GARRA ROBÓTICA E VISÃO COMPUTACIONAL COM TOPCONDES

Almeres
E-mail:
Danilo Martins de Brito Fialho
E-mail: danilo__danilo@hotmail.com
Everton
E-mail:
Jaderson
E-mail:
Murilo
E-mail:
Paullo
E-mail:
Wallace
E-mail:

Escola Técnica Estadual Professor Agamemnon Magalhães
Disciplina de Metodologia Científica

AGRADECIMENTOS

A todos que contribuíram para o aprendizado e desenvolvimento deste projeto, onde alguns serão citados a seguir em forma de agradecimento por seus esforços, e por sempre acreditara que isso poderia ser possível. Regison Venâncio, professor de Metodologia Científica, que nos acompanhou e nos orientou na elaboração deste Artigo Científico. Jener Toscano, professor de Microcontroladores e Eletrônica Digital, que nos auxiliou nas duvidas a respeito do Arduino e Eletrônica. Décio Ferraz, professor de Fabricação Mecânica e Tecnologia dos Materiais, que contribuiu com sua vasta experiência para a elaboração de toda a parte Mecânica do presente projeto. Humberto Junior, Bacharel em Sistema da Informação, que auxiliou em toda a parte de programação Java.

RESUMO

Este trabalho visa o desenvolvimento de um sistema automato de coleta seletiva para identificação de objetos e execução da uma tarefa programada. Foi utilizada uma aplicação Java que executa o processamento de imagens, e um Arduino para o controle de uma garra robótica. A pesquisa foi fundamentada nos conhecimentos adquiridos no decorrer do curso de Mecatrônica e dividida em três vertentes: Informática, Eletrônica e Mecânica. Com isso é possível conceituar a funcionalidade da visão computacional na robótica, e sua importância para desenvolvimento da tecnologia priorizando a didática e a simplicidade para detalhar pontos importantes e as dificuldades na pesquisa e execução do projeto.

Palavras-chave: Visão Computacional. Arduino. Garra Robótica. TopCodes. OpenCV. Java. Eclipse.

INTRODUÇÃO

A robótica mundial vem se desenvolvendo a passos largos, e a necessidade de uma maior interação entre o homem e máquina é cada vez mais importante. Para que os robôs tenham a possibilidade de entrar em locais inacessíveis ou até mesmo executar tarefas mais simples como servir um café, eles necessitam de uma persepção a mais. É neste ponto que a visão computacional se destaca, dando a possibilidade de perceber multiplos objetos, o seu formato, tamanho e distância parece ser algo simples para quem enxerga, mas já pensou em como seria sem toda essa percepção? Com toda certeza esse é um fato limitador para a robótica atual. Por este motivo tivemos a ideia de abordar um assunto novo, que tem muito a se desenvolver ainda. A ilustração abaixo exemplifica os níveis de abordagem atingidos por este artigo.

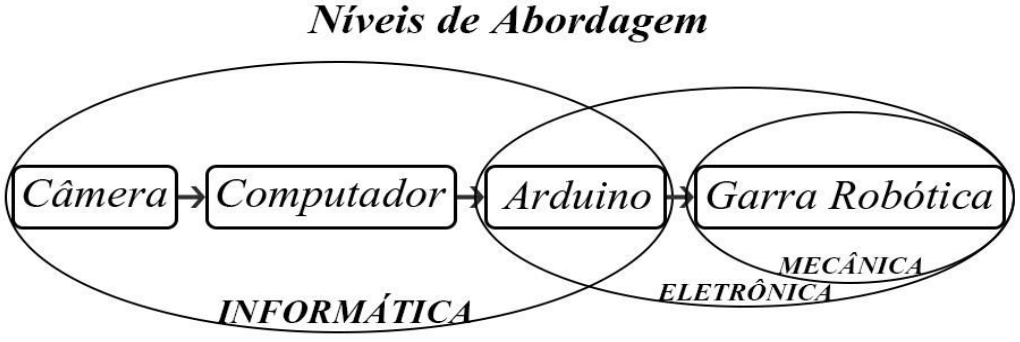


Ilustração 1 - Diagrama dos níveis de abordagem do projeto.

1. INFORMÁTICA

A informática é um termo usado para descrever o conjunto das ciências relacionadas ao armazenamento, transmição e processamento de informações em meios digitais e é considerada uma das bases da robótica. Por meio da programação, atravez de um determinado estímulo digital, pode-se definir a transmição e resposta que se deseja de um sistema, executando assim funções complexas com alta eficiência e precisão. A programação está descrita nas considerações finais onde seu link se encontrará para livre acesso, mas mesmo com as principais linhas do programa comentadas algumas partes necessitam de enfase neste artigo.

