UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

Controle de robôs móveis utilizando Kinect

MARCOS VINICIUS DA CRUZ CORREA

Orientador: Maximilian Luppe

São Carlos
2014

Marcos Vinicius da Cruz Correa

Controle de robôs móveis utilizando Kinect

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

Curso de Engenharia Elétrica com ênfase em
eletrônica

ORIENTADOR: Maximillian Luppe

São Carlos
2014

da Cruz Correa, Marcos Vinicius
D824c Controle de robôs móveis utilizando Kinect / Marcos Vinicius da Cruz Correa; orientador Maximilian Luppe. São Carlos, 2014.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Kinect. 2. Robô. 3. Arduino. I. Título.

FOLHA DE APROVAÇÃO

Nome: Marcos Vinicius da Cruz Correa

Título: "Controle de robôs móveis utilizando Kinect"

Trabalho de Conclusão de Curso defendido e aprovado em 27/11/2014,

com NOTA 9,3 (nove, três), pela Comissão Julgadora:

Prof. Dr. Maximiliam Luppe - (Orientador - SEL/EESC/USP)

Prof. Dr. Marcelo Andrade da Costa Vieira - (SEL/EESC/USP)

Mestre Henrique Cunha Pazelli - (Opto Eletrônica S/A)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Aos meus pais Marcelo e Joana, aos meus irmãos Mauricio e Izabela e aos meus amigos de Rep Tocaia.

Agradecimentos

- A minha família pelo apoio e força que me ajudaram a conquistar esse sonho.

- Aos amigos pelo companheirismo e lealdade.

- Ao Departamento de Engenharia Elétrica da Escola de Engenharia Elétrica de São Carlos (EESC) – Universidade de São Paulo.

- Ao professor doutor Maximilian Luppe por toda ajuda, orientação e dedicação durante a minha formação.

Resumo

CORREA, M. V. C. Controle de robôs móveis utilizando Kinect. 2014. 77f. Trabalho de Conclusão de Curso – Departamento de Engenharia Elétrica da escola de Engenharia de São Carlos, São Carlos: Universidade de São Paulo, 2014.

A alta velocidade de avanço da tecnologia, seja na criação de novos recursos ou no método de produção, possibilita que tecnologias, que antes eram utilizadas apenas por grandes companhias ou grandes centros tecnológicos, possam ser aplicadas em projetos mais simples. O controle de robôs de forma remota é um exemplo, essa aplicação é de grande impacto em automação e pode ser utilizada desde controle de braços mecânicos a controle de robôs móveis. Este trabalho introduz algumas plataformas robóticas simples que são controladas de formas diferentes e possuem distintos propósitos, demonstrando a facilidade de criação desse tipo de tecnologia e como ela está se difundindo facilmente, entretanto é dado um maior foco na utilização do sensor Kinect para controle remoto de um robô construído utilizando um Arduino UNO. Esse projeto é composto em montagem do robô, aprendizado da utilização da comunicação sem fio Bluetooth, utilização do sensor Kinect para captura de gestos, criação dos controles entre usuário e robô e comunicação entre as plataformas Kinect e Arduino. Por fim foi possível controlar remotamente o robô utilizando duas formas distintas de controle relacionadas a gestos capturados pelo Kinect, o resultado foi satisfatório com respostas rápidas e eficientes do robô permitindo uma interface dinâmica entre o sistema e o usuário, mas com algumas limitações a movimentos bruscos.

Abstract

CORREA, M. V. C. Control of mobile robots using Kinect. 2014. 77f. Course Conclusion Work – Electrical Engineering – São Carlos Engineering School, São Carlos: University of São Paulo, 2014.

The high speed of advancement of technology, on the creation of new resources or on production methods, enables technologies, that were previously used only by large companies or large technology centers, can be applied to simpler designs. The controlling robots remotely is an example, this application is of great impact in automation and can be used from control of mechanical arms to control of mobile robots. This paper introduces some simple robotic platforms that are controlled in different ways and have different purposes, demonstrating the facility of creating this type of technology and how it is spreading easily, however is given a greater focus on the use of the Kinect sensor for control remotely a robot built using an Arduino UNO. This project consists in assembling the robot, the learn of the use of wireless Bluetooth, utilization of the Kinect sensor to capture gestures, creation of the controls between user and robot and communication between the Kinect and Arduino platforms. Finally was possible to remotely control the robot using two different forms of control related to gestures captured by Kinect, the result was satisfactory with quick and efficient responses from the robot allowing a dynamic interface between the system and the user, but with some limitations to sudden movements .

Sumário

Lista de figuras

Lista de tabelas

1. Introdução

A robótica é um tema de repercussão na atualidade, que gera grandes expectativas para um futuro próximo e libera a imaginação do homem que extrapola do mundo teórico da engenharia e computação para o mundo dos filmes e livros que imaginam as utilizações e consequências dessa tecnologia.

Inicialmente a robótica foi idealizada para fins de automatização de processos industriais, em que as empresas procuravam se equipar com máquinas que fossem 00capazes de produzir, automaticamente, determinadas tarefas. E assim foi por muito tempo à única utilização dessa forma de tecnologia, com alto custo de implementação e de manutenção. No entanto o desenvolvimento e melhoria da tecnologia, como um todo, se expandiu rapidamente, seus principais componentes de fabricação se tornaram baratos e surgiu diversificação de seu uso ganhando expressão significativa no dia a dia das pessoas. Hoje qualquer pessoa pode ter conhecimentos mínimos, acesso fácil à informação e condições de se “aventurar“ com a robótica.

É possível encontrar vários projetos que utilizam a robótica com diversas funcionalidades distintas, a Figura 1 ilustra uma plataforma robótica que se movimenta de forma autônoma e evita colisões utilizando um sonar.

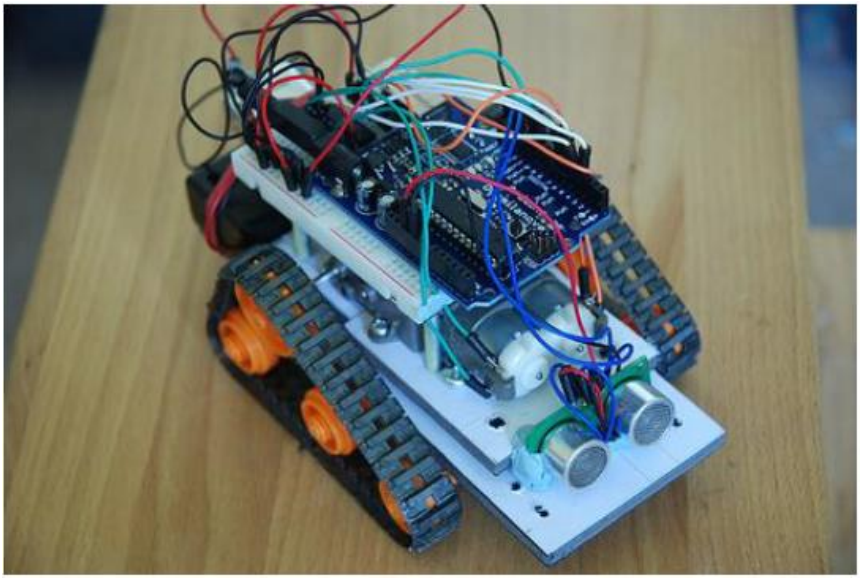

Figura 1 - Plataforma robótica móvel [1]

A Figura 2 apresenta um robô que utiliza um Arduino para controle e um sonar para evitar colisões frontais, no entanto essa plataforma se diferencia do exemplo anterior por ser controlada remotamente, para controle é utilizado um controle do videogame PlayStation ou WII (Figura 3).

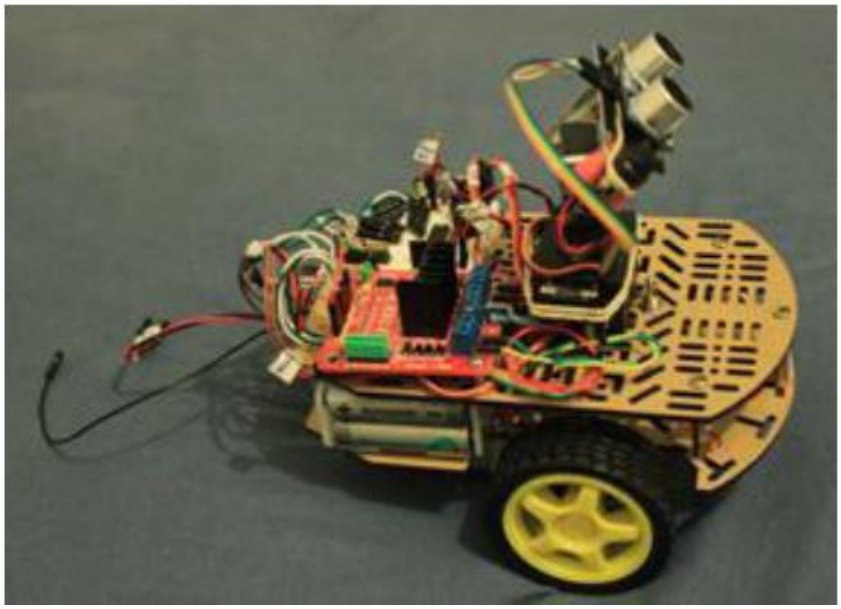

Figura 2 - Plataforma robótica controlada remotamente [2]

Figura 3 - Controle da plataforma robótica [2]

Existem vários competições que agrupam pessoas de diversas idades que visam disseminar e ampliar o conhecimento sobre a robótica, um exemplo desse tipo de evento é o LARC (*Latin American Robotics Competition*), o robô da Figura 4 foi um dos participantes da edição da competição de 2014 e é um robô autônomo que utiliza um Ardunio e uma Raspberry Pi para controle, diversos sensores e uma câmera, o objetivo desse robô é identificar cubos de uma determinada cor em um ambiente, recolher o máximo possível e posicionar em um outo robô aquático para fazer o transporte.

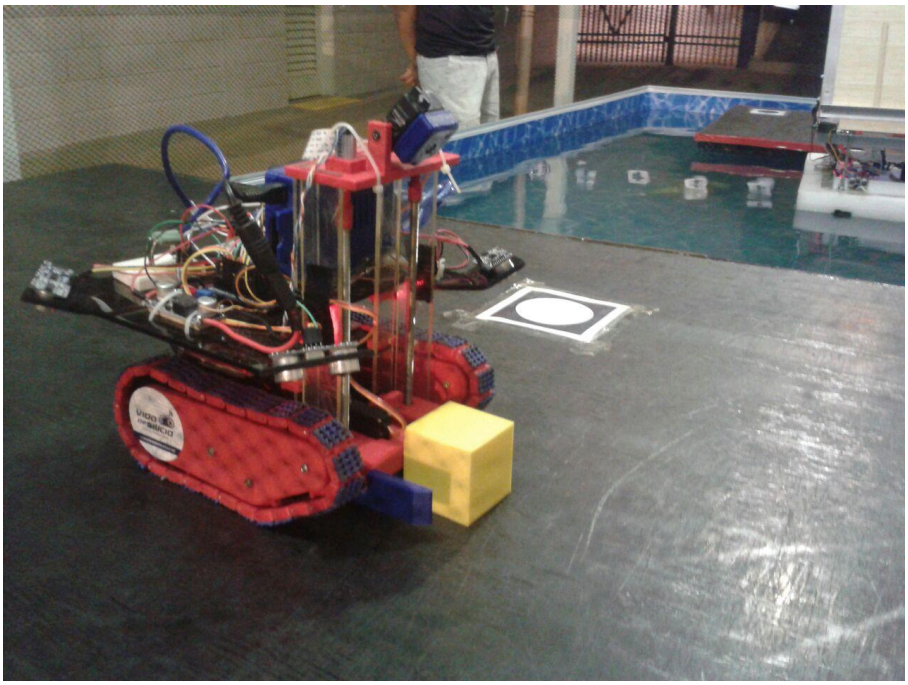

Figura 4 - Robô participante da competição LARC 2014

Esse tipo de iniciativa e a diversidade de projetos que se pode encontrar que utilizam robótica demonstra como essa prática se expandiu para além das áreas comerciais e conquistou as universidades, escolas e a qualquer individuo que se interesse.

Esse projeto tem como intuito o controle de um robô móvel utilizando o sensor Kinect da Microsoft, criado para utilização em jogos com movimentos pelo console Xbox360, o sensor capta os movimentos do usuário e passa para o computador por cabo USB, este por sua vez possui um programa com alguns movimentos pré-estabelecidos que se relacionam com comandos que devem ser enviados ao robô, dessa forma, quando um movimento é reconhecido o seu comando correspondente é enviado via Bluetooth para o controlador do robô, este controlador é um Arduino UNO, ele recebe os comandos e identifica qual deve ser a ação aplicada no robô e então atua nos motores fazendo com

que ele se movimente, relacionando assim o movimento do usuário com o movimento do robô, todo o processo de funcionamento está ilustrado na figura 5.

No capítulo 2 serão apresentados os materiais e métodos utilizados no projeto, definindo as características e funções dos componentes utilizados e conexões feitas entre eles. Serão apresentados também os fluxogramas dos softwares criados introduzindo o funcionamento dos programas.

No capítulo 3 será demonstrado toda a implementação dos *softwares* de controle que foram feitas para atingir o resultado pretendido, explicando toda a lógica e sequência de criação.

No capítulo 4 serão apresentados os resultados, demonstrando que o objetivo foi alcançado e que todo o sistema tem boa resposta aos estímulos do usuário.

No capítulo 5 foi feita uma análise de todo o projeto, expondo os pontos em que foi necessário executar algumas mudanças inesperadas ao decorrer da criação do sistema e qual foram essas mudanças. Nessa seção também são citados pequenos erros que podem ocorrer no sistema implementado e idealiza algumas formas de correção.

No capítulo 6 foi feita uma conclusão sobre o projeto, discutindo os conhecimentos adquiridos e utilizados durante o trabalho. Foi confirmado que o objetivo foi atingido satisfatoriamente tendo como resultado final um sistema que possui o funcionamento pretendido.

2. Materiais e Métodos

Conforme foi dito na introdução esse projeto tem como intuito utilizar o sensor Kinect da Microsoft para capturar e reconhecer movimentos de um usuário e atribuir a eles comandos, os quais serão enviados via comunicação Bluetooth para o controlador do robô, um Arduino UNO, que traduz os comandos recebidos e atua nos motores do robô fazendo com que ele se movimente, criando assim um sistema que controla um robô móvel através de gestos feitos por um usuário.

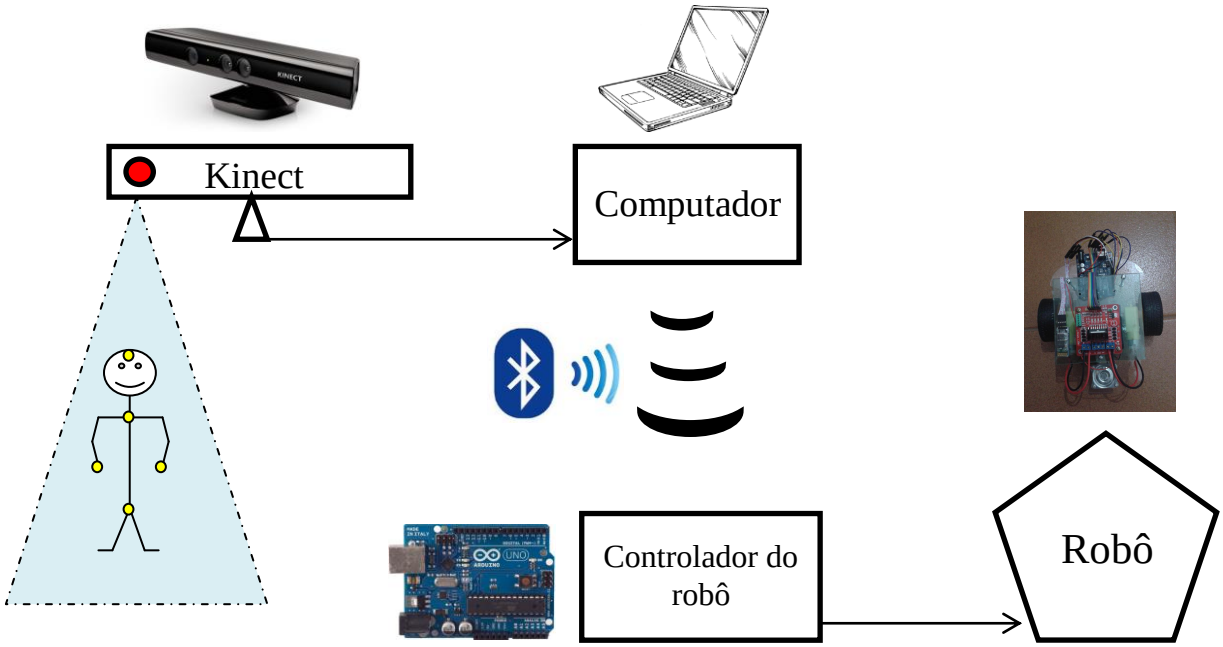

Figura 5 - Controle do Robô Utilizando um Sensor Kinect

Para criar a lógica de reconhecimento de movimentos do usuário será utilizado o *software* Microsoft Visual C#, no qual será usada a linguagem de programação orientada a objeto C#, e o código que interpretará o comando enviado remotamente a fim de controlar o robô será escrito na linguagem e plataforma própria do Arduino.

2.1. Robô

Para a montagem da plataforma é necessário as peças que correspondem a sua movimentação e força, como as rodas e os motores, além disso, é necessário o circuito de controle que nesse caso será um Arduino UNO e ainda do circuito de transmissão e recepção de dados, ou seja, o *shield* do Bluetooth, também se deve utilizar uma eletrônica de controle para os motores, para isso foi escolhida uma ponte H e por fim como alimentação haverá quatro pilhas do tipo AA de 1,5V. Todos esses componentes serão integrados em um chassi e corretamente interligados, de acordo com a Figura17 e Figura 18, para criar a plataforma robótica móvel.

- Arduino UNO
- 2 Motores DC
- 2 Rodas
- 1 Roda de rolamento
- 1 Chassi
- *Shield* Ponte H
- *Shield* Bluetooth
- 4 Pilhas

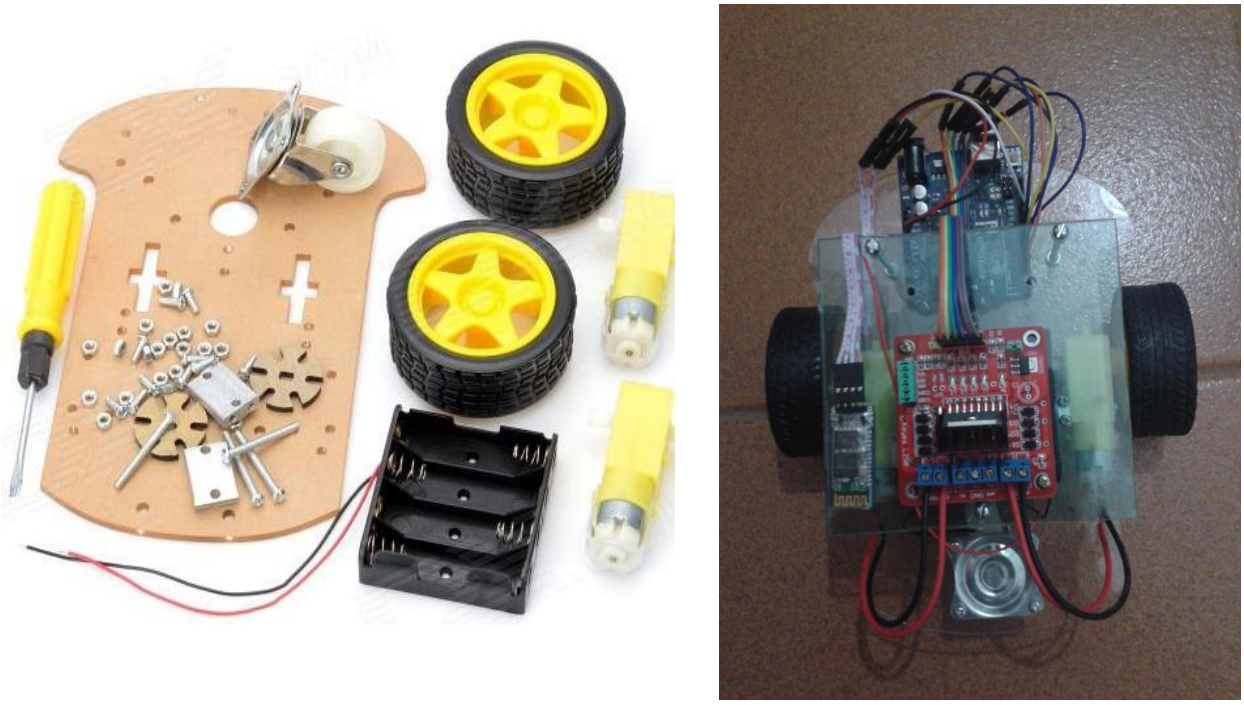

Figura 6 - Robô

2.2. Kinect

O Kinect[3] tem mudado a forma como as pessoas interagem com a tecnologia, criando uma forma mais natural e espontânea com a utilização de simples movimentos ou por comandos de voz.

