plataforma de controle

O esquemático da plataforma de controle foi desenvolvido na ferramenta Eagle e pode ser visto na figura 35. Nele podemos identificar o Arduino, cérebro do circuito, os controles de Playstation e Wii, e o módulo APC220 que faz a comunicação RF.

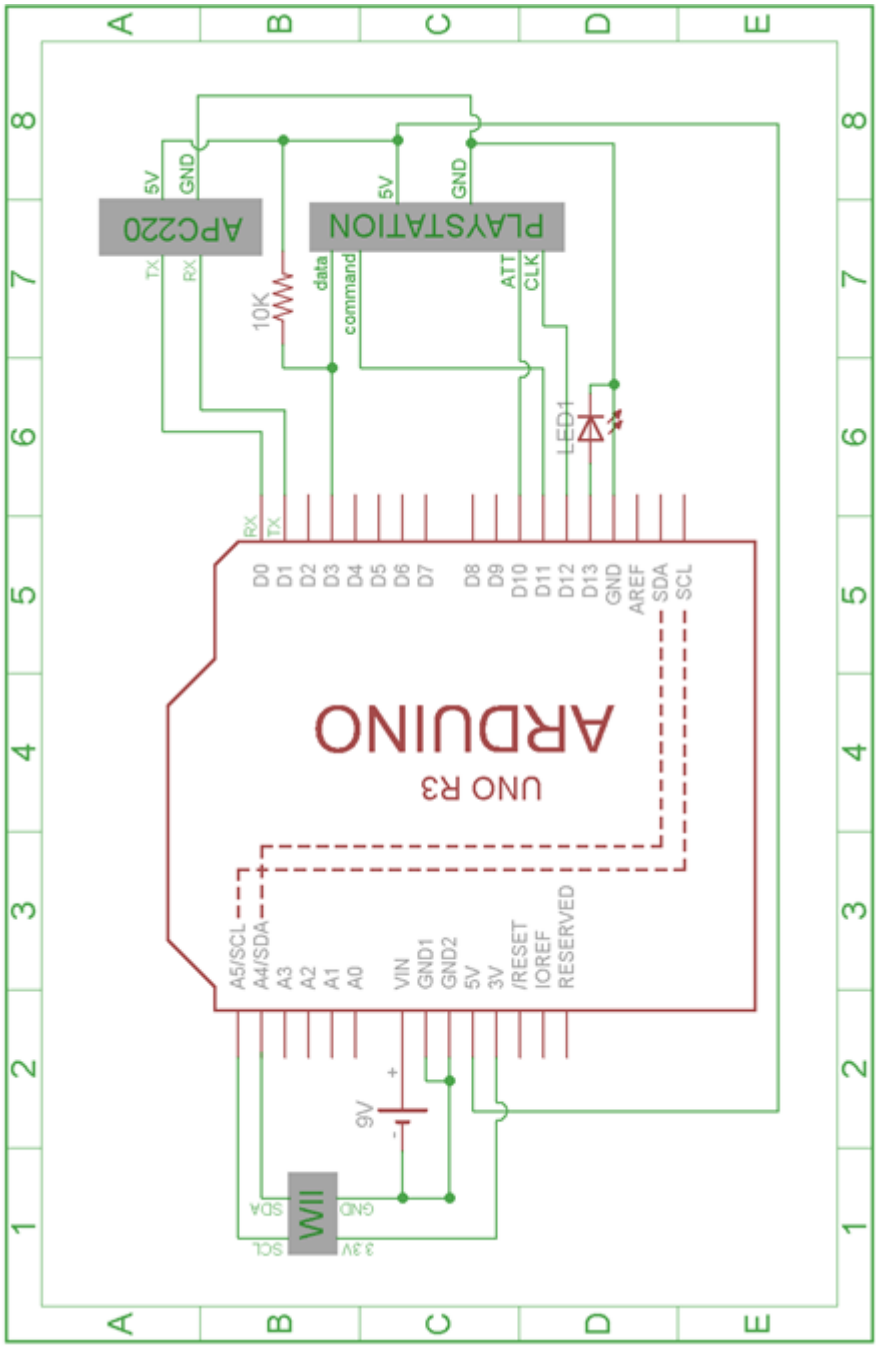


Figura 35: Esquemático da plataforma de controle

Anexo III – Código fonte do robô

ARQUIVO: droid_v4.pde

```
//#define RANGE_RF_TEST
//#define SEM_SOM

*=================>  Ligaçoes  <=======
 *  DIGITAL:
 *
 *  0 - Tx do radio                          |    7 - Trigger Sonar
 *  1 - Rx do radio - ver OBS                |    8 - buzzer *apita em LOW , foi ligado
invertido
 *  2 -                                      |    9 - Servo X
 *  3 -                                      |   10 - Servo Y
 *  4 - Echo Sonar                           |   11 - dir1PinA
 *  5 - PWM motor 2                          |   12 - dir1PinB
 *  6 - PWM motor 1                          |   13 - laser
 *
 *  ANALOGICOS
 *  A0 - led synch (led amarelo no shield)   |    A3 -
 *  A1 -                                     |    A4 -
 *  A2 -                                     |    A5 -
 *
 */

#include <Servo.h>           // servos
#include <NewPing.h>          //sonar
#include <MsTimer2.h>        // Timer2OverFlow

//soft  reset ->  nao reset os registradores
void(* resetFunc) (void) = 0; //declare reset function @ address 0

#define FRENTE 0
#define TRAS 1
#define DIREITA 2
#define ESQUERDA 3
#define PARADO 4

// motor 1
#define dir1PinA 11
#define speedPinA 6//10

//motor 2
#define dir1PinB 12
#define speedPinB 5//9

//botoes, leds, buzzer, laser
#define buzzerPin 8
#define led_synch A0
#define laserPin 13

long timebuzzer;// usando junto com o sonar
/* =========> led_synch  <=========
 * 1) se estiver piscando rapido ->  Esperando synch do controle
 * 2) se estiver acesso          ->  Sonar esta detectando algo
 * 3) se estiver apagado         ->  Sonar nao esta detectando algo
 */

//servos
#define pinServoX 9
#define pinServoY 10
Servo Servo_x;
Servo Servo_y;
unsigned long time_to_detach_servo;    // usado para detached o servo

//sonar
#define TRIGGER_PIN  7
#define ECHO_PIN     4
#define MAX_DISTANCE 200 // distancia maxima sonar
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
unsigned long time_last_read;          // tempo em que se realizou a ultima leitura do sensor
int valid_dist_sonar = -1;            //leitura valida do sonar usado para parar o carrinho
com distancia minima
```

```
char comando;// comando recebido do controle
static int direcao;

//usando para comunicacao
int nada;

//parametro para ajuste do sistema
const byte pwmA = 255;  //pwm do motor A ( 0  -  255 )
const byte pwmB = 249;  //pwm do motor B ( 0  -  255 )
const byte PWMA = 180;  //pwm do motor A ( 0  -  255 )
const byte PWMB = 180;  //pwm do motor B ( 0  -  255 )
unsigned int time_overFlow = 100; //100ms para overflow do timer2
const byte safe_dist = 20;  //distancia minima(cm) para parar o carrinho pelo sonar

#ifdef RANGE_RF_TEST
long TimeSend_ack = millis();
unsigned int send_value = 0;
#endif

void setup()
{
  pinMode_setup();

  Serial.begin(9600);

  direcao = PARADO;//comecar parado sempre
  aplly_vel(0, 0);

  Serial.println("Waiting for the controller ...");

  timebuzzer = millis();

  while( comando != '$' && comando != '&'  && comando != 'q') // char para iniciar os comandos
  {
    //comando == $ => RF controle
    //comando == & => bluetooth
    //comando == q => RF pc

    if( millis() - timebuzzer > 50 ) // just blinking the led
    {
      digitalWrite(led_synch , digitalRead(led_synch) ^ 1);
      timebuzzer =  millis();
    }

    if( Serial.available() > 0 )
    {
      comando =  (char)Serial.read();

      if( comando == '-1')
        comando = '*';

      delay(3);

      if( comando != '$' )
        Serial.print(comando);
    }
  }

  if(  comando == '&' )//bluetooth setup e ajuste inicial dos servos
  {
    /*
     Serial.println("Bluetooth Connection On");
     */
    time_overFlow = 10000;// ajustando timer para ao bluetooth
    servo_setup(108, 180); // coloca os servos na posicao inicial x e y
  }
  else
    servo_setup(100, 90);


  if( comando == 'q' ) // controle pelo teclado do pc - debug
    time_overFlow = 3600000;

  TimerOverFlow_setup();// ajusta o timer2, tem que ser chamada apos o bluetooth_setup  para
corrigir o valor do timer2 caso a conexao seja por bluetooth
```

```
  /*
  Serial.print("Timer_OverFlow: ");
   Serial.println(time_overFlow);
   */

  time_last_read = time_to_detach_servo = millis();

  do// pegar uma medida valida do sonar
  {
    valid_dist_sonar = sonar_read();
  }
  while( valid_dist_sonar == -1 );//inicializando o sonar

  comando ='*';

  Serial.println("\nOK Ready to run");
  digitalWrite(led_synch , LOW);//ready to run
}

void loop()
{
  comando ='*';

  if( Serial.available() > 0 )
  {
    comando =  Serial.read();
    delay(2);
  }

  switch(comando)
  {

  case 'w' :
    frente(&direcao);
    break;

  case 's':
    tras(&direcao);
    break;

  case 'd':
    direita(&direcao);
    break;

  case 'a' :
    esquerda(&direcao);
    break;

  case 'p' : // parar
    parar(&direcao);
    break;

     case 'W' : //debug teclado
    frenteW(&direcao);
    break;

  case 'S': //debug teclado
    trasS(&direcao);
    break;

  case 'D': //debug teclado
    direitaD(&direcao);
    break;

  case 'A' : //debug teclado
    esquerdaA(&direcao);
    break;

  case 'b' :  //buzzer
    buzzer();
    break;

  case '%' : // entrar no servo mode

      if( Servo_x.attached() == true && Servo_y.attached() == true)// so entra se os servos
estiverem attached
```

```
    {
      manual_servoMode();
    }
    else // senao tiverem attached , attach os servos
    {
      Servo_y.attach(pinServoY);
      Servo_x.attach(pinServoX);
      time_to_detach_servo = millis();
    }
    break;

  case 'r' :  // reset
    resetFunc();
    break;

  case 'u' :
    automaticMode();
    break;

  case 'l' :
    digitalWrite(laserPin, digitalRead(laserPin) ^ 1 );
    break;

  }

  if_necessary_detach_servo();

  sonar_verify_rotine();//safe distance rotine
  sonar_stop_with_control(safe_dist);

#ifdef RANGE_RF_TEST
  if( millis() - TimeSend_ack > 700 )
  {
    Serial.println(send_value++);
    TimeSend_ack =  millis();
  }
#endif
}

void read_input_and_set_param(char input_string[] ,int *param1 , int *param2 , int *param3)
{
  int i = 0;

  delay(12);//ausencia causa problemas na comunicacao

  while( Serial.available() > 0 )
  {
    char aux = (char)Serial.read();

    if( aux != ',' )
      input_string[i++] = aux;
    else // iremos obter o param1
    {
      input_string[i] = '\0';

      *param1 = atoi(input_string);

      i = 0;
      while( Serial.available() > 0 )// lendo pwmA
      {
        aux = (char)Serial.read();

        if( aux != ',' )
          input_string[i++] = aux;
        else
        {
          input_string[i] = '\0';
          *param2 = atoi(input_string);

          i = 0;
          while( Serial.available() > 0 )// lendo pwmB
          {
            aux = (char)Serial.read();

            if( aux != '\n' )
              input_string[i++] = aux;
```

```
            else
            {
              input_string[i] = '\0';
              *param3 = atoi(input_string);

              //        Serial.println(param3);
              break;
            }
          }
        }
      }
    }
  }
}
```