1.1. FERRAMENTAS DE PROGRAMAÇÃO

No desenvolvilmento do programa foi utilizado a Linguagem de Programação Java, por sua portabilidade e facilidade, e o programa Eclipse, por sua confiabilidade. O Java necessita de um JDK[1] para editar o programa, e de um JRE[2] para executar o programa, e para entender o processamento do programa é necessário entender um pouco sobre a estrutura de funcionamento do Java. As aplicações em Java se utilizam de uma JVM[3] que executa a leitura do programa em linguagem de máquina, sendo isolada do sistema operacional pela JRE que se encarrega de toda a compreeção independente do sistema local, pois ja é arquitetada para cada sistema operacional individualmente. Quando um programa é feito pode ser lido em qualquer computador que tenha uma JRE devidamente instalada.

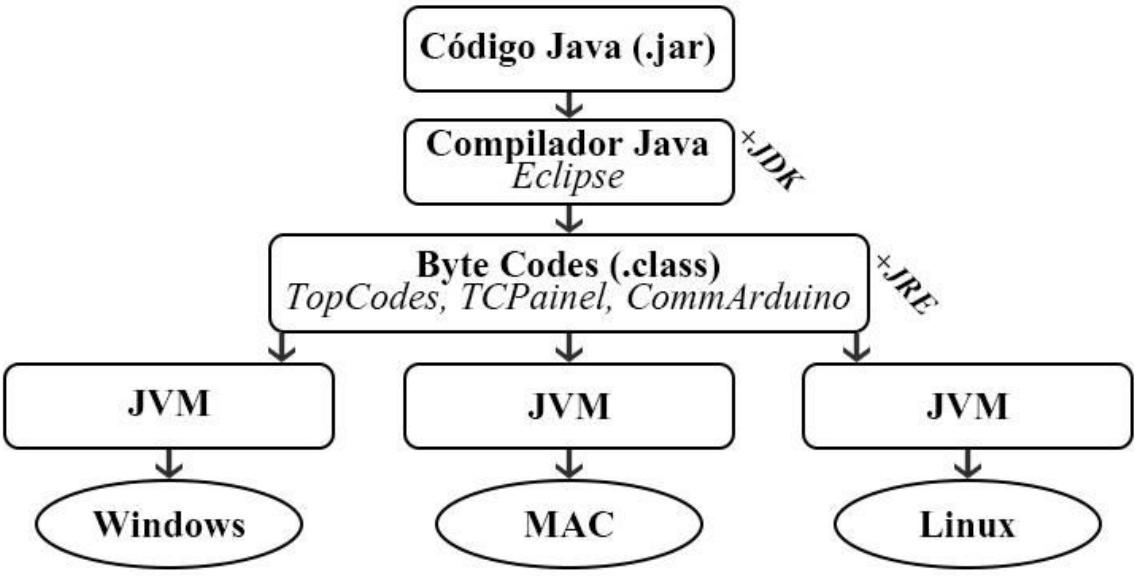


Ilustração 2 - Esquema de funcionamento Java.

[1] JDK: Pacote de desenvolvimento para aplicações em Java, que já vem incluso a JRE² da mesma versão.
[2] JRE: Pacote necessário para executar aplicações em Java.
[3] JVM: Java Virtual Machine. “Máquina Virtual Java”.

O Eclipse é grátis e a versão instalada foi a "*MARS 4.5.0 - Java EE IDE*", ja que é a versão configurada para programação em Java. As bibliotecas necessárias para a execução do programa foram apenas para referênciar a utilização dos arquivos ". *jar*", que são os reais necessários para a execução.

1.2. BIBLIOTECAS

A biblioteca *TopCodes*[4] é a responsável por amazenar os algorítimos de reconhecimento dos códigos captados pela câmera, utilizando-se também da biblioteca *WebCam*. Ela pode reconhecer noventa e nove tipos de códigos diferentes representados por uma circunferência preta com interrupções brancas, formando assim o código de 13 *bits* para leitura onde se tem os zeros representados pelo preto e os ums representados pela cor branca. O *TopCode* ainda pode fornecer sua localização do centro (eixos x e y) em relação a imagem, a distância (eixo z), a rotação do código e o seu diâmetro. A biblioteca *RxTx*[5] é a respossável pela comunicação com o Arduino, enviando os dados configurados que serão explanados na classe *CommArduino*. Outra biblioteca importante é a *JavaCV*[6], que é também utilizada pela biblioteca *TopCodes*. Ela tem como papel principal a filtragem da imagem para a identificação do código, e funciona simplesmente transformando a imagem em escala de cinza e comparando os *pixels* com a do algorítimo interno que determina o código a ser identificado, podendo até identificar um *TopCode* com apenas 25x25 *pixels*.