O baixo custo ligado ao alto desempenho do sensor Kinect fez com que esse dispositivo, criado pela Microsoft para fins de entretenimento com o videogame Xbox360, ganhasse espaço nas áreas de computação, eletrônica e engenharia, onde desenvolvedores tem utilizado os seus muitos recursos para criar interações entre homem e máquina de uma forma mais direta.

O Kinect é composto por diversos sensores que trabalham em conjunto recolhendo dados do usuário e processando por diversos algoritmos robustos como o de rastreamento do esqueleto. Ele possui um sistema de imageamento 3D que é composto por dois dispositivos, um projetor infravermelho e um detector infravermelho. O funcionamento deles em conjunto permite a criação de imagens com informações de posição e profundidade da seguinte forma, o projetor emite uma rede de pontos infravermelho que sai do Kinect em projeção cônica (Figura 7) eles incidem na superfície dos objetos presentes no ambiente e refletem parte da luz, o detector capta essas reflexões e um processamento no Kinect calcula o tempo que cada ponto levou para retornar ao dispositivo (Figura 8), com essa medida é possível então estimar a distância do sensor referente a cada ponto criando um vetor de dados tridimensional que vai de 40 a 2000 pontos, o valor de cada ponto se relaciona diretamente com a distância em milímetros[4]. Cada distância corresponde a uma valor na escala de cinzas, o preto representa que não há valor válido de distância para o pixel, isso pode ocorrer por três motivos, primeiro, o ponto está muito distante e não se pode determinar com precisão a distância, segundo, o ponto está muito perto o que se caracterizaria por um ponto cego da câmera e do projetor devido a suas limitações, ou terceiro, houve pouca reflexão oriunda desse ponto, o que pode ocorrer devido a superfícies polidas [5].

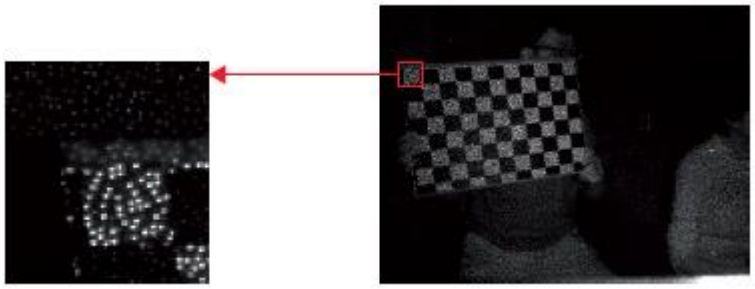

Figura 7 - Rede de pontos captados pelo detector infravermelho[4]

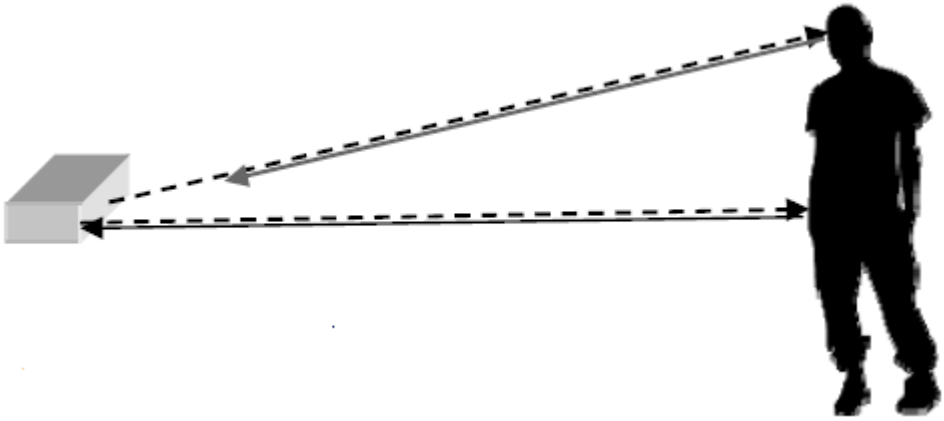

Figura 8 - Mecanismo de detecção de profundidade[4]

O Kinect possui também uma câmera colorida responsável por captar as imagens coloridas que são utilizadas para reconhecimento de movimentos e para projeção em tela e ainda é composto por quatro *arrays* de microfones, os quais captam o áudio para utilização nos comandos de voz, utilizando os dados de imagens ele também é capaz de fazer reconhecimento facial [5]. O Kinect possui também um motor que possibilita o sensor mudar sua inclinação, utilizado para fins de calibração (Figura 9).

Figura 9 - Kinect [5]

Algumas características importantes do Kinect estão listadas na sequência:

- Campo de visão (Figura 10)
  - Horizontal: 57º.
  - Vertical: 43º.
  - Inclinação: 27º.
  - Sensor de profundidade: 1,2m – 3,5m.

Figura 10 - Campo de Visão do sensor Kinect [6]

- Fluxo de dados
  - 320 x 240, 16-bits, sensor profundidade, 30 frames/sec.
  - 640 x 480, 32-bits, câmera colorida, 30 frames/sec.
  - 16-bit áudio, 16KHz.

- *“Skeletal Traking System”*
  - Detecta até 6 pessoas, somente duas ficam ativas.
  - Detecta 20 juntas por pessoa ativa.

O Kinect possui seu próprio processamento de imagem conhecido como o Primesense’s PS1080-A2 *System on Chip* (SoC), processador que trabalha com as imagens capturadas pela câmera RGB e infravermelha [7] (Figura 11).

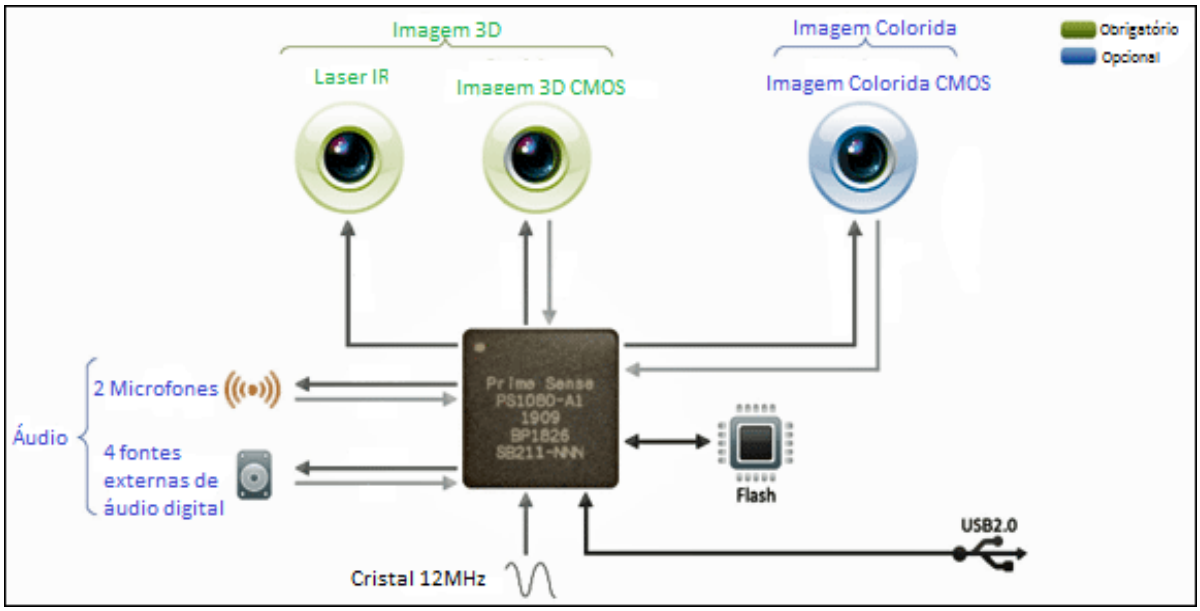

Figura 11 - Esquemático da Arquitetura do Sistema do Kinect [7]

A grande inovação por traz do sensor Kinect está no seu algoritmo de rastreamento de esqueleto, em que o corpo de uma pessoa é representado por um número de juntas às quais representam partes do corpo como cabeça, mão, ombro e braços. Para conseguir chegar a um algoritmo eficaz e eficiente a equipe da Microsoft, liderada por Jamie Shotton, escolheu fazer um reconhecimento das partes do corpo por pixel como um passo intermediário. Eles consideraram a segmentação das imagens de profundidade como uma classificação por pixel. Aavaliar cada pixel separadamente evitou a procura por combinações entre diferentes juntas do corpo. A equipe gerou diversas imagens de profundidade de pessoas de diversos portes e tamanhos em diversas poses elas foram amostradas para um banco de dados de captura de movimentos o qual foi utilizado como dados de treinamento, em seguida foi realizado um treino de decisão de classificação aleatória, o que evita super ajuste por utilização de muitas imagens de

treino. Por fim, modelos espaciais inferidas pela distribuição por pixel são calculadas utilizando variação média resultando em um plano composto pelas juntas em 3D. A sequência do rastreamento do esqueleto pelo Kinect está ilustrada na Figura 12, primeiramente se realiza a classificação das partes do corpo por pixel, depois se supõem as juntas do corpo achando um centroide global de probabilidade de massa através de variação média, em seguida se mapeia as juntas do esqueleto para então traçar a forma de um esqueleto considerando continuidade temporal e conhecimento prévio de um banco de dados de traino [5].

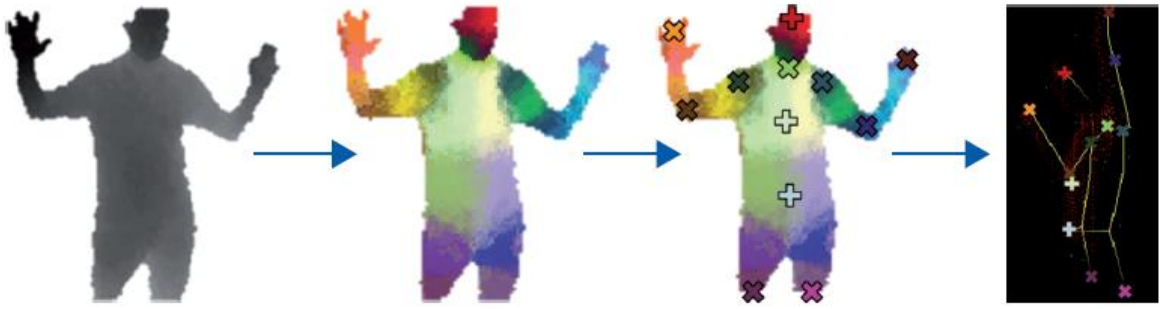

Figura 12 - Rastreamento do esqueleto [5]

2.3. Arduino **UNO**

O Arduino[8] é uma plataforma de *software* aberto (*open-core*) utilizado amplamente para prototipagem, cujo objetivo é a flexibilidade, é de fácil aprendizado, com muitas referências e exemplos online e em livros, possui diversas bibliotecas já implementadas pela equipe Arduino ou por usuários que as disponibilizam de forma gratuita, isso acelera a criação de novos projetos e evita o retrabalho.

O microcontrolador da plataforma é programado utilizando uma linguagem padrão do Arduino que é baseada em *Wiring[9]* e a plataforma de desenvolvimento é baseada em *Processing[10]*. Ambos os sistemas tem em comum os seguintes aspectos:

- São funcionais em Linux/ GNU/ Mac OS/ Windows.
- Mais de 100 bibliotecas.
- Boa documentação.

O *Wiring* é uma linguagem de programação *open-source* para *framework* para microcontroladores, ele permite criar programas com o intuito de controlar diversos periféricos e gerar experiências físicas ou iterativas. Foi criada por designers com a ideia de encorajar os iniciantes a interagir com experts e trocarem ideias e conhecimentos. Os microcontroladores da Atmel [11] são compatíveis com o *Wiring* e eles que são utilizados nas placas Arduino.

O *Processing* é uma linguagem de programação e uma plataforma de desenvolvimento. Foi idealizado para facilitar o aprendizado de *software* com artes visuais e também do aprendizado da arte visual utilizando tecnologia. Inicialmente criado como um programa que funcionava como um caderno de desenho para ensinar fundamentos de linguagem de programação de forma visual.

Utilizando esses dois sistemas, juntamente com microcontroladores da Atmel, foi criado o sistema do Arduino.

Nesse projeto será utilizada a placa Arduino UNO (Figura 13) que possui as seguintes especificações:

- Microcontrolador ATmega328.
- 14 pinos de entrada/saída digitais, no qual 6 podem ser usados como saídas PWM.
- 6 pinos de entradas analógicas.
- Gerador de clock de 16MHz.
- Conexão USB.
- Um Botão de Reset.
- Tensão de operação de 5V.
- Tensão de alimentação de 7-12V.
- Flash Memory (ATmega328) 32Kb.
- SRAM (ATmega328) 2Kb.
- EEPROM (ATmega328) 1Kb.

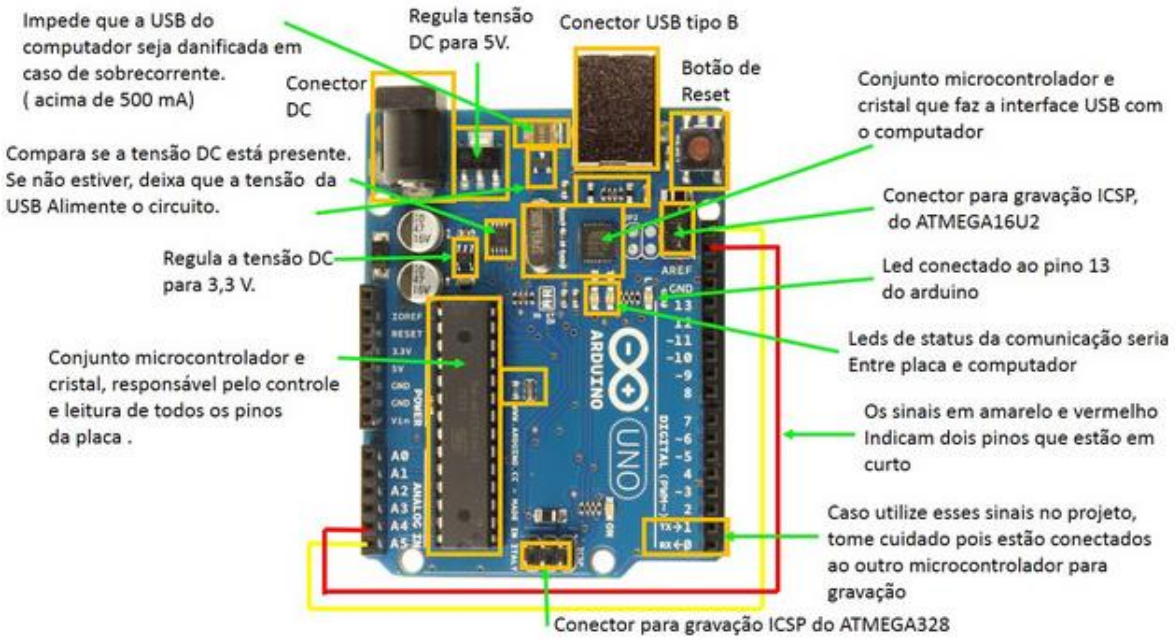

Figura 13 - Arquitetura da Placa Arduino Uno [11]

Diferentes microcontroladores são utilizados para as diversas plataformas Arduino, nesse projeto onde será utilizado o Arduino UNO o microcontrolador é o ATmega328 (Figura 14), é um componente que trabalha com 8 bits com arquitetura RISC e encapsulamento DIP28. Ele possui uma memória Flash de 32Kb (512 bytes são utilizados pro *bootloader*), 2Kb de Ram e 1Kb de EEPROM, ele pode operar em até 20MHz, no entanto o Arduino UNO funciona com 16Mhz. Possui 28 pinos, nos quais 23 podem ser utilizados como I/O. Esse microcontrolador pode operar com tensões baixas, como 1,8V, no entanto com o limitante de clock de 4MHz, tem como periféricos uma USART de 250kbps, uma SPI que vai até 5MHz e uma I2C que pode ir até 400KHz. É composto ainda por um comparador analógico interno, timers e 6PWMs. A corrente máxima por pino é 40mA, todavia a soma da corrente no CI não pode ser superior a 200mA [12]. A placa possui também um ATmega16U2 que serve como “ponte” entre a porta USB do computador e a porta serial principal do processador. A seguir estão listadas suas características:

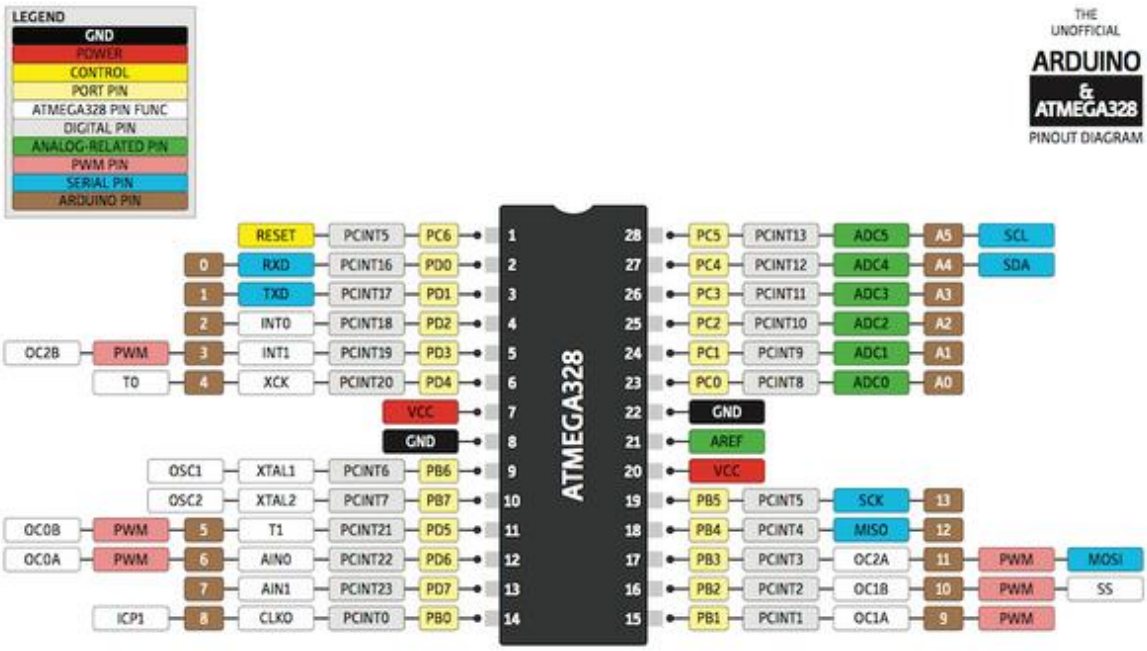

Figura 14 - ATMEGA328 [12]

- Arquitetura Harvard
- 8 bits
- RISC
- Flash Memory 32Kb
- SRAM 2Kb
- EEPROM 1Kb
- 32 registradores de uso geral
- 3 temporizadores/contadores
- USART
- Comunicação SPI
- 6 conversores AD de 10 bits
- Tensão de operação entre 1,8 - 5,5V

2.4. Bluetooth

O Bluetooth (Figura 15) é uma tecnologia de comunicação sem fio, foi criado em 1994 por um grupo de engenheiros da empresa sueca Ericsson, essa ideia foi idealizada como alternativa aos cabos de dados serial RS-232. Ela tornou possível à troca de dados entre pequenas distâncias usando transmissão a radio. Ele opera em uma banda de 2,4 a 2,485 GHz, usando espectro de dispersão, sinal *full-duplex* a uma velocidade de 1600 hops/sec [13].

Em 1998 as companhias Ericsson, Intel, Nokia, Toshiba e IBM se juntaram para formar o Grupo de Interesse Especial Bluetooth (*Bluetooth Special Interest Group* - SIG), dessa forma o Bluetooth pertence a todas as empresas que trabalham em conjunto para preservar e melhorar a tecnologia.

Alguns outros dispositivos também utilizam as ondas de radio para transmissão de dados, exemplos são a Televisão, radio FM e celulares, a diferença é que esses dispositivos transmitem para longas distâncias, enquanto que o Bluetooth trabalha somente na Rede de Área Pessoal (*Personal Area Network* - PAN) [14].

