UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PEDRO DORIGHELLO FOLTRAN

CAPTURA E TRANSMISSÃO DE DADOS DE IMAGENS DE VISÃO GLOBAL ESPECÍFICA PARA O FUTEBOL DE ROBÔS

BAURU
2017

PEDRO DORIGHELLO FOLTRAN

CAPTURA E TRANSMISSÃO DE DADOS DE IMAGENS DE VISÃO GLOBAL ESPECÍFICA PARA O FUTEBOL DE ROBÔS

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Renê Pegoraro

BAURU
2017

Foltran, Pedro Dorighello.
CAPTURA E TRANSMISSÃO DE DADOS DE IMAGENS DE VISÃO GLOBAL ESPECÍFICA PARA O FUTEBOL DE ROBÔS/ Pedro Dorighello Foltran, 2017
28 p. : il.

Orientador: Prof. Dr. Renê Pegoraro

Monografia (Graduação)–Universidade Estadual Paulista.
Faculdade de Ciências, Bauru, 2017

1. Futebol de Robôs. 2. Módulo de Visão Global. 3. Banana Pi.

Pedro Dorighello Foltran

CAPTURA E TRANSMISSÃO DE DADOS DE IMAGENS DE VISÃO GLOBAL ESPECÍFICA PARA O FUTEBOL DE ROBÔS

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Renê Pegoraro
Orientador

Profª. Drª. Simone das Graças Domingues Prado
Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Prof. Dr. Wilson Massashiro Yonezawa
Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Bauru, 06 de Fevereiro de 2017.

Saol, Grá, Gáire.

Resumo

A UNESP Bauru participa desde 1998 de competições de Futebol de Robôs. Para o funcionamento dos robôs são necessários diversos módulos, como o módulo de Estratégia e o módulo de Visão Global. Atualmente, o módulo de Visão Global da UNESP de Bauru possui atrasos na transferência de informações das imagens, em virtude do meio utilizado para transferência e de compactação realizada pela câmera antes do envio das imagens, trazendo consequências para o funcionamento dos outros módulos. Uma nova câmera que incorpora parte do módulo de Visão Global é proposto com o objetivo de reduzir este problema, utilizando um Banana Pi e uma câmera acoplada, o qual realiza pré-processamento das imagens, reduzindo seu tamanho e compactando-a, para então, enviar as imagens via rede Ethernet com protocolo UDP.

Palavras-chave: Futebol de Robôs; Visão Global; Banana Pi.

Abstract

The UNESP Bauru participates since 1998 in Robot Soccer competitions. To the robot be able to operate, many different modules are necessary, such as the strategy module and the Global Vision Module. Nowadays, the Global Vision Module from UNESP Bauru has delays to transfer the images, due to the mean of transmission and the compaction performed by the camera before sending the image, bringing consequences to the operation of the other modules. A new camera, which is part of the Global Vision module is proposed with the aim to reduce this problem, using a Banana Pi and a camera coupled, which perform a preprocessing of the images, reducing their size and compacting, and then, sending those images through Ethernet cable with UDP protocol.

Keywords: Robot Soccer; Global Vision; Banana Pi.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

As competições de futebol de robôs tem como objetivo o desenvolvimento de pesquisas na área de inteligência artificial e robótica. O esporte, por ser dinâmico e em tempo real, proporciona um desafio maior se comparado ao Xadrez e devido a isso envolve diversas tecnologias e algoritmos necessários para conseguir que o robô jogue futebol de maneira razoável (KITANO, 1998). Para incentivar essas pesquisas, desde o final do século passado, diversos campeonatos de Futebol de Robôs tem sido propostos em variadas categorias.

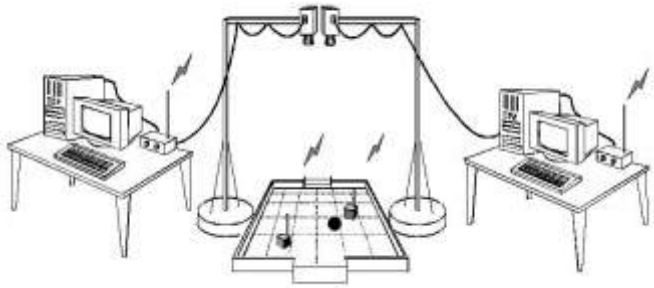
A UNESP de Bauru participou pela primeira vez em uma competição de Futebol de Robôs em 1998, na primeira Copa Brasil de Futebol de Robôs, tendo participado, no mesmo ano, da terceira edição da *Micro-Robot World Cup Soccer Tournament*, realizada na França e organizada pela FIRA (*Federation of International Robot-soccer Association*), com o time "Guaraná" em parceria com a POLI-USP onde conquistou o vice-campeonato da categoria (COSTA; PEGORARO, 2000). Atualmente, possui uma equipe em expansão, desenvolvendo tanto o *software* quanto o *hardware* dos robôs.

Esses *softwares* e *hardwares* podem ser divididos em módulos: módulo de controle, responsável pela movimentação do time, módulo de estratégia, responsável por definir os objetivos e decisões dos robôs e o módulo de visão, responsável pelo processamento e captura das imagens. O foco neste estudo é a analise do módulo de visão global que é constituído pela câmera, situada 2 metros acima do campo, conforme figura 1 e o *software* de visão. A câmera possibilita determinar todos os estados – velocidades e posições – dos elementos – robôs e a bola – em campo (KOOIJ, 2003) e para qual equipe determinado robô faz parte, baseando-se em etiquetas coloridas fixadas acima dos robôs e pela cor da bola (laranja) (COSTA; PEGORARO, 2000), conforme figura 2. A visão global deve transmitir os dados a pelo menos 25 quadros por segundo (25*fps*) (KOOIJ, 2003), contudo, um maior valor torna melhor a fidelidade das posições e velocidades dos elementos e as consequentes informações obtidas.

O módulo de visão é responsável, portanto, por fornecer dados para que outros módulos determinem as ações a serem tomadas pelos robôs. Estes módulos funcionam em um ciclo de execução, que se repete na frequência das imagens obtidas da câmera.

