IMAGEM E MOVIMENTO: IMPLEMENTANDO O MOVIMENTO AUTÔNOMO ATRAVÉS DO ARDUINO E OPENCV

Ewerton Silva Braga, Romualdo Monteiro de Resende Costa
Centro de Ensino Superior de Juiz de Fora (CESJF)
Rua Halfeld, nº1179 - Centro – Juiz de Fora/MG
braga.ewerton@gmail.com

RESUMO

Com o avanço da tecnologia ligada à robótica, inovações permitem que entusiastas possam criar protótipos com diversos recursos e funcionalidades. Como exemplo, neste trabalho foi desenvolvido um robô composto por uma placa de prototipagem de baixo custo (Arduino Uno) capaz de, através de um aplicativo Java, realizar a detecção facial através de processamento de imagens e fazer com que esse robô localize e siga até o seu objetivo. Para realizar a computação visual foram utilizadas as bibliotecas OpenCV e JavaCV e a tecnologia Bluetooth para realizar a comunicação do Software com o Hardware, fazendo com que o robô localize e siga seu objetivo.

ABSTRACT

With the advancement of technology related to robotics, innovations allow enthusiasts can create prototypes with various features and functionality. As an example, this work was developed a robot composed of a low-cost prototyping board (Arduino Uno) able to , through a Java application perform face detection through image processing and make this robot locate and follow up your goal. To carry out visual computing used the OpenCV libraries and JavaCV and Bluetooth technology to make communication of the Software with Hardware , causing the robot to locate and follow your goal.

1. INTRODUÇÃO

O campo da robótica é fascinante, pois, a cada dia, inovações e tecnologias permitem a esses dispositivos programáveis a realização de mais tarefas (Romero, 2014). Adicionalmente, componentes com baixo custo e, portanto, cada vez mais acessíveis (Banzi, 2011) além de contribuir para o desenvolvimento dessa tecnologia, permitem que um número cada vez maior de pessoas estejam envolvidas nesse desenvolvimento.

Dentre as áreas de pesquisas envolvidas no projeto da robótica, aquela que trata do planejamento da sua trajetória tem recebido grande atenção por parte dos pesquisadores. Em parte, isso é devido à necessidade inerente de oferecer mobilidade aos robôs. Sem essa característica, aplicações domésticas, de exploração ou mesmo automotivas, além de muitas outras, seriam inviáveis. Considere, como exemplo, o projeto Google[1] de

[1] www.google.com

carro sem motorista (Google Car)[2]. Nesse projeto, o automóvel consegue, de forma autônoma, deslocar-se da origem até o destino, obedecendo às regras de trânsito do local em coexistência com carros tradicionais.

Diversas estratégias podem ser utilizadas para o planejamento da trajetória de robôs autônomos. Na realidade, mais de uma estratégia diferente deve ser usualmente adotada, principalmente em ambientes não estruturados, como é o caso daquele empregado no projeto Google Car, onde o objetivo principal deve ser estabelecer uma trajetória livre de colisão. O planejamento da trajetória pode ainda ser estático ou dinâmico, dependendo, sobretudo, de quando a informação sobre a trajetória ou sobre obstáculos estarão disponíveis. Em um problema estático, toda a informação é conhecida a priori, enquanto que no planejamento dinâmico, grande parte da informação, principalmente sobre obstáculos, somente vai estar disponível durante a execução do trajeto.

Uma das principais estratégias para a implementação de movimentos em ambientes não estruturados consiste na utilização de imagens sobre esse ambiente. Nesse caso, usualmente, as informações extraídas de um vídeo são utilizadas no reconhecimento de padrões que definem o ambiente em questão. Reconhecer um padrão é uma atividade que faz parte das atividades humanas básicas. O ato, por exemplo, de leitura de um texto envolve o reconhecimento de símbolos como letras de um alfabeto e no agrupamento desses símbolos em palavras com algum significado. Assim, define-se que um padrão pode ser uma forma de agrupar objetos semelhantes dentro de uma determinada classe ou categoria, sendo a extração de características relevantes desses objetos automática.

Visando o estudo das atividades de movimento de robôs em ambientes não estruturados, este trabalho tem como objetivo aproveitar as oportunidades oferecidas por componentes de baixo custo para construir um robô móvel. Este projeto, desenvolvido em uma placa, através do processo de prototipagem eletrônica, é capaz de controlar o movimento de uma unidade autônoma baseada em imagens recebidas do ambiente. O microcontrolador do Arduino[3]é responsável por carregar a programação para que o robô móvel execute as funções e responda aos comandos pré-estabelecidos. Essa programação inclui respostas aos sensores que são conectados às portas digitais ou analógicas, dependendo da necessidade do próprio sensor.

Para a análise do ambiente este trabalho utiliza algoritmos baseados na biblioteca de programação OpenCV (Open Source Computer Vision)[4]. Bibliotecas desse tipo são necessárias uma vez que, na maioria dos casos, os métodos de reconhecimento de padrões necessitam de um pré-processamento da imagem antes do reconhecimento. Além disso, a própria biblioteca pode oferecer algumas funções úteis ao rastreamento (Bradski, 2008) que serão abordadas na terceira seção.

Para atuar de maneira satisfatória, sem que haja colisões, os sensores são fundamentais para extrair informações sobre o ambiente ao seu redor. Existem dois tipos de sensores: os Sensores ativos e Sensores passivos (Romero, 2014). Sensores passivos realizam medições a partir de informações externas, presentes no ambiente que incluem, por exemplo, a temperatura, a umidade e a luminosidade. Este projeto emprega um sensor

[2]http://www.ted.com/talks/sebastian_thrun_google_s_driverless_car
[3]www.arduino.cc
[4]www.opencv.org

ultrassonico, considerado um sensor ativo, ou seja, ele é o responsável em gerar e receber a informação, já que ele emite pulsos e realiza estimativas baseadas na energia que foi retornada. A câmera IP utilizada neste projeto é considerada como um sensor passivo, mas pode ser considerada um sensor ativo, caso possua flash ou outro tipo de iluminação.

Os aspectos relacionados ao desenvolvimento do robô são abordados na próxima seção, que apresenta características importantes do Arduino[5], utilizado como base deste trabalho. A Seção 3 trata do processo de reconhecimento de imagens, que inclui a biblioteca OpenCV[6] e JavaCV[7]. A Seção 4 apresenta os testes realizados com o robô autônomo construído a partir das plataformas anteriormente descritas com as conclusões e os trabalhos futuros.

2. CARACTERISTICAS DO PROJETO

O robô (Figura 1) foi desenvolvido com o objetivo de percorrer uma pequena distância definida pela imagem identificada através de sua câmera. Toda a montagem teve como característica principal buscar um equilíbrio das baterias (três no total), uma vez que o seu peso poderia influenciar na rotação dos motores.

Como pode ser observado na Figura 1, o robô possui quatro rodas que, além do movimento em direção ao alvo, identificado pela câmera, permitem a rotação, para que a imagem seja buscada. Uma vez identificado o alvo, a rotação é interrompida e é iniciado um movimento em direção ao alvo, que pode ser complementado, com novas rotações, caso o alvo saia da área de identificação da câmera.