Essa tecnologia foi criada para facilitar a utilização de alguns periféricos a fim de substituir os cabos, como em fone de ouvido e teclado, no entanto com a sua popularidade, baixo custo e bom desempenho ganhou o mercado em diversas áreas como a automobilística, onde é possível realizar ligações telefônicas através do veículo, e saúde com monitoramento e aquisição de dados para uso médico.

Conexões entre dispositivos eletrônicos com Bluetooth ativo permite a comunicação em pequenas distâncias, essas conexões são conhecidas como *Piconets*. *Piconets* são estabelecidos dinamicamente e automaticamente quando dispositivos com a tecnologia ativa estão dentro do raio de alcance.

Os dispositivos em um *Piconet* podem se comunicar simultaneamente com até sete dispositivos dentro de um único *Piconet,* e ainda, cada dispositivo também pode pertencer a vários *Piconets* ao mesmo tempo, o que significa que existem diversas formas de conexões entre os dispositivos Bluetooth [15].

2.4.1. Bluetooth especificações

- Espectro: a tecnologia Bluetooth opera na indústria, ciência e área médica em uma banda de 2,4 a 2,485GHz, usando espectro de dispersão, sinal *full-duplex* e velocidade de 1600 hops/sec [15].

- Interferência: a capacidade de salto de frequência adaptativa da tecnologia Bluetooth (*Adaptative frequency hopping* - AFH) foi criada para minimizar a interferência entre dispositivos que utilizam a banda de 2,4GHz. Isso funciona da seguinte maneira, a tecnologia detecta outros dispositivos que estão utilizando o espectro e evita as frequências que eles estão utilizando. Esse pulo adaptativo entre 79 frequências com intervalos de 1 MHz gera uma grande imunidade a interferências, para os usuários isso mantem a performance mesmo se outras tecnologias estão sendo utilizadas ao mesmo tempo do Bluetooth [15].

- Alcance: o alcance varia de acordo com a classe sendo utilizada, existem três [15].

  - Classe 3 : alcance de 1 metro a 3 metros
  - Classe 2: alcance de até 10 metros – mais comum e utilizado em celulares
  - Classe 1: alcance de até 100 metros – utilizado na indústria

- Potência: A classe mais utilizada é a classe 2 a qual utiliza 2,5mW de potência, uma das principais características da tecnologia é o baixo consumo [15].

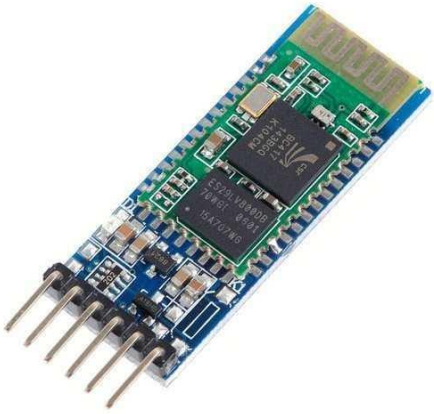

Figura 15 - Shield Bluetooth para Arduino

2.5. Linguagem de programação C#

À medida que houve um avanço na computação, com o passar dos anos, tanto o *hardware* como o *software* foram sofrendo mudanças e adaptações. No caso da programação as linguagens de alto nível ganharam grande atenção por facilitar a criação e implementação de *softwares*. A linguagem C é um grande exemplo de linguagem de alto-nível muito utilizada na atualidade.

No entanto a alta demanda de *softwares* mais robustos gerou a necessidade de acelerar a construção de programas, a alternativa encontrada foi um novo tipo de programação, que deixaria de ser estruturada e passaria a ser orientada a objeto, exemplos são o C++ e o Java.

Sem muita demora surgiram novas variáveis no cotidiano dos programadores, a primeira foi a *Word Wide Web*, devido a isso se tornou necessário à criação de *softwares* que funcionassem nos PCs e também de aplicativos baseados na web para serem acessados e usados via internet, a segunda variável foi o aparecimento dos dispositivos móveis. Para tratar dessas necessidades a Microsoft criou a iniciativa .NET e a linguagem de programação C#.

A plataforma .NET possibilita que aplicativos baseados na web possam ser distribuídos para uma grande variedade de dispositivos e para PCs, a linguagem de programação C# foi desenvolvida por uma equipe liderada por Anders Hejlsberg e Scott Wiltamuth, projetada especificamente para a plataforma .NET, ela tem raízes em C, C++ e Java.

O C# é uma linguagem de programação visual dirigida por eventos e orientada a objeto, onde os programas são criados usando-se uma IDE (*Integrated Development Enviroment* - ambiente de desenvolvimento integrado). Utilizando a IDE o desenvolvedor cria, executa, testa e depura os programas, o que acelera e facilita a criação dos *softwares*. O processo de criação rápida de aplicativos usando uma IDE é denominado RAD (*Rapid Application Development* – desenvolvimento rápido de aplicativos) [16].

2.5.1. Programação orientada a objeto

A programação orientada a objeto é um esquema de empacotamento que facilita a criação de unidades de *software* significativa. Essas unidades são grandes e focalizadas em áreas de aplicação específica. Os objetos têm propriedades, ou seja, atributos (cor,

tamanho e peso) e executam ações, isto é, possuem comportamentos (comer, dormir e correr). As “Classes” representam grupos de objetos relacionados, um exemplo seria a “Classe” carro, mesmo que carros individuais variem de marca ou modelo todos pertencem a mesma “Classe”. Uma “Classe” especifica o formato geral de seus objetos, as propriedades e ações de um objeto dependem de sua classe.

Um grande problema observado pelos desenvolvedores foi à perda de tempo criando e recriando *softwares* com características semelhantes para projetos distintos. Com a tecnologia de objetos, as entidades de *software* (objetos) podem ser reutilizadas em futuros projetos, dessa forma, trabalhar com bibliotecas de componentes reutilizáveis reduz a quantidade de trabalho na criação de programas. A linguagem C# utiliza a biblioteca de classe da plataforma .NET, conhecida como FLC *(.NET Framework Class Library*) [16].

2.5.2. Visual Studio

O Visual Studio . NET é o IDE da Microsoft para criação, documentação, execução e depuração de programas escritos em diversas linguagens de programação .NET. O Visual Studio .NET também oferece ferramentas de edição para manipular vários tipos de arquivos.

A versão utilizada nesse projeto é o Visual Studio Express 2010 que utiliza a linguagem C#, é importante ressaltar esse ultimo detalhe, pois existe o *software* para linguagem C e Basic também, que igualmente podem ser utilizados para criação dessa aplicação, no entanto foi optado pela utilização da linguagem C# por maior facilidade de compreensão e implementação. A IDE pode ser obtida de forma gratuita no endereço eltrônico [17].

2.6. Eletrônica do motor

Para Controle dos motores DC é necessário à construção de um *hardware* que possibilite o direcionamento (rotação horaria e anti-horária) dos motores, para que assim se possa movimentar o robô no sentido desejado (frente, atrás, esquerda e direita). Para isso foi escolhido um circuito simples e bem conhecido para essa aplicação, a ponte H (Figura 17) [18].

A ponte H é composta pelo driver L298N cujo circuito está na Figura 16. Para controlar a direção do motor se deve manipular a direção da corrente que passa por ele,

para isso basta chavear corretamente os transistores para modificar como a corrente passa através do motor DC.

O circuito controla dois motores, assim ele se espelha, ou seja, são dois circuitos idênticos, um para cada motor, para efeito de entendimento vamos analisar somente o lado esquerdo composto pelas entradas EnA, In1 e In2 e saídas OUT1 e OUT2.

A entrada Ena é o *enable*, utilizado para ativar ou desativar o controle do motor, ela se conecta diretamente com as portas lógicas AND, assim se o seu sinal analógico for “0” o sistema está inativo e se “1” o sistema está ativo para manipulação, as entradas In1 e In2 determinam a direção da corrente pelo motor, a Tabela 1 relaciona os valores lógicos com a direção do motor, os transistores ativos e inativos e a direção da corrente nas saídas OUT1 e OUT2.

EnA	In1	In2	T1	T2	T3	T4	OUT1	OUT2	Motor
0	X	X	Inativo	Inativo	Inativo	Inativo	X	X	Parado
1	1	0	Ativo	Inativo	Inativo	Ativo	+	-	>
1	0	1	Inativo	Ativo	Ativo	Inativo	-	+	<
1	1	1	Ativo	Ativo	Inativo	Inativo	+	+	Parado
1	0	0	Inativo	Inativo	Ativo	Ativo	-	-	Parado

Tabela 1 - Lógica Ponte H

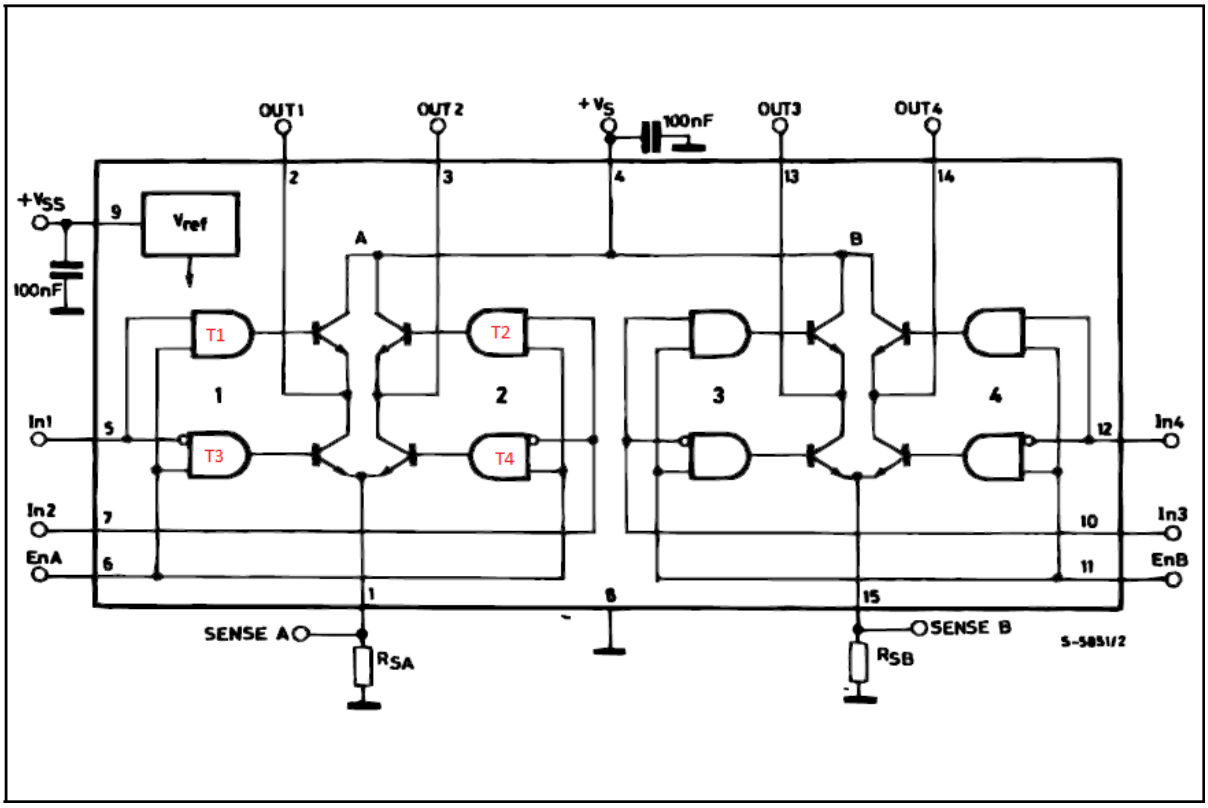

Figura 16 - Circuito L298N [18]

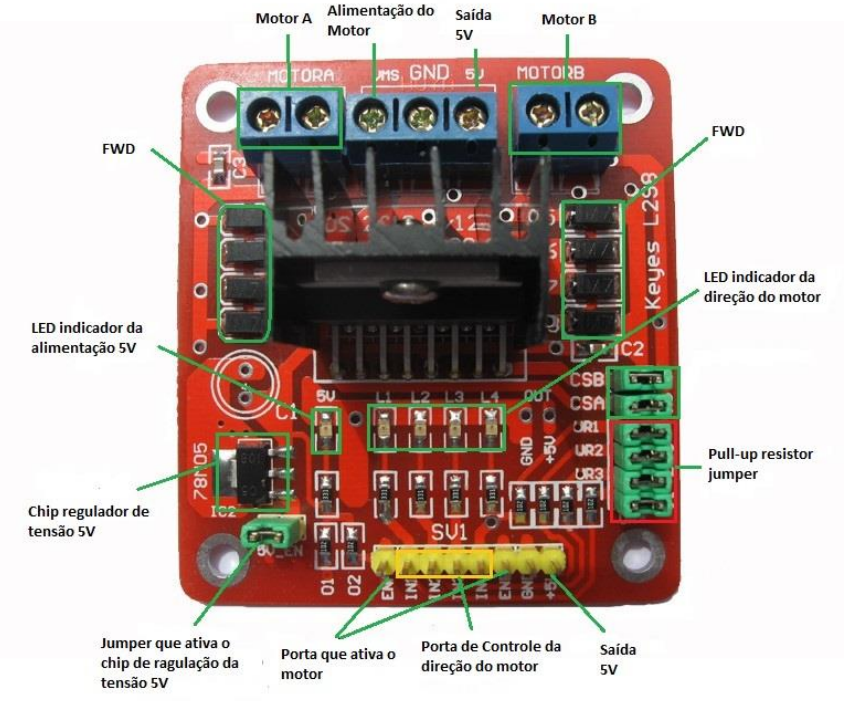

Figura 17 - *Shield* Ponte H para Arduino[19]

2.7. Conexões

Para ilustrar as conexões foi utilizado o *software* Fritzing [21], com ele é possível criar as ligações entre os componentes (Figura 18) de forma ilustrativa e juntamente com isso é gerado um esquemático com as interligações dos CIs (Figura 19) o que facilita o entendimento de como todo o circuito se comunica e funciona.

Todo sistema é alimentado pelas 4 baterias AAA, sendo que cada uma delas possui 1,5V, criando uma tensão total de 6V. A bateria alimenta diretamente os motores através da ponte H, essa por sua vez possui uma saída de tensão de 5V que alimenta o Arduino UNO pelo pino Vin, e por fim a saída de 5V do Arduino é utilizada para alimentar o módulo Bluetooth.

Como já foi dito na seção 2.5 a ponte H possui três sinais de controle para cada motor, nesse esquemático os motores estão separados entre motor1 e motor2, sendo os sinais de controle INA1, INB1 e PWM1 o do primeiro motor e o INA2, INB2 e PWM2 o do segundo.

O módulo Bluetooth possui apenas quatro ligações, sendo duas delas a alimentação e o terra e as outras duas a comunicação serial, ou seja, um canal de leitura RX e um canal de escrita TX.

Os motores são ligados diretamente à ponte H, sendo o motor1 colocado nas saídas A1, B1 e o motor2 nas saídas A2, B2.

A Tabela 2 abaixo resume as ligações entre o Arduino e seus módulos, sendo o GND comum a todos eles.

Cor Conexão	Pino Arduino	Ponte H	Bluethooth	Motor 1	Motor2
Azul Escuro	10~	PWM2			
Roxo	9~	PWM1			
Amarelo	8	INB2			
Verde	7	INA2			
Laranja	6	INB1			
Marrom	5	INA1			
Azul Claro	4		TX		
Cinza	3		RX		
Vermelho	Vin	5V			
Vermelho	5V		VCC		
Preto		A1		-	
Preto		B1		+	
Vermelho		A2			-
Vermelho		B2			+

Tabela 2 - Conexões

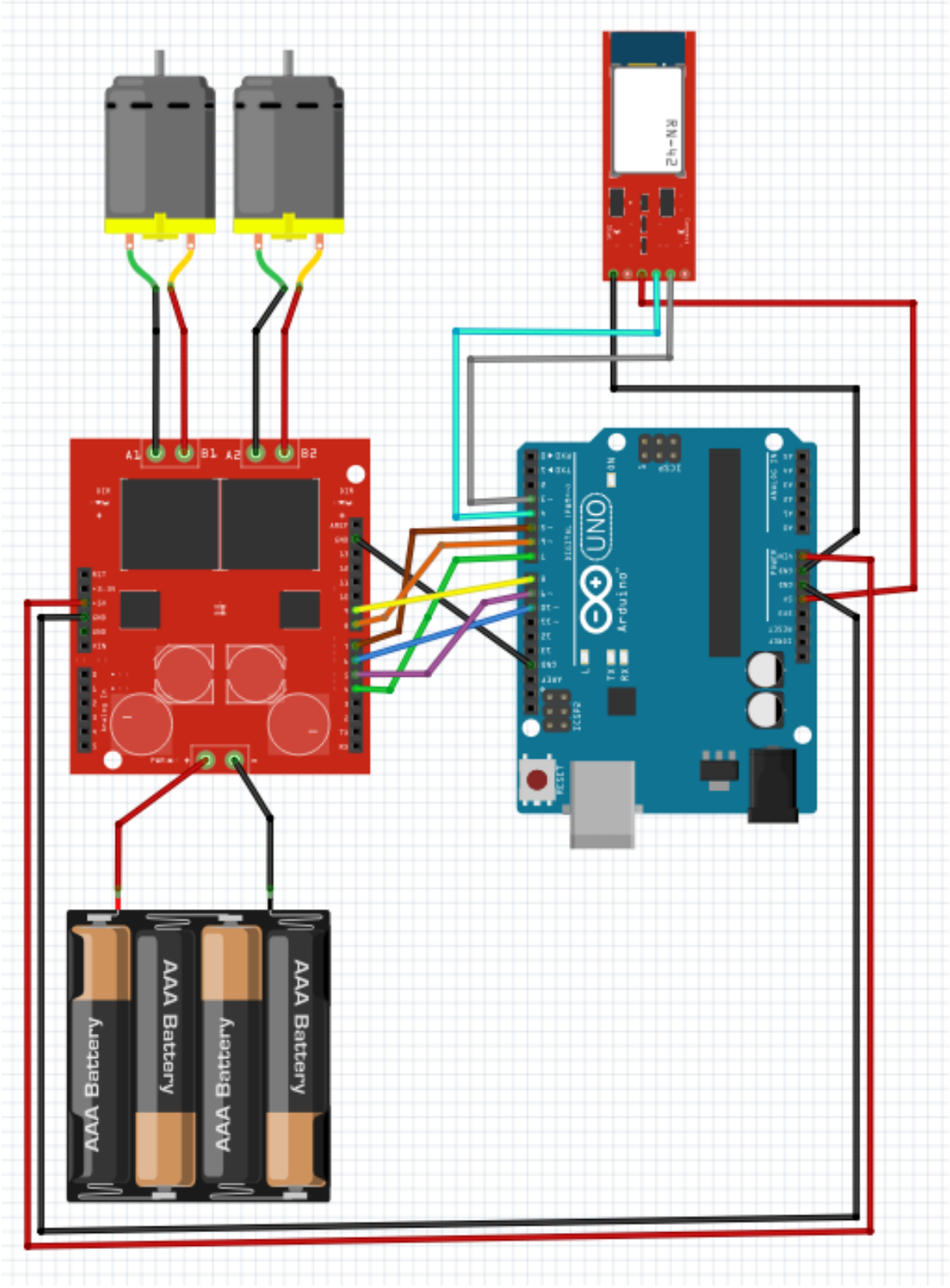

Figura 18 - Esquemático de ligações com os componentes

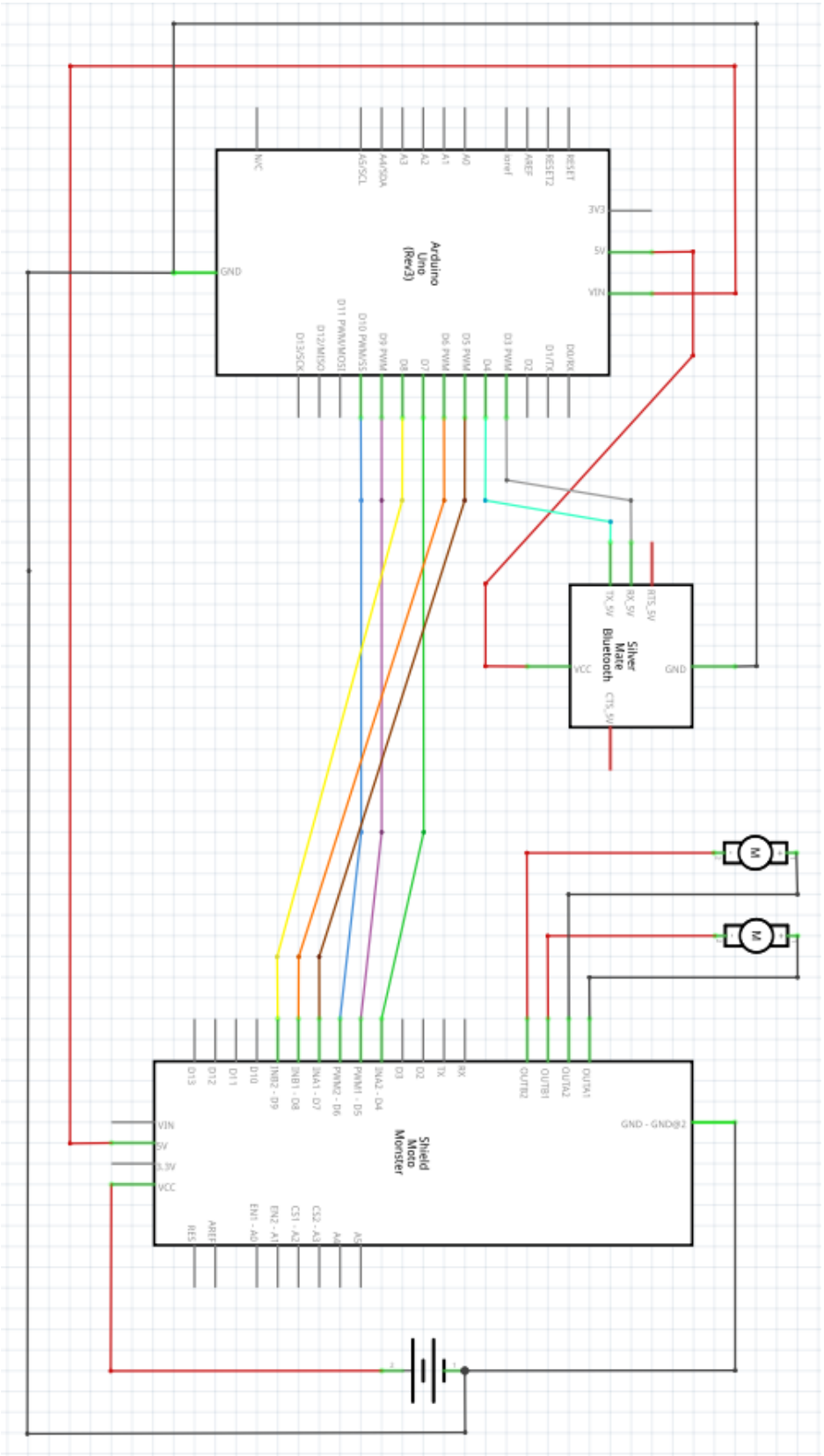

Figura 19 - Esquemático de ligações

2.8. *Softwares*

Para controle do robô foram criados dois *softwares* que se comunicam via Bluetooth. O primeiro *software* criado na linguagem C# (Figura 21) reconhece o usuário via sensor Kinect e mapeia as junções das mãos, cabeça, cintura e ombro, utilizando suas posições cartesianas em um plano (X,Y) o *software* processa qual comando o usuário está executando e o envia para o Arduino. Ao receber essa instrução o *software* no controlador do Arduino (Figura 20) atua no motor de acordo com o requerido e aguarda nova instrução.