Atualmente, o módulo de visão da UNESP de Bauru utiliza uma câmera conectada a um computador via USB fornecendo 30 imagens por segundo, mas com um atraso de cerca de 5 quadros. Esse atraso é prejudicial ao Futebol, na medida em que as imagens atrasadas não permitem com que a real posição dos Robôs naquele instante seja determinada. Para se evitar este problema, busca-se o aprimoramento desse módulo, trocando a câmera atual

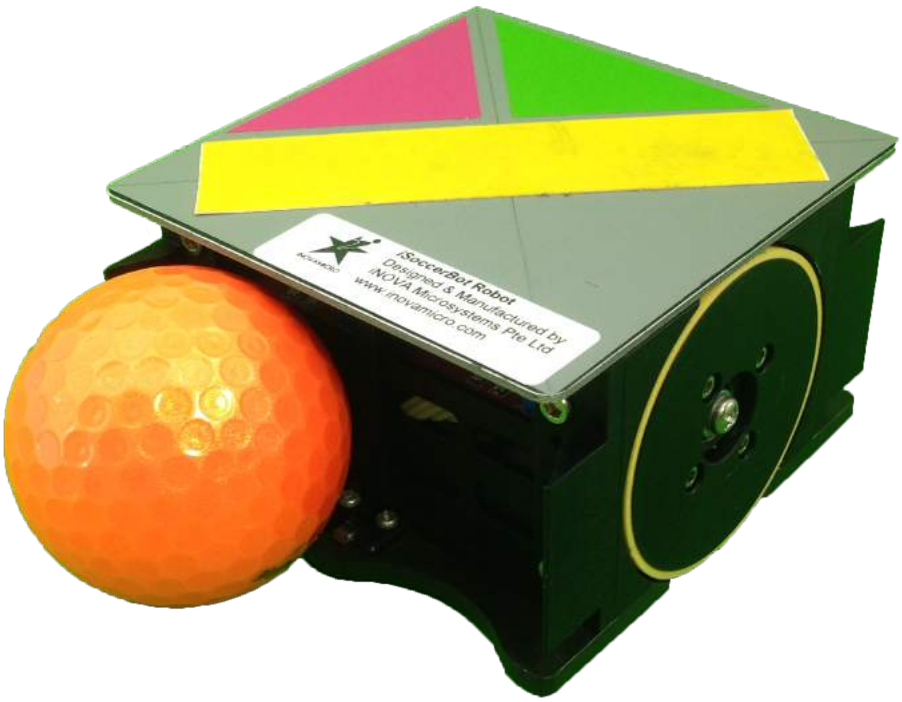
Figura 1 – Esquema de Campo do Futebol de Robôs



Fonte: (COSTA; PEGORARO, 2000)

por um módulo conectado a um Banana Pi. Ele realizará um processamento inicial das imagens e posterior transmissão para o computador responsável pelo controle e tática dos robôs, utilizando protocolo de transmissão já existente em C++, como o protocolo UDP, em substituição do USB para assim reduzir o atraso das imagens que chegam ao computador.

Figura 2 – Robô e Bola Utilizados nos Jogos

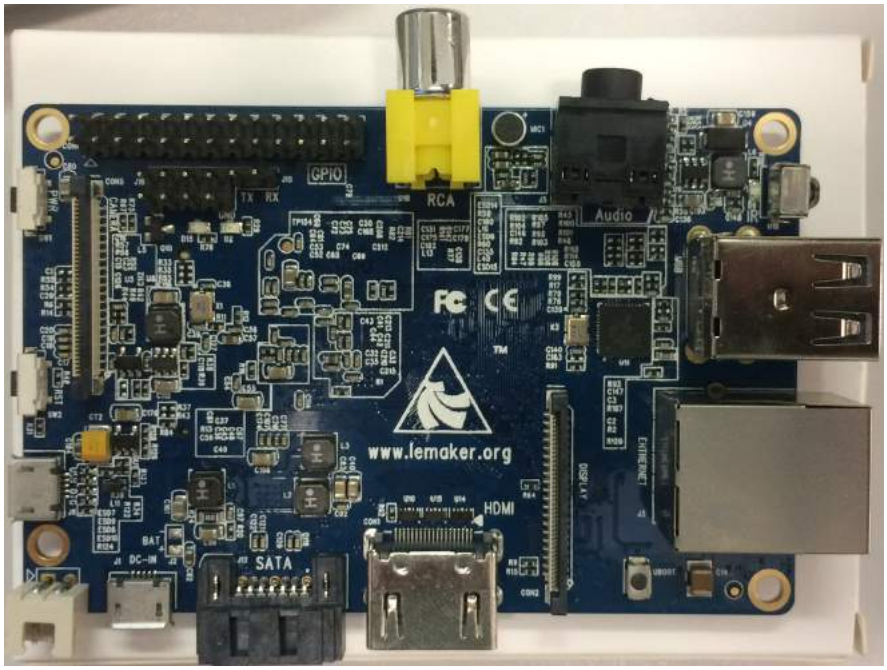


Fonte: <http://www.inovamicro.com/isoccerbot.html>. Acesso em 10/02/2017.

Um dos principais componentes, responsável pelo processamento das imagens, explicado no capitulo 2, é o Banana Pi (representado na figura 3). Ele é uma das “versões”

do *Raspberry Pi*, o qual, segundo Suryawanshi e Annadate (2016, p. 149) é um computador de placa única do tamanho de um cartão de crédito, onde é possível instalar diversos sistemas operacionais e conectar módulos externos. Um desses módulos é a câmera, conectada através da porta CSI (*Camera Serial Interface*), já presente no equipamento.

Figura 3 – Imagem de um Banana Pi



Fonte: elaborada pelo Autor

A câmera deve possuir compatibilidade com a porta CSI e um mínimo de 640x480 de resolução com 30 quadros por segundo, possibilitando uma aquisição das imagens para posterior localização dos elementos em campo do futebol de robôs.

1.1 Objetivo

Desenvolver um sistema utilizando Banana Pi que capture as imagens utilizadas pelo *software* do futebol de robôs compactando-as e transmitindo-as utilizando protocolo UDP.