ARQUIVO manual_servo.pde

```
char pos_stick[20] = "*"; // armazena em char as posicoes do joystick
int num_stick2[2] = {
  90 , 90 }; // armazena em int a posicao dos joystick

void manual_servoMode()
{
  delay(5);

  int i = 0;

  int pos_x;
  int pos_y;

  read_input_and_set_param(pos_stick , &nada , &pos_x , &pos_y);

  if( nada == 2 )
    Nunchuck_command(pos_x,pos_y);
  else
  {

  pos_x = map(pos_x , 0 , 255 , 180 , 0 );
  pos_y = map(pos_y , 0 , 255 , 180 , 0 );

  Servo_x.write(pos_x);
  Servo_y.write(pos_y);


  Serial.print("x ");
  Serial.print(Servo_x.read(),DEC);
  Serial.print("y ");
  Serial.println(Servo_y.read(),DEC);

  delay(20);
  }
}

void servo_setup(int x_ini , int y_ini)
{
  Servo_y.attach(pinServoY);
  Servo_x.attach(pinServoX);


  Servo_x.write(x_ini);
  Servo_y.write(y_ini);

  delay(20);
  time_to_detach_servo = millis();

}

void if_necessary_detach_servo()
{
  if( millis() - time_to_detach_servo > 1000 ) // time_to_detach_servo eh atualizada somente
quando se utiliza o servo no modo manual
  {
```

```
    Servo_y.detach();
    Servo_x.detach();
  }
}
```

ARQUIVO automatic_mode.pde

```
int automaticMode ()
{

  int state_led = LOW;    //led utilizado para indicar o automatic mode
  long blink_led_time =  millis();

  Servo_y.attach(pinServoY);
  Servo_x.attach(pinServoX);

  set_servos_frente();
  randomSeed(analogRead(3));

  while(1)
  {

    sonar_verify_rotine();//safe distance rotine
    sonar_stop_automatic_mode(25);// distancia de reacao



    if( Serial.available() > 0 )// qualquer comando volta pra o modo manual
      return 0;

    blink_led_time = blink_led_automatic_mode( blink_led_time);
  }

  Serial.println("Automatic Mode End");
}


void sonar_stop_automatic_mode(byte safe_dist)
{
  if( valid_dist_sonar <= safe_dist && valid_dist_sonar > 2  ) // problemas com o sonar -
alguma coisa esta perto
  {

    int randNumber = random(3);// numero aletoria de 0 ate 2

    if( randNumber == 0 )
    {

      setpin_tras();
      aplly_vel(pwmA, pwmB);
      delay(700);

      int randNumber2 = random(2);// virar para algum lado

      if( randNumber2 == 0 )
        setpin_direita();
      else if (randNumber2 == 1 )
        setpin_esquerda();

    }
    else if( randNumber == 1 )
      setpin_direita();
    else if (randNumber == 2 )
      setpin_esquerda();


    aplly_vel(pwmA, pwmB);
    delay(1000);
  }
  else// anda para frente
  {
    setpin_frente();
```

```
    aplly_vel(pwmA, pwmB);
  }
}


void set_servos_frente()
{
  Servo_x.write(108);
  Servo_y.write(180);
}

void set_servos_tras()
{
  Servo_x.write(108);
  Servo_y.write(0);
}

void set_servo_esquerda_extremo_tras()
{
  Servo_x.write(0);
  Servo_y.write(0);//posicionando servo y
}

void set_servo_esquerda_extremo_frente()
{
  Servo_x.write(180);
  Servo_y.write(180);//posicionando servo y
}

void set_servo_direita_extremo_tras()
{
  Servo_x.write(180);
  Servo_y.write(0);//posicionando servo y
}

void set_servo_direita_extremo_frente()
{
  Servo_x.write(0);
  Servo_y.write(180);//posicionando servo y
}
```

ARQUIVO comandos_buzzer.pde

```
void frente(int *direcao)
{
  *direcao = FRENTE;
 // Serial.println(*direcao);
  setpin_frente();
  aplly_vel(pwmA, pwmB);

  MsTimer2::start();
}

void tras(int *direcao)
{
  *direcao = TRAS;
//  Serial.println(*direcao);
  setpin_tras();
  aplly_vel(pwmA, pwmB);

  MsTimer2::start();
}

void direita(int *direcao)
{
  *direcao = DIREITA;
//  Serial.println(*direcao);
  setpin_direita();
  aplly_vel(pwmA, pwmB);

  MsTimer2::start();
}

void esquerda(int *direcao)
```

```
{
  *direcao = ESQUERDA;
 // Serial.println(*direcao);
  setpin_esquerda();
  aplly_vel(pwmA, pwmB);

  MsTimer2::start();
}

void parar( int *direcao)
{
 *direcao =  PARADO;
//  Serial.println(*direcao);
  aplly_vel(0, 0);


}

void frenteW(int *direcao)
{
  *direcao = FRENTE;
 // Serial.println(*direcao);
  setpin_frente();
  aplly_vel(PWMA, PWMB);

  MsTimer2::start();
}

void trasS(int *direcao)
{
  *direcao = TRAS;
//  Serial.println(*direcao);
  setpin_tras();
  aplly_vel(PWMA, PWMB);

  MsTimer2::start();
}

void direitaD(int *direcao)
{
  *direcao = DIREITA;
//  Serial.println(*direcao);
  setpin_direita();
  aplly_vel(PWMA, PWMB);

  MsTimer2::start();
}

void esquerdaA(int *direcao)
{
  *direcao = ESQUERDA;
 // Serial.println(*direcao);
  setpin_esquerda();
  aplly_vel(PWMA, PWMB);

  MsTimer2::start();
}

void aplly_vel(int pwmA , int pwmB)
{
  analogWrite(speedPinA , pwmA);
  analogWrite(speedPinB, pwmB);
}

void buzzer()
{
    digitalWrite(buzzerPin, LOW);
    delayMicroseconds(10000);
    digitalWrite(buzzerPin, HIGH);
    delayMicroseconds(10000);  // acho que pode tirar esse cara
}
```