Dessa forma foi construído um controle iterativo entre o usuário e o robô, em que este responde sempre que o primeiro muda o seu comando.

2.8.1. Fluxograma Arduino

A Figura 20 representa o funcionamento do programa criado para a plataforma Arduino, todo *software* utiliza para o seu funcionamento bibliotecas e variáveis, dessa forma inicialmente se adiciona as bibliotecas e se cria as devidas variáveis definindo seus nomes e tipos, em seguida é necessário fazer algumas configurações de *hardware*, Primeiramente foi configurado o *shield* Bluetooth, definindo quais as portas do Arduino serão responsáveis pela transmissão e recepção dos dados seriais, em seguida se configura as demais portas a serem utilizadas, definindo se elas serão *inputs* ou *outputs.* Com todas as configurações prontas se inicia a lógica do *software*. Esse programa tem como funcionalidade receber dados, processa-los e atuar nos motores, por isso o primeiro passo é receber o dado via canal serial, no entanto o Bluetooth pode estar com transmissão ou recepção em andamento o que impossibilita o recebimento ou o envio de novos dados, portanto antes de qualquer ação é verificado se o Bluetooth está disponível, caso não esteja significa que o canal serial está sendo utilizado e se deve esperar até que ele seja liberado, mas se estiver disponível então o *software* está pronto para receber um dado e este é então armazenado em uma variável. Cada dado recebido representa uma ordem a ser executado no robô, dessa forma a variável de comando recebida é então processada e se verifica qual a instrução a ser passada para os motores, ou seja, velocidade, direção e sentido. Por fim é então aguardado um novo comando a ser executado e por isso o programa retorna a espera de um novo dado.

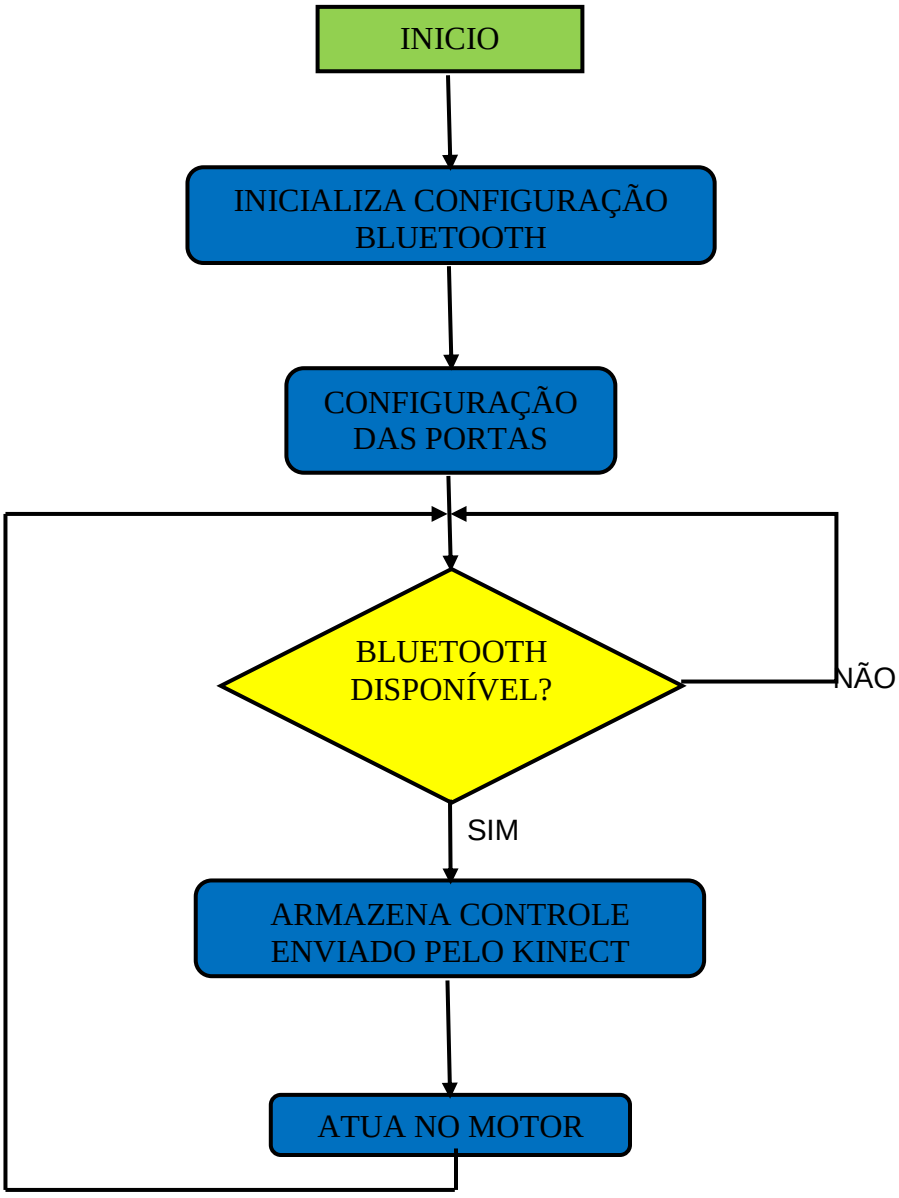

Figura 20 - Fluxograma do Software do Arduino

2.8.2. Fluxograma Kinect

A Figura 21 representa o funcionamento do programa criado para utilização do sensor Kinect. Assim como para o *software* explicado anteriormente é necessário à utilização de bibliotecas e variáveis, por isso no inicio são adicionadas as bibliotecas e são criadas as variáveis utilizadas no decorrer da lógica definindo seus nome e tipos.

Essa aplicação necessita do sensor Kinect, caso ele não esteja conectado o funcionamento do *software* fica comprometido e por isso é necessário verificar se há um sensor conectado e caso não haja um aviso é mostrado e a aplicação encerrada, no entanto se um Kinect estiver em funcionamento ele é inicializado e configurado. É ativada então a captura da imagem colorida com resolução de 320 x 240 e velocidade de 30 Fps.

O próximo passo é verificar o recebimento dessa imagem, caso nenhuma imagem seja recebida então a função de imagem colorida retorna nulo e o *software* continua verificando o processamento do esqueleto, mas se há uma imagem é conferido se os bits da imagem colorida já foram capturados, se não foram ou caso houve mudança então a variável referente aos bits coloridos é atualizada.

Por fim essa imagem deve ser colocada na tela da aplicação e para isso são configuradas as características do *bitmap* que será enviado, como tamanho, resolução, pontos por polegadas e formato do pixel.

Com a imagem colorida pronta é então verificado se há dados de “esqueleto”, caso não haja a função retorna nulo e o *software* retorna ao inicio, mas se houver dados referentes ao esqueleto de um usuário é então conferido se eles já foram capturados, se não foram ou houve mudança a variável é então atualizada.

O sensor pode captar mais de um esqueleto e como nessa aplicação a interação é apenas com um usuário por vez se captura somente os dados de esqueleto do usuário mais próximo, com esses dados é possível mapear uma série de juntas e manipula-las, nesse projeto serão utilizados cinco juntas, sendo elas, a cabeça, as mão direita e esquerda, o centro do ombro e o centro do quadril.

Para visualização de controle uma elipse é colocada para acompanhar os movimentos das mãos.

Depois de adquirir os dados das juntas do esqueleto eles são processados de forma a verificar, de acordo com as suas coordenadas, qual o comando que o usuário está aplicando, essa ordem é então enviada via Bluetooth para o *software* do Arduino para atuar no motor do robô.

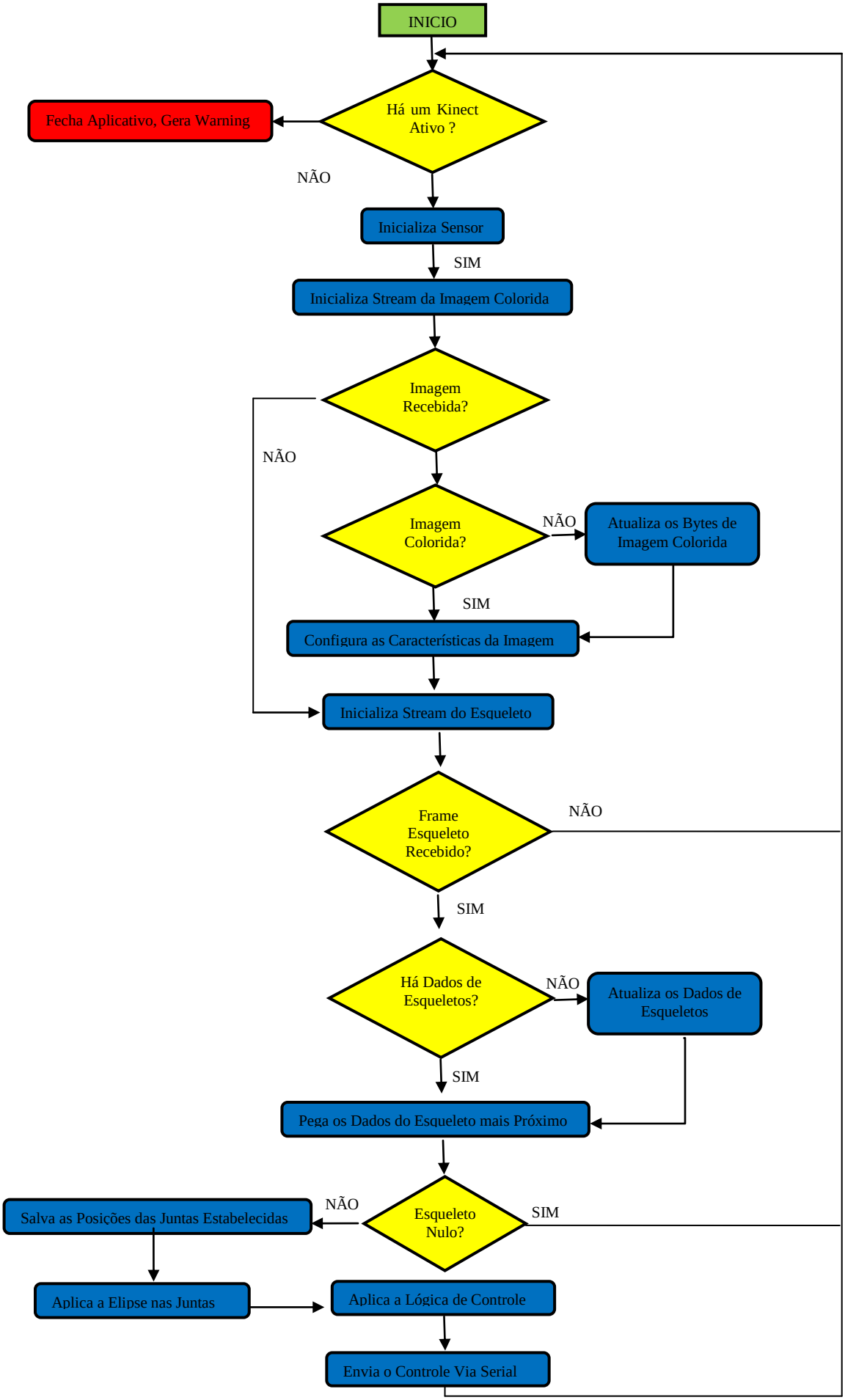
INICIO
Há um Kinect Ativo ?
Fecha Aplicativo, Gera Warning
NÃO
Inicializa Sensor
SIM
Inicializa Stream da Imagem Colorida
Imagem Recebida?
NÃO
Imagem Colorida?
NÃO
Atualiza os Bytes de Imagem Colorida
SIM
Configura as Características da Imagem
Inicializa Stream do Esqueleto
Frame Esqueleto Recebido?
NÃO
SIM
Há Dados de Esqueletos?
NÃO
Atualiza os Dados de Esqueletos
SIM
Pega os Dados do Esqueleto mais Próximo
Esqueleto Nulo?
NÃO
Salva as Posições das Juntas Estabelecidas
SIM
Aplica a Elipse nas Juntas
Aplica a Lógica de Controle
Envia o Controle Via Serial

Figura 21 - Fluxograma Kinect

2.9. Controle

Foram criados dois controles para o robô e verificado qual apresenta uma melhor resposta para a aplicação. Os dois possuem o mesmo intuito, que é movimentar o robô utilizando os movimentos através do Kinect, o que os diferencia é a lógica do que se refere à atuação dos motores que resultam na direção e velocidade. Para ambos, as juntas para controle utilizadas são as mesmas e estão ilustradas na Figura 22

Juntas para controle

- Cabeça - A
- Ombro - B
- Mão Direita - C
- Mão Esquerda - D
- Cintura - E

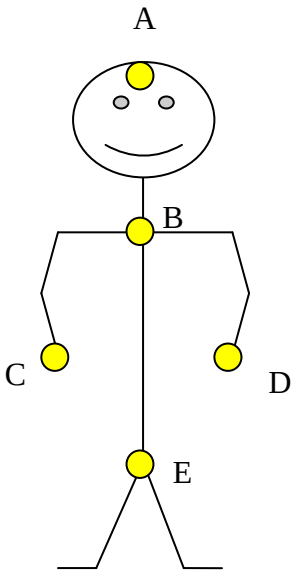

Figura 22 - Juntas de Controle

O primeiro controle (Figura 23) possui quatro movimentos, cada um dos movimentos é equivalente à movimentação do robô (para frente, para traz, para a direita e para a esquerda), nesse não há controle da velocidade. A figura 23 ilustra as regiões que correspondem a cada controle. Para parar o robô, o controle é deixar as duas mãos abaixo da cintura.

No segundo controle (Figura 24) cada mão corresponde a uma roda, assim sendo a mão direita controla a roda da direita e a mão esquerda a roda da esquerda, para controle da direção o limiar fica sendo a posição do ombro, acima do ombro a direção é para frente e abaixo para traz. Esse controle possui controle de velocidade, para cada direção a forma de aceleração é diferente.

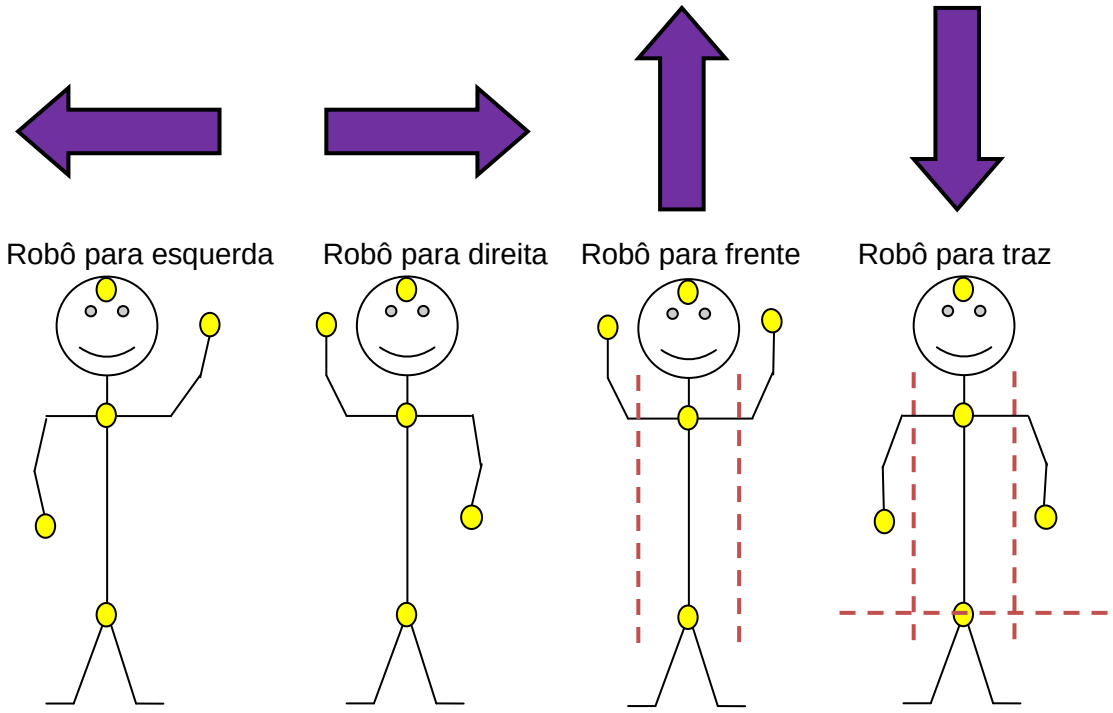

Figura 23 - Controle 1

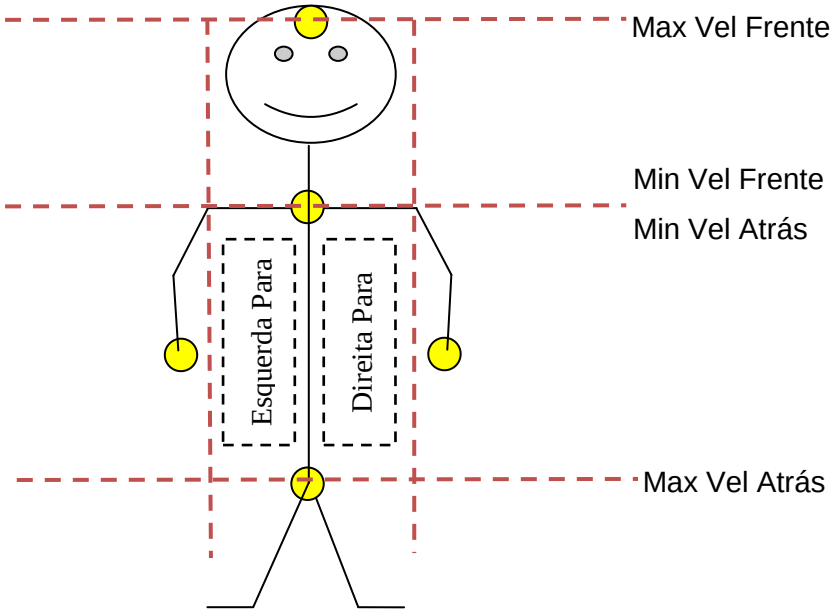

Figura 24 - Controle 2

3. Implementação.

Nessa parte estarão presentes os resultados de uma forma mais detalhada, o que foi feito para alcançar o objetivo desejado e, além disso, será verificando se o atingido foi o esperado e caso não a possível razão será apresentada.