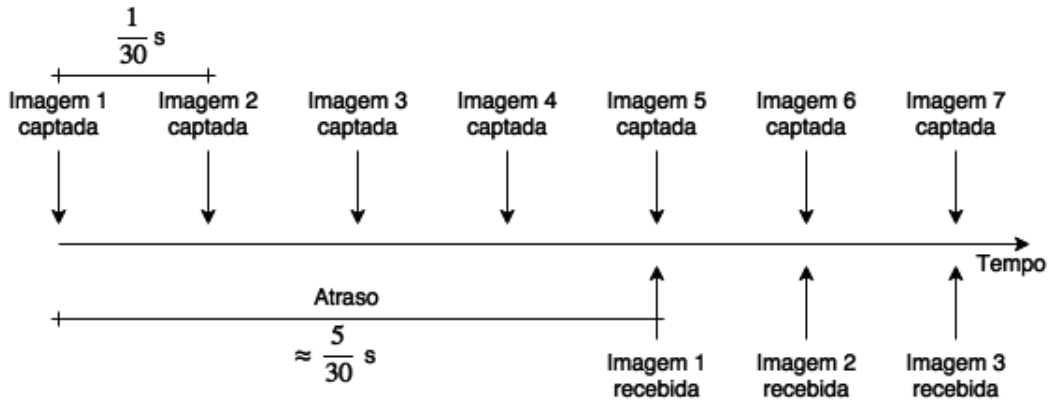
1.2 Problema

O ciclo de execução dos módulos do Futebol de Robôs se inicia nas imagens obtidas pela câmera e da qualidade destas depende o bom funcionamento do sistema. No caso do atual módulo, o problema consiste nas consequências do atraso nas informações das imagens transmitidas entre a câmera, o módulo de visão e os outros módulos, os quais controlam a equipe. O atraso gera uma diferença entre a posição real do robô e a posição

obtida das imagens pelo módulo de visão, mesmo que por alguns instantes, e assim, tal diferença é propagada para a determinação da velocidade e até mesmo para as estratégias de ataque ou defesa, gerando dados não consistentes com a realidade no instante atual.

A figura 4 representa o problema. Nela, a parte superior indica os momentos em que as imagens são captadas pela câmera, com frequência de uma imagem a cada trinta segundos e na parte inferior representa os momentos em que as informações obtidas das imagens são fornecidas pelo módulo de visão aos outros módulos, existindo um atraso, no caso, de cinco quadros. Assim, no exemplo da imagem 4, o atraso pode ser observado nas informações da imagem 1 que serão recebidas pelo módulo de estratégia 5/30 segundos após a imagem 1 ter sido capturada pelo módulo de visão, ocorrendo no mesmo instante em que a imagem 5 está sendo captada pela câmera.

Figura 4 – Explemplo de atraso do sistema atual



Fonte: elaborada pelo Autor

1.2.1 Taxa de Transferência

O padrão do USB 2.0 (*Universal Serial Bus* 2.0) proporciona uma taxa de transferência de, no máximo, 480Mbits/s (ou 60MBytes/s), de acordo com Jolfaei et al. (2009, p.1), caso o padrão seja USB 1.1, ele possui uma taxa de transferência inferior, sendo, um possível gargalo ao se fazer a transferência de vídeo em tempo real: uma imagem, cuja proporção seja de 640x480 *pixels* sem compactação, que é inferior, em resolução e tamanho, ao HD, terá em torno de 0.92 MBytes (baseando-se na equação $Tamanho = Largura \times Altura \times 3$, onde 3 corresponde ao número de bytes de cada uma das cores que compõe uma imagem: vermelho, azul e verde), e considerando a necessidade de 30 imagens por segundo, produzirá 27,6 MBytes/s, quase metade do poder de transferência nominal do USB.

Em uma comparação entre as taxas de transferências do USB (utilizando FTP) e de protocolo de rede (utilizando *iperf*) observadas no Banana Pi, realizada por HTPCGuides (2017, p.1), o protocolo de rede possui uma taxa de transferência de 58,7 *MBytes* e a USB

de 24,4 *MBytes*, ou seja o protocolo de rede é superior em 31 *MBytes* se comparado ao USB. Tal fato é confirmado por Mikronauts.com (2017, p.1), ao realizar testes com discos rígidos ligados via USB ao Banana Pi, onde há uma taxa de transferência de no máximo 30,6 *MBytes*, novamente menor que os 58,7 *MBytes* alcançados utilizando protocolos de rede da comparação anterior.

Apesar da transmissão pela USB ser suficientemente rápida para enviar 30 imagens por segundo com resolução 640x480x3, as câmeras frequentemente compactam os dados por motivos técnicos, como economia de banda da USB e economia do espaço em disco (LOGITECH, 2012), exigindo mais tempo de processamento no interior dessas câmeras, aumentando o atraso na transmissão das imagens ao computador.

Por fim, os tamanhos dos cabos é outro aspecto considerado. No caso do futebol de robôs há uma necessidade técnica de pelo menos 5 metros de cabo (distância entre o local onde é instalada a câmera ao computador que recebe as informações). O padrão USB permite no máximo 5 metros de cabo, sem a colocação de repetidores, o que faz com que se trabalhe no limite técnico necessário ao futebol de robôs (AXELSON, 2015). Já um cabo do tipo *CAT6*, utilizados em conexões *Ethernet*, permite tamanhos de cabo de até 100 metros (JONES et al., 2013).

Estas características de desempenho observadas no Banana Pi e o comprimento máximo nominal limitado pela USB, bem como prévio conhecimento do protocolo UDP, levou este projeto a utilizar *Ethernet* com protocolo UDP.

1.2.2 Compactação e Descompactação

Uma câmera USB genérica realiza uma compactação das imagens (quadro a quadro) antes do envio para o computador(LOGITECH, 2012). A compactação é parte do problema, pois insere atraso, ao não enviar a imagem logo após a captura, mas fazendo-a com que passe por um processo próprio da câmera de compactação.

Portanto, há a necessidade do desenvolvimento de algoritmos próprios de compactação, descompactação e tratamento dessas imagens para o envio, mantendo a taxa de quadros por segundo de obtenção das imagens e reduzindo o atraso dos quadros.

2 Materiais e Métodos

O sistema de visão global é composto tanto por *hardware* quanto por *software*. Para o desenvolvimento do *software* dividiu-se em três etapas, as quais são executadas em momentos diferentes, variando com a necessidade do ambiente do Futebol de Robôs.

