AMANDA VIEIRA FERNANDES
RENAN RICARDO MARCHETTO

Módulo de visão integrado ao kit da LEGO MINDSTORMS EV3

São Paulo

2015

AMANDA VIEIRA FERNANDES
RENAN RICARDO MARCHETTO

Módulo de visão integrado ao kit da LEGO MINDSTORMS EV3

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a conclusão do curso de graduação em Engenharia Mecatrônica, junto ao Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos (PMR)

Orientador: Prof. Dr. Thiago de Castro Martins

São Paulo

2015

Este relatório é apresentado como requisito parcial para obtenção do grau de Engenharia Mecatrônica na Escola Politécnica da Universidade de São Paulo. É o produto do nosso próprio trabalho, exceto onde indicado no texto. O relatório pode ser livremente copiado e distribuído desde que a fonte seja citada.

FICHA CATALOGRÁFICA

Vieira Fernandes, Amanda; Marchetto, Renan Ricardo.
Módulo de visão integrado ao kit da LEGO MINDSTORMS EV3 / Vieira Fernandes, Amanda; Marchetto, Renan Ricardo.. – São Paulo, 2015- 93 p.

Monografia – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos (PMR), 2015.

1. LEGO MINDSTORMS EV3. 2. Intel Edison. 3. Visão Computacional. I. Prof. Dr. Thiago de Castro Martins. II. Universidade de São Paulo. III. Escola Politécnica. IV. Módulo de visão integrado ao kit da LEGO MINDSTORMS EV3

AGRADECIMENTOS

Considerando essa monografia como o resultado de uma caminhada que não começou na Escola Politécnica, agradecer pode não ser uma tarefa nem fácil, nem justa. Para não correrrmos o risco da injustiça, agradecemos de antemão a todos que, de alguma forma, passaram pelas nossas vidas e contribuíram para a construção de quem somos hoje.

Gostaríamos de agradecer, em especial, algumas pessoas pela contribuição direta na construção deste trabalho:

Ao nosso orientador Prof. Dr. Thiago de Castro Martins, pelo estímulo acadêmico, pela orientação constante, pelas contribuições teóricas e, principalmente, por nos mostrar na prática que podemos nos superar a cada dia.

Ao Prof. Nilson Noris Francischetti, pelo suporte eletrônico oferecido nos momentos de maior necessidade.

Ao Laboratório de Veículos Não Tripulados, pelo empréstimo do ítem mais valioso desse projeto: o analisador lógico.

Ao Peter Thesbjerg e Per Christoffersen, Diretor Sênior de Marketing e Engenheiro Sênior de Pesquisa e Desenvolvimento Eletrônicos do Grupo LEGO de Billund, pela disponibilidade e atenção durante a nossa visita, pelo interesse acadêmico no nosso projeto e pela inspiração da criação de uma ferramenta tão interessante quanto o LEGO MINDSTORMS.

Aos professores e amigos da Escola Politécnica da Universidade de São Paulo, pelo incentivo e inspiração que nos ofereceram durante este e tantos outros trabalhos ao longo desses 6 anos.

Aos amigos que, mesmo não estando presentes no nosso dia-a-dia, nos deram força e foco para continuar avançando.

Aos nossos irmãos e pais, pelo carinho e apoio durante os momentos mais difíceis e pela comemoração a cada pequena vitória alcançada.

RESUMO

Em 1998 o Grupo LEGO, em parceria com o MIT, criou o primeiro kit de robótica para crianças: o RCX. Desde então, com o desenvolvimento da tecnologia, o produto vem sendo amplamente utilizado em cursos de graduação, especialmente de mecatrônica, computação e eletrônica. Além do bloco programável, o kit contém diversos tipos de sensores: toque, luz, ultrassom, infravermelho, entre outros. Eles fazem do produto uma excelente ferramenta robótica, permitindo a construção de projetos bastante complexos. Entretanto, até então, o kit não possui oficialmente um sensor de visão, o que permitiria aos usuários a criação de robôs ainda mais eficazes e complexos. O projeto tem como objetivo final o desenvolvimento de um módulo de visão integrado ao kit da terceira geração da LEGO MINDSTORMS. Para tal finalidade, ambos hardware e software serão desenvolvidos de modo que uma câmera possa captar o ambiente através de fotos e/ou vídeos, estas imagens sejam processadas por um microcomputador embarcado ao módulo que enviará ao bloco inteligente EV3 da LEGO um conjunto de informações pós-processadas que serão, então, integradas à programação de blocos própria ao produto. Envolto em um invólucro correspondente aos padrões de encaixe da LEGO, o módulo seria uma ferramenta reprogramável de processamento de imagens em tempo real completamente integrável ao kit, permitindo tanto uma utilização simplificada das funções pré-programadas na linguagem de blocos, quanto uma programação personalizada de diferentes funções segundo as necessidades do usuário.

Palavras-chave: LEGO MINDSTORMS EV3. Intel Edison. Visão Computacional.

ABSTRACT

In 1998 the LEGO Group, in partnership with the MIT, created the first robotic kit for children: the RCX. Ever since, with the development of this tecnology, the product has been widely used in undergraduate courses, especially mechatronics, computer science and electronics. In addition to the programmable brick, the kit contains a variety of sensors: touch, light, ultrasound, infrared and others. They make the product an excellent robotics tool, allowing the construction of very complex projects. However, the kit does not offitially have a vision sensor which would allow users to create even more effective and complex robots. The project's ultimate goal is the development of a vision module integrated with the LEGO MINDSTORMS third generation kit. For this purpose, both hardware and software will be developed so that the camera can capture its environment through photos and/or videos, these images being processed by a microcomputer embedded to the module that will send to the EV3 a set of post-processed data that will be then integrated into the LEGO's icon-based programming software. The module will be wrapped in a LEGO matching casing and it would be then a reprogrammable tool for real-time image processing completely integrated to the kit, allowing both simplified use of the pre-programmed functions and custom programming different functions according to users needs.

Key-words: LEGO MINDSTORMS EV3. Intel Edison. Computational vision.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EPUSP	Escola Politécnica da Universidade de São