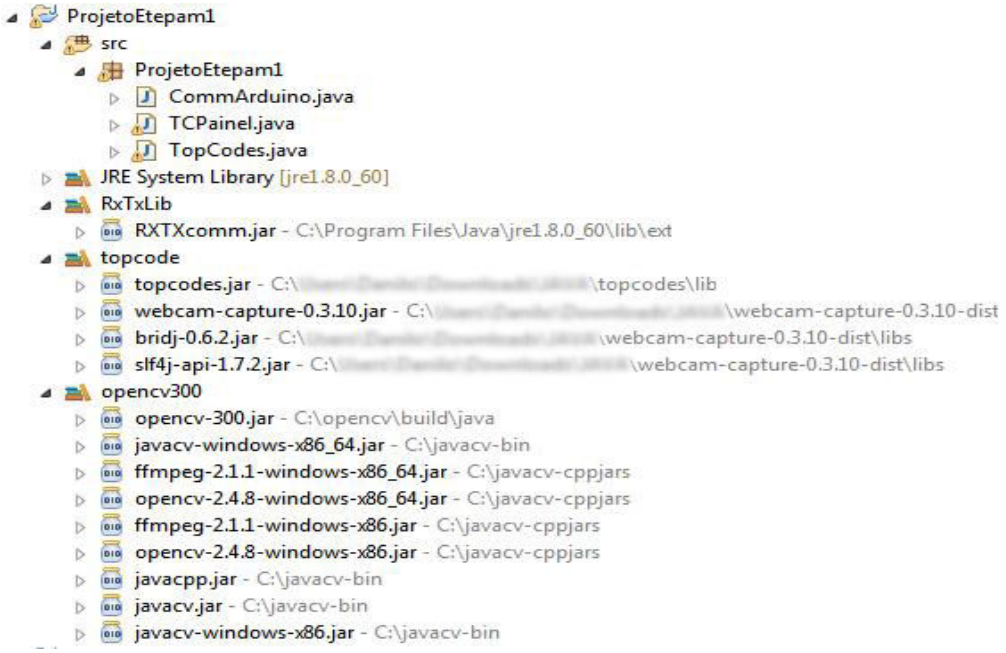


Ilustração 3 - Imagem das bibliotecas e a localização dos arquivos "*.jar"*.

[4] *TopCodes*: Tangible Object Placement Codes. "Código de Localização de Objeto Tangível".
[5] *RxTx*: *Rx* é sigla para recepção de dados, e *Tx* para a transmissão.
[6] *JavaCv*: Funciona como um emulador da biblioteca *OpenCv* (que é nativa da linguagem C++) para aplicações Java.

1.3. CLASSE TOPCODES

Foi tomado como referência principal para esta parte do programa o livro "*Vision-based User Interface Programming in Java*", de Andrew Davison, sendo esta a classe responssável pela execução da função *main*[7] e de todas as outras responsáveis pela execução do programa.

Ilustração 4 - Ilustração dos TopCodes.

1.4. CLASSE TCPAINEL

Nesta classe é iniciado o protocolo de comunicação com o Arduino, as propriedades da janela, as configurações de entrada, as configurações de desenho do topcode, a preparação e o envio das variáveis utilizando-se da terceira classe do programa, a "*CommArduino.java*", entre outras configurações. A função principal desta classe é a *TrackTarget*, que é responsável pela indentificação do código e armazenamento do *TopCode* numa variável inteira i*d*. A câmera USB está sendo inicializada na variável *CAMERA_ID,* que é correspondente ao número ordenado de conecção em seu computador, sendo zero como padrão e um como a primeira conecção USB estabelecida. A tela foi configurada para ter 640 por 480 *pixels*[8], e é a responsável pela referência do *TopCode* em relação a imagem da câmera, onde o centro da tela tem as coordenadas 320 (no eixo x) e 240 (no eixo y).

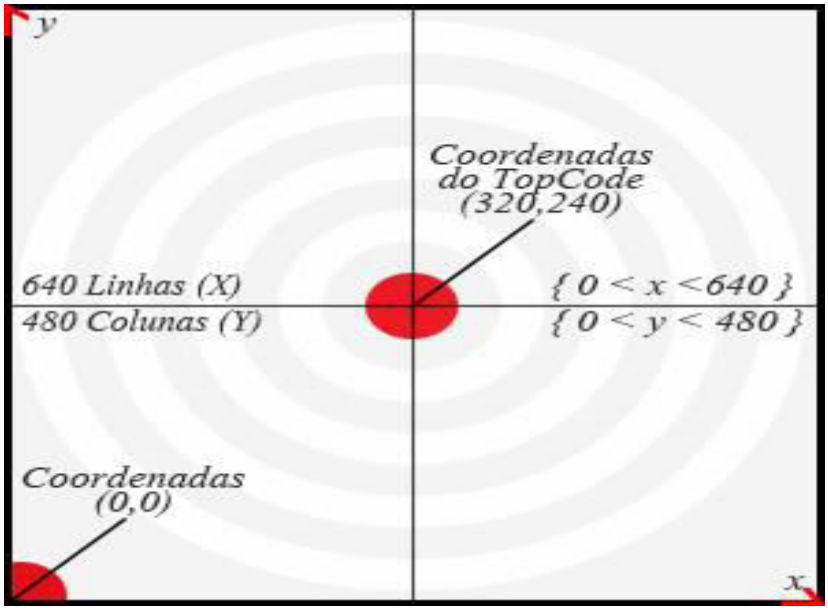


Ilustração 5 - Representação do TopCode em relação a câmera.