No momento 0, denominado calibração, o Banana Pi envia ao computador responsável pelo controle do time imagens completas de 24*bits* para a calibração do sistema (seção 2.4 deste capítulo).

Por sua vez, o momento 1, denominado configuração, é quando o Banana Pi recebe a tabela de configuração de cores que serão utilizadas no próximo momento (seção 2.5 deste capítulo) e por fim, o momento 2, ou operação, onde as imagens compactadas de acordo com a configuração de cores, são enviadas ao computador responsável pelo controle do time (seção 2.6).

É denominado como cliente o computador responsável por controlar e se comunicar com o time. O Banana Pi, responsável pelo processamento e envio das imagens é referenciado como servidor. Os principais elementos de *hardware* serão comentados nas seções 2.1 e 2.2. Já a transmissão de dados será descrita na seção 2.3.

2.1 Banana Pi

Por possuir o *hardware* de código aberto, existem diversos "modelos" de *Raspberry Pi*, sendo um deles o Banana Pi (figura 3) que é usado neste projeto. O Banana Pi desenvolvido pela empresa LeMaker, possui um *gigabyte* de memória RAM e uma CPU dual-core, há também entrada para um cartão de memória, o qual armazena o sistema operacional utilizado. Justifica-se seu uso em virtude de sua saída *Ethernet* ser do tipo *gigabit*, possibilitando transferências de até um *gigabit/s* e também possuir a entrada CSI para a câmera.

O projeto contém o módulo de câmera conectado ao Banana Pi e uma estrutura plástica ao redor destes componentes para a devida proteção física.

2.1.1 Lubuntu

O sistema operacional escolhido é o Lubuntu, uma distribuição GNU/Linux baseada no Ubuntu. Devido a baixa exigência de processamento e *hardware* (CORREA et al., 2016) em sua interface gráfica, e portanto, exigindo menos processamento e ainda assim, permite a utilização de um ambiente de desenvolvimento amigável.

2.2 Câmera

A câmera a ser utilizada é a OV5640 da empresa Omni Vision, devido ao fato de ser possível conectar a câmera via porta CSI (*Camera Serial Interface*) e portanto sendo compatível com o Banana Pi. Além de possuir a resolução necessária (640x480 *pixels*) e taxa de transferência entre 30 quadros por segundo e 90 quadros por segundo. Portando, se necessária uma atualização do atual sistema de 30 quadros por segundo, será possível o re-aproveitamento deste módulo.

2.3 Protocolo UDP

O protocolo *User Datagram Protocol* (UDP) foi escolhido para a comunicação entre o cliente e servidor, pois por ser um protocolo não voltado a conexão, ele economiza banda de transmissão, não verificando possíveis erros decorridos nas trocas de dados. Sendo assim, este protocolo não proporciona garantia de que a comunicação foi realizada com sucesso, economizando assim o tempo exigido para que se garanta o recebimento das mensagens (KUROSE; ROSS, 2010).

Embora pareça uma desvantagem a perda de pacotes, há uma preocupação maior com que as imagens sejam entregues ao cliente o quanto antes, evitando assim o atraso dos dados, em detrimento à possível perda de imagens. Há também a vantagem do *overhead* (em torno de 8 *bytes*) ser menor no caso do UDP, se comparado ao USB ou ao protocolo TCP (em torno de 20 *bytes*, de acordo com Kurose e Ross (2010)), necessitando de menos dados acrescentados no pacote e deixando-o com menor quantidade de *bytes* de controle. O servidor e o cliente foram desenvolvidos em C++ em ambiente Linux.

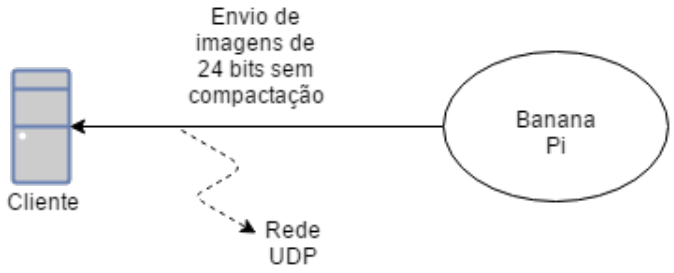
2.4 Calibração – Momento 0

É o primeiro passo do sistema. Nele, o servidor (Banana Pi) envia continuamente ao cliente imagens 640x480 com os 15 *bits* de cor, estes 15 *bits* são divididos em três canais: vermelho, verde e azul (ou *RGB*), que determinam a cor de cada pixel. Essas imagens iniciais são necessárias para a definição das cores por um operador humano: é ele que indicará quais as combinações vermelho, verde e azul formam as cores dos times e da bola, bem como ele confirmará que suas escolhas produzem uma imagem onde é possível determinar com clareza os elementos apresentados. Não é possível uma pré-determinação dessas combinações devido as variadas formações de sombras e intensidade da luz.

Com isso, o operador irá colocar um *flag* – um valor do tipo inteiro, o qual representa uma cor – em um vetor, cujo o índice é a concatenação dos valores de vermelho, verde e azul da respectiva cor. O *flag* irá determinar o elemento identificado por aquela cor. Não é de responsabilidade deste trabalho determinar o vetor de cores, fornecer

ferramentas para o operador indicar as cores ou desenvolver um sistema inteligente para a detecção das cores. Na imagem 5 há uma representação deste momento.

Figura 5 – Representação do Momento 0

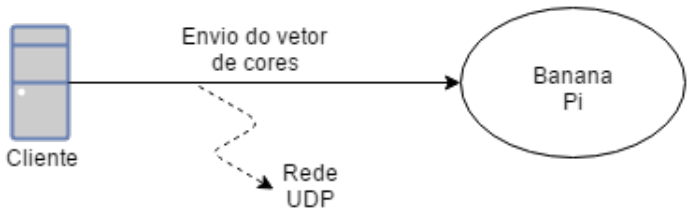


Fonte: Elaborado pelo Autor

2.5 Envio do Vetor – Momento 1

É o momento do servidor receber o vetor de conversão de cores de 32 *kbytes* – tamanho suficiente para armazenamento de todas as possibilidades de cores – criado no momento 0 (seção 2.4) utilizando o protocolo de transmissão UDP definido na seção 2.3. Essa etapa deve ocorrer pelo menos uma vez. Na imagem 6 há uma representação deste momento.