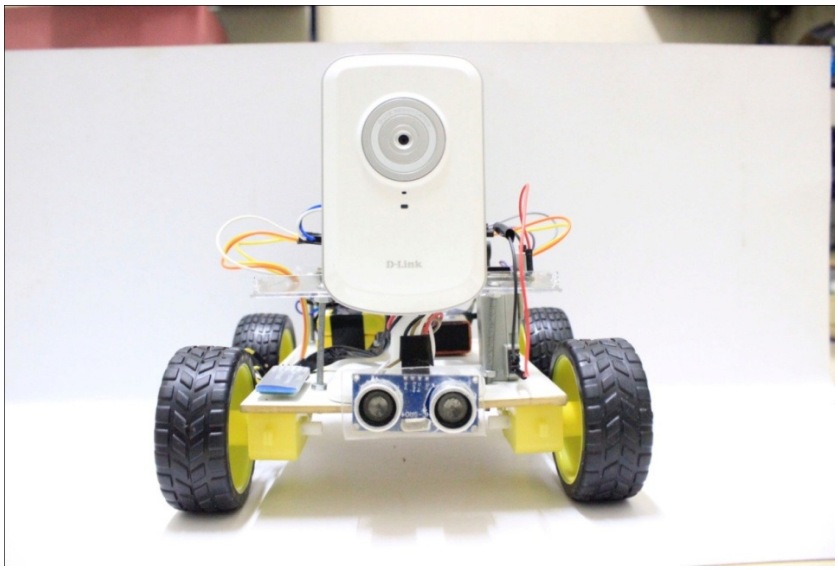

Figura 1. Visão frontal do robô.

[5]www.arduino.cc
[6]opencv.org
[7]code.google.com/p/javacv

2.1 ArduinoUno R3

O Arduino Uno R3 (Figura 2) é a placa de prototipagem, que possui os pinos de entrada/saída analógicos e digitais, além do microcontrolador que é onde são processados os comandos enviados pela aplicação Java[8] (descrita na Seção 3). Nele são conectados outros componentes que irão controlar o robô. O Microcontrolador é um ATMega328, onde a memória flash tem 32K, dos quais 0,5K são utilizados pelo bootloader (inicializador do processo), a SRAM tem 2KB e a EEPROM tem 1KB. A velocidade do clock desse microcontrolador é de 16MHz. Essa placa é alimentada com uma fonte externa de 9V (apesar de o Arduino trabalhar com 5V, a placa possui um regulador de tensão que faz com que a tensão seja regulada para não danificar a placa).

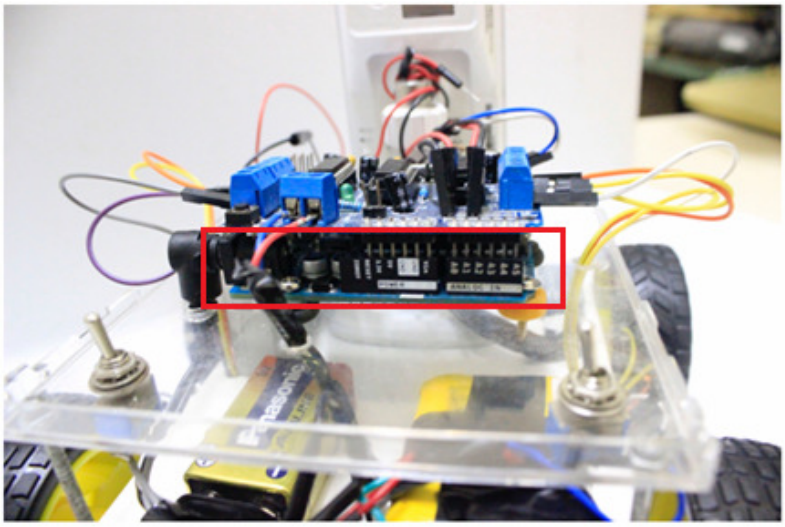

Figura 2. Arduino Uno R3, embaixo do MotorShield, que é encaixado em cima do Arduino.

2.2 MotorShield (l293D)

O MotorShield (Figura 3) é o que controla os motores do robô. É ele quem define a direção da rotação de cada motor, fazendo com que ele tenha os movimentos para frente, para trás, para a esquerda ou para a direita. Além disso, o MotorShield é responsável por controlar a velocidade. Com esta placa é possível controlar 4 motores DC (corrente contínua) ou 2 motores de passo e 2 servos motores. Uma observação importante consiste na necessidade de utilizar uma fonte de alimentação externa para controlar os motores, uma vez que a alimentação utilizada no Arduino não é suficiente para alimentar os motores DC. Assim foram utilizadas duas baterias de 3.7v - 1.3mAh[9] ligadas em série, totalizando 7.4V (além da bateria de alimentação do Arduino, totalizando, portanto, as três baterias). Como pode ser observado na Figura 3, esse componente é encaixado perfeitamente em cima do Arduino, sem a necessidade de utilizar fios para fazer a conexão.

---

[8]www.java.com
[9]HTTPS://PT.wikipedia.org/wiki/Amp%C3%A8re-hora

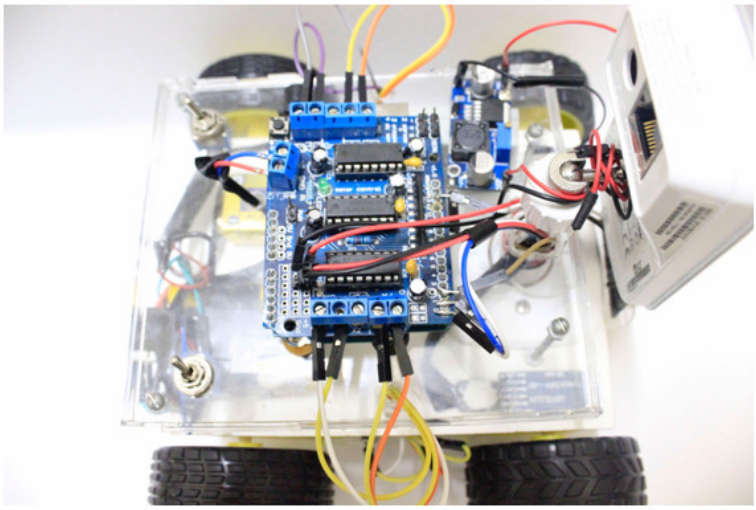

Figura 3. MotorShield L293D, responsável por controlar velocidade e sentido da rotação dos motores.

2.3 Alimentação

No robô foram inseridas duas chaves liga/desliga com o intuito de economizar a alimentação enquanto o robô não está sendo usado, sem a necessidade, portanto, de remoção da fiação. Essas chaves podem ser visualizadas na Figura 4, juntamente com as fontes de alimentação (uma bateria de 9V para alimentar o Arduino e duas baterias de 3,7V ligadas em série para alimentar o MotorShield).