[7] *Main*: Significa principal. Neste caso é utilizada como a função principal do programa.
[8] *Pixels*: São os pontos que formam a imagem da tela.

1.5. CLASSE COMMARDUINO

A classe *CommArduino* é responssável apenas pela definição de configurações de comunicação entre o computador e o Arduino. Nela está definida a teixa de comunicação (9600 *Baud's*[9]), e a porta em que ele está conectado (*COM3*). Para determinar a quantidade máxima de bits que o arduino pode transmitir levamos em consideração a frequência do oscilador e o protocolo de sincronização que é o *Serial_8N1* (padrão do Arduino). Este protocolo define a transmição de 8bits de transmição sem *paridade*[10] e um bit de parada, tendo o intervalo de mudança de 104µs de um bit a outro. Três variáveis serão enviadas nesta comunicação serial: o valor do eixo x (*pixels* da câmera em relação ao centro do *TopCode)*, distância entre a câmera e o objeto, e o valor do código. São definidas, portanto, como valores inteiros decimais e logo após são convertidos em bits para serem enviados em forma de *sinal*[11] da porta USB para o Arduino. É necessário então configurar o programa para não enviar nenhuma informação que seja maior que o valor decimal "255", pois pode ocasionar *Overflow*[12]. Neste caso o limite da comunicação é de 2bytes (8bits), mas o valor necessário para se enviar valores maiores que o inteiro decimal 255 é de 3bytes, como mostra a figura abaixo:

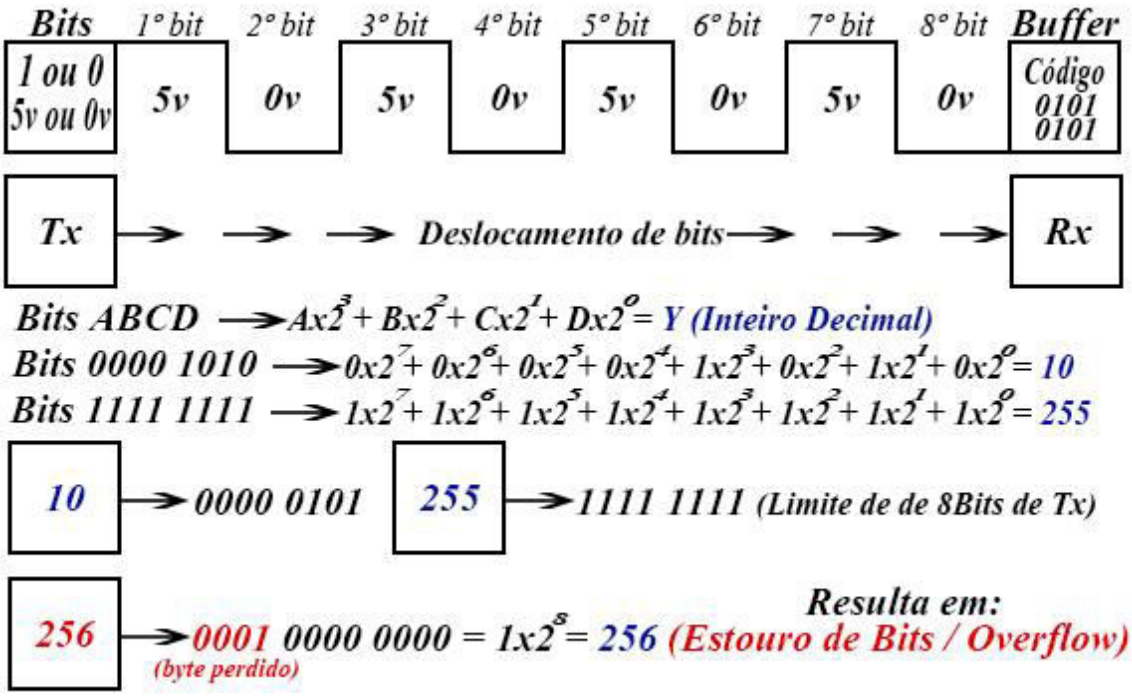


Ilustração 6 - Comunicação *Serial_8N1*.

---

[9] *Baud's*: Unidade de transmissão equivalente ao número de mudanças de fase do sinal transmitido.
[10] *Paridade*: É o estabelecimento de um paralelismo entre as duas partes da comunicação.
[11] *Sinal*: Neste caso se refere à ausência ou presença de tensão, que provoca a leitura da porta receptora do Arduino, formando assim um código binário.
[12] *Overflow*: Quando a resposta de uma adição ou subtração excede a magnitude em que o valor binário que pode ser apresentado.