ARQUIVO: set_pin.pde

```
void setpin_frente()
{
  motorA_frente();
  motorB_frente();
}

void setpin_tras()
{
  motorA_tras();
  motorB_tras();
}

void setpin_direita()
{
  motorA_frente();
  motorB_tras();
}

void setpin_esquerda()
{
  motorA_tras();
  motorB_frente();
}

void pinMode_setup()
{
  //botoes , leds, buzzer
  pinMode(buzzerPin, OUTPUT);
  digitalWrite(buzzerPin, HIGH);
  pinMode(led_synch, OUTPUT);
  pinMode(laserPin, OUTPUT);

  //motor A
  pinMode(dir1PinA, OUTPUT);
  //pinMode(dir2PinA, OUTPUT);
  pinMode(speedPinA, OUTPUT);

  //motor B
  pinMode(dir1PinB, OUTPUT);
 // pinMode(dir2PinB, OUTPUT);
  pinMode(speedPinB, OUTPUT);

  pinMode(2,INPUT);//interrupcao 0
  pinMode(3,INPUT);//interrupcao 1

}

void motorA_frente()
{
  digitalWrite(dir1PinA, HIGH);
}

void motorA_tras()
{
  digitalWrite(dir1PinA, LOW);
}

void motorB_frente()
{
  digitalWrite(dir1PinB, HIGH);
}

void motorB_tras()
{
  digitalWrite(dir1PinB, LOW);
}
```

ARQUIVO timer2_sonar.pde

```
void TimerOverFlow_setup()// inicializando o timer2 para gerar interrupcoes internas
{
  MsTimer2::set(time_overFlow, tOverF);// chama a funcao tOverF a cada 100 ms
}

void tOverF ()
{
  MsTimer2::stop();

  direcao = PARADO;// parando o carro
  aplly_vel(0, 0);

}

/************************************ SONAR
*****************************************************/

int sonar_read()
{
  int sonar_distance;
  if( millis() - time_last_read > 55 ) // permite a leitura do sensor com um diferenca de
tempo de pelo menos 50ms para nao sobrecarregar o sensor
  {
    sonar_distance = sonar.ping_cm();
    time_last_read = millis();
    return sonar_distance;  //return igual 0 significa que nao foi recebido retorno ou que nao
tem nada na frente dele
  }

  return -1;//return -1 indica que esta tentando ler
}

void sonar_verify_rotine()
{

  int dist_sonar = sonar_read();

  if( dist_sonar >= 0) // mostra tudo, inclusive que nao esta vendo nada ( 0 cm )
  {
    //Serial.print(dist_sonar);Serial.println(" cm");

    if( dist_sonar > 2 || dist_sonar == 0)// atualizando valid_dist_sonar
      valid_dist_sonar =  dist_sonar;
  }

}

void sonar_stop_with_control(byte safe_dist)
{
   if( valid_dist_sonar <= safe_dist && valid_dist_sonar > 2  )
  {
      digitalWrite(led_synch,HIGH);
        aplly_vel(0,0);

      if( millis() - timebuzzer > 1000 )//buzzer
      {
        //Serial.println("AQUI");
        #ifndef SEM_SOM
        digitalWrite(buzzerPin, LOW);
        delay(20);
        digitalWrite(buzzerPin, HIGH);
        delay(20);
        #endif

        timebuzzer = millis();
      }
  }
  else
      digitalWrite(led_synch,LOW);
}
```