Figura 6 – Representação do Momento 1



Fonte: Elaborado pelo Autor

O vetor armazena em cada posição números de 1 *byte* (o *flag*), ou seja, de 0 a 254, sendo o 255 reservado para indicar o fim de imagem). Os índices do vetor são equivalentes a todas as cores possíveis no padrão RGB de 5 *bits*, uma vez que cada cor deve ter seu valor correspondente no vetor para que, durante o jogo, o programa ao calcular o valor do *pixel* coloque-o no índice do vetor e dele se obtenha o valor de conversão daquele *pixel* (*flag*).

2.6 Jogo – Momento 2

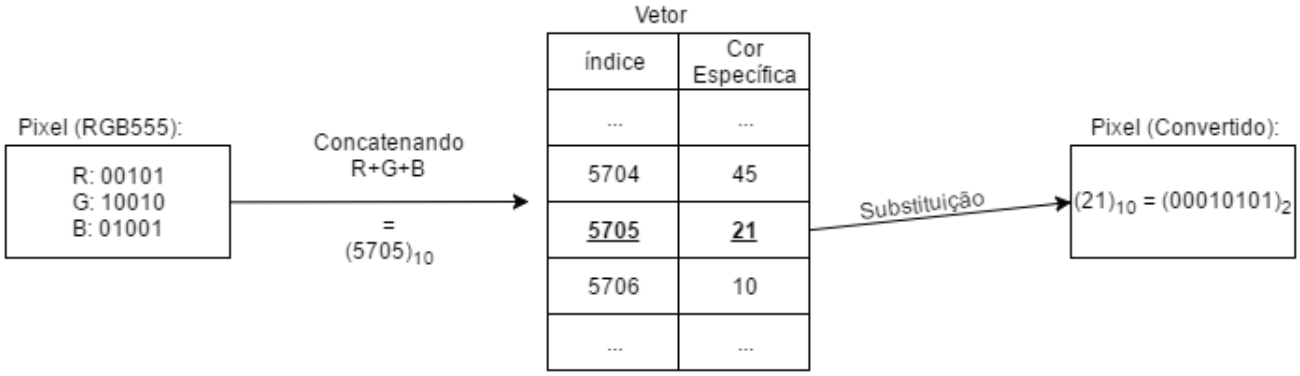
O Banana Pi recebe de seu módulo de câmera uma lista de 614400 *bytes* – a imagem – sendo cada elemento dessa lista um canal de cor no padrão YUYV, descrito

em 3.2.1. Cada *pixel* é convertido para RGB, possuindo três canais (que são a quantidade de cada uma das cores primárias necessárias para compor as outras cores) de 8 *bits* cada, perfazendo 24 *bits*. São, então, excluídos os 3 *bits* menos significativos de cada canal o que resulta em um valor combinado com 5 *bits* de R (vermelho), 5 *bits* de G (verde) e 5 *bits* de B (azul).

Os 15 *bits* dos canais de cores são concatenados (na sequência: $R + G + B$) e formam o índice do vetor recebido no momento 1 (seção 2.5), este índice é utilizado para recuperar o valor do *flag* correspondente à cor do *pixel* no vetor, resultando na cor específica identificada, a qual possui um *byte*. Cada *pixel* agora é representado por um *flag* que possui 8 *bits*, ao invés de 15 *bits*.

A imagem 7 exemplifica essa conversão. O *pixel*, à esquerda, é representado pelos valores de cada canal em binário. Após a concatenação dos três canais, (na ordem R, G e B) o valor 5705, em decimal, é determinado e usado como índice no vetor, determinando o *flag* de cor específica como sendo 21. Então, o valor 21 passará a representar o *pixel* daquela posição.

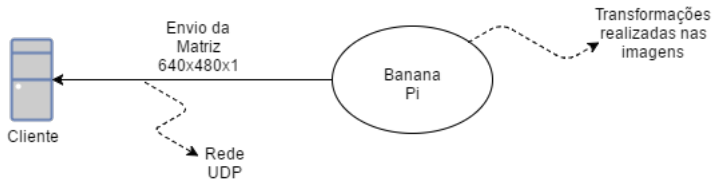
Figura 7 – Exemplo de conversão



Fonte: Elaborado pelo Autor

Os dados gerados passam por um processo de compactação e são enviados para o cliente. Este momento ocorre constantemente, com uma taxa desejada de 30 quadros por segundo. Após o envio das imagens, o cliente é responsável por determinar o posicionamento dos robôs e bola. Na imagem 8 há uma representação deste momento.

Figura 8 – Representação do Momento 2



Fonte: Elaborado pelo Autor

2.6.1 Compactação

Após a diminuição da quantidade de *bits* que formam um *pixel*, ocorrido no momento 2 (2.6) e encontrada a cor específica no vetor de conversão de cores (após utilizar o valor do *pixel* como índice no vetor e receber dele o valor do *flag* da cor específica), o *pixel* passa a ser apenas um *byte* correspondente à cor específica identificada como correspondente.

A compactação é realizada neste momento, seguindo o padrão de compactação RLE (*Run-lenght encoding*), a qual realiza a substituição de cadeias de caracteres por uma combinação da identificação desse caractere com a quantidade de vezes que ele repete na cadeia (SADLER; MARTONOSI, 2006), empregada no padrão BMP do *Windows*[1]. O método de compactação é descrito em 3.3.

[1] Informações sobre o padrão BMP podem ser encontradas em <https://msdn.microsoft.com/pt-br/library/windows/desktop/dd183383(v=vs.85).aspx>

3 O Sistema

A fim de facilitar o entendimento do código, o mesmo é dividido em duas partes, denominadas transmissão e captura, os quais são associados em apenas um único código. O código denominado "transmissão" é responsável pela envio das imagens entre o cliente e servidor utilizando um cabo de rede e é detalhado na seção 3.1, já o código denominado "captura" é responsável pala captura das imagens recebidas pelo *driver* da câmera e é detalhado na seção 3.2. O algoritmo desenvolvido e os detalhes da compactação das imagens estão descritos em 3.3.