Foi feito então a divisão dos *pixels* dos eixos x e y por quatro, tendo assim uma mudança de posição no Servo a cada quatro *pixels*. A câmera usada tem três milhões de *pixels*, e por este motivo ainda conserva-se a precisão reduzindo o tamanho dos dados e obedecendo as limitações da comunicação. Esta configuração se da a partir da função "*private static CommArduino gui1 = new CommArduino("COM3", 9600);*", onde é criada uma variável *gui1* para enviar os dados por meio de outra função, a "*gui1.enviadados("~~Dado a ser enviado~~");*". E apenas com estas duas linhas de código é possível estabelecer uma comunicação e enviar uma variável ao Arduino.

```
System.out.println("Enviando Coordenada X:"+ xc+" ->"+cx);
gui1.enviaDados(cx);
try {
    Thread.sleep(10);
} catch (InterruptedException e) {
    e.printStackTrace();
```

Ilustração 7 - Ilustração da parte responsável pelo envio, delay e amostra da variável no Eclipse.

1.6. PROGRAMAÇÃO ARDUINO

A comunicação com os servomotores é feita em microsegundos (10^{-6}) atravéz da função *writeMicroseconds*, para obter uma precisão maior em relação a função padrão, pois pode escrever mil posições (1000 até 2000, sendo 1500 o centro equivalente a 90º). Foi utilizada uma técnica para atualização da variável, e em cada acrescimo da função write existe uma atualização de valor para executar a centralização código com o centro da imagem.

```
void loop(){
fim:
dx = meiox - cx;{Distância em relação ao centro}
posxms = posx;{Atualização da variável}

if (dx >= -16 && dx <= 16){
if (dx > 0){servoX.writeMicroseconds(posxms+=1);
posx = posxms + 1;{Atualização da variável}
goto fim;{Saltar para o laço fim do loop}
```

Ilustração 8 - Técnica para atualização da variável.

As funções *void* estão sendo uzadas apenas como atalhos por questão de organização e espaço, ja que os movimentos se repetem diversas vezes. Existem duas destas funções que merecem uma atenção maior, a "*void RESET*" que serve para zerar as variáveis, finalizar a comunicação serial, e a "*void SETUP"* que é a configuração de início do programa depois da execução de um ciclo de tarefa, juntamente com o reinício da comunicação serial com o Eclipse. A função *T_CODE* é a responsável pela execução relacionada pelo código captado, separando as tarefas de acordo com a variável de entrada do código emitida pelo Eclipse, que são: *AUTO_METAL, AUTO_PLASTICO, AUTO_ORGÂNICO*.

2. ELETRÔNICA

É a cinência que estuda as propriedades, aplicações e desenvolvimento de dispositivos que dependem do movimento dos elétrons em semicondutores para representar, armazenar, transmitir, controlar ou processar informações. Partindo desta definição e do objetivo deste artigo precisamos limitar a abrangência da abordagem do assunto neste artigo, afinal tudo que se aplica na Robótica depende principalmente desta ciência. Sendo assim serão definidos os principais pontos sobre o Arduino e Servomotores com a utilização de imagens simples para demonstrar a montagem do circuito.

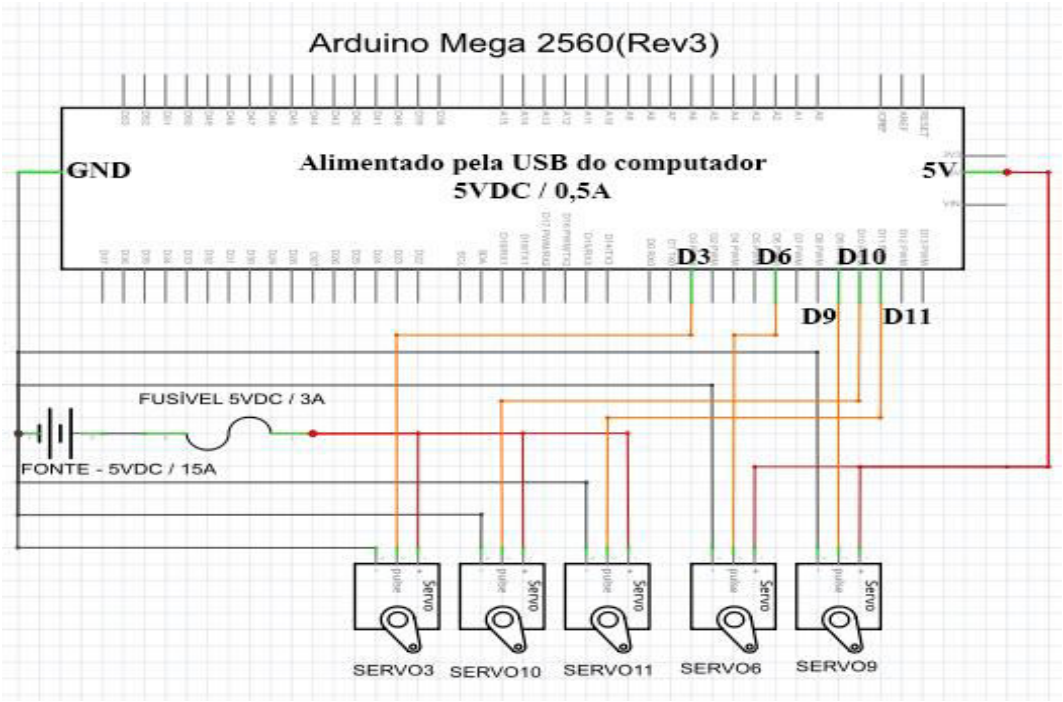


Ilustração 9 - Representação da montagem dos Servos com o Arduino e a Fonte de Alimentação.