Anexo IV – Código fonte da plataforma de controle

ARQUIVO droid_controle_v4.pde

```
/*=================> special char <=================
 *
 *   $ end char -> usado para fazer synch entre controle e o carro
 *   % usado para indicar o ServoMode
 *   @ usado para indicar controle analogico
 *
 *********************************************
 **   Controle Playstation ligado em +5V     **
 **   Controle Nunchk Wii  ligado em +3.3V   **
 *********************************************
 *
 *===> Arduino I2C pinos de comunicacao <===
 *   Analog In 4 is SDA signal
 *   Analog In 5 is SCK signal
 *
 * Desenho do conector e suas conexoes
 *  _______                 _______
 *  |     |                 |     |
 *  |     |_________________|     |
 *  |    SCK      N/C     GND     |
 *  |(azul/verde)       (preto)   |
 *  |                             |
 *  |   +3.3V     N/C     SDA     |
 *  |  (amarelo)        (branco)  |
 *   \___________________________/
 */

#include <Wire.h>      //Nunchuck
#include <PS2X_lib.h>  //for v1.6 Controle PsOne

// nesse caso ela eh usada apena para fazer o sync entre o carro e o controle
// pois os sync entre o controle e o arduino nano ocorre na funcao setup()
void(* resetFunc) (void) = 0; //declare reset function @ address 0

PS2X ps2x; // create PS2 Controller Class
//right now, the library does NOT support hot pluggable controllers, meaning
//you must always either restart your Arduino after you conect the controller,
//or call config_gamepad(pins) again after connecting the controller.

//led que indica se o controle esta funcionando ou nao
#define led_ok 13//A0

//defines do controle
#define botao_start PSB_START
#define botao_select PSB_SELECT

#define botao_cima PSB_PAD_UP
#define botao_direita PSB_PAD_RIGHT
#define botao_esquerda PSB_PAD_LEFT
#define botao_baixo PSB_PAD_DOWN

#define botao_L2  PSB_L2
#define botao_R2 PSB_R2

#define botao_L1 PSB_L1
#define botao_R1 PSB_R1

#define botao_X PSB_BLUE
#define botao_quadrado PSB_PINK
#define botao_circulo PSB_RED
#define botao_triangulo PSB_GREEN

//variaveis de controle do Controle do playstation
int error = 0;
byte type = 0;

//variaveis de ajuste
const int tempo_minimo_comunicacao = 60;
const int tempo_para_inicio_envio = 10;

char aux = '*';// comunicacao serial ... receber dados
```

```
void setup()
{
  pinMode( led_ok , OUTPUT);

  Serial.begin(9600);

  //iniciando controle do playstation
  //Conexao do dos fios: branco, azul(marrom)1,azul(marrom)2, verde
  error = ps2x.config_gamepad(12,11,10,3, !true, !true);   //GamePad(clock, command,
attention, data, Pressures?, Rumble?) check for error
  serch_error(error);
  type = ps2x.readType();
  //serch_type(type);
  delay(10);


  //inicializando controle Wii
  nunchuk_init();
  delay(20);

  nunchuk_get_data();// escrevendo no buffer de saida do wii pela primeira vez

  delay(300);
  Serial.println("Controle Wii ... OK\nSynch Completo$");
  Serial.print("$");
}

void loop()
{
  if(error == 1) //playstation nao encontrado
  {
    digitalWrite(led_ok , HIGH);
    return;
  }
  else { //DualShock Controller

    digitalWrite(led_ok , LOW); // indicando que esta tudo certo com o controle

    //lendo playstation
    delay(tempo_para_inicio_envio/2);
    ps2x.read_gamepad();

    //lendo wii
    delay(tempo_para_inicio_envio/4);
    nunchuk_get_data();

    //Hierarquia de comandos iniciais:
    //
    // 1) Nunchuck com botoes C e Z presionados
    // 2) Botao C do nunchuck  (mutuamente exclusivo em relacao ao botao Z)
    // 3) Botao Z do nunchuck
    // 4) Controle do playstation

    if( nunchuk_cbutton() && nunchuk_zbutton() )
    {
      WiiNunchkMode_CarControl();
    }
    else
    {

      if( nunchuk_cbutton() )
      {
        WiiNunchkMode_x();
      }
      else if (nunchuk_zbutton() )
      {
        WiiNunchkMode_y();
      }
      else
      {
        delay(tempo_para_inicio_envio/4);
        search_command();
      }
    }
  }
  delay(tempo_minimo_comunicacao);
```

```
}

void print_f()
{
  Serial.print("caccelx: ");
  Serial.print(nunchuk_caccelx());
  Serial.print(" caccely: ");
  Serial.print(nunchuk_caccely());
  Serial.print(" caccelz: ");

  Serial.print( nunchuk_caccelz());
  Serial.print(" joy: ");
  Serial.print(nunchuk_joyangle());
  Serial.print(" roll: ");
  Serial.print(nunchuk_rollangle());
  Serial.print(" pitch: ");
  Serial.println(nunchuk_pitchangle());

}

void serch_error(int error )
{
  if(error == 0)
    Serial.print("Controle Playstation ... OK\n");

  else if(error == 1)
    Serial.println("WARNING: NO CONTROLLER FOUND, CHECK WIRING\n");

  else if(error == 2)
    Serial.println("WARNING: CONTROLLER FOUND BUT NOT ACCEPTING COMMANDS\n");

  else if(error == 3)
    Serial.println("WARNING: CONTROLLER REFUSING TO ENTER PRESSURES MODE, MAY NOT SUPPORT
IT\n");
}

void serch_type(int type)
{
  switch(type)
  {
  case 0:
    Serial.println("UNKNOWN CONTROLLER TYPE");
    break;
  case 1:
    Serial.println("DUALSHOCK CONTROLLER FOUND");
    break;
  case 2:
    Serial.println("NO DUALSHOCK CONTROLLER FOUND");
    break;
  }
}

/*============================================================================*/
/*=========================> Nunchuk Library <================================*/
/*============================================================================*/
/*
 * File  : wiinunchuk.h V0.9
 * Author: Tim Teatro
 * Date  : Feb 2012
 *
 * Description:
 *
 *   Library to set up and poll a Wii nunchuk with Arduino. There are
 * many libraries available to do this, none of which I really liked.
 * I was fond of Tod Kurt's, but his was incomplete as it did not work
 * with knockoff nunchuks, it did not consider the least significant
 * bits of accelerometer data and didn't have any advanced functions
 * for processing the data such as calculating pitch and roll angles.
 *
 *
 * Provides functions:
 *  void nunchuk_setpowerpins()
 *  void nunchuk_init()
 *  int nunchuk_get_data()
 *  void nunchuk_calibrate_joy()
 *  inline unsigned int nunchuk_zbutton()
```

```
 *  inline unsigned int nunchuk_cbutton()
 *  inline int nunchuk_joy_x()
 *  inline int nunchuk_cjoy_x()
 *  inline int nunchuk_cjoy_y()
 *  inline uint16_t nunchuk_accelx()
 *  inline uint16_t nunchuk_accely()
 *  inline uint16_t nunchuk_accelz()
 *  inline int nunchuk_caccelx()
 *  inline int nunchuk_caccely()
 *  inline int nunchuk_caccelz()
 *  inline int nunchuk_joyangle()
 *  inline int nunchuk_rollangle()
 *  inline int nunchuk_pitchangle()
 *  void nunchuk_calibrate_accelxy()
 *  void nunchuk_calibrate_accelz()
 *
 * This library is inspired by the work of Tod E. Kurt,
 *     (http://todbot.com/blog/bionicarduino/)
 *
 * (c) 2012 by Tim Teatro
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#if (ARDUINO >= 100)
#include <Arduino.h>
#else
#include <WProgram.h>
#endif

//
// These are suitable defaults for most nunchuks, including knockoffs.
// If you intend to use the same nunchuk all the time and demand accu-
// racy, it is worth your time to measure these on your own.
//     If you may want to use various nunchuks, you may want to
// calibrate using functions
//   nunchuk_calibrate_joy()
//   nunchuk_calibrate_accelxy()
//   nunchuk_calibrate_accelz()
//
#define DEFAULT_CENTRE_JOY_X 124
#define DEFAULT_CENTRE_JOY_Y 132
#define ACCEL_ZEROX 490
#define ACCEL_ZEROY 500
#define ACCEL_ZEROZ 525

//
// Global vars are kept to a minimum.
//
uint8_t ctrlr_type[6];   // Used externally?
uint8_t nunchuk_buf[6];  // Keeps data payload from nunchuk
// Accelerometer values and callibration centres:
uint16_t accel_zerox, accel_zeroy, accel_zeroz;
// Joystick values and calibration centres:
int joy_x, joy_y, joy_zerox, joy_zeroy;

//
//
// Uses port C (analog in) pins as power & ground for nunchuk
//
void nunchuk_setpowerpins()
{
#define pwrpin PORTC3
#define gndpin PORTC2