A Câmera IP tem como alimentação 5V - 1.2mAh, sendo assim, foi necessário adicionar um regulador de tensão, já que não foi encontrada nenhuma bateria que atendesse a essa necessidade. Foi utilizada uma bateria de 7.2V - 1.3mAh e com ajuda do regulador de tensão, a saída da bateria que era de 7.2V foi regulada para 5V, aproximadamente, fazendo com que a câmera não precisasse estar conectada a uma rede elétrica, dando mobilidade ao projeto.

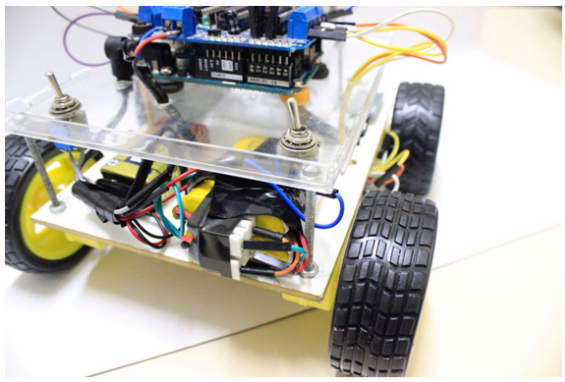

Figura 4. Chaves Liga/Desliga e baterias de alimentação para os motores e o Arduino.

2.4 Regulador de Tensão:

O regulador de tensão (LM2596) apresentado na Figura 8 tem como função ajustar a voltagem de entrada para o valor desejado, sendo capaz de reduzir carga de até 3A. A tensão de saída pode ser ajustada entre 1,5V a 35V, tendo como entrada 3,2V até 40V. O projeto possui uma entrada de 7,2V e a saída gerada é de 5V como pede a câmera IP, tudo regulando através do parafuso localizado na parte azul do regulador.

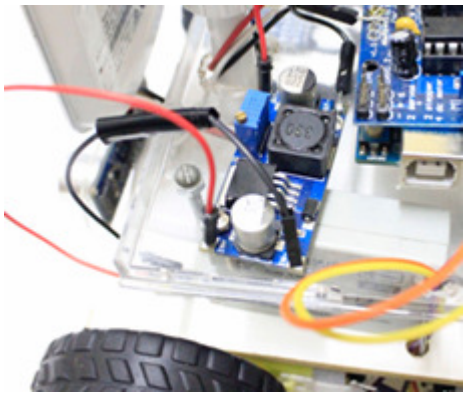

Figura 8. Regulador de Tensão regulando a entrada de acordo com as necessidades.

2.5 Módulo Bluetooth:

O Módulo Bluetooth HC-06permiteestabelecer a conexão entre o Arduino e qualquer outro dispositivo que tenha a tecnologia Bluetooth (Bluetooth, 2015). Ele suporta ambos os modos Slave (Escravo) e Master (Mestre). Tem um alcance de 10 metros e sua frequência de trabalho é de 2,4GHz Banda ISM (banda destinada ao uso industrial, científico e médico). Utiliza o protocolo V2.0+EDR, o que lhe permite uma taxa de transmissão de 3Mbit/s. Possui protocolos de segurança que incluem autenticação e encriptação. Ele é usado para receber as instruções da aplicação Java no Arduino. Esse módulo encontra-se em destaque na Figura 5.

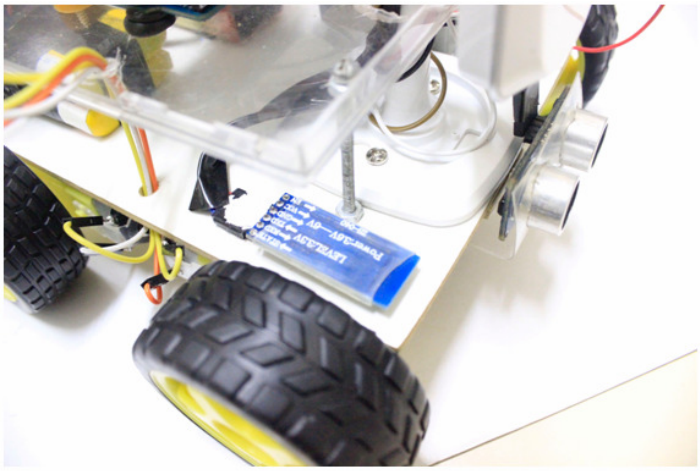

Figura 5. Módulo Bluetooth HC-06, responsável pela conexão entre Arduino e Aplicação.

2.6 Sensor Ultrassonico:

Este sensor envia uma onda sonora que, ao encontrar o obstáculo reflete, assim este sensor é capaz de calcular a distância entre o obstáculo e o sensor. O ângulo de efeito é de 15º e o seu alcance aproximado é de 2cm até 4m, com uma precisão de 3mm. No projeto, o sensor foi utilizado na frente do robô (Figura 6) para evitar a colisão com o "alvo" ou com outros objetos no percurso. Pelo fato de ter um ângulo de efeito de 15º é importante definir a posição dependendo do tamanho do robô para que ele consiga evitar a colisão em qualquer um dos lados.

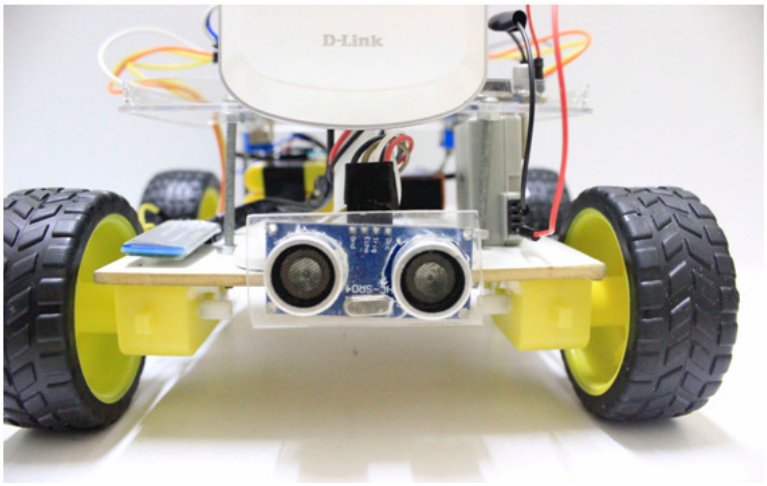

Figura 6. Sensor ultrassonico posicionado na parte frontal do robô.

2.7 Câmera IP

A Câmera IP (DLink DCS-930L) apresentada na Figura 7 é conectada na mesma rede Wifi (Kurose, 2014) do computador que executa a aplicação de reconhecimento facial. O acesso a ela é realizado através da biblioteca JavaCV, que é uma interface para o OpenCV e cuja implementação será discutida na próxima seção.