2.1. ARDUINO

Utilizamos neste projeto o *Arduino Mega 2560* por possuir um bom desempenho e uma grande quantidade de portas. Tem uma interface para comunicação USB atravez da placa microcontroladora *ATMEGA16U2*, e um *ATMEL ATMGA2560,* que é a principal microcontroladora, possuindo arquitetura *RISC*[13] avançada operando em 8bits. O arduino foi alimentado pela porta USB do computador, ja que há a necessidade da comunicação atravéz do Java, estabelecendo um ponto comum (GND) entre o Arduino e a Fonte externa, para obter a referência durante o envio de dados atravéz das portas *PWM*[14], pois o Arduino tem um limite de *500mA* de corrente total que não deve ser ultrapassado.

[13] *RISC*: Reduced Instruction Set Computer ou Computador com um Conjunto Reduzido de Instruções (RISC).
[14] *PWM*: Pulse Width Modulation. Modulação por Largura de Pulso. Necessária para se enviar sinais analógicos por níveis de resolução digital, atravéz da variação da largura do pulso de 5VDC.

2.2. SERVOMOTORES

Os Servomotores são motores de posicionamento, capazes de se mover e se estabilizar numa posição fixa previamente determinada. Utilizamos neste projeto o Servo *"Tower Pro Micro Servo 9g SG90".* O principal fator avaliado para sua compra foi sem dúvida o custo e a facilidade de uso. Ele tem apenas três pinos (VCC, GND, DATA), capazes de receber informações em microsegundos, posicionando-se assim com até mil posições. Apesar do seu baixo consumo tem um torque equivalente ao seu peso, apenas 1,8kg.cm na ponta do seu eixo. Funciona basicamento com ciclos de valores de pulso para escrever sua posição em microsegundos pela placa controladora interna que controla a rotação do Motor.