  DDRC |= _BV(pwrpin) | _BV(gndpin);
```

```
  PORTC &=~ _BV(gndpin);
  PORTC |=  _BV(pwrpin);
  delay(100); // wait for things to stabilize
}

// Initialize and join the I2C bus, and tell the nunchuk we're
// talking to it. This function will work both with Nintendo
// nunchuks, or knockoffs.
//
// See http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1264805255
//
void nunchuk_init()
{
  Wire.begin();
  delay(1);
  Wire.beginTransmission(0x52);  // device address
#if (ARDUINO >= 100)
  Wire.write((uint8_t)0xF0);  // 1st initialisation register
  Wire.write((uint8_t)0x55);  // 1st initialisation value
  Wire.endTransmission();
  delay(1);
  Wire.beginTransmission(0x52);
  Wire.write((uint8_t)0xFB);  // 2nd initialisation register
  Wire.write((uint8_t)0x00);  // 2nd initialisation value
#else
  Wire.send((uint8_t)0xF0);   // 1st initialisation register
  Wire.send((uint8_t)0x55);   // 1st initialisation value
  Wire.endTransmission();
  delay(1);
  Wire.beginTransmission(0x52);
  Wire.send((uint8_t)0xFB);   // 2nd initialisation register
  Wire.send((uint8_t)0x00);   // 2nd initialisation value
#endif
  Wire.endTransmission();
  delay(1);
  //
  // Set default calibration centres:
  //
  joy_zerox = DEFAULT_CENTRE_JOY_X;
  joy_zeroy = DEFAULT_CENTRE_JOY_Y;
  accel_zerox = ACCEL_ZEROX;
  accel_zeroy = ACCEL_ZEROY;
  accel_zeroz = ACCEL_ZEROZ;
}

// T.T.
// Standard nunchuks use a byte-wise encryption using bit-wise XOR
// with 0x17. This function decodes a byte.
//
// This function is not needed since the way we initialize the nunchuk
// does not XOR encrypt the bits.
//
//static inline char nunchuk_decode_byte (char x)
//      {
//      x = (x ^ 0x17) + 0x17;
//      return x;
//      }

static void nunchuk_send_request()
{
  Wire.beginTransmission(0x52);// transmit to device 0x52
#if (ARDUINO >= 100)
  Wire.write((uint8_t)0x00);// sends one byte
#else
  Wire.send((uint8_t)0x00);// sends one byte
#endif
  Wire.endTransmission();// stop transmitting
}

// Gets data from the nunchuk and packs it into the nunchuk_buff byte
// aray. That array will be processed by other functions to extract
// the data from the sensors and analyse.
//
int nunchuk_get_data()
{
  int cnt=0;
```

```
  // Request six bytes from the chuck.
  Wire.requestFrom (0x52, 6);
  while (Wire.available ())
  {
    // receive byte as an integer
#if (ARDUINO >= 100)
    nunchuk_buf[cnt] = Wire.read();
#else
    nunchuk_buf[cnt] = Wire.receive();
#endif
    cnt++;
  }

  Wire.beginTransmission(0x52);// transmit to device 0x52
#if (ARDUINO >= 100)
  Wire.write((uint8_t)0x00);// sends one byte
#else
  Wire.send((uint8_t)0x00);// sends one byte
#endif
  Wire.endTransmission();// stop transmitting

  if (cnt >= 5)
  {
    return 1;   // success
  }
  return 0; // failure
}

// Calibrate joystick so that we read the centre position as (0,0).
// Otherwise, we use the default values from the header.
//
void nunchuk_calibrate_joy()
{
  joy_zerox = joy_x;
  joy_zeroy = joy_y;
}

// Returns c and z button states: 1=pressed, 0=not
// The state is in the two least significant bits of the 6th byte.
// In the data, a 1 is unpressed and 0 is pressed, so this will be
// reversed. These functions use a bitwise AND to determine the value
// and then the () ? true : false; conditional structure to pass out
// the appropriate state.
//
static inline unsigned int nunchuk_zbutton()
{
  return ((nunchuk_buf[5] >> 0) & 1) ? 0 : 1;
}

static inline unsigned int nunchuk_cbutton()
{
  return ((nunchuk_buf[5] >> 1) & 1) ? 0 : 1;
}


// Returns the raw x and y values of the the joystick, cast as ints.
//
static inline int nunchuk_joy_x()
{
  return (int) nunchuk_buf[0];
}

static inline int nunchuk_joy_y()
{
  return (int) nunchuk_buf[1];
}

// Return calibrated x and y values of the joystick.
//
static inline int nunchuk_cjoy_x()
{
  return (int)nunchuk_buf[0] - joy_zerox;
}

static inline int nunchuk_cjoy_y()
{
```

```
  return (int)nunchuk_buf[1] - joy_zeroy;
}


// Returns the raw 10-bit values from the 3-axis accelerometer sensor.
// Of the six bytes recieved in a data payload from the nunchuk, bytes
// 2, 3 and 4 are the most significant 8 bits of each 10-bit reading.
// The final two bits are stored in the 6th bit along with the states
// of the c and z button. These functions take the most significant
// 8-bits and stacks it into a 16 bit unsigned integer, and then tacks
// on the least significant bits from the 6th byte of the data
// payload.
//
// Load the most sig digits into a blank 16-bit unsigned int leaving
// two bits in the bottom ( via a 2-bit shift, << 2) for the least sig
// bits:
//     0x0000 | nunchuk_buff[*] << 2
// Add to the above, the least sig bits. The code for x:
//     nunchuk_buf[5] & B00001100
// for example selects the 3rd and 4th bits from the 6th byte of the
// payload to be concatinated with nunchuk_buff[2] to complete the 10-
// bit datum for a given axis.
//
static inline uint16_t nunchuk_accelx()
{
  return (  0x0000 | ( nunchuk_buf[2] << 2 ) +
    ( ( nunchuk_buf[5] & B00001100 ) >> 2 )  );
}

static inline uint16_t nunchuk_accely()
{
  return (  0x0000 ^ ( nunchuk_buf[3] << 2 ) +
    ( ( nunchuk_buf[5] & B00110000 ) >> 4 )  );
}

static inline uint16_t nunchuk_accelz()
{
  return (  0x0000 ^ ( nunchuk_buf[4] << 2 ) +
    ( ( nunchuk_buf[5] & B11000000 ) >> 6 )  );
}

// Returns the x,y and z accelerometer values with calibration values
// subtracted.
//
static inline int nunchuk_caccelx()
{
  return (int)(nunchuk_accelx() - accel_zerox);
}

static inline int nunchuk_caccely()
{
  return (int)(nunchuk_accely() - accel_zeroy);
}

static inline int nunchuk_caccelz()
{
  return (int)(nunchuk_accelz() - accel_zeroz);
}


// Returns joystick angle in degrees. It uses the ratio of calibrated
// x and y potentiometer readings to find the angle, zero being direct
// right (positive x) and measured counter-clockwise from there.
//
// If the atan2 function returns a negative angle, it is rotated back
// into a positive angle. For those unfamiliar, the atan2 function
// is a more inteligent atan function which quadrant the vector <x,y>
// is in, and returns the appropriate angle.
//
static inline int nunchuk_joyangle()
{
  double theta;
  theta = atan2( nunchuk_cjoy_y(), nunchuk_cjoy_x() );
  while (theta < 0) theta += 2*M_PI;
  return (int)(theta * 180/M_PI);
}
```

```
// Returns roll angle in degrees. Under the assumption that the
// only acceleration detected by the accelerometer is acceleration due
// to gravity, this function uses the ratio of the x and z
// accelerometer readings to gauge pitch. This only works while the
// nunchuk is being held still or at constant velocity with zero ext-
// ernal force.
//
static inline int nunchuk_rollangle()
{
  return (int) (  atan2( (double) nunchuk_caccelx(),
  (double) nunchuk_caccelz() ) * 180 / M_PI  );
}

// Returns pitch angle in degrees. Under the assumption that the
// only acceleration detected by the accelerometer is acceleration due
// to gravity, this function uses the ratio of the y and z
// accelerometer readings to gauge pitch.  This only works while the
// nunchuk is being held still or at constant velocity with zero ext-
// ernal force.
//
static inline int nunchuk_pitchangle()
{
  return (int) (  atan2( (double) nunchuk_caccely(),
  (double)nunchuk_caccelz() ) * 180 / M_PI  );
}

// Because gravity pulls down on the z-accelerometer while the nunchuk
// is upright, we need to calibrate {x,y} and {z} separately. Execute
// this function while the nunchuk is known to be upright and then
// execute nunchuk_calibrate_accelz() when the nunchuk is on its side.
//
void nunchuk_calibrate_accelxy()
{
  accel_zerox = nunchuk_accelx();
  accel_zeroy = nunchuk_accely();
}

// See documentation for nunchuk_calibrate_xy()
//
void nunchuk_calibrate_accelz()
{
  accel_zeroz = nunchuk_accelz();
}
```