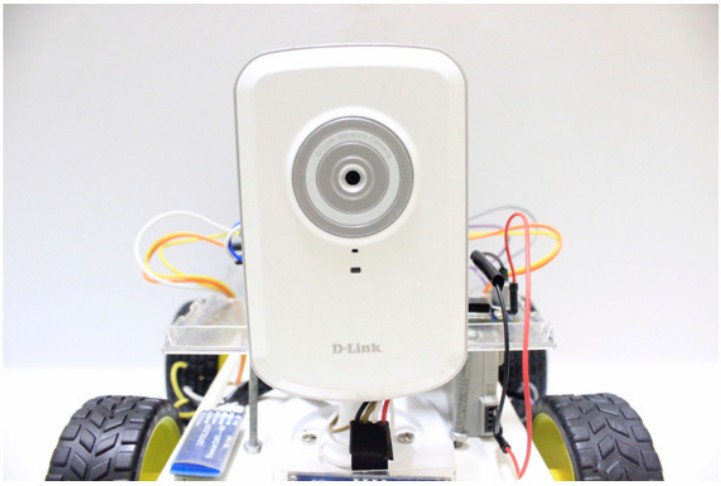

Figura 7. Câmera IP posicionada a frente do robô, neste exemplo, sendo um Sensor Passivo.

2.8 Sketch do Arduino

Sketch é o nome que o Arduino usa para um programa. É a sequência de códigos, sejam elas funções, condições ou outros comandos que são enviados para o Arduino para que ele execute as instruções.

Na Figura 9 estão as declarações de variáveis e os parâmetros de velocidade dos motores, bem como a velocidade da porta serial. Mais especificamente, nas linhas 1 e 2 estão as importações das bibliotecas utilizadas. A primeira é a AFMotor.h, uma biblioteca desenvolvida pela Adafruit [10] para o MotorShield, que permite alterar a velocidade dos motores através do método setSpeed().Esse método recebe valores inteiros (*Integer*) entre 0 e 255, onde 0 é desligado e 255 é velocidade máxima, que pode variar de acordo com a amperagem da bateria, quanto maior, mais rápido é o desempenho do motor. Outro método desta biblioteca é o run(), que será comentado mais a diante.

Outra importação utilizada é a biblioteca, Ultrasonic.h que foi desenvolvida pelo espanhol J.Rodrigo e pode ser encontrada no github[11].

Neste projeto os pinos do sensor ultrassonico ligados são Trigger e Echo. O primeiro(trigger) é responsável por enviar o sinal sonoro e o segundo é o retorno, dado pelo cálculo da média simples de distância percorrida mantendo-se a aceleração constante, exemplificado pela fórmula a seguir:

$$d = \frac{(V * t)}{2}$$

onde:

d = distância entre o sensor e o obstáculo;

V = Velocidade do som no ar (343m/s aproximadamente);

t = tempo gasto pelo sinal desde a saída do sinal do trigger, colisão com o obstáculo e retorno para o echo.

A variável serialByte do tipo char, irá armazenar o valor recebido pela porta serial. Ela é utilizada na função loop(); e será melhor explicada na Figura 11.

O comando AF_DCMotor realiza a inicialização de cada motor.
O motor1 está ligado na entrada1 do MotorShield e o motor2 na entrada 2 do MotorShield e assim sucessivamente para os motores 3 e 4.

A função Setup() é chamada no Arduino assim que o programa inicia. Geralmente, essa função é usada para inicializar variáveis, definir pinos como entrada ou saída. No

[10] www.adafruit.com
[11] https://github.com/JRodrigoTech/Ultrasonic-HC-SR04

Setup() deste projeto esse método é empregado para definir a velocidade dos motores através do método setSpeed() (mencionado anteriormente). Adicionalmente é definida a velocidade da comunicação serial em bits por segundo (bauds), neste exemplo, 9600 através do método Serial.begin().

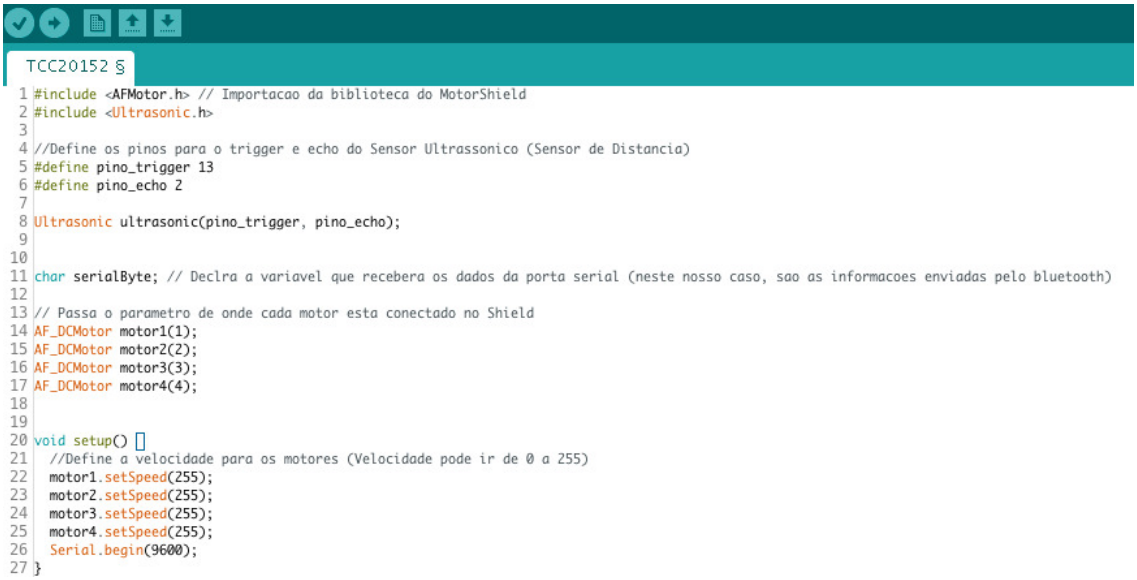

Figura 9. Código de inicialização do Arduino

A Figura 10 apresenta os comandos para controle dos motores, através do método run() onde RELEASE interrompe o funcionamento dos motores, FORWARD realiza o giro em sentido horário, BACKWARD no sentido anti-horário. O método delay() define o tempo de execução da função, especificado em milissegundos. Nas funções frente(), direita() e esquerda() são informadas as direções respectivas para que o motor gire por 80ms, seguido da chamada ao método parar(). Como exemplo, ao invés de invocar a função parar(), o comando run(RELEASE) poderia ser diretamente invocado.