3.1 Transmissão

Para a transmissão utiliza-se o protocolo UDP. Os códigos do cliente e servidor são semelhantes, existindo alterações na forma com que o *socket*, cuja função é criar uma "ponte" entre o programa e o sistema operacional (KUROSE; ROSS, 2010), é criado. Ao iniciar o servidor, o valor de porta pré-definido é associado ao *socket*, indicando onde o mesmo deve esperar a comunicação. No código do servidor, três rotinas são as principais: *receive*, *send* e *sendcompact*.

A função *receive* é responsável pelo recebimento de informações enviadas pelo cliente, tanto para a escolha das opções no servidor (as quais são as mesmas oferecidas ao cliente: calibrar a imagem, envio do vetor e início do jogo), quanto para que aguarde receber o Vetor de Conversão de Cores do cliente, neste último caso há uma rotina para que este vetor seja salvo em um arquivo binário no Banana Pi para ser posteriormente carregado anteriormente ao jogo.

A função *send*, por sua vez, tem por objetivo enviar as imagens durante a etapa de calibração, descrita em 2.4, para isso a rotina deve armazenar uma quantidade de *pixels* RGB555, para então enviar este conjunto de dados ao cliente até que se complete os 307200 *pixels* que constituem uma imagem completa (totalizando um total de 600 kB de informação enviada – 307200 *pixels* multiplicados pelo tamanho de cada um, ou seja, 2 *bytes*). Para a melhor utilização do protocolo UDP a imagem é divida em pacotes de 30 *kbytes* para então cada um destes pacotes serem enviados.

Por fim, a função *sendcompact* é responsável por enviar a imagem de forma compactada, sendo uma alteração da função *send*, uma vez que, neste caso, a quantidade de dados não é igual ao tamanho da imagem original, devido a compactação. Para indicar fim da transmissão da imagem compactada é enviado o valor 255. Entretanto, de forma semelhante à função *send*, ela deve armazenar os dados em uma quantidade

pré-determinada, sendo os dados organizados de forma a atender a compactação utilizada e descrita em 3.3.

Para o cliente, o código de transmissão funciona de forma semelhante, existindo duas rotinas de recebimento de dados (com e sem compactação) e a rotina de envio. Para as rotinas de recebimento sem compactação há um laço aguardando o recebimento total da imagem (espera-se receber todos os 307200 *pixels*, já no caso de existir a compactação, aguarda-se receber o valor 255, ou o *byte* de fim de imagem).

As rotinas de envio de dados são responsáveis por enviar a escolha do usuário, como iniciar o jogo, iniciar as funções de envio do vetor ou iniciar a calibração do sistema, bem como para enviar o Vetor de Conversão de Cores ao servidor. Para o envio desse vetor, deve-se criar um arquivo binário, de nome pré-determinado, e com 32767 valores – o total de combinações do padrão RGB de 15 *bits* – entre 0 e 254 (255 é reservado para o fim de imagem). Para iniciar o cliente, o usuário deverá indicar o *IP* do servidor.

Para o compartilhamento de rede entre os dispositivos, há a necessidade de alteração do cliente para ele compartilhar a rede via cabo com o Banana Pi. Para isso altera-se a opção nas configurações da rede cabeada do *Ubuntu* para "Compartilhado com outros computadores".

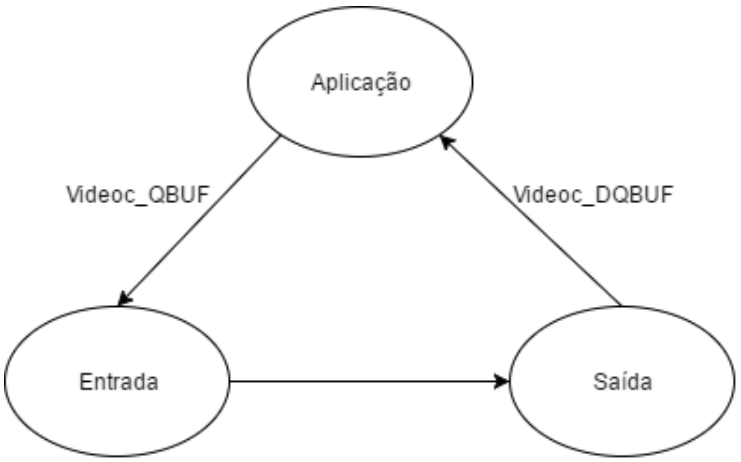
3.2 Captura

Para a captura, é utilizada uma API (*Application Programming Interface*), denominada *Video For Linux 2* (ou *V4L2*) a qual fornece rotinas para a captura das imagens da câmera. A opção por esta API deve-se ao fato dela trabalhar em nível próximo ao *hardware* e por possuir compatibilidade com a câmera utilizada, embora que com algumas restrições (uma delas a ser descrita em 3.2.1).

A câmera do sistema permite o acesso da API através da opção de *Memory Mapped*, a qual cria mapas de memória a partir do endereço anteriormente alocado pelo programa (BOVET; CESATI, 2005). Sendo assim, a API cria uma quantidade pré-determinada de *buffers* do tipo *memory mapped* e os coloca em uma fila de entrada, onde aguarda o recebimento da imagem completa para então seguir para a fila de saída, aguardando ser chamado pelo programa desenvolvido. A figura 9 representa as filas e as transições pelos quais o *buffer* passa.

Os atributos de *VIDIOC_DQBUF* e *VIDIOC_QBUF* são utilizados para tirar o *buffer* pronto (com uma imagem alocada pelo *driver*) da fila de saída e colocar o *buffer* na fila de entrada, respectivamente (SCHIMEK et al., 1999). Após a captura, a imagem é enviada para uma rotina de conversão, passando do formato YUYV (detalhado em 3.2.1) para RGB de 5 *bits*. Durante o jogo, a imagem, além de convertida para o padrão desejado,

Figura 9 – Representação dos Buffers



Fonte: Adaptada pelo Autor do original em <http://stackoverflow.com/questions/10634537/v4l2-difference-between-enque-deque-and-queueing-of-the-buffer>. Acesso em 25/01/2017.