3.1. Implementação *software* Arduino.

O fluxograma do *software* criado para o Arduino UNO já foi apresentado na seção 2.8. *Software*, na Figura 20, agora será apresentado os comandos utilizados para efetuar cada etapa do programa.

3.1.1. Inicio

Todo *software* utiliza bibliotecas que possuem as funções que serão utilizadas para a lógica do programa, nessa aplicação será utilizado à comunicação serial e por isso é necessário adicionar a biblioteca responsável para esse fim, ela é adicionada através do comando:

```
#include <SoftwareSerial.h>
```

Além disso, foram criadas algumas variáveis que serão manipuladas no decorrer do *software*, cada uma delas possui um tipo e um nome e estão listadas abaixo com os comentários que dão uma breve ideia da sua utilização. Como existem dois controles cada um deles possuem suas próprias variáveis e não são necessariamente as mesmas.

- Controle 1:

```
int caractere;  //Variável que recebe o valor do comando enviado pelo Kinect
int IN1 = 5;    //Pino de controle de direção do motor da direita
int IN2 = 6;    //Pino de controle de direção do motor da direita
int IN3 = 7;    //Pino de controle de direção do motor da esquerda
int IN4 = 8;    //Pino de controle de direção do motor da esquerda
int ENA = 9;    //Pino enable motor direita
int ENB = 10;   //Pino enable motor esquerda
```

- Controle 2:

```
int IN1 = 5;              //Pino de controle de direção do motor da direita
int IN2 = 6;              //Pino de controle de direção do motor da direita
int IN3 = 7;              //Pino de controle de direção do motor da esquerda
int IN4 = 8;              //Pino de controle de direção do motor da esquerda
int ENA = 9;              //Pino enable motor direita
int ENB = 10;             //Pino enable motor esquerda
String crtl;              // [0] a [2] direção, [3] sentido, [4] a [6] valor PWM
String velocR, velocL; // Recebe o valor da velocidade de cada motor
String sentiR, sentiL;    // Recebe o sentido de cada motor
String direcR, direcL;    // Recebe a identificação de que motor está sendo ativado
int velR, velL;           // Recebe o valor da velocidade no tipo inteiro
                          // Estas variáveis são inicializadas com o valor 150
```

No Controle 2, com a utilização do PWM, foi constatado a necessidade de modificar a configuração do PWM utilizado, isso será analisado mais a frente na seção 3.3.Análise, o comando para isso está abaixo.

```
TCCR1B = TCCR1B & 0b11111000 | 0x05;
```

3.1.2. Inicializa configuração Bluetooth

É necessário configurar e ativar a comunicação serial do Bluetooth, para isso primeiramente se identifica quais serão os pinos que serão o TX e o RX no Arduino e um nome é atribuído para a comunicação, nesse caso o nome dado foi “blue”.

```
SoftwareSerial blue (11,12)        // Pino 11 RX, Pino 12 TX
blue.begin(9600)                   //Baud rate em 9600
```

3.1.3. Configuração das portas

É necessário também configurar o sentido de cada porta digital utilizada do Arduino, ou seja, se ela é uma saída (OUTPUT) ou entrada de dados (IMPUT), nesse caso todas as portas utilizadas serão saídas e o comando utilizado para a configuração é:

```
pinMode(Nome do Pino, OUTPUT);
```

3.1.4. Bluetooth disponível?

O programa deve ficar aguardando um comando vindo do Kinect para então atuar no motor, para isso é utilizado uma lógica condicional que verifica se a comunicação está disponível.

```
if(blue.available())
{
        // Recebe commando do Kinect
}
```

3.1.5. Armazena controle enviado pelo Kinect

Cada controle envia um comando diferente por Bluetooth, o Controle 1 envia um valor hexadecimal que se refere a direção a ser aplicada no robô (Frente, Atrás, Esquerda ou Direita), já o Controle 2 envia uma string que possui três tipos de informação, o motor a ser ativado, o sentido dele e a velocidade.

- Controle 1:

```
caractere = blue.read();  // A variável caractere armazena o controle enviado pelo
                             Kinect
```

- Controle 2:

```
crtl = blue.readString();        // A variável crtl recebe a String de comando
                                    enviado pelo Knect
```

A String de controle é composta por 10 caracteres (Tabela 3) e a cada 5 deles corresponde a um motor, sendo assim os cinco primeiros correspondem ao motor direito e os cinco últimos ao motor esquerdo. Se utiliza então o comando ".substring()" para desmembrar o comando enviado e atribuir para a variável de cada motor os valores.

0	1	2	3	4	5	6	7	8	9

Motor Direito	Sentido Motor Direito	Velocidade Motor Direito	Motor Esquerdo	Sentido Motor Esquerdo	Velocidade Motor Esquerdo

Tabela 3 - String de velocidade do controle 2

```
direcR = crtl.substring(0,1);
sentiR = crtl.substring(1,2);
velocR = crtl.substring(2,5);
velR= velocR.toInt();            //Converte o valor da velocidade em inteiro

direcL = crtl.substring(5,6);
sentiL = crtl.substring(6,7);
velocL = crtl.substring(7,10);
velL = velocL.toInt();           //Converte o valor da velocidade em inteiro
```

3.1.6. Atua no motor

Para atuar no motor deve-se enviar comando de controle para os pinos da ponte H (IN1, IN2, IN3, IN4, ENA e ENB), primeiro se verifica o controle enviado pelo Kinect e a partir desse valor se atua nas saídas.

- Controle 1:

  Nesse caso o controle enviado foi para o robô se movimentar para frente, o que irá variar são os valores enviados para as portas, para cada movimento há uma combinação específica, esses valores já foram estabelecido na seção 2.5. Ponte H.

```
if(caractere == 'w')     //Verifica qual comando que foi enviado
 {
        digitalWrite(ENA,HIGH);       //ativa motor da direita
        digitalWrite(ENB,HIGH);       //ativa motor da esquerda
        digitalWrite(IN1,LOW);        //motor A e B a frente
        digitalWrite(IN2,HIGH);
        digitalWrite(IN3,LOW);
        digitalWrite(IN4,HIGH);
        delay(25);                    //delay para manter a movimentação ativa por 25ms
        caractere = 'q';              //limpa a variável caractere para que o movimento
                                      não  seja continuo e sim a cada comando
}
```

- Controle 2:

  Esse caso se refere à roda da direita, para a roda da esquerda a lógica é a mesma, o que irá variar é o valor da String que identifica a roda da esquerda e as saídas a serem atuadas (IN3 e IN4);

```
if (direcR == "D"){              //Verifica se roda direita foi ativada
 analogWrite(ENA,velR);
  if (sentiR == "F") {           //Verifica se sentido escolhido foi para frente
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);}
 if (sentiR == "A") {            //Verifica se sentido escolhido foi para atrás
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);}
}
  else{
   digitalWrite(ENA,LOW);}     //Se roda não foi ativada a mantém desligada
```

3.2. Implementação *software* C#.

O funcionamento desse *software* está ilustrado na Figura 21, nessa seção será aprofundada somente a lógica dos controles que foram criados, as partes de inicialização e configuração podem ser encontradas no Apêndice, nele está disponibilizado o código na sua integra com comentários para ajudar no entendimento.

3.2.1. Controle 1

O Controle 1 foi constituído para identificar movimentos pré-determinados e gerar comandos correspondentes a cada um deles, dessa forma foram criados cinco, onde quatro deles para movimentação (frente, trás, esquerda e direita) já ilustrados na Figura 23 e um de parada. Cada movimento é determinado identificando as posições das juntas e comparando-as em conjunto em uma lógica condicional, quando um movimento é então reconhecido o controle correspondente é enviado via Bluetooth para o Arduino.

```
//Identifica o movimento, se verdadeiro envia o comando correspondente via Bluetooth, nesse exemplo o movimento reconhecido foi o que movimenta o robô para frente.
if (  rightHand.Position.Y > centerShoulder.Position.Y &
      leftHand.Position.Y < centerShoulder.Position.Y        )
       {
              rightHandactive = true;     //Ativa a elipse referente a mão direita.
                                          Usado para verificar que foi reconhecido o
                                          movimento.
              System.Windows.Forms.SendKeys.SendWait("{d}");//Robô para a direita.
       }
```

Para envio do comando via Bluetooth inicialmente foi utilizado um *software* de comunicação serial chamado Tera Term (Figura 25 e Figura 26), quando um movimento era identificado o *software* aplicava a função “SendKeys(comando)”, ou seja, ela aplicava o valor do comando na janela ativa no PC, nesse caso a janela do Tera Term, o qual estava conectado com o Bluetooth do robô e por isso o comando era enviado.

Após verificar o funcionamento correto da lógica o *software* Tera Term foi substituído por uma comunicação serial criada em C#, dessa forma o próprio *software* criado para utilizar o Kinect ficou responsável pela comunicação serial, isso será mais bem explicado na seção 3.2.3 Comunicação Serial. Dessa forma ao reconhecer o movimento o *software* chama a função “SerialCmdSend()” a qual envia o comando.

```
SerialCmdSend(comando);   //Função que recebe o comando referente ao movimento e o
                            envia para a serial
```

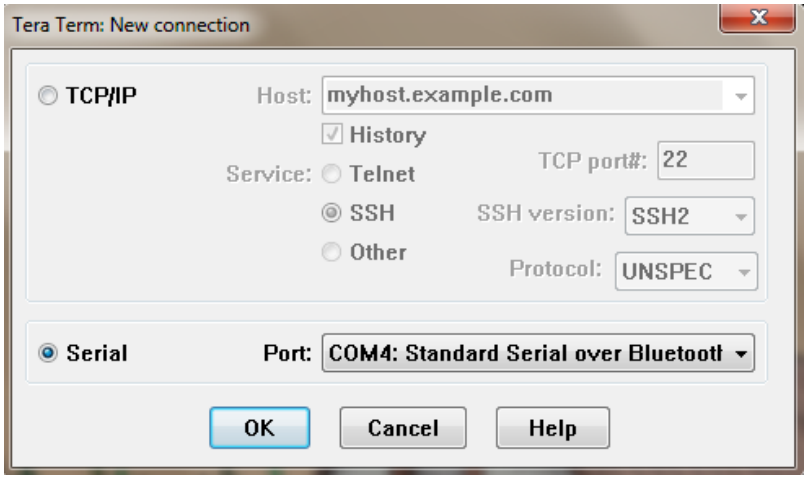

Figura 25 - Conexão Tera Term

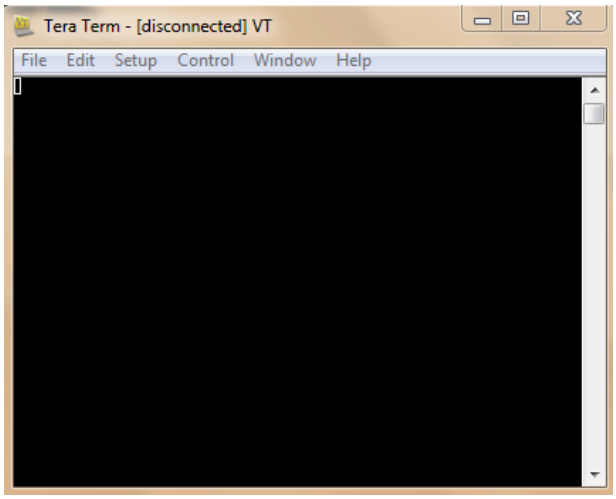

Figura 26 - Tela de envio e recebimento de dados Tera Term

3.2.2. Controle 2

O Controle 2 foi criado para ser mais intuitivo, ou seja, cada mão seria referente a um motor (Mão Direita -> Motor Direito / Mão Esquerda -> Motor esquerdo) e a escolha do sentido e velocidade seria referente a posição das mãos (Figura 24). Assim como a lógica anterior o comando a ser enviado depende da posição das juntas comparadas entre si em uma lógica condicional, o comando de cada motor é uma String composta de cinco elementos (Tabela 3), então ao identificar o que o usuário pretende uma String com o comando correspondente é criada e armazenada em uma variável, ao se obter o comando de ambos os motores as Strings de cada um deles são agrupadas, sendo a do motor direito a frente da do motor esquerdo, por fim se verifica se houve mudança do comando a ser enviado em relação ao anterior e se caso houve esse novo comando é então enviado, isso foi necessário para que os motores recebam somente uma vez um certo comando, pois se vários comandos são enviado sucessivamente a uma velocidade muito rápida os motores não conseguiam responder a tempo e por isso permaneciam parados, assim caso não haja mudança do comando ele permanece o mesmo, o robô mantem o seu movimento mas não é enviado o comando novamente.

Abaixo serão apresentadas e brevemente explicadas partições do controle 2 do motor da direita, para o motor da esquerda a lógica é a mesma variando somente a *String* a ser gerada.

Primeiramente, é verificado se o movimento executado é referente a parar o motor, caso seja, é então criado uma *String* de parada.

```
if (rightHand.Position.X <= (centerShoulder.Position.X + 0.3))
 {
  RodaDireita = "DX000";   //Para o motor direito
  rightHandactive = false; //Desativa a elipse referente a mão direita.
                           Usado para verificar que foi reconhecido o
                           movimento.
 }
```

Se o movimento não for de parada o próximo passo é identificar o sentido que o motor deve girar, para frente ou para trás, nesse caso será explicado para quando o motor deve se movimenta para frente, no outro caso a lógica é a mesma mudando apenas a condição do comando “IF” e a *String* a ser gerada.

Ao verificar o movimento referente ao sentido o próximo passo é calcular a velocidade, na Figura 24 é possível verificar as regiões em que a velocidade é considerada máxima e mínima para cada sentido, a velocidade foi considerada linear à posição da mão na região, com esse pensamento foi então calculado uma equação de reta que determina todos os valores possíveis para o PWM de acordo com a posição da mão na região, para essa equação foram colocadas como máximo do PWM o valor 255 e o mínimo 100, as equações de reta para cada sentido do motor podem ser visualizadas na Figura 27 e Figura 28 e o gráfico das retas está ilustrado na Figura 29.

As relações do gráfico referente às juntas da Cintura, Ombro e Cabeça no eixo Y, estão relacionadas na Tabela 4.

1	Cintura.Y
2	Ombro.Y
3	Cabeça.Y

Tabela 4 - Relação gráfico e juntas

$$Vel\ Atrás = \frac{-155}{Ombro.Y - Cintura.Y} * (Mão.Y - Cintura.Y) + 255$$

Figura 27 - Equação de velocidade de rotação para trás

$$Vel\ Frente = \frac{155}{Cabeça.Y - Ombro.Y} * (Mão.Y - Ombro.Y) + 100$$

Figura 28 - Equação de velocidade de rotação para frente

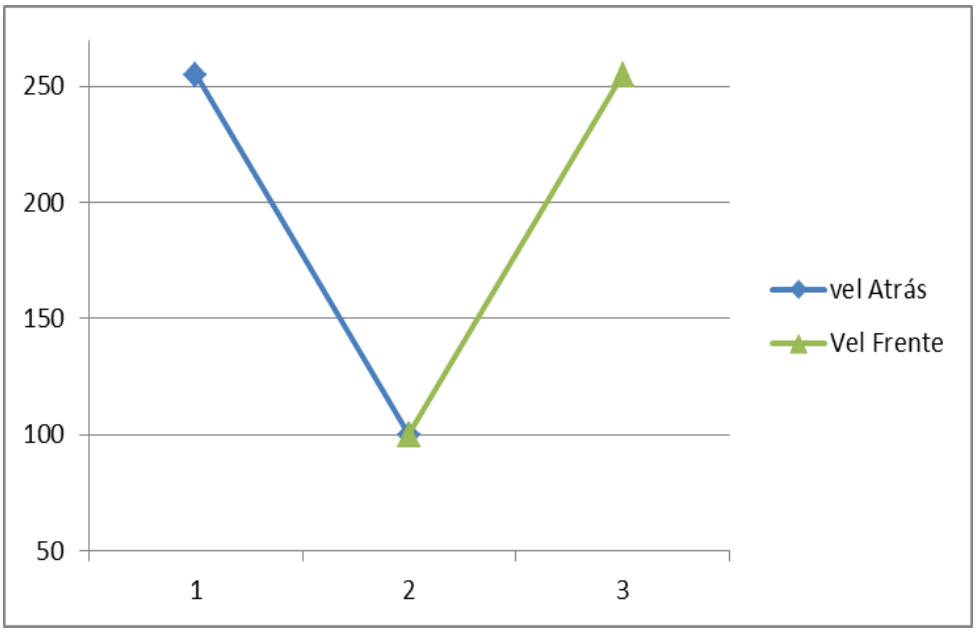

Figura 29 - Velocidade dos Motores

No entanto devido à sensibilidade do sistema os valores de velocidade aplicada ao motor ficaram limitados em dois, velocidade mínima de 100 e velocidade máxima de 240, o limiar ficou sendo o valor de 180, ou seja, quando a equação de velocidade atingia valores inferiores a 180 a velocidade aplicada ao motor era de 100 e quando superior a velocidade ficaria sendo de 240.

Para finalizar é então criada a *String* do motor com a identificação do motor, do sentido e velocidade (Tabela 5).

Motor	D	E
Sentido	F	A
Velocidade	Veloc Max	Veloc Min

Tabela 5 - Valores Correspondentes a *String*

```
else
 {
  if (rightHand.Position.Y < head.Position.Y & rightHand.Position.Y >
       centerShoulder.Position.Y)
  {
  //equação de reta para obtenção do valor da velocidade de acordo com a posição da
       mão [conversão de float para int]
       VelRodaDireita = (int)Math.Ceiling((155 / (head.Position.Y -
       centerShoulder.Position.Y)) * (rightHand.Position.Y -
       centerShoulder.Position.Y) + 100);

       if (VelRodaDireita < 180) //Verifica valor calculado da velocidade e aplica
                                 velocidade máxima ou mínima
       {
        VelDireita = "100";
       }
       else
      {
        VelDireita = "240";
      }

    RodaDireita = "DF" + VelDireita;      //Liga motor DIREITO para FRENTE com
                                                velocidade 100 Ex: DF100
    rightHandactive = true;
  }
```

Com os valores de *String* de cada motor é então criado o controle concatenado as duas *Strings*, então se verifica se houve mudança no controle enviado com o valor do ultimo comando enviado, e se houve o valor é então impresso em tela, apenas para verificação, enviado para a serial e se atualiza a variável referente ao último valor enviado.

```
Roda = RodaDireita + RodaEsquerda; //concatenação dos comandos de cada motor
if (Roda != RodaOld)               //verifica se houve variação no comando
 {
   Verify.AppendText(Roda);        //Imprime o valor do comando em tela
   SerialCmdSend(Roda);            //Envia para a serial o comando
   RodaOld = Roda;                 //Atualiza a variavel
  }
```

3.2.3. Comunicação Serial

A comunicação entre os *softwares* do Arduino e o *software* C# é serial via Bluetooth, para que isso ocorra é necessário configurar uma porta serial e conecta-la entre o PC e o modulo ligado ao Arduino, para fazer isso foi criado um código em C#, nele são escolhidas todas as características da porta serial e gerado os comando de envio e recebimento de dados.

O nome da variável referente à comunicação é “serial”, e a ela foi atribuído cada característica listada abaixo, como por exemplo, nome da porta a ser conectado, *baud rate*, se há *handshake* e paridade, tamanho do dado e numero de bits de parada, e por fim ela é aberta.

```
serial.PortName = Comm_Port_Names.Text;              //Nome da porta serial
serial.BaudRate = Convert.ToInt32(9600);             //Baud rate
serial.Handshake = System.IO.Ports.Handshake.None;   //Handshake
serial.Parity = Parity.None;                         //Bit de paridade
serial.DataBits = 8;                                 //Número de bits de dado
serial.StopBits = StopBits.One;                      //Número de stop bits
serial.ReadTimeout = 200;                            //temporização
serial.WriteTimeout = 50;
serial.Open();                                       //Conecta porta serial
```

Com a serial conectada ela já está apta a enviar os comandos para o robô, para isso primeiramente ela verifica se a serial está realmente aberta e se sim envia o dado passado para a função “SerialCmdSend(data)”, se durante o processo houver um erro o envio é então parado e um aviso é escrito em tela.

```
public void SerialCmdSend(string data)   //Função de envoi de dados via serial
{
 if (serial.IsOpen)                      //verifica se serial está aberta
 {
  try
  {
   serial.Write(data);                   //Envia dado
  }
  catch (Exception ex)                   //Se houver erro interrompe o processo e
                                         envia um aviso
  {
   para.Inlines.Add("Failed to SEND" + data + "\n" + ex + "\n");
   mcFlowDoc.Blocks.Add(para);
   Commdata.Document = mcFlowDoc;
  }
 }
}
```

4. Resultados

Nessa seção serão apresentados os resultados atingidos após toda a implementação do projeto, demonstrando o funcionamento do sistema dos controles e a interface entre o usuário e o robô.