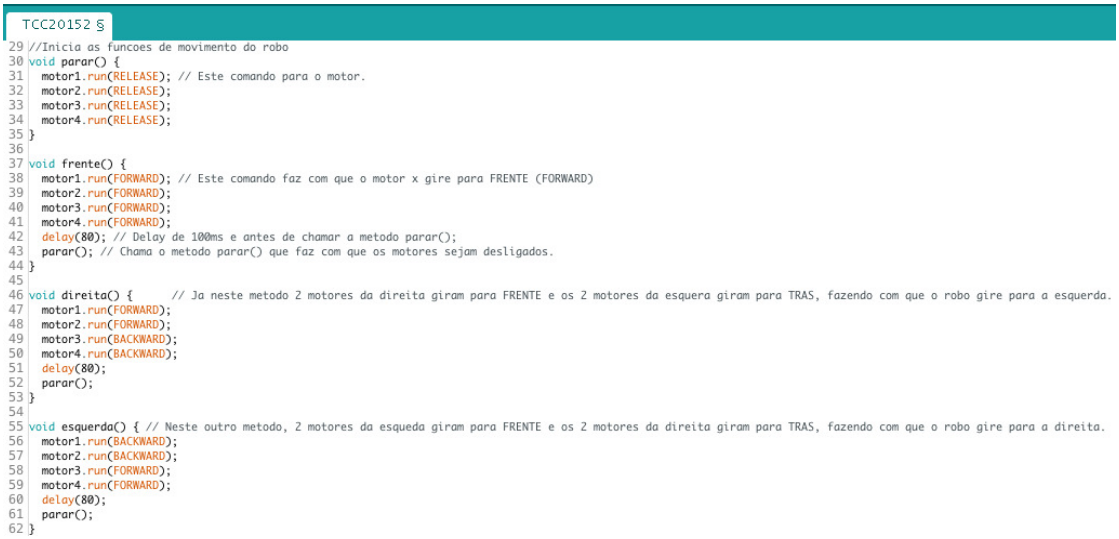

Figura 10. Código para movimento do protótipo.

A Figura 11 apresenta a função principal do sketch, a função loop, que repete contínua e indefinidamente para controlar o Arduino. Dentro dessa função existe um array que receberá os valores da aplicação Java, através do Bluetooth, para controlar o movimento. Na quarta seção serão abordados os testes, e exemplificado o uso desse array. A variável serialByte recebe apenas o valor da quarta posição do array,do total de cinco que foram recebidos através da porta serial, e realiza a comparação, caso o valor da variável for igual a “f” ele invocará a função frente(), se for igual a “d” ele invocará a função direita(), se for igual a “e”invoca a função esquerda() e se for igual a “p”invocará a função parar().

TCC20152 §

```
void loop() {

  char ton[5]; // criamos um array do tipo char com tamanho 5
  int i = 0;
  /* A instrução abaixo verifica enquanto o i for menor que 5 executa a linha o proximo, que é verificar se existe informação na porta serial,
     enquanto a condição for verdadeira, adiciona o valor recebido na porta serial dentro do array, após isso ele */
  while (i < 5) {
    while (Serial.available() > 0) {
      ton[i++] = Serial.read();
      Serial.println(i);
      serialByte = ton[4];
      Serial.print("Serial: ");
      Serial.println(serialByte);
      if (i > 10) {
        loop();
      }
      if (i == 5) {
        if (serialByte == 'f') frente(); // Verifica o valor recebido, se for igual a f o robo anda para frente e assim sucessivamente.
        if (serialByte == 'd') direita();
        if (serialByte == 'e') esquerda();
        if (serialByte == 's') parar();
        else {
          loop();
        }
      }
    }
  }
}
```

Figura 11. Código que recebe os comandos externos e controla o movimento.

3.APLICAÇÃO JAVA

3.1 Telas

Considerando o relativo baixo poder de processamento do Arduino (um microcontrolador de 16MHz), para o processamento de imagens seria necessário um hardware específico como, por exemplo, Raspberry[12], BeagleBone[13] ou similares, integrados ao Arduino. Outra opção para esse processamento seria a utilização de equipamento externo para o processamento com uma comunicação externa, através de protocolo como o Bluetooth com Arduino. Essa segunda opção foi escolhida neste projeto, sendo realizado o processamento através da linguagem Java com utilização da biblioteca OpenCV. Uma das dificuldades encontrada no desenvolvimento da aplicação foi a comunicação Bluetooth. No inicio do projeto a aplicação estava sendo desenvolvida no sistema operacional da *Apple*, o *Mac OS X Yosemite*, ao implementar a

[12]HTTPS://www.raspberrypi.org/
[13]http://beagleboard.org/

comunicação *Bluetooth* utilizando a biblioteca *BlueCove* [14], foram encontrados problemas de incompatibilidade pois não funciona com Java 64-bits. Foi definido então migrar para o Windows 7 32-bit, quando, pesquisando sobre o *BlueCove* comunicando com o Arduino, foi encontrada a biblioteca *Ardulink*[15], que é uma solução Java Open Source para controle da placa Arduino que inclui protocolos de comunicação, componentes Java Swing e muito mais. Esta biblioteca utiliza o BlueCove para a comunicação Bluetooth entre a aplicação Java Desktop e Arduino. A tela principal (figura 12) da aplicação apresenta as tecnologias que foram utilizadas no projeto, o Arduino, Java e OpenCV, além de chamar a classe responsável por estabelecer a comunicação entre a aplicação e o Arduino.

Figura 12. Tela inicial da aplicação Java.

Aberta à tela de conexão (Figura 13), possui um Combobox (componente Swing) que recebe uma lista com os dispositivos Bluetooth, o botão procurar, que inicia a busca pelos dispositivos, os botões connect e disconnect além do botão Iniciar, que inicia a classe responsável pelo processamento das imagens recebidas e envio das coordenadas para o Arduino. O botão iniciar só é habilitado após existir uma conexão, do contrário ele permanece o tempo todo desabilitado.

[14]http://bluecove.org/
[15]http://www.ardulink.org/

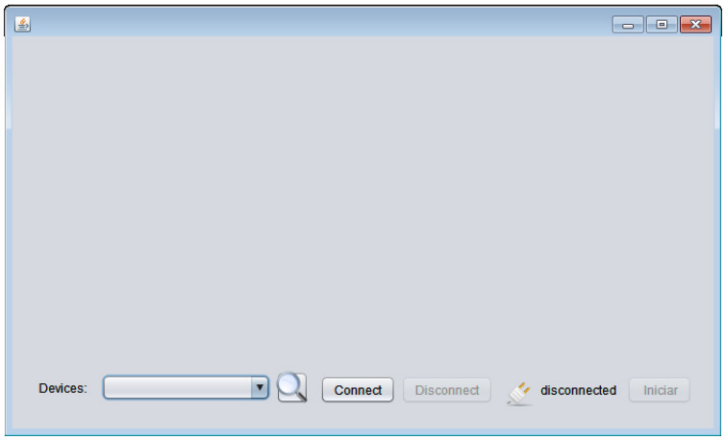

Figura 13. Tela onde é feita a conexão e inicialização da câmera IP.

A figura 14 mostra a busca por dispositivos e no console de saída, a versão do BlueCove que é utilizada neste projeto,2.1.1-SNAPSHOT.