ARQUIVO playstation_search_cmd.pde

```
void search_command()
{
  if(ps2x.Button(botao_select)) // envia comando de soft rest para o carro
  {
    select();
  }
  else if(ps2x.Button(botao_quadrado))   // parar
  {
    quadrado();
  }

  else if(ps2x.Button(botao_cima)) // andar para frente
  {
    botao_cima_command();
  }

  else if(ps2x.Button(botao_direita))// rotacionar para direita
  {
    botao_direita_command();
  }

  else if(ps2x.Button(botao_esquerda))// rotacionar para esquerda
  {
    botao_esquerda_command();
  }
```

```
  else if(ps2x.Button(botao_baixo))// andar para tras
  {
    botao_baixo_command();
  }
  else if(ps2x.Button(botao_circulo))    //automatic mode
  {
    circulo();
  }
  else if(ps2x.Button(botao_R1))
  {
    buzzer();
  }
  /*
  else if(ps2x.Button(botao_triangulo))
  {

  }
  else if(ps2x.Button(botao_X))
  {

  }*/
  // comandando o laser do servo
  if(ps2x.Button(botao_R2))
  {
    laser();
  }

  //L1 + L2
  if( ps2x.Button(botao_L1) && ps2x.Button(botao_L2) )// controla os carrinho pelo controle
analogico
  {
    L1_and_L2();
  }
  else // pode ter sido so o L1 ou so o L2
  {
    if(ps2x.Button(botao_L1))// controla os dois servos usando somente o joystick2
    {
      L1();
    }
    else if(ps2x.Button(botao_L2) )// joystick 2 controla eixo x e joystick 1 controla eixo y
    {
      L2();
    }
  }

  if(ps2x.Button(botao_start))// comando de synch do controle
  {
    start();
  }
}
```