A Figura 30 e Figura 31 ilustra a tela de aplicação utilizada pelo PC como interface para ativar a comunicação Bluetooth, para visualizar a imagem captada pelo sensor Kinect, para aplicar comandos a serem enviado para o robô e verifica-los em caso de testes.

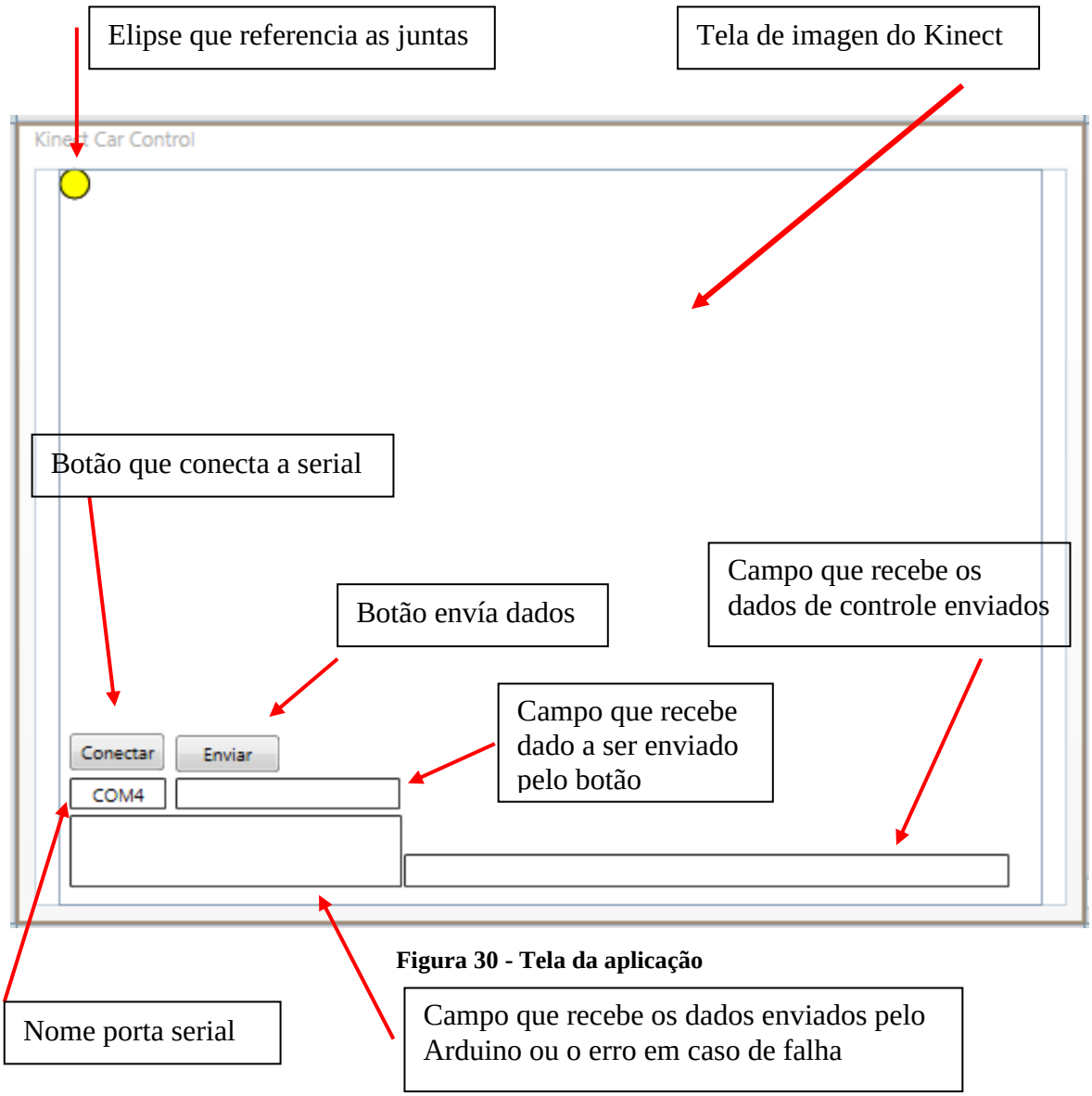

Figura 30 - Tela da aplicação

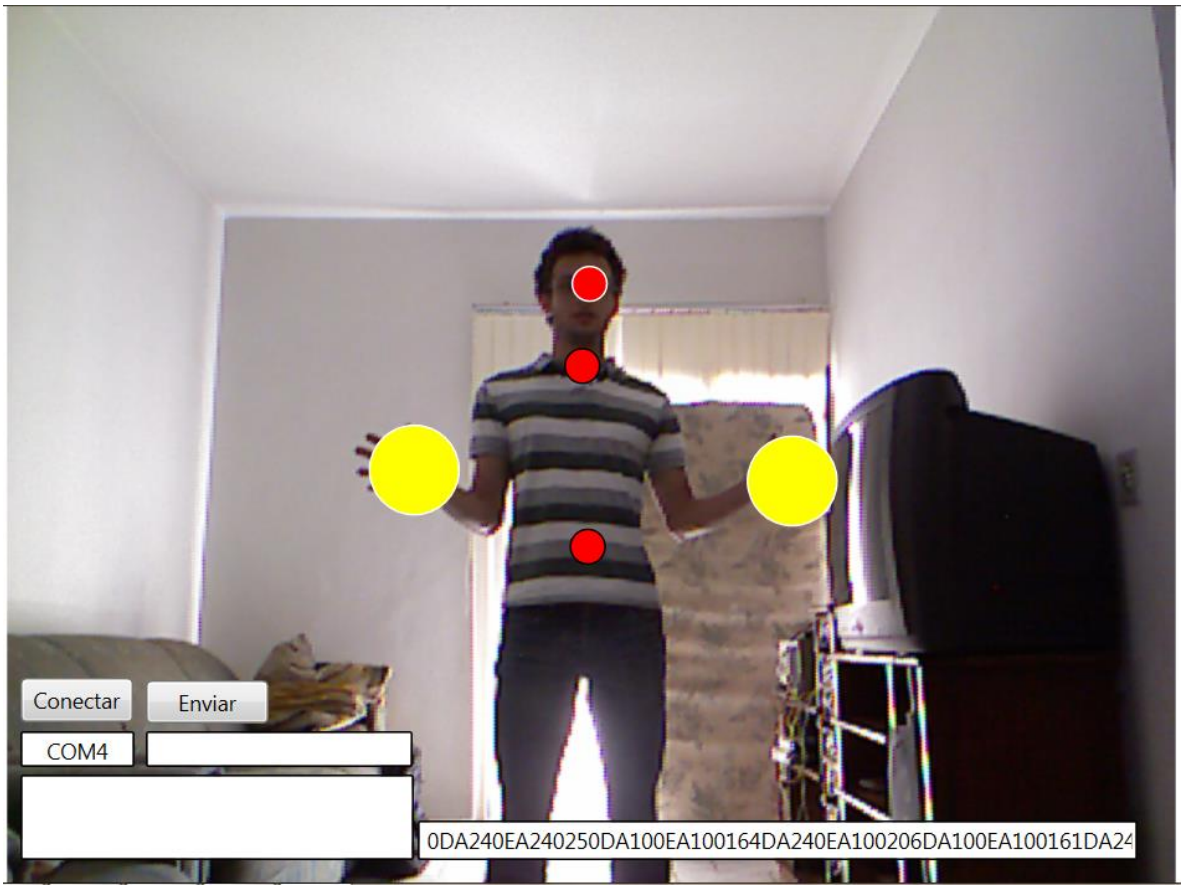

Figura 31 - Aplicação em funcionamento

Nessa imagem é possível verificar a aplicação em funcionamento com o Bluetooth já conectado e os movimentos sendo reconhecidos pelo Kinect. As elipses marcam as juntas que foram escolhidas para serem combinadas e definir os movimento referente a cada controle.

4.1. Resultado Controle 1.

A seguir se poderá verificar o funcionamento do Controle 1 explicado na seção 2.9. Controle e Figura 23.

A Figura 32 demonstra o movimento para manter o robô parado, nessa condição as elipses referente as mãos se mantem em um tamanho menor para verificar que nenhum comando está sendo enviado para o robô.

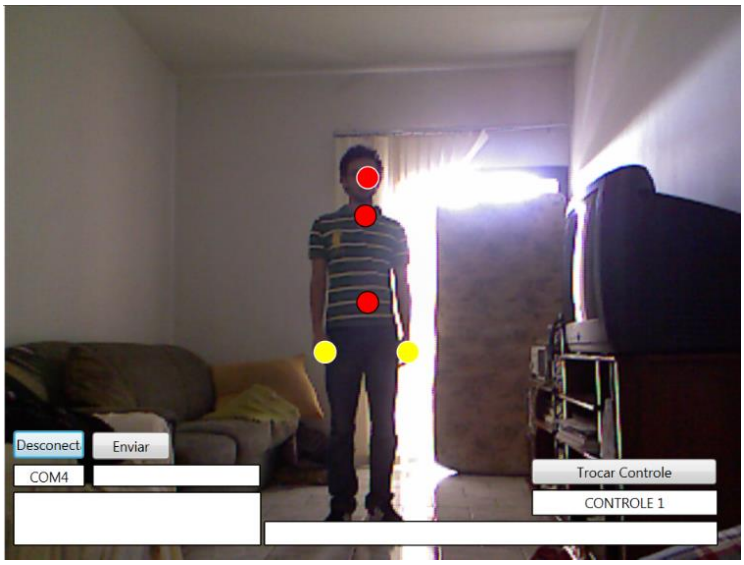

Figura 32 - Controle 1- robô parado

A figura 33 demonstra o movimento referente ao controle de virar o robô para a esquerda, o funcionamento ocorre da seguinte forma, ao se reconhecer esse movimento a roda da esquerda se mantém parada e a da direita é ligada o que resulta no robô virando para a esquerda.

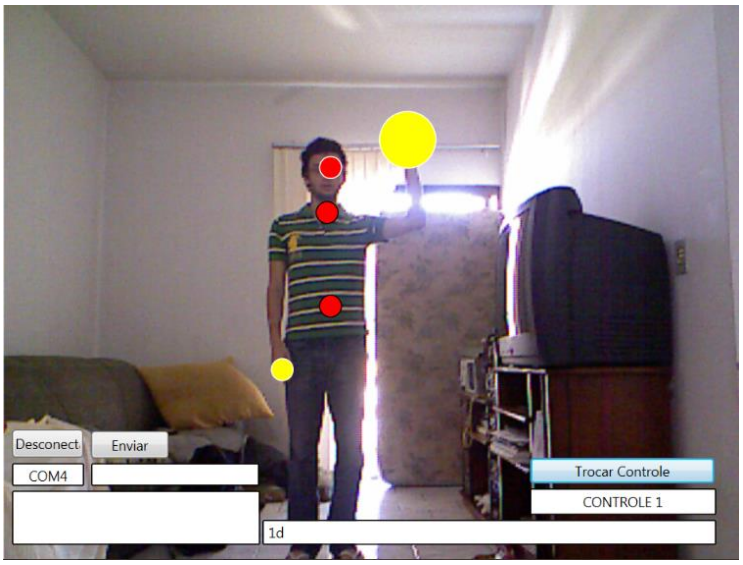

Figura 33 - Controle 1 - robô vira para esquerda

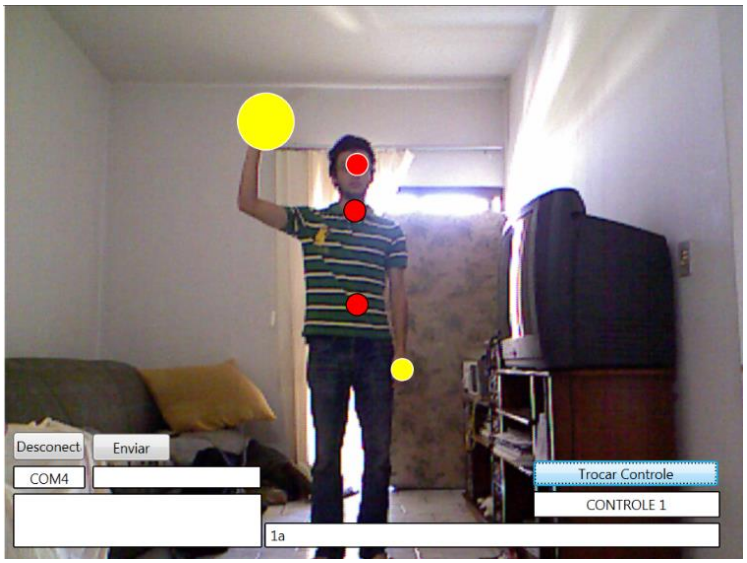

Figura 34 - Controle 1 - robô vira para a direita

A figura 34 demonstra o movimento referente ao controle de virar o robô para a direita, o funcionamento ocorre da seguinte forma, ao se reconhecer esse movimento a roda da direita se mantém parada e a da esquerda é ligada o que resulta no robô virando para a direita.

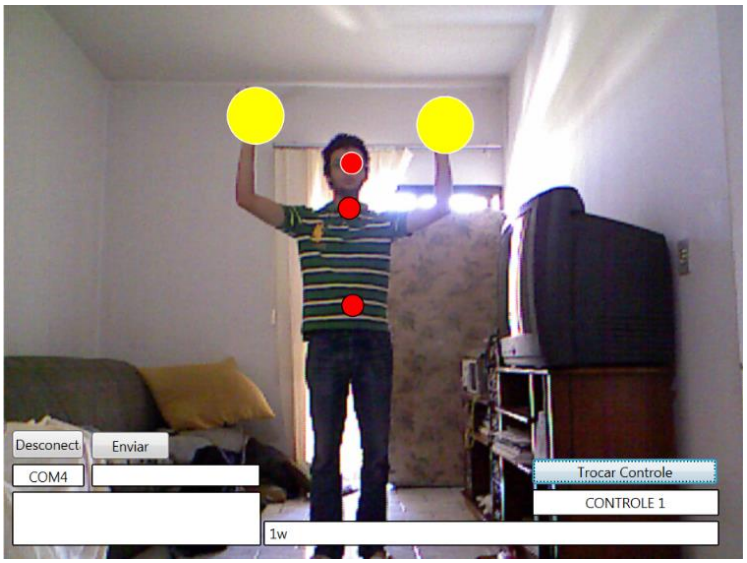

Figura 35 - Controle 1 - robô se movimenta para frente

A figura 35 ilustra o movimento para controlar o robô para frente, o que ocorre é a execução dos dois motores, motor direito e motor esquerdo, com a mesma velocidade, no caso do Controle 1 a velocidade é a máxima, pois não há variação de velocidade e mesmo sentido.

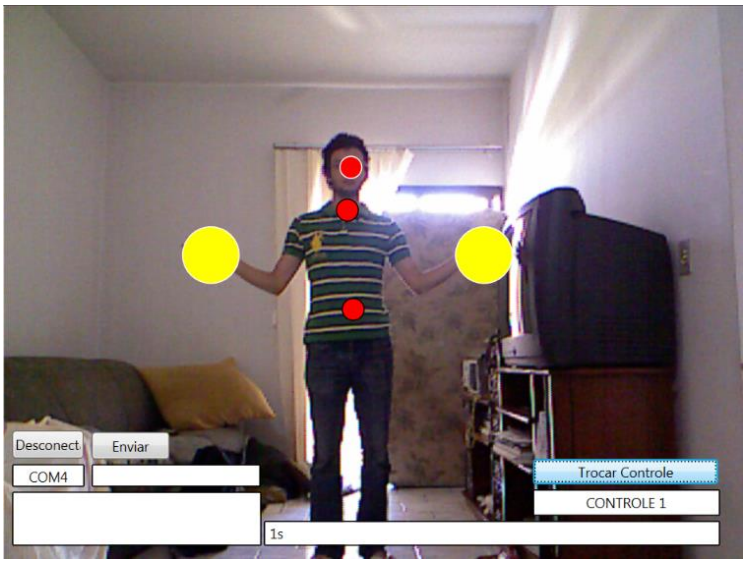

Figura 36 - Controle 1 - robô se movimenta para traz

A figura 36 ilustra o movimento para controlar o robô para traz, o funcionamento é idêntico ao do movimento ilustrado na figura 35, a diferença nesse controle é o sentido dos motores, que agora giram no sentido contrario

4.2. Resultado Controle 2.

A seguir se poderá verificar o funcionamento do Controle 2 explicado na seção 2.9. Controle e Figura 24.

A Figura 37 demonstra o movimento para manter o robô parado, nessa condição as elipses referente as mãos se mantem em um tamanho menor para verificar que nenhum comando está sendo enviado para o robô.

A figura 38 demonstra o movimento para controlar o robô para frente na região de velocidade mínima do robô, nesse controle, como cada mão é referente a uma roda o que se vê são as duas mão acima do limiar do ombro, o qual se refere à região de movimento para frente e ao mesmo tempo as duas mãos se encontra na mesma altura, logo mesma velocidade e por isso o robô se movimenta para frente.

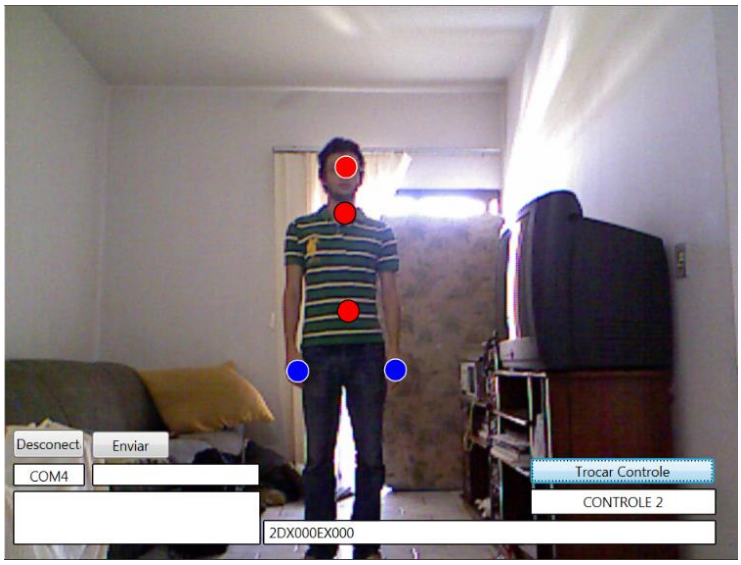

Figura 37 - Controle 2 - robô parado

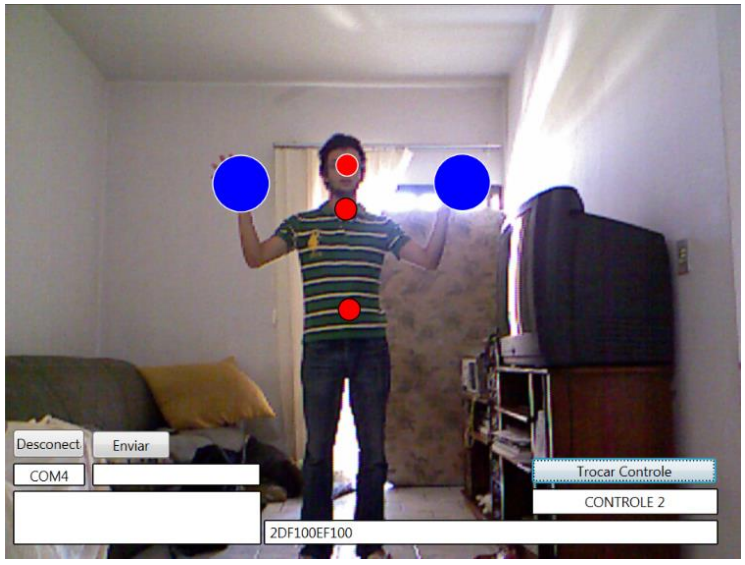

Figura 38 - Controle 2 - robô para frente com velocidade mínima

O movimento da figura 39 é análogo ao da figura 38, a diferença está na altura das mãos, dessa forma o que ocorre é um aumento da velocidade do robô.