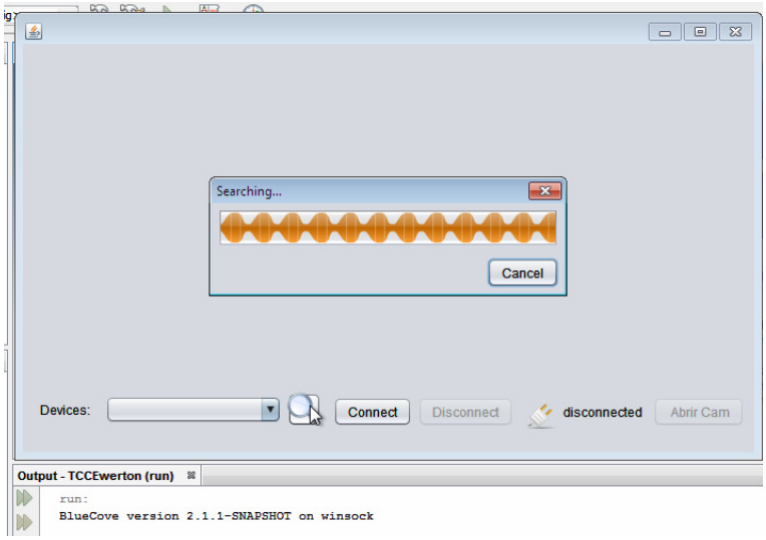

Figura 14. Versão do BlueCove utilizada.

Após finalizar a busca, caso o dispositivo foi encontrado, na lista de *Devices* irá aparecer o nome do dispositivo, e após clicar em *Connect*, a *labeldisconnected* será alterada para *Connected*, o botão *Connect* será desabilitado e os botões *Disconnect* e Abrir Cam serão habilitados, como pode ser observado a seguir (Figura 15).

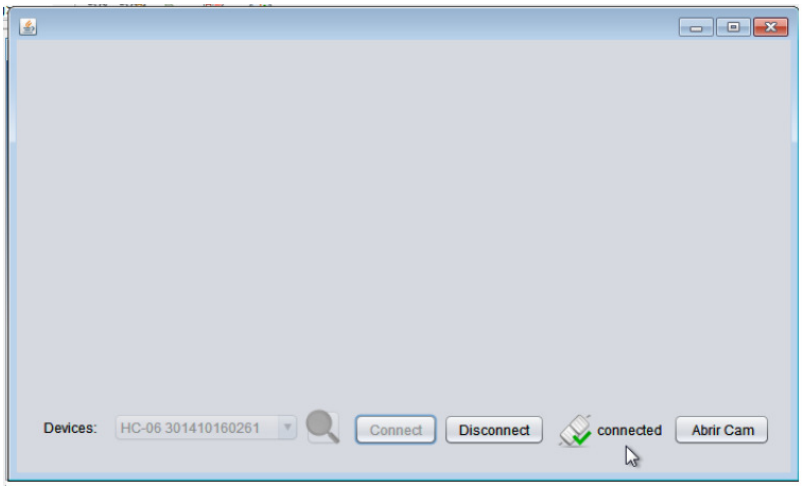

Figura 15. A label altera de Disconnected para Connected e o botão Abrir Cam e habilitado.

Quando pressionado o botão "Abrir Cam", será aberto um *frame* com a imagem ao vivo da Câmera IP, já executando o processamento de imagem em tempo de execução, buscando encontrar uma face, que, quando encontrada, desenha um círculo verde com o centro do círculo centralizado na face, conforme exibido a seguir (Figura 16).

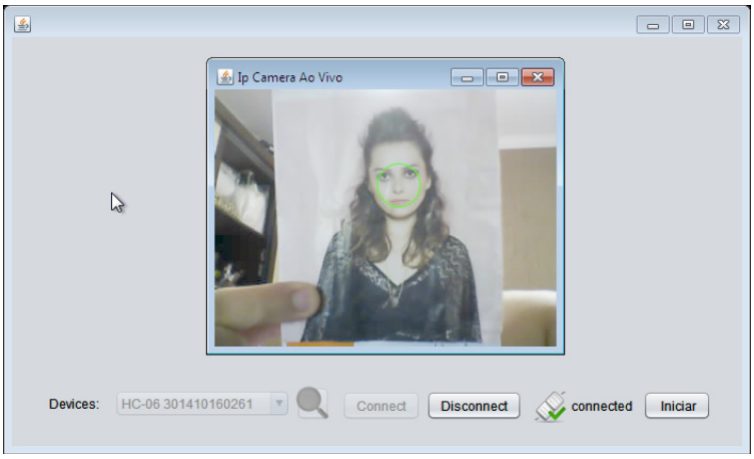

Figura 16. Face detectada e identificada através do círculo verde.

3.2 Classes

A aplicação possui 3 classes, a *Principal.java*, *ConexaoBluetoothArdulink.java* e *ProcessamentoImagem.java*, representadas pelo diagrama de classes abaixo (figura 17).

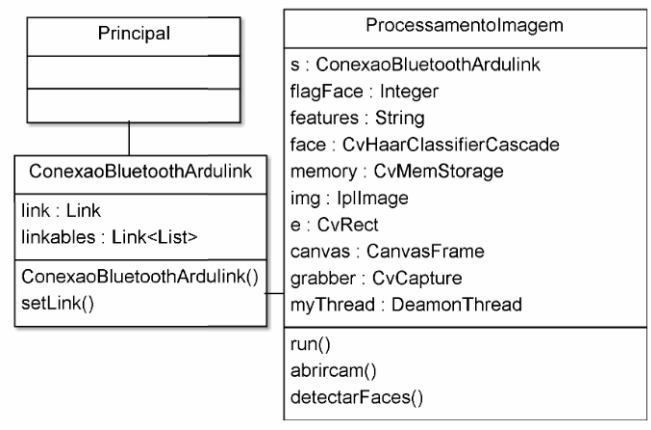

Figura 17. Diagrama de classes da Aplicação Java.

Como a classe *Principal.java* só possui um botão, não se faz necessário descrevê-la aqui. Já a classe *ConexaoBluetoothArdulink.java* possui métodos e atributos, além das importações.

A classe *ConexaoBluetoothArdulink.java* possui um *JPanel* onde é adicionado o *BluetoothConnectionPanel()* que possui a *combobox* com todos os *devices* descobertos, é ele que busca todos os ao clicar no botão *Connect* ele pega o item selecionado e conecta através do *link.connect(deviceName)*, onde *deviceName* é o valor na combobox. Pode ser melhor visualizado na imagem a seguir (Figura 18).

```
133
134
135         bluetoothConnectionPanel = new BluetoothConnectionPanel();
136         connectionPanel.add(bluetoothConnectionPanel);
137
138         btnConnect = new JButton("Connect");
            btnConnect.addActionListener(new ActionListener() {
141             public void actionPerformed(ActionEvent e) {
142                 if(link != null) {
143                     String deviceName = bluetoothConnectionPanel.getSelectedDevice();
144                     link.connect(deviceName);
145                 }
146             }
147         });
            connectionPanel.add(btnConnect);
```

Figura 18. Método de conexão, feita pelo Ardulink.