ARQUIVO wiinunchuck_search_cmd.pde

```
//intervalos
//        Y                          X
//      cima:  -120              direta:  +105
//      baixo: +120              esquerda: -105
void WiiNunchkMode_y()
{
  int pos_y =  nunchuk_pitchangle();

  //filtrando

  if( pos_y > 35 )
    pos_y = 35;

  if( pos_y < -35 )
    pos_y = -35;

  pos_y = map( pos_y , -30 , 35 , 1 ,179 );

  Serial.print("%2,0");   Serial.print(",");
  Serial.println(pos_y);
}
```

```
void WiiNunchkMode_x()
{
  int pos_x =  nunchuk_rollangle();

  //  Serial.print("\t pos x: ");
  // Serial.println(pos_x);

  if( pos_x > 53 )
    pos_x = 53;

  if( pos_x < -20 )
    pos_x = -20;

  pos_x = map( pos_x , -20 , 53 , 1 ,179 );

  Serial.print("%2,");
  Serial.print(pos_x);
  Serial.println(",0");
}

void WiiNunchkMode_CarControl()
{
  int pos_y =  nunchuk_pitchangle();

  if( pos_y > 35 )
  {
    pos_y = 35;

  }

  if( pos_y < 0 )
  {
    pos_y = 0;

  }
  int pos_x =  nunchuk_rollangle();
  //Serial.print("\t ");
   // Serial.println(pos_x);

  if( pos_x > 54 )// direita
  {
    pos_x = 54;

  }
  if( pos_x < -27 )// esquerda
  {
    pos_x = -27;
  }

  if( pos_x == 54 )
    botao_direita_command();

  else if( pos_x == -27 )
    botao_esquerda_command();

  else if( pos_y == 35 )
    botao_cima_command();

  else if( pos_y == 0 )
    botao_baixo_command();

  else
    quadrado(); //parar sempre !
}
```