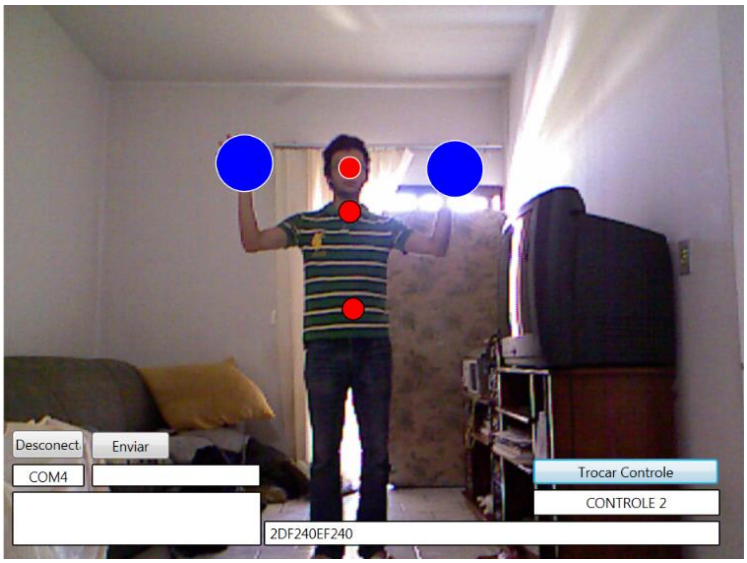

Figura 39 - Controle 2 - robô para frente com velocidade máxima

A figura 40 demonstra o movimento para controlar o robô para traz na região de velocidade mínima do robô, nesse controle, como já foi dito, cada mão é referente a uma roda o que se vê são as duas mão abaixo do limiar do ombro, o qual se refere à região de movimento para traz e ao mesmo tempo as duas mãos se encontra na mesma altura, logo mesma velocidade e por isso o robô se movimenta para traz.

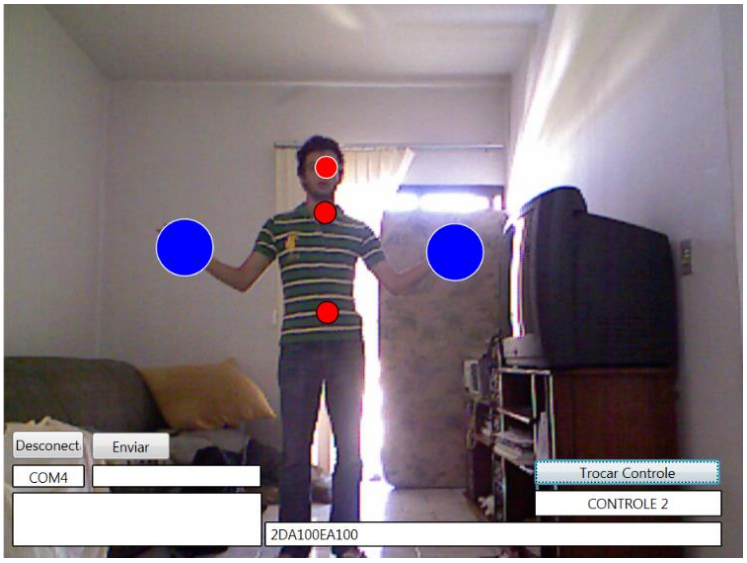

Figura 40 - Controle 2 - robô para traz com velocidade mínima

O movimento da figura 41 é análogo ao da figura 40, a diferença está na altura das mãos, dessa forma o que ocorre é um aumento da velocidade do robô.

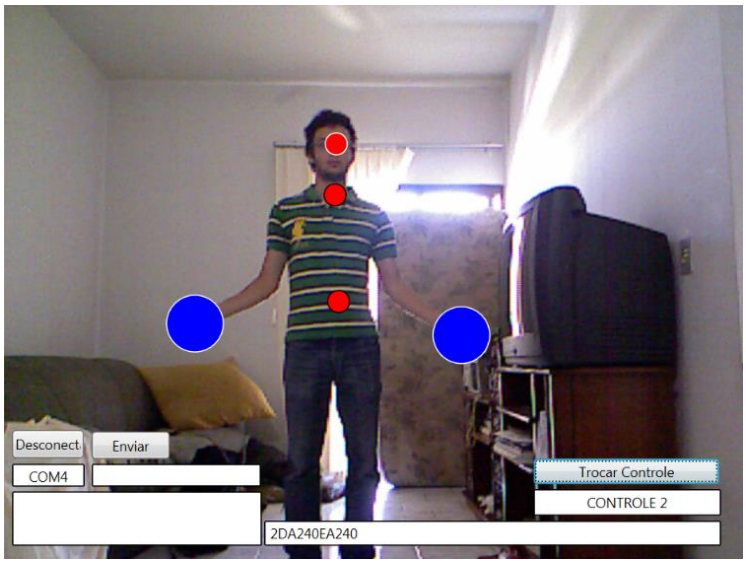

Figura 41 - Controle 2 - robô para traz com velocidade máxima

Na figura 42 e figura 43 o que se ilustra é uma diferença entre as posições das mãos direita e esquerda, o que não havia ocorrido até o momento. No caso da figura 42 a mão direita está abaixo do limiar do ombro o que leva a roda direita a ter um movimento para traz, já a mão esquerda está acima do limiar do ombro, assim a roda esquerda se movimenta para frente, essa combinação faz com que o robô vire para a direita. Na figura 43 já se verifica o oposto, então nesse caso a roda direita possui movimento para frente e a roda esquerda para traz, então o robô vira para a esquerda.

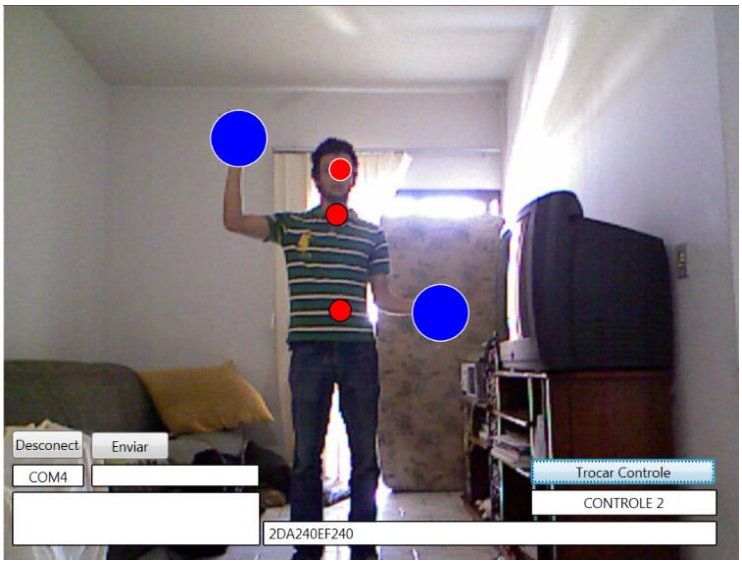

Figura 42 - Controle 2 - roda direita para traz, roda esquerda para frente

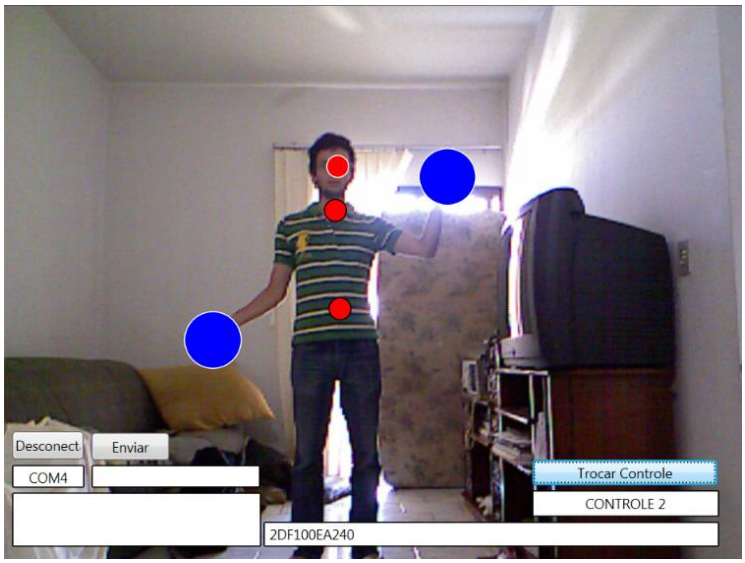

Figura 43 - Controle 2 - roda direita para frente, roda esquerda para traz

Para verificar o funcionamento da variação da velocidade no Controle 2 foi utilizado um osciloscópio e visualizado a variação do *duty cycle* do PWM. Quando o motor se encontra parado foi verificado uma tensão nula, como se ilustra na figura 44.

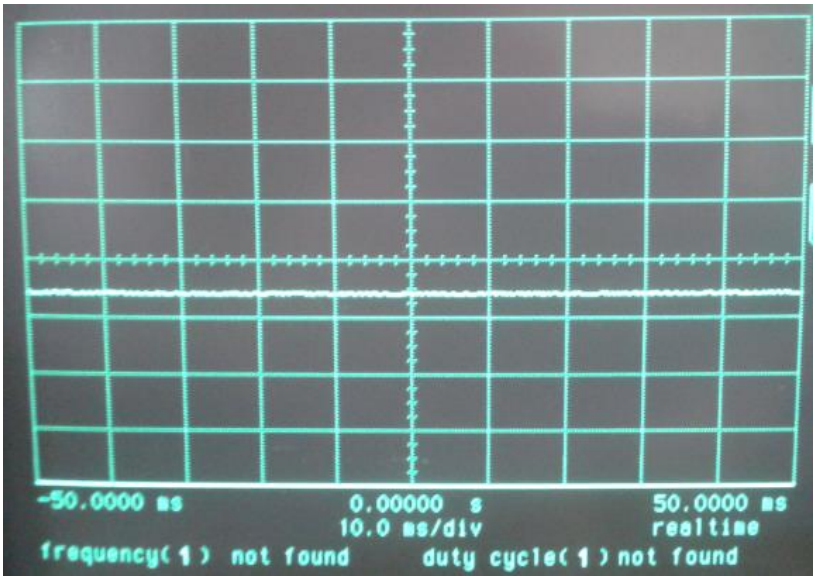

Figura 44 - PWM com duty cycle 0%

Ao iniciar um movimento em um sentido o PWM começa a responder e foi encontrado o resultado da figura 45, onde se tem um *duty cycle* de 39%.

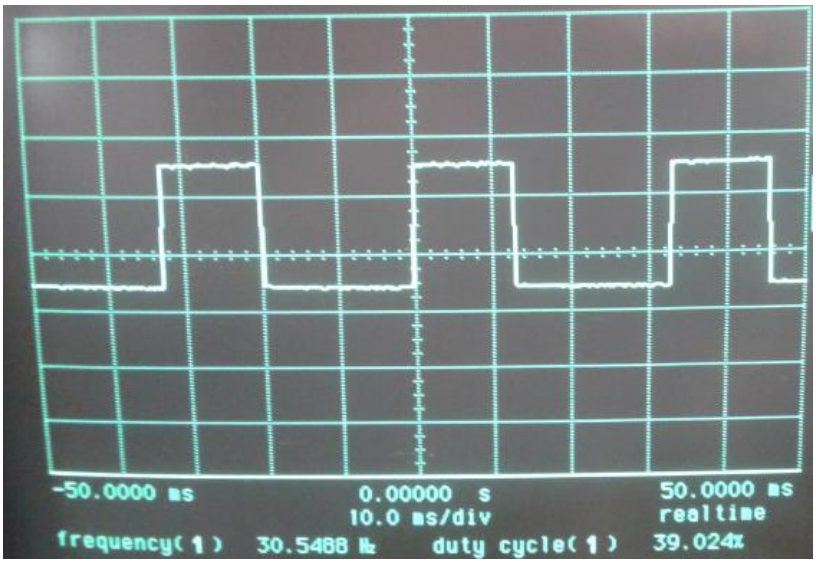

Figura 45 - PWM com duty cycle 39%

Variando o movimento tendendo a aumentar a velocidade o PWM aumenta o seu *duty cycle*, isso é notado na figura 46, nela é possível identificar um *duty cycle* de aproximadamente 94%. A frequência do PWM foi ajustada para 30,5 Hz, aproximadamente.

Figura 46 - PWM com duty cycle 94%

5. Análise

O Atmega328 tem três *timers* para PWM que controlam seis saídas PWM (Figura 47). É possível manipular os registradores dos *timers* diretamente e dessa forma é possível obter mais controle sobre os PWMs. Nesse caso o que foi buscado foi diminuir a frequência do PWM utilizado (Pinos 9 e 10), que é de 500Hz, o máximo possível (30,52Hz). Isso foi necessário devido as características do motor (120 RPM). Foi verificado que para velocidades baixas, ou seja, valores a baixo de 130 no comando analogWrite(Pin,PWM) o motor não respondia bem, dessa forma para aumentar a faixa possível de velocidades do motor a frequência do PWM foi reduzida. A causa desse problema é que para valores menores de PWM o *duty cycle* é menor e como a frequência era alta não gerava tensão suficiente para tirar o motor da sua inércia, então diminuindo a frequência do PWM significa aumentar o *duty cycle*.

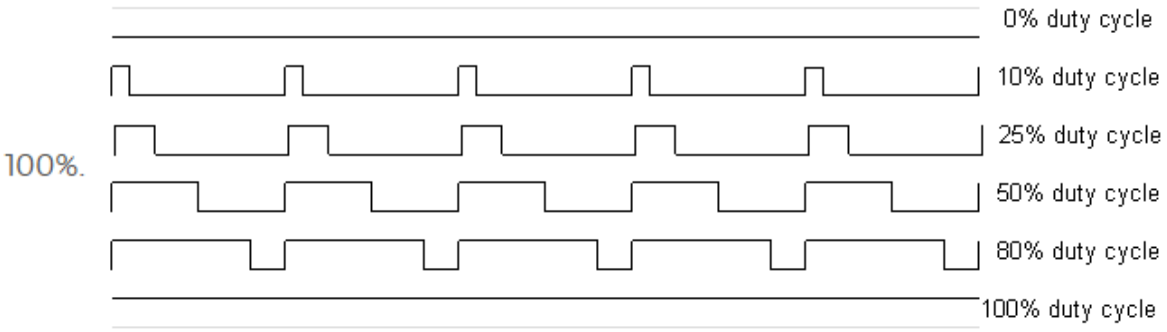

Figura 47 - PWM

O Atmega328 possui três timers, sendo eles, Timer 0, Timer 1 e Timer 2, o timer correspondente aos pinos 9 e 10 é o Timer 1, para modifica-lo se deve manipular o registrador TCCRnB (Timer/Counter Control Register) em que “n” é o número do timer. O que deve ser manipulado são os últimos três bits desse registrador, assim é feito um “AND” com o valor 0bx11111000 para zerar esses bits e então é aplicada uma lógica “OU” com o valor correspondente a frequência requerida, que nesse caso é 0x05 [22].

```
TCCR1B = TCCR1B & 0b11111000 | 0x05;
```

Os dois controles implementados funcionam e deixa o usuário capaz de controlar o robô, no entanto cada um deles possui as suas vantagens e desvantagens. No Controle 1 não há controle da velocidade, o que o torna mais simples e com menos recursos, no entanto na prática ele se mostrou mais fácil de ser utilizado e gerou um controle melhor e mais estável. O controle 2 não possui movimento fixo para gerar um comando, cada mão se refere a um motor (Mão direita -> Motor direito / Mão esquerda -> Motor esquerdo) o que torna esse tipo de interação mais intuitiva, ele também possui controle da velocidade, mas está possui apenas duas variações devido a sensibilidade encontrada no método utilizado, o qual gera variações muito rápidas da velocidade e o robô não era capaz de interpretar, a solução encontrada foi limitar essa variação a apenas dois valores.

Uma outra análise é em relação aos motores que mesmo ativando ambos a mesma velocidade o robô não se movimenta em linha reta, foram encontradas duas causas desse efeito. A primeira é o mau alinhamento das rodas, uma roda desalinhada em relação à outra pode gerar um deslocamento em curva. O segundo é a diferença de redução entre os motores, apesar de serem do mesmo modelo e possuírem as mesmas especificações não necessariamente eles são idênticos devido à fabricação, para corrigir isso se deve descobrir empiricamente a diferença entre os motores e aplica-las em *software* ou criar um controle em *hardware* que monitore a rotação de cada motor e através de um *software* atuar com correções.

5.1. Trabalhos futuros

O Kinect possui um grande recurso que não foi explorado nesse projeto, as quatro *arrays* de microfones. Com esses sensores é possível criar reconhecer comandos por voz, seria assim mais uma possibilidade para controlar o robô móvel criado.

Outro trabalho interessante seria criar um sistema escalonado de velocidades ao invés de linear (Figura 29), esse sistema pode evitar os erros em relação a variação da velocidade discutidos na análise, deixando o sistema mais robusto e com mais variações possíveis da velocidade.

6. Conclusão

A criação de uma plataforma robótica de baixo custo que pode ser controlada remotamente é uma boa demonstração dos conhecimentos adquiridos em um curso de engenharia elétrica, pois é requerido compreender o funcionamento da eletrônica, de lógica digital, de linguagem de programação, de métodos de comunicação e seus protocolos.

A montagem da plataforma na prática gera a oportunidade de verificar as dificuldades encontradas para a implementação do sistema como um todo e principalmente em como lidar e identificar erros inesperados e como corrigi-los, essa prática gera experiência que na eletrônica é de fato muito importante, pois como em muitos casos não se é possível, literalmente, ver o erro e sim somente perceber o mau funcionamento a experiência limita as possibilidades do que pode ser o problema levando a uma rápida solução ou até aperfeiçoamento.

A utilização do sensor Kinect é uma experiência muito interessante, pois esse dispositivo foi criado por uma empresa mundialmente conhecida com o intuito simplesmente de entretenimento, mas por sua versatilidade, bom desempenho e baixo custo foi identificado nele um grande potencial para diversas aplicações que necessitam de um bom processamento em visão computacional, percebendo isso a criadora do sensor tornou livre a utilização do seu próprio *software* para que desenvolvedores autônomos se aventurassem nas possibilidades oferecidas pelo Kinect e por isso é possível facilmente encontrar diversas aplicações que usufruem do dispositivo.

O resultado obtido ao final desse trabalho foi satisfatório, foi conseguido controlar o robô por meio de dois controles utilizando o reconhecimento de gestos pelo Kinect, no entanto um problema encontrado foi referente a movimentos bruscos, os quais podem causar mal entendimento pelo robô e ele pode responder de forma errada ou não responder.