O botão Iniciar faz uma nova instância da classe *ProcessamentoImagem.java,*que é o responsável pelo processamento das imagens recebidas pela câmera IP, além de já chamar o método que inicia a *Thread* principal. (Figura 19).

```
167
168             abrircam = new JButton("Iniciar");
169             abrircam.setIcon(ton);
            abrircam.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
172                     ProcessamentoImagem c = new ProcessamentoImagem();
173                 c.abrircam();
174                     }
175
176         });
```

Figura 19. Instanciação da classe ProcessamentoImagem.java e invocação do método abrircam();

A classe mais complexa deste projeto é a *ProcessamentoImagem.java*, onde é feito todo processamento das imagens recebidas pela câmera IP, chama o método que, através de um *XML*, procura uma face e desenha um círculo 'e'. Este círculo possui *e.x()*, *e.y()*, *e.width()* e *e.heigh()*, Será utilizado como parâmetro o centro do círculo, dado pela seguinte formula:

$$ton = \frac{e.x() - e.width()}{2}$$

3.2.1 Atributos

Alguns atributos do projeto exibidos na tabela abaixo e na figura a seguir(Figura 20):

```
29  public class ProcessamentoImagem {
30
31      ConexaoBluetoothArdulink s = new ConexaoBluetoothArdulink();
32      IplImage img = new IplImage();
33      int t1 = 0;
34      int ultimox = 0;
35      String features = "haarcascade_frontalface_alt.xml";  //classifiers para detectar faces
36      CvHaarClassifierCascade face = new CvHaarClassifierCascade(cvLoad(features));
37      CvMemStorage memory = cvCreateMemStorage(0);
38      CvRect e;
39      CanvasFrame canvas = new CanvasFrame("Ip Camera Ao Vivo");
40      CvCapture grabber = null;
41      private ProcessamentoImagem.DaemonThread myThread = null;
42
```

Figura 20. Declaração dos atributos existentes na classe.

Tabela 1. Atributos da aplicação, tipo e breve descrição.

Atributo	Tipo	Descrição
s	*ConexaoBluetoothArdulink*	Classe responsável pela conexão.
img	*IplImage*	Suporta subconjunto de formatos.
t1	*Int*	Flag relacionada à detecção de face.
ultimox	*Int*	Recebe último valor de x.
features	*String*	Diretório do arquivo XML ClassifierCascade.
face	*CvHaarClassifierCascade*	Carrega o ClassifierCascade.
memory	*CvMemStorage*	Armazena dados dinâmicos.
e	*CvRect*	Armazena as coordenadas de um retângulo.
canvas	*CanvasFrame*	Frame que exibe o stream das imagens recebidas.
grabber	*CvCapture*	Permite compartilhar imagens entre diferentes APIs.
myThread	*DaemonThread*	Thread que inicializa a câmera e invoca demais métodos.

3.2.2 Métodos

Os métodos utilizados nesta classe são:

Método run(): Inicializa a câmera IP, chama o método *detectarFaces()* e faz com que a o vídeo da câmera seja exibido no *frame* através da linha: *canvas.showImage()*. Além disso é no *run()* que é feita a verificação da *flag* t1, caso ela seja igual a 0, isto significa que nenhuma face foi detectada, então é feita outra verificação, desta vez o ultimo valor de *x* dado pela variável *ultimox*, caso seja maior que a largura da resolução dividida por

2 + 40 (para dar uma margem) ele envia *"d"* para o Arduino, que irá girar para a sua direita, realizando a busca da face. Se o valor for menor que a largura da resolução dividida por 2 – 40 a aplicação envia *"e"* para o Arduino que, ao contrário da primeira condição, irá iniciar suas buscas para o lado esquerdo. Se o *ultimox* não possuir nenhum valor (primeira busca, sem ainda não ter detectado nenhuma face) ficou definido que ele irá inicializar as buscas pela direita. (Figura 21)

```
52  public void run() {
53      grabber = cvCreateFileCapture("http://admin:romualdo123@192.168.43.10/video.cgi?.mjpg");
54      grabber = cvCreateCameraCapture(0);
55        while(runnable){
56
57        while (true) {
58
59               img = cvQueryFrame(grabber);
60
61               if (img != null) {
62                    t1 = 0;
63                   detectarFaces(img);
64                   canvas.showImage(img);
65
66              if (t1 == 0){
67                  if (ultimox > img.width()/2+40)
68                  {
69                 s.link.writeSerial("d");
70                       System.out.println("Procurando para a DIREITA!");
71                  }
72
73                  else if (ultimox < img.width()/2-40)
74                  {
75                 s.link.writeSerial("e");
76                   System.out.println("Procurando para a ESQUERDA!");
77                  }
78                  else {
79                      s.link.writeSerial("1");
80                        System.out.println("Procurando para ALEATORIAMENTE");
81                  }
82                                                  }
83                                              }
84                                          }
85                                      }
86  }
```

Figura 21. Thread principal inicializando a câmera IP, invocando o método de detecção facial e execução da busca.

Método abrircam(): Instancia e inicializa a *Thread* e inicializa a mesma. (Figura 22)

```
88  public void abrircam  () {
89        myThread = new ProcessamentoImagem.DaemonThread();
90        Thread t = new Thread(myThread);
91        t.setDaemon(true);
92        myThread.runnable = true;
93        t.start();
94  }
95
```

Figura 22. Instanciando uma nova Thread e *startando.*

Método detectarFaces: Neste método é onde é realizado a busca pela face e, após detectada, envia as orientações para o Arduino. Caso não seja encontrada nenhuma face ele retorna, caso encontre, ele desenha um círculo de cor Verde definido por *CvScalar.GREEN*, onde *CVScalar* trabalha com cores RGB, onde o centro do círculo é o centro da face detectada. Dentro do método é crida uma variável para receber o centro do círculo, já descrito acima. Após a detecção da face ele verifica o valor da variável que armazenou a posição do centro do círculo, se estiver entre centro da resolução dividido por 2 + 40 ou centro da resolução dividido por 2 - 40 significa que está

“centralizado”, então a aplicação envia *“f”* para o Arduino, para que siga em Frente pois o objetivo está centralizado no vídeo, caso o valor seja maior que o centro da resolução / 2 + 40 ele altera a *flag* t1 de 0 para 1, ou seja, informa que uma face foi localizada, sendo assim pode parar as buscas e envia para o Arduino *“d”*, para que o robô vire para a direita, se o valor foi menos que o centro da resolução / 2 – 40 também altera a *flag* t1 de 0 para 1 ordenando que parem as buscas e envia *“e”* para que o Arduino vire para a esquerda. (Figura 23)

```
103  public void detectarFaces(IplImage img){
104
105    CvSeq faces = cvHaarDetectObjects(img, face, memory, 2.0, 3, 0);
106    if (faces.total() == 0) return;
107
108    e = new CvRect(cvGetSeqElem(faces, 0));
109
110   cvCircle(img,cvPoint( (2*e.x()+e.width())/2,(2*e.y()+e.height())/2), e.width()/4,
111                          CvScalar.GREEN, 1, CV_AA, 0);
112
113    int ton = (e.x()+e.width()/2);
114
115     if (ton > img.width()/2-40 && ton < img.width()/2+40){
116          t1 = 1;
117          System.out.println("Centralizado! Mover para Frente!");
118          s.link.writeSerial("f");
119
120       if (ton > img.width()/2+40)
121       {
122          t1 = 1;
123          System.out.println("direita ---->");
124          s.link.writeSerial("d");
125          ultimox = ton;
126       }
127
128       if (ton < img.width()/2-40)
129       {
130          t1 = 1;
131          System.out.println("ESQUERDA <----");
132          s.link.writeSerial("e");
133          ultimox = ton;
134       }
135  cvClearMemStorage(memory); // clear memory for next frame.
136     }
137    }
138
```

Figura 23. Detecção de faces, caso seja detectada, desenha o círculo e verifica a posição de x, enviando as coordenadas de acordo com as condições.