é compactada (descritos na subseção 2.6.1 e seção 3.3) e então é enviada, conforme descrito na seção 3.1.

3.2.1 YUYV

A câmera não possui no *driver* a opção para envio imediato em RGB555, embora ela possua suporte conforme informado pelo manual (OMNIVISION TECHNOLOGIES, INC, 2011), sendo um problema de compatibilidade entre o *V4L2* e a câmera. Como alternativa, usa-se o formato YUYV (em acordo com o especificado pelo manual do *V4L2* (SCHIMEK et al., 1999)), devido a ordem em que as informações são dispostas na estrutura de dados, o que leva a dois *pixels* de RGB555 serem obtidos a partir de quatro *bytes* seguidos dos canais de cor do YUYV.

Cada pixel no formato YUYV passa por uma função no servidor, a qual converte para RGB de 15 *bits*. Esta função, descrita no algoritmo 1, calcula a partir dos valores dos canais Y, U e V, os valores do canal RGB e, em seguida, concatena-os utilizando o operador binário "ou" em uma única variável de tipo inteiro. A conversão demanda tempo extra de processamento.

Algoritmo 1 – Algoritmo YUYV para RGB 555

Entrada: Valores inteiros de Y, U e V.

1. ***R*** *é igual a* $Y + (1.4065 * (V - 128))$
2. ***G*** *é igual a* $Y - (0.3455 * (U - 128)) - (0.7169 * (V - 128))$
3. ***B*** *é igual a* $Y + (1.7790 * (U - 128))$
4. ***Desloca-se*** *B em 3* bits *para direita.*

5. ***Altera*** *os 3* bits *menos significativo de G para 0 e* ***desloca-o*** *em 2* bits *para esquerda.*
6. ***Altera*** *os 3* bits *menos significativo de R para 0 e* ***desloca-o*** *em 7* bits *para esquerda.*
7. *Utilizando o operador binário "**ou**" concatena-se R, G e B em uma variável.*

3.3 Compactação

O recebimento e utilização do Vetor de Conversão de Cores faz com que a probabilidade de repetição de dados seja maior, pois restringe para 255 opções de cores, e no domínio do Futebol, onde os robôs são identificados por cores em suas etiquetas e a bola por sua cor laranja, normalmente utiliza-se menos de 10 cores. Conforme descrito em 2.6.1, a compactação segue o padrão BMP, que consiste em enviar duas informações, a primeira delas sendo a quantidade de valores repetidos e em seguida o valor que se repete. Caso o dado não se repita, o padrão envia primeiramente o valor 0, seguido da quantidade de dados que não são repetidos e então os valores. A tabela 1 representa à esquerda os dados a serem compactados e à direita como estão após a compactação.

Tabela 1 – Exemplo de Compactação BMP

	Descompactado	Compactado
	04 04 04 06 06 06 06 06 45 56 67	03 04 05 06 00 03 45 56 67
Total	12 bytes	9 bytes

Fonte: Elaborado pelo autor.

A porcentagem de redução após a compactação, no caso das imagens, varia de acordo com o Vetor de Conversão de Cores e da distribuição das cores na imagem. Se o vetor possuir muitas cores e seus tons com um mesmo valor de conversão e essas cores e tons estejam em *pixels* adjacentes da imagem, então as repetições serão mais frequentes e por consequência a redução será maior. Para indicar ao cliente o final de imagem é reservado o valor 255.

O algoritmo de compactação, sendo descrito no algoritmo 2, funciona de forma a comparar o dado *atual* (o último) de cor especifica recebido com o valor *anterior*. Caso sejam iguais e já exista a *estrutura* para colocar as repetições (ou seja, a quantidade de valores repetidos, seguida do valor), apenas aumenta-se um na quantidade de repetidos. Caso sejam diferentes, e já exista a estrutura, basta atualizar a quantidade de dados não repetidos e colocá-lo no final.

Se o fato dos dados serem diferentes tenha ocorrido pela *primeira vez*, não se faz nada, para assim verificar a relação deste valor com o próximo dado a ser recebido. Sendo assim, na situação onde o dado recebido é diferente do anteriormente "ignorado", cria-se a estrutura para dados diferentes, caso contrário, a estrutura de dados iguais é criada. O

algoritmo 2 não realiza as verificações que são realizadas para o momento de iniciar o sistema.

Algoritmo 2 – Algoritmo para Compactação

Entrada: Valor a ser compactado: Atual

1. ***Se*** *Anterior é igual Atual,* ***então:***
2. ***Se*** *EstruturaIgual é falso,* ***então:***
3. ***BufferCompactacao***[Última$Posicao - 1$] *recebe 2.*
4. ***BufferCompactacao***[Última$Posicao$] *recebe Anterior.*
5. ***EstruturaIgual*** *recebe verdade.*
6. ***EstruturaDiferente*** *recebe falso.*
7.
8. ***Se não****, incrementa-se BufferCompacatacao*[Última$Posicao - 1$]
9.
10. ***Se não:***
11. ***Se*** *PrimeiraVez é falso e EstruturaDiferente é falso,* ***então***
12. ***PrimeiraVez*** *recebe verdade.*
13.
14. ***Se não, se*** *PrimeiraVez é falso e EstruturaDiferente é verdade,* ***então***
15. ***BufferCompactacao***[Última$Posicao - Quantidade - 1$] *é incrementado.*
16. ***BufferCompactacao***[Última$Posicao$] *recebe Anterior.*
17. ***Quantidade*** *é incrementado.*
18.
19. ***Se não, se*** *PrimeiraVez é verdade,* ***então***
20. ***BufferCompactacao***[Última$Posicao - 2$] *recebe 0.*
21. ***BufferCompactacao***[Última$Posicao - 1$] *recebe 1.*
22. ***BufferCompactacao***[Última$Posicao$] *recebe Anterior.*
23. ***Quantidade*** *recebe 1.*
24. ***EstruturaDiferente*** *recebe verdade.*
25. ***EstruturaIgual*** *recebe falso.*
26.

4 Conclusão

O objetivo, conforme exposto na seção 1.1, é o desenvolvimento de parte de um sistema de visão global para o Futebol de Robôs da UNESP de Bauru, sendo tal objetivo obtido.