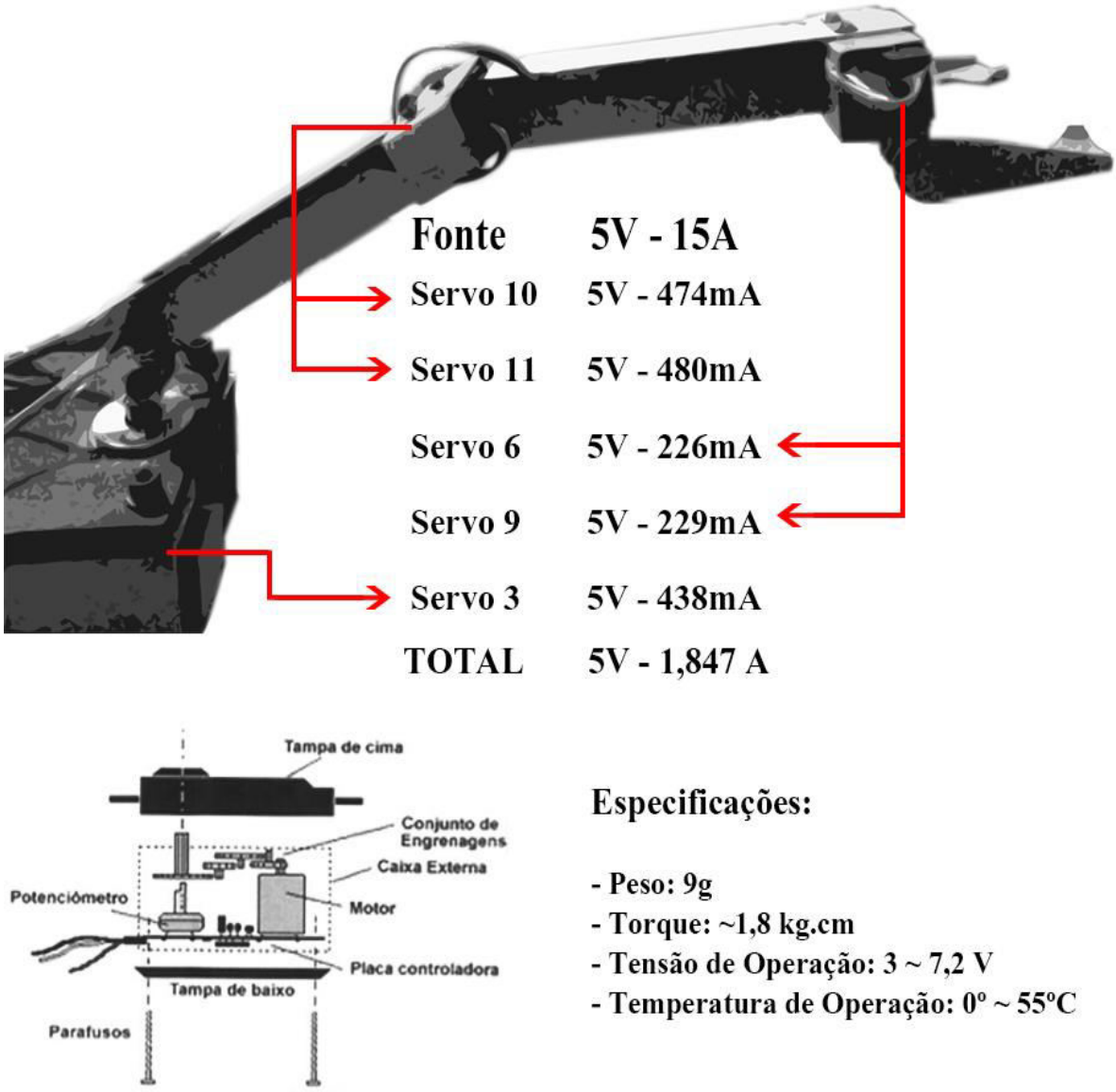


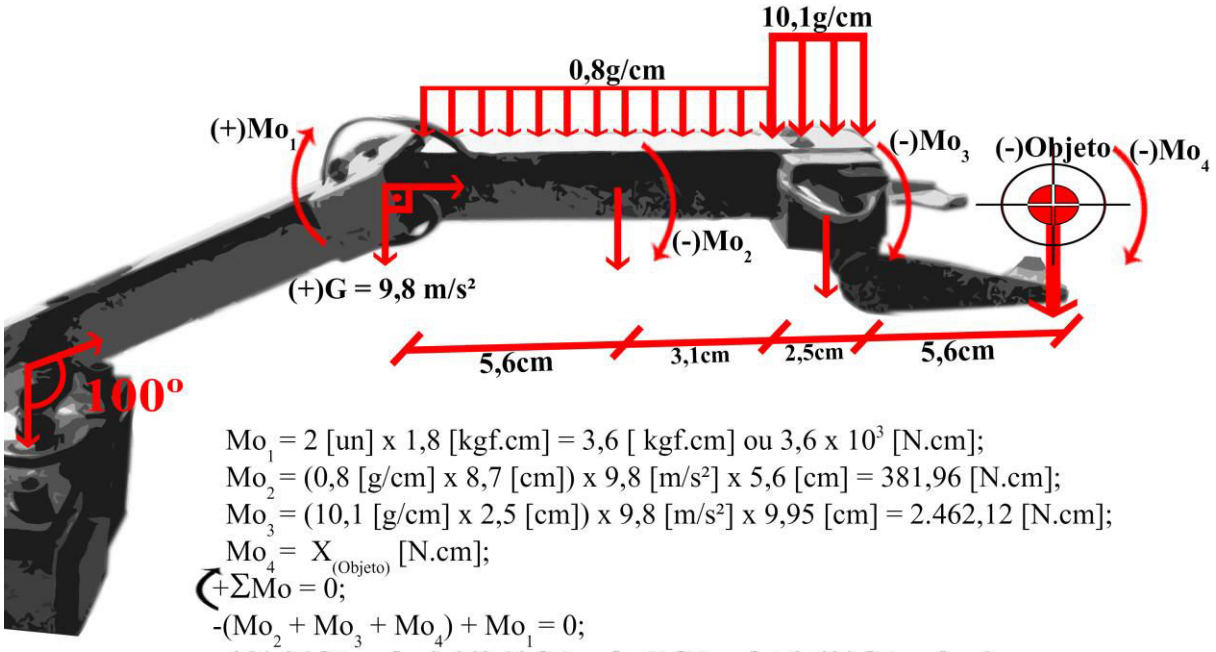
Ilustração 10 - Visão interna do Servomotor, e seus respectivos valores.

3. MECÂNICA

Consiste no ramo da física que estuda o comportamento de sistemas submetidos à ação de uma ou mais forças, análisando movimento e repouso dos corpos, e sua evolução no tempo, seus deslocamentos e seus efeitos subsequentes sobre seu ambiente. Usamos então as necessidades para a escolha de um bom material e a tarefa a ser executada, tendo ciência da necessidade de uma sobra de torque equivalente para um bom trabalho dos equipamentos longe de sua zona limite. Para os calculos mecânicos foram considerados principalmente o peso do material e da carga a ser movimentada, ambos sendo admitidos como uniformes e tendo suas centróides admitidas como regiões retangulares planas. Sendo assim consideramos os esforços em ambito bidimensional, para uma representação mais simples.

3.1. DIAGRAMA DE ESFORÇOS

Foi elaborado um diagrama para uma representação e calculo das principais grandezas físicas responssáveis pela reação a ação gravitacional do corpo em repouso, para saber a quantidade ideal de peso suportado pela Garra Robótica, considerando-se o fator gravitacional em atuação máxima, estando assim a 90º em relação ao solo. Foi também modificada a unidade padrão de Momento (g.m/s².cm, equivalente à N.cm) para simplificação dos calculos.