Arquivo command.pde

```
int t = 10;

void start()
{
  resetFunc();
}

void select()
{
  Serial.print("r");
}

void quadrado()
{
  Serial.print("p");
}

void botao_cima_command()
{
  Serial.print("w");
}

void botao_direita_command()
{
  Serial.print("d");
}

void botao_esquerda_command()
{
  Serial.print("a");
}

void botao_baixo_command()
{
  Serial.print("s");
}

void circulo()
{
  Serial.print("u");
}

void buzzer()
{
  Serial.print("b");
}

void laser()
{
  Serial.print("l");
  delay(200);
}

void L1()
{
    Serial.print("%"); // char para indicar servoMode

    int pos_y2 = ps2x.Analog(PSS_RY);
    int pos_x2 = ps2x.Analog(PSS_RX);

    Serial.print("0,"); // just for comunication

    Serial.print(pos_x2);

    Serial.print(",");

    Serial.println(pos_y2);
}
```

```
void L2()
{
  Serial.print("%"); // char para indicar servoMode

  int pos_y2 = ps2x.Analog(PSS_LY);
  int pos_x2 = ps2x.Analog(PSS_RX);

  Serial.print("0,");

  Serial.print(pos_x2);

  Serial.print(",");

  Serial.println(pos_y2);

}

//========> IMPLEMENTACAO COMANDO ESPECIAIS(COMPOSTOS)

void L1_and_L2()
{
  Serial.print("@");  //char para indicar controle analogico

  // joystick parado proximo de : X: 116 Y: 122
  int pos_y2 = ps2x.Analog(PSS_RY);
  int pos_x2 = ps2x.Analog(PSS_RX);

  float conta = sqrt(pow( (pos_x2 - 122),2) + pow( (pos_y2 - 122),2) );
  /*
   Serial.print("Conta ");
   Serial.println(conta);
   */
  if( conta > 26 ) // folga inicial -> folga do joystick
  {
    int correcao_x = abs(255 -pos_x2);
    int correcao_y = abs(255 -pos_y2);

    //faixa de comando para cima 156 - 255
    //faixa de comando para baixo 103 - 0
    //faixa de comando para direita 93 - 0
    //faixa de comando para esquerda 161 - 255

    //comando de direita e esquerda
    //sentido = 0 -> parado
    //sentido = 1 -> ESQUERDA
    //sentido = 2 -> DIREITA
    //sentido = 3 -> FRENTE
    //sentido = 4 -> FRENTE ESQUERDA
    //sentido = 5 -> FRENTE DIREITA
    //sentido = 6 -> TRAS
    //sentido = 7 -> TRAS ESQUERDA
    //sentido = 8 -> TRAS DIREITA
    int sentido = determinar_sentido( correcao_x,correcao_y);

    send_command_analog(sentido , correcao_x, correcao_y);

    /*
        Serial.print(correcao_x);//invertendo o sentido do eixo
     Serial.print(",");
     Serial.println(correcao_y);//invertendo o sentido do eixo
     //delay(700);
     */
  }
  else
  {
    //Serial.print("PARADO ");
    Serial.println("0,0,0"); //sentido

  }
}
```

Referências

[1] Arduino.

Disponível em:< http://arduino.cc/ >

Acesso em 16 de Abril. 2013.

[2] Brasil Robotics.

Disponível em: < http://brasilrobotics.blogspot.com.br/ >

Acesso em 16 de Abril. 2013.

[3] Luck Larry - website

Disponível em: < http://luckylarry.co.uk/arduino-projects/obstacle-avoidance-robot-build-your-own-larrybot/ >

Acesso em 16 de Abril. 2013.

[4] Monk, Simon. *30 Arduino Projects for the Evil Genius*. New York: McGraw-Hill, 2010.

[5] C. Cardeira and J. S. Da Costa "A low cost mobile robot forengineering education", *IECON Proc.*, pp.2162 -2167 2005

[6] J. Hau-Shiue and L. Kai-Yew “Design and control of a two-wheel self-balancing robot using the arduino microcontroller board”, in Control and Automation (ICCA), 2013 10th IEEE International Conference on, pp. 634 – 639.

[7] P. Dong, G. Bilbro, and M. Chow, "Controlling a path-tracking unmanned ground vehicle with a field-programmable analog array," in Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on. IEEE, pp. 1263-1268.

[8] Atmel.

Disponível em: < http://www.atmel.com/devices/atmega328.aspx >

Acesso em 16 de Abril. 2013.

[9] HCSR04 datasheet.

Disponível em: < http://www.micropik.com/PDF/HCSR04.pdf >

Acesso em 16 de Abril. 2013.

[10] Bill Porter. Playstation 2 controller Arduino Library

Disponível em: < http://www.billporter.info/2010/06/05/playstation-2-controller-arduino-library-v1-0/ >
Acesso em 16 de Abril. 2013.

[11] Tim Teatro. A libray for using the Wii Nunchuck in Arduino Sketches.
Disponível em: < http://www.timteatro.net/2012/02/10/a-library-for-using-the-wii-nunchuk-in-arduino-sketches/ >
Acesso em 16 de Abril. 2013

[12] Zuchi. Módulo Serial Bluetooth.
Disponível em: < http://www.zuchi.com.br/download/0400402/0400402.zip >
Acesso em 16 de Abril. 2013.

[13] APC220. Datasheet.
Disponível em: < http://www.dfrobot.com/image/data/TEL0005/APC220_Datasheet.pdf >
Acesso em 16 de Abril. 2013.

[14] Motor DC, Suntek Store.
Disponível em :< http://www.suntekstore.co.uk/product-14003464-plastic_dc_gear_motor_for_smart_car_-_yellow.html >
Acesso em 18 de Abril. 2013.

[15] L298N. Datasheet.
Disponível em: < http://www.datasheetcatalog.org/datasheet/SGSThomsonMicroelectronics/mXxwur.pdf >
Acesso em 16 de Abril. 2013.

[16] Ponte H L298N. Deal Extreme
Disponível em: < http://dx.com/p/l298n-stepper-motor-driver-controller-board-for-arduino-120542 >
Acesso em 18 de Abril. 2013.

[17] Arduino. Timer2
Disponível em: < http://playground.arduino.cc/Main/MsTimer2 >
Acesso em 16 de Abril. 2013.

[18] Servo 9g. Deal Extreme
Disponível em < http://dx.com/p/towerpro-sg90-9g-mini-servo-with-accessories-12859 >
Acesso em 18 de Abril. 2013.