Com base nas especificações apresentadas no capítulo 2, o sistema encontra-se funcional, tendo se mostrado promissor para substituir a câmera e parte do antigo sistema de visão global do Futebol de Robôs da UNESP Bauru. O sistema desenvolvido é escalável, permitindo atualizações, como a taxa de quadros por segundo ou a melhora na qualidade de imagem.

Contudo, as atualizações sugeridas no capítulo 5 são necessárias para tornar viável a integral substituição do antigo sistema. Além de testes mais aprofundados para comprovar a diminuição dos erros apontados na seção 1.2.

5 Trabalhos Futuros

Como trabalhos futuros sugere-se a atualização do *driver* do dispositivo para que ele passe a capturar imagens diretamente em RGB555, removendo as necessidades de conversão do YUYV (descrito em 3.2.1). O trabalho no *driver* deverá também atualizar a taxa de quadros, que atualmente é restrita em 25 *fps*, para pelo menos 30 *fps* (de acordo com o manual do fabricante, há a possibilidade de até 90 *fps* com qualidade VGA de 640x480 (OMNIVISION TECHNOLOGIES, INC, 2011)).

Há também a necessidade de um melhor controle em caso de perda de partes das imagens, durante a transmissão. A necessidade existe devido ao fato do protocolo de transmissão UDP não ter garantia na entrega de dados, podendo acontecer casos em que partes da imagem não sejam recebidas pelo cliente sem que o cliente perceba.

Referências

AXELSON, J. *USB complete*. 5. ed. Lakeview Research, 2015. Disponível em: <https://books.google.com.br/books?id=pkefBgAAQBAJ>. Acesso em: 01/11/2016.

BOVET, D.; CESATI, M. *Understanding the Linux Kernel: From I/O Ports to Process Management*. O'Reilly Media, 2005. ISBN 9780596554910. Disponível em: <https://books.google.com.br/books?id=h0lltXyJ8aIC>. Acesso em: 16/01/2017.

CORREA, A.; OLIVEIRA, P.; ASSIS, G.; FICHEMAN, I.; NASCIMENTO, M.; GOBARA, S.; ARAúJO, E.; LOPES, R. *Projeto UCA-Assistiva: mapeamento e avaliação de ferramentas assistivas nos laptops educacionais do PROUCA*. 2016. Disponível em: <http://www.seer.ufrgs.br/index.php/renote/article/view/44706/28406>. Acesso em: 24/11/2016.

COSTA, A. H. R.; PEGORARO, R. Construindo robôs autônomos para partidas de futebol: o time guaraná. *SCBA Controle & Automação*, v. 11, n. 93, p. 141–149, dec 2000. Disponível em: <http://www.sba.org.br/revista/vol11/v11a259.pdf>. Acesso em: 25/10/2016.

HTPCGUIDES. *RASPBERRY PI VS PI 2 VS BANANA PI PRO BENCHMARKS*. 2017. Disponível em: <http://www.htpcguides.com/raspberry-pi-vs-pi-2-vs-banana-pi-pro-benchmarks/>. Acesso em: 13/10/2017.

JOLFAEI, F. A.; MOHAMMADIZADEH, N.; SADRI, M. S.; FANISANI, F. High speed usb 2.0 interface for fpga based embedded systems. *2009 Fourth International Conference on Embedded and Multimedia Computing*, 2009. Disponível em: <http://googoolia.com/downloads/papers/emcom2009.pdf>. Acesso em: 25/10/2016.

JONES, G.; LAYER, D.; OSENKOWSKY, T.; WILLIAMS, E. *National Association of Broadcasters Engineering Handbook: NAB Engineering Handbook*. Taylor & Francis, 2013. ISBN 9781136034107. Disponível em: <https://books.google.com.br/books?id=K9N1TVhf82YC>. Acesso em: 13/10/2017.

KITANO, H. *RoboCup-97*. Springer, 1998. Disponível em: <https://books.google.com.br/books?id=7Tp4s9i3N_UC>. Acesso em: 25/10/2016.

KOOIJ, N. S. *The development of a vision system for robotic soccer*. Dissertação (Mestrado) — University of Twente, 2003. Disponível em: <http://hmi.ewi.utwente.nl/robotsoccernieuw/documents/thesis_niek_kooij.pdf>. Acesso em: 25/10/2016.

KUROSE, J. F.; ROSS, K. W. *Computer networking*. Pearson, 2010. Disponível em: <http://www.nylxs.com/docs/cmpnet.pdf>. Acesso em: 01/11/2016.

LOGITECH. The h.264 advanced video coding (avc) standard. In: . [s.n.], 2012. Disponível em: <http://www.logitech.com/assets/45120/logitechh.pdf>. Acesso em: 23/01/2017.

MIKRONAUTS.COM. *Banana Pi & Pro SATA and USB Hard Drive Tests*. 2017. Disponível em: <http://www.mikronauts.com/banana-pi/banana-pi-pro-sata-and-usb-hard-drive-tests/>. Acesso em: 13/10/2017.

OMNIVISION TECHNOLOGIES, INC. *OV5640 Datasheet PRODUCT SPECIFICATION*. [S.l.], 2011. Disponível em: <https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/OV5640_datasheet.pdf>. Acesso em: 23/01/2017.

SADLER, C. M.; MARTONOSI, M. Data compression algorithms for energy-constrained devices in delay tolerant networks. In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM, 2006. (SenSys '06), p. 265–278. ISBN 1-59593-343-3. Disponível em: <http://mrmgroup.cs.princeton.edu/papers/csadler-sensys2006.pdf>. Acesso em: 25/11/2016.

SCHIMEK, M. H.; DIRKS, B.; VERKUIL, H.; RUBLI, M. Video for linux two api specification. *History*, v. 6, p. 11, 1999.

SURYAWANSHI, S.; ANNADATE, S. Raspberry pi based interactive smart home automation system through e-mail using sensors. *International Journal of Advanced Research in Computer and Communication Engineering*, v. 5, n. 2, p. 148–151, 2016. Disponível em: <http://www.ijarcce.com/upload/2016/february-16/IJARCCE%2031.pdf>. Acesso em: 31/10/2016.