Mo_1 = 2 [un] x 1,8 [kgf.cm] = 3,6 [kgf.cm] ou 3,6 x 10^3 [N.cm];
Mo_2 = (0,8 [g/cm] x 8,7 [cm]) x 9,8 [m/s²] x 5,6 [cm] = 381,96 [N.cm];
Mo_3 = (10,1 [g/cm] x 2,5 [cm]) x 9,8 [m/s²] x 9,95 [cm] = 2.462,12 [N.cm];
Mo_4 = $X_{(Objeto)}$ [N.cm];
$+\Sigma Mo = 0$;
$-(Mo_2 + Mo_3 + Mo_4) + Mo_1 = 0$;
- 381,96 [N.cm] - 2.462,12 [N.cm] - X [N.cm] + 3.600 [N.cm] = 0;
- 2.844,08 [N.cm] + 3.600 [N.cm] = X [N.cm];
X [N.cm] = 755,92 [N.cm];
Mo_4 = 755,92 [g.m/s².cm] / (9,8 [m/s²] x 16,8 [cm]) ~ 4,59 [g];
Podendo assim levantar objetos com aproximadamente 4,59g;

Ilustração 11 - Diagrama de Esforços da Garra.

CONSIDERAÇÕES FINAIS

O presente Artigo se solidificou na simplicidade, com o intuito de divulgar não só os resultados, mas os pontos mais importantes que envolveram este projeto, e apesar da grande dificuldade em encontrar materiais para esta pesquisa conseguiramos comprovar os conceitos e alcançar o objetivo proposto.

A parte mais importante foi sem dúvida a utilização da biblioteca *TopCodes,* que tornou tudo mais simples, e mesmo com a escassez de materiais para referência em português o entendimento do inglês foi de vital importância para o andamento do projeto. A utilização do Java veio após sucessivas decepções, pela enorme dificuldade de se configurar a biblioteca em linguagem C#/C++, a limitação de programas como o *MyrobotLab*, e novamente pela falta de referências, tornando extremamente favorável a utilização do Java.

A maior dificuldade em Eletrônica foi na utilização de uma fonte, já que um circuito preparado especialmente para este projeto havia falhado. Mecânicamente o pricipal fator prejudicial ao andamento foi o torque muito baixo dos motores, mas o seu preço foi de fato compensador. A estrutura de alumínio exedeu o limite de peso, fazendo com que os motores não respondessem as coordenadas corretamente, impossibilitando a ultilização de materiais mais pesados, ou o aumento do raio de atuação da Garra. Assim foi feito o redimensionamento para atender as necessidades do objetivo do trabalho.

O gasto total com o projeto inclui a compra da lista dos seguintes materiais:

- 03 Cantoneiras de Alumínio em T de 10cm (R$ 25,00);
- 20 Parafusos (R$ 5,00);
- 05 Servomotores (R$ 50,00);
- 30 Fios de Encaixe (R$ 10,00);
- 01 Câmera Clone 3Mp (R$30,00);
- 01 Arduino Mega (R$ 80,00);
- 01 Fonte de Alimentação 5V/15A (R$ 30,00);

Todo o programa, e alguns arquivos importantes se encontrarão para livre acesso neste link.

REFERÊNCIAS

DAVISON, Andrew. **Vision-based User Interface Programming in Java (VBI)**. Disponível em: <http://fivedots.coe.psu.ac.th/~ad/index.html>. Acessado em 25 de Outubro de 2015.

DIAS, Klauder. **Comunicação Serial Java-Arduino**. Disponível em: <http://www.embarcados.com.br/comunicacao-serial-java-arduino/>. Acessado em 26 de Outubro de 2015.

FIRYN, Bartosz. **WebCam Capture API**. Disponível em:

<http://webcam-capture.sarxos.pl/>. Acessado em 11 de Novembro de 2015

GROG, Supertick. **MyRobotLab**. Disponível em:

<http://myrobotlab.org/>. Acessado em 11 de Novembro de 2015.

HORN, Michael. **Tangible Object Placement Codes Library (TopCodes)**. Disponível em: <http://users.eecs.northwestern.edu/~mhorn/topcodes/>. Acessado em 25 de Outubro de 2015.

ITSEEZ. **Open Computer Vision**. Disponível em:

<http://opencv.org/>. Acessado em 11 de Novembro de 2015.

JIMBO, SparkFun. **Rules of Serial Communication**. Disponível em: <https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial>. Acessado em 26 de Outubro de 2015.

LINDSAY, Andy. **How to Use the Arduino Servo Library**. Disponível em:

< http://learn.parallax.com/node/179>. Acessado em 11 de Novembro de 2015.

PICORETI, Rodolfo. **Comunicação Serial Arduino**. Disponível em: <http://blog.vidadesilicio.com.br/arduino/basico/comunicacao-serial-arduino>. Acessado em 26 de Outubro de 2015. https://rafaelcoronel.wordpress.com

RANDOM, James. **RxTx Communication**. Disponível em:

<http://rxtx.qbang.org/wiki/index.php/Main_Page>. Acessado em 11 de Novembro de 2015.