Referências Bibliográficas

[1] http://luckylarry.co.uk/arduino-projects/obstacle-avoidance-robot-build-your.own-larrybot/
[2] André Crepaldi Geiger Smidt, "Implemetação de uma plataforma robótica controlada remotamente utilizando o Arduino", EESC – USP, 2013.
[3] http://www.microsoft.com/en-us/kinectforwindows/discover/features.aspx
[4] Tanner Bryce Blair, Chad Eric Daves. "Innovate Engineering Outreach: A Special Application of the Xbox 360 Kinect Sensor", School of Electrical and Computer Engineering ,University of Oklahoma.
[5] Z.Zhang. "Microsoft Kinect Sensor and Its Effect",IEEE MultiMedia, 27 Abril 2012
[6] Microsoft Corporation, "Kinect foe Windows – Human Interface Guidelines V1.8.0
[7] El-laithy, R.A. ;"Study on the Use of Microsoft Kinect for Robotics Applications" California State Univ., Fullerton, CA, USA ; Jidong Huang ; Yeh, M.
[8] http://arduino.cc/
[9] http://wiring.org.co/
[10] http://www.processing.org/
[11] http://www.atmel.com/pt/br/devices/ATMEGA328.aspx
[12] http://www.embarcados.com.br/arduino-uno/
[13] http://www.bluetooth.com/Pages/Fast-Facts.aspx
[14] McDermott-Wells, P. "What is Bluethooth?" ; Mega-Data Services Inc., FL, USA
[15] http://www.bluetooth.com/Pages/Basics.aspx
[16] Deitel, H. M. ; C# - Como Programar ; P. J. Deitel; J. Listfield; T. R. Nieto; C. Yager; M. Zlatnika. São Paulo, Pearson Makron Books, 2003.
[17] http://www.visualstudio.com/downloads/download-visual-studio-vs
[18] Vibhor Grupta. "Working and Analysis of the H-Bridge Motor Driver Circuit Designed for Wheeled Mobile Robots"
[19] https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf
[20] http://softwaresouls.com/softwaresouls/category/arduino/
[21] http://fritzing.org/home/
[22] http://arduino.cc/en/Tutorial/SecretsOfArduinoPWM

Apêndice 1: Arduino Controle 1

```
#include <SoftwareSerial.h>

SoftwareSerial blue(11,12); //RX TX
int caractere;
int IN1 = 5;
int IN2 = 6;
int IN3 = 7;
int IN4 = 8;
int ENA = 9;  //motor direita
int ENB = 10; //motor esquerda

void setup ()
{
  blue.begin(9600);
  pinMode(IN1,OUTPUT);
  pinMode(IN2,OUTPUT);
  pinMode(IN3,OUTPUT);
  pinMode(IN4,OUTPUT);
}

void loop (){
  if(blue.available())
  {
    caractere = blue.read();
  }
  if(caractere == 'w')
  {
    blue.println("Robo a frente");
    digitalWrite(ENA,HIGH);  //ativa os dois motores
    digitalWrite(ENB,HIGH);
    digitalWrite(IN1,LOW);   //motor A e B a frente
    digitalWrite(IN2,HIGH);
    digitalWrite(IN3,LOW);
    digitalWrite(IN4,HIGH);
    delay(25);
    caractere = 'q';      //limpa a variavel caractere para q o movimento nao seja continuo
e sim a cada comando
  }
  if(caractere == 's')
  {
    blue.println("Robo para traz");
    digitalWrite(ENA,HIGH);  //ativa os dois motores
    digitalWrite(ENB,HIGH);
    digitalWrite(IN1,HIGH);   //motor A e B para traz
    digitalWrite(IN2,LOW);
```

```
    digitalWrite(IN3,HIGH);
    digitalWrite(IN4,LOW);
    delay(25);
    caractere = 'q';
  }
  if(caractere == 'a')
  {
    blue.println("Robo para esquerda <---");
    digitalWrite(ENA,HIGH);  //ativa motor A
    digitalWrite(ENB,LOW);   //desliga motor B
    digitalWrite(IN1,LOW);  //motor A para frente
    digitalWrite(IN2,HIGH);
    digitalWrite(IN3,LOW);
    digitalWrite(IN4,LOW);
    delay(25);
    caractere = 'q';
  }
  if(caractere == 'd')
  {
    blue.println("Robo para direita --->");
    digitalWrite(ENA,LOW);  //desliga motor A
    digitalWrite(ENB,HIGH); //ativa motor B
    digitalWrite(IN1,LOW);
    digitalWrite(IN2,LOW);
    digitalWrite(IN3,LOW);  //motor B para frente
    digitalWrite(IN4,HIGH);
    delay(25);
    caractere = 'q';
  }
  if(caractere == 'q')
  {
    blue.println("Robo freio ---");
    digitalWrite(IN1,LOW);
    digitalWrite(IN2,LOW);
    digitalWrite(IN3,LOW);
    digitalWrite(IN4,LOW);
    caractere = 0;
  }
}
```

Apêndice 2: Arduino Controle 2

```
#include <SoftwareSerial.h>

SoftwareSerial blue(11,12); //RX TX
String crtl; // [0] a [2] direc, [3] sentido, [4] a [6] valor PWM
String velocR, velocL;
String sentiR, sentiL;
String direcR, direcL;
int velR, velL;
boolean StringComplete = false;

int IN1 = 5;
int IN2 = 6;
int IN3 = 7;
int IN4 = 8;
int ENA = 9; //motor direita
int ENB = 10; //motor esquerda

//**********************************
void setup ()
{
  blue.begin(9600);
  //Serial.begin(9600);
  pinMode(IN1,OUTPUT);
  pinMode(IN2,OUTPUT);
  pinMode(IN3,OUTPUT);
  pinMode(IN4,OUTPUT);
  pinMode(ENA,OUTPUT);
  pinMode(ENB,OUTPUT);
  velR = 150;
  velL = 150;
  TCCR1B = TCCR1B & 0b11111000 | 0x05;
}
//**********************************
void loop (){

  crtl="";
  if (blue.available())
  {
    crtl = blue.readString();

    direcR = crtl.substring(0,1);
    sentiR = crtl.substring(1,2);
    velocR = crtl.substring(2,5);
    velR = velocR.toInt();
```

```
    direcL = crtl.substring(5,6);
    sentiL = crtl.substring(6,7);
    velocL = crtl.substring(7,10);
    velL = velocL.toInt();
  }

//**********************************
  if (direcR == "D"){      //ativa roda da direita
    analogWrite(ENA,velR);
    if (sentiR == "F") {     //escolhe sentido do motor
        digitalWrite(IN1,LOW);
        digitalWrite(IN2,HIGH);}
    if (sentiR == "A") {
        digitalWrite(IN1,HIGH);   //motor A e B a frente
        digitalWrite(IN2,LOW);}
  }
  else{
    digitalWrite(ENA,LOW);}

  if (direcL == "E"){         //ativa roda da esquerda
    analogWrite(ENB,velL);
    if (sentiL == "F") {        //escolhe sentido do motor
        digitalWrite(IN3,LOW);
        digitalWrite(IN4,HIGH);}
    if (sentiL == "A") {
        digitalWrite(IN3,HIGH);
        digitalWrite(IN4,LOW); }
  }
  else{
    digitalWrite(ENB,LOW);}
  //direcR = "";
  //direcL = "";
//**********************************************
}
```

Apêndice 3: Software Kinect

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;
using Microsoft.Speech.Recognition;
using System.Threading;
using System.IO;
using Microsoft.Speech.AudioFormat;
using System.Diagnostics;
using System.Windows.Threading;
using System.IO.Ports; //biblioteca para comunicaçao serial

namespace KinectPowerPointControl
{
    public partial class MainWindow : Window
    {
        #region Variaveis
        KinectSensor sensor;
        SpeechRecognitionEngine speechRecognizer;

        DispatcherTimer readyTimer;

        byte[] colorBytes;
        Skeleton[] skeletons;

        bool isCirclesVisible = true;

        bool rightHandactive = false;
        bool leftHandactive = false;
        string RodaDireita;
        string RodaEsquerda;
        string Roda;
        string RodaOld;
        int VelRodaDireita;
        int VelRodaEsquerda;
        string VelDireita;
        string VelEsquerda;
        SolidColorBrush activeBrush = new SolidColorBrush(Colors.Yellow);
        SolidColorBrush inactiveBrush = new SolidColorBrush(Colors.Red);

        //Variaveis da serial
        SerialPort serial = new SerialPort();
        string recieved_data;
        FlowDocument mcFlowDoc = new FlowDocument();
        Paragraph para = new Paragraph();

        #endregion

        public MainWindow()
        {
            InitializeComponent();
            Conectar.Content = "Conectar";


            this.Loaded += new RoutedEventHandler(MainWindow_Loaded);
```

```
            this.KeyDown += new KeyEventHandler(MainWindow_KeyDown);
        }

        #region Conectar Serial
        private void Conectar_Click_1(object sender, RoutedEventArgs e)
        {
            if (serial.IsOpen == false)
            {
                try
                {
                    serial.PortName = Comm_Port_Names.Text;
                    serial.BaudRate = Convert.ToInt32(9600);
                    serial.Handshake = System.IO.Ports.Handshake.None;
                    serial.Parity = Parity.None;
                    serial.DataBits = 8;
                    serial.StopBits = StopBits.One;
                    serial.ReadTimeout = 200;
                    serial.WriteTimeout = 50;
                    serial.Open();
                }
                catch { return; }
                if (serial.IsOpen)
                {
                    Conectar.Content = "Desconectar";
                    serial.DataReceived += new
System.IO.Ports.SerialDataReceivedEventHandler(Recieve);
                }
            }
            else
            {
                try
                {
                    serial.Close();
                    Conectar.Content = "Conectar";
                }
                catch { return; }
            }
        }
        #endregion

        #region Reading
        private delegate void UpdateUiTextDelegate(string text);
        private void Recieve(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {

            recieved_data = serial.ReadExisting();
            Dispatcher.Invoke(DispatcherPriority.Send, new UpdateUiTextDelegate(WriteData),
recieved_data);
        }
        private void WriteData(string text)
        {

            para.Inlines.Add(text);
            mcFlowDoc.Blocks.Add(para);
            Commdata.Document = mcFlowDoc;
        }
        #endregion

        #region Sending
        private void Enviar_Click_1(object sender, RoutedEventArgs e)
        {
            SerialCmdSend(SerialData.Text);
            SerialData.Text = "";
        }

        public void SerialCmdSend(string data)
        {
            if (serial.IsOpen)
            {
                try
                {
```

```
                    serial.Write(data);
                }
                catch (Exception ex)
                {
                    para.Inlines.Add("Failed to SEND" + data + "\n" + ex + "\n");
                    mcFlowDoc.Blocks.Add(para);
                    Commdata.Document = mcFlowDoc;
                }
            }
            else
            {
            }
        }
        #endregion

        void MainWindow_Loaded(object sender, RoutedEventArgs e)
        {
            sensor = KinectSensor.KinectSensors.FirstOrDefault();

            if (sensor == null)
            {
                MessageBox.Show("This application requires a Kinect sensor.");
                this.Close();
            }

            sensor.Start();

            sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
            sensor.ColorFrameReady += new
EventHandler<ColorImageFrameReadyEventArgs>(sensor_ColorFrameReady);

            sensor.DepthStream.Enable(DepthImageFormat.Resolution320x240Fps30);

            sensor.SkeletonStream.Enable();
            sensor.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(sensor_SkeletonFrameReady);

            //sensor.ElevationAngle = 10;

            Application.Current.Exit += new ExitEventHandler(Current_Exit);

            InitializeSpeechRecognition();
        }

        void Current_Exit(object sender, ExitEventArgs e)
        {
            if (speechRecognizer != null)
            {
                speechRecognizer.RecognizeAsyncCancel();
                speechRecognizer.RecognizeAsyncStop();
            }
            if (sensor != null)
            {
                sensor.AudioSource.Stop();
                sensor.Stop();
                sensor.Dispose();
                sensor = null;
            }
        }

        void MainWindow_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.C)
            {
                ToggleCircles();
            }
        }

        void sensor_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
        {
            using (var image = e.OpenColorImageFrame())
```

```
            {
                if (image == null)
                    return;

                if (colorBytes == null ||
                    colorBytes.Length != image.PixelDataLength)
                {
                    colorBytes = new byte[image.PixelDataLength];
                }

                image.CopyPixelDataTo(colorBytes);

                int length = colorBytes.Length;
                for (int i = 0; i < length; i += 4)
                {
                    colorBytes[i + 3] = 255;
                }


                BitmapSource source = BitmapSource.Create(image.Width,
                    image.Height,
                    96,
                    96,
                    PixelFormats.Bgra32,
                    null,
                    colorBytes,
                    image.Width * image.BytesPerPixel);
                videoImage.Source = source;
            }
        }

        void sensor_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
        {
            using (var skeletonFrame = e.OpenSkeletonFrame())
            {
                if (skeletonFrame == null)
                    return;

                if (skeletons == null ||
                    skeletons.Length != skeletonFrame.SkeletonArrayLength)
                {
                    skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
                }

                skeletonFrame.CopySkeletonDataTo(skeletons);
            }

            Skeleton closestSkeleton = skeletons.Where(s => s.TrackingState ==
SkeletonTrackingState.Tracked)
                                                .OrderBy(s => s.Position.Z *
Math.Abs(s.Position.X))
                                                .FirstOrDefault();

            if (closestSkeleton == null)
                return;

            var head = closestSkeleton.Joints[JointType.Head];
            var rightHand = closestSkeleton.Joints[JointType.HandRight];
            var leftHand = closestSkeleton.Joints[JointType.HandLeft];
            var centerHip = closestSkeleton.Joints[JointType.HipCenter];
            var centerShoulder = closestSkeleton.Joints[JointType.ShoulderCenter];

            if (head.TrackingState == JointTrackingState.NotTracked ||
                rightHand.TrackingState == JointTrackingState.NotTracked ||
                leftHand.TrackingState == JointTrackingState.NotTracked ||
                centerHip.TrackingState == JointTrackingState.NotTracked ||
                centerShoulder.TrackingState == JointTrackingState.NotTracked)
            {

                return;
            }
```

```
            SetEllipsePosition(ellipseHead, head, false);
            SetEllipsePosition(ellipseLeftHand, leftHand, leftHandactive);
            SetEllipsePosition(ellipseRightHand, rightHand, rightHandactive);
            SetEllipsePosition(ellipsecenterShoulder, centerShoulder, false);
            SetEllipsePosition(ellipsecenterHip, centerHip, false);

            Controle2(head, rightHand, leftHand, centerHip, centerShoulder);
        }

        private void SetEllipsePosition(Ellipse ellipse, Joint joint, bool isHighlighted)
        {
            if (isHighlighted)
            {
                ellipse.Width = 50;
                ellipse.Height = 50;
                ellipse.Fill = activeBrush;
            }
            else
            {
                ellipse.Width = 20;
                ellipse.Height = 20;
                ellipse.Fill = inactiveBrush;
            }

            CoordinateMapper mapper = sensor.CoordinateMapper;

            var point = mapper.MapSkeletonPointToColorPoint(joint.Position,
sensor.ColorStream.Format);

            Canvas.SetLeft(ellipse, point.X - ellipse.ActualWidth / 2);
            Canvas.SetTop(ellipse, point.Y - ellipse.ActualHeight / 2);
        }

        #region Controle2
            private void Controle2(Joint head, Joint rightHand, Joint LeftHand, Joint
centerHip, Joint centerShoulder)
        {
//************************************** MotorDIREITO ************************
            if (rightHand.Position.X <= (centerShoulder.Position.X + 0.3))
            {
                RodaDireita = "DX000";          //Para o motor direito
                rightHandactive = false;
            }
            else
            {
                if (rightHand.Position.Y < head.Position.Y & rightHand.Position.Y >
centerShoulder.Position.Y)
                {
                    //equaçao de reta para obtençao do valor da velocidade de acordo com a
posiçao da mao [conversao de float para int]
                    VelRodaDireita = (int)Math.Ceiling((155 / (head.Position.Y -
centerShoulder.Position.Y)) * (rightHand.Position.Y - centerShoulder.Position.Y) + 100);
                    if (VelRodaDireita < 180)
                    {
                        VelDireita = "100";
                    }
                    else
                    {
                        VelDireita = "240";
                    }
                    //VelDireita = Convert.ToString(VelRodaDireita); //converte o valor da
velocidade em string
                    RodaDireita = "DF" + VelDireita;      //Liga motor dieito para frente
                    rightHandactive = true;
                }
                if (rightHand.Position.Y < centerShoulder.Position.Y & rightHand.Position.Y
> centerHip.Position.Y)
                {
                    VelRodaDireita = (int)Math.Ceiling((-155 / (centerShoulder.Position.Y-
centerHip.Position.Y)) * (rightHand.Position.Y - centerHip.Position.Y) + 255);
```

```
                    if (VelRodaDireita < 180)
                    {
                        VelDireita = "100";
                    }
                    else
                    {
                        VelDireita = "240";
                    }
                    RodaDireita = "DA" + VelDireita;      //Liga motor direito para traz
                    rightHandactive = true;
                }
            }
//*****************************************Motor ESQUERDO***********************
             if (LeftHand.Position.X >= (centerShoulder.Position.X - 0.3))
            {
                    RodaEsquerda = "EX000";      //Para o motor esquerdo
                    leftHandactive = false;
            }
            else
            {
                if (LeftHand.Position.Y < head.Position.Y & LeftHand.Position.Y >
centerShoulder.Position.Y)
                {
                        RodaEsquerda = "EF100";     //Liga motor dieito para frente
                        leftHandactive = true;
                }
                if (LeftHand.Position.Y < centerShoulder.Position.Y & LeftHand.Position.Y >
centerHip.Position.Y)
                {
                        RodaEsquerda = "EA100";     //Liga motor direito para traz
                        leftHandactive = true;
                }
            }
             Roda = RodaDireita + RodaEsquerda;
             if (Roda != RodaOld)
             {
                 Verify.AppendText(Roda);
                 Verify.AppendText(Convert.ToString(VelRodaDireita));
                 SerialCmdSend(Roda);
                 //SerialCmdSend(SerialData.Text);
                 //SerialData.Text = "";
                 RodaOld = Roda;
             }
        }
        #endregion

        #region Controle1
            private void Controle1(Joint head, Joint rightHand, Joint leftHand, Joint
centerHip, Joint centerShoulder)
        {
            if (rightHand.Position.Y > centerShoulder.Position.Y & leftHand.Position.Y <
centerShoulder.Position.Y)
            {
                //if (!rightHandactive)
                //{
                    rightHandactive = true;
                    System.Windows.Forms.SendKeys.SendWait("{d}"); // carro anda para a
direita
                //}
            }
            //else
            //{
            //    rightHandactive = false;
            //}
            //-------------------------------------------------------------------------------
---------
            else if (leftHand.Position.Y > centerShoulder.Position.Y & rightHand.Position.Y
< centerShoulder.Position.Y)
            {
                    leftHandactive = true;
```

```
                    System.Windows.Forms.SendKeys.SendWait("{a}"); //carro anda para a
esquerda
            }
            //-------------------------------------------------------------------------------
---------
            else if (leftHand.Position.Y > centerShoulder.Position.Y & rightHand.Position.Y
> centerShoulder.Position.Y &
                 rightHand.Position.X > centerHip.Position.X + 0.3 & leftHand.Position.X <
centerHip.Position.X - 0.3)
            {
                    leftHandactive = true;
                    rightHandactive = true;
                    System.Windows.Forms.SendKeys.SendWait("{w}"); //carro anda para frente
            }
            //-------------------------------------------------------------------------------
---------
            else if (rightHand.Position.Y > centerHip.Position.Y & leftHand.Position.Y >
centerHip.Position.Y &
                rightHand.Position.Y < centerShoulder.Position.Y & leftHand.Position.Y <
centerShoulder.Position.Y &
                rightHand.Position.X > centerHip.Position.X + 0.3 & leftHand.Position.X <
centerHip.Position.X - 0.3)
            {
                    leftHandactive = true;
                    rightHandactive = true;
                    System.Windows.Forms.SendKeys.SendWait("{s}"); //carro anda para traz
            }
            //-------------------------------------------------------------------------------
---------
            else if (rightHand.Position.Y < centerHip.Position.Y & leftHand.Position.Y <
centerHip.Position.Y &
                rightHand.Position.X > centerHip.Position.X + 0.3 & leftHand.Position.X <
centerHip.Position.X - 0.3 )
            {
                    leftHandactive = true;
                    rightHandactive = true;
                    System.Windows.Forms.SendKeys.SendWait("{q}"); //carro para
            }
             else
            {
                leftHandactive = false;
                rightHandactive = false;
            }
            //-------------------------------------------------------------------------------
----------
        }
        #endregion
```

Apêndice 4 : Configuração da tela de aplicação

```
<Window x:Class="KinectPowerPointControl.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Kinect Car Control"
        Height="480"
        Width="640"
        WindowState="Maximized">
    <Viewbox Stretch="Uniform">
        <Grid>
            <Image Name="videoImage"
                   Width="640"
                   Height="480"></Image>
            <Canvas Background="Transparent">
                <Ellipse Fill="Red"
                         Height="20"
                         Width="20"
                         Name="ellipseLeftHand"
                         Stroke="White" />
                <Ellipse Fill="Red"
                         Height="20"
                         Width="20"
                         Name="ellipseRightHand"
                         Stroke="White" />
                <Ellipse Fill="Red"
                         Height="20"
                         Width="20"
                         Name="ellipseHead"
                         Stroke="White" />
                <Ellipse Fill="Yellow"
                         Height="20"
                         Width="20"
                         Name="ellipsecenterShoulder"
                         Stroke="Black" />
                <Ellipse Fill="Yellow"
                         Height="20"
                         Width="20"
                         Name="ellipsecenterHip"
                         Stroke="Black" />
                <RichTextBox Height="47" Name="Commdata" Width="216" Canvas.Left="7"
Canvas.Top="421" Background="White" BorderBrush="Black" />
                <Button Content="Conectar" Height="24" Name="Conectar" Width="62"
Canvas.Left="7" Canvas.Top="368" Click="Conectar_Click_1" />
                <TextBox Height="20" Name="Comm_Port_Names" Width="63" Text="COM4"
Canvas.Left="7" Canvas.Top="397" BorderBrush="Black" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center" />
                <Button Content="Enviar" Height="24" Name="Enviar" Width="67"
Canvas.Left="76" Canvas.Top="369" Click="Enviar_Click_1" />
                <TextBox Height="20" Name="SerialData" Width="146" Canvas.Left="76"
Canvas.Top="397" BorderBrush="Black"/>
                <TextBox Canvas.Left="237" Canvas.Top="420" Height="47" Name="Verify"
Width="394" BorderBrush="Black" />
            </Canvas>
        </Grid>
    </Viewbox>
</Window>
```