Quando o Arduino aproxima-se do seu objetivo, o sensor de distância irá fazer com que o Arduino execute a função *parar()* fazendo com que o Arduino pare a 5 cm do objetivo. Esta parte é realizada apenas no Arduino, já que a aplicação Java não controla o sensor de distância.

4. Testes, conclusões e projetos futuros.

Os testes com o robô autônomo foram bastante satisfatórios, os resultados surpreenderam apesar de alguns problemas encontrados ao longo do projeto.

O primeiro teste foi a conexão com *Bluetooth* do Arduino. O tempo gasto pela Aplicação para buscar e encontrar o dispositivo *Bluetooth* em média foi de 20 segundos. Quando se realizava a tentativa de conectar ao dispositivo, foi constatado que em algumas vezes o dispositivo não respondia após um período de tempo e em muitos

casos era necessário fechar e abrir a aplicação novamente. Após alguns testes em relação a este erro foi constatado que este problema acontecia quando a tentativa de conexão era feita após um determinado tempo, quando a tentativa era realizada imediatamente após a busca do dispositivo, a conexão foi bem sucedida em quase 100% das tentativas. Os testes da busca foram realizados apenas com 1 dispositivo, sendo assim, não existem dados em relação à duração da busca caso existam mais de 1 dispositivo disponível no mesmo ambiente. O tempo gasto para a conexão era praticamente imediato, menos que 2 segundos.

Os testes com a câmera IP já exigiram um pouco mais de estudos, sendo que o primeiro problema foi descobrir que o *OpenCV* não trabalha bem com *Stream* no formato *mjpg*[16], a solução encontrada foi através do *grabber* do *JavaCV*, mencionado nos atributos.

O primeiro teste com a Câmera estava configurada a resolução máxima suportada, de 640x480, e, por se tratar de uma Câmera IP, há um *delay* inicial de aproximadamente 10 segundos após a inicialização da *Thread*. Após o inicio do *Stream*, a imagem capturada durante este tempo é passado em velocidade aumentada até chegar ao estado real. Caso nenhuma face seja detectada, o *delay* da imagem é de aproximadamente 1 segundo e com face detectada o *delay* aumenta para 2.5 segundos. Após pesquisas e, seguindo a lógica, com uma resolução menor a tendência é que o *delay* diminuísse e foi o que realmente aconteceu. O tempo do *delay* inicial para iniciar o *stream* foi de aproximadamente 8 segundos, quando nenhuma face é detectada, menor que 1 segundo e com a face detectada o *delay* permanecia o mesmo.

Devido ao tempo de resposta do Arduino ser imediato, com o *delay* maior o objetivo nunca era alcançado, com isso, reduzir a resolução para 320x160 foi a solução para o problema do *delay* da Câmera IP.

Com uma fotografia de um rosto de aproximadamente 5 cm, com uma boa iluminação a distância máxima em que a face foi detectada foi de 52cm. A partir desta distância a face era detectada e perdida, detectada e perdida, fazendo com que o robô se movesse para frente e depois executava a busca inúmeras vezes. O mesmo aconteceu com uma face de aproximadamente 10cm, com a diferença de que a distância aumentou de 52cm para 80cm. Já uma face real, o alcance se estendeu para aproximadamente 2m (ver tabela 1).

Um problema encontrado em relação ao Arduino era a quantidade de informações recebidas pela porta serial. A *Thread* do Java é executada **N** vezes, fazendo com que a quantidade de comandos, sejam eles *"d"* para virar a direita, *"e"* para virar à esquerda e *"f"* para seguir em frente eram muitas e como o Arduino recebe essas informações sequenciais, ele acabava se perdendo, por executar muitas vezes o mesmo comando. A solução encontrada foi criar um *Array* de 5 posições que receberia estas informações, e comparava apenas 1 posição, com isso, mesmo que aplicação *Java* enviasse 10 vezes

[16]HTTPS://tools.ietf.org/html/rfc2435

"e" para o Arduino, ele só executaria este comando 2 vezes. Com isto, o problema de comandos enviados foi resolvido e a precisão da busca aumentou satisfatoriamente.

A iluminação também se faz bastante necessária, uma vez que, quanto melhor, mais detectável a face se tornava.

As conclusões dos testes foram boas, uma vez que com uma fotografia de boa qualidade, uma iluminação adequada e um ambiente fixo o robô atendeu as expectativas e conseguiu chegar ao seu objetivo.

Na tabela a seguir podem ser observadas as comparações com diferentes faces:

Tabela 2. Comparações dos resultados em faces diferentes.

Tipo	**Distância de Reconhecimento**	**Distancia limite próxima ao objetivo**	**Precisão**
Rosto Humano Real	2m	80 cm (aprox.)	100%
Rosto Humano Fotografia (5cm)	52cm	10 cm (aprox.)	100%
Rosto Humano Fotografia (10cm)	80cm	20 cm (aprox.)	100%

Os próximos projetos prevêem a implementação do Reconhecimento Facial, fazendo com que o robô, não só detecte face como também diferencie uma da outra e siga apenas a face que lhe foi solicitada, além da migração da placa Arduino para o RaspberryPI ou Intel Galileo Gen 2[17], sendo possível executar o processamento de imagens na própria placa, eliminando a necessidade uma rede Wifi e o uso da câmera IP.

Com a possibilidade de alterar o arquivo Cascade Classifier (XML), é possível alterar o padrão de busca do robô, fazendo com que ele procure objetos por sua forma, cor, além de poder criar o seu próprio Cascade Classifier a partir de treinamento melhorando o desempenho da detecção.

[17]http://www.intel.com.br/content/www/br/pt/embedded/products/galileo/galileo-overview.html

5.REFERÊNCIAS

ARDULINK, “a complete, open source, Java solution for the control and coordination of Arduino boards. Communication protocol, java SWING components collection, Network Server and more…”Disponível em: <http://www.ardulink.org/ardulink-messages/> , Acesso em Setembro de 2015.

BANZI, M. Primeiros Passos com o Arduino. ISBN 978-85-7522-290-4. NOVATEC, 2011.

BRADSKI, G.; KACHLER, A. LearningOpenCVComputerVisionwiththeOpenCV. O’Reilly Media, Safari Books Online, 2008.

KUROSE, J. Redes de Computadores e a Internet: uma abordagem top down. ISBN 858-14-3677-3. Editora Pearson, 2014.

ROMERO, R. Robótica Móvel. ISBN 978-85-2162-230-38. LTC, 2014.

BLUETOOTH. Specification of the Bluetooth System. Bluetooth SIG Property.Julho 2015.

WENDLING JR., J. M. A. (2009) “CI Reguladores de Tensão”, Disponível em: <http://www2.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/2---ci-reguladores-de-tensao---v1.0.pdf>,Acesso em 20 de outubro de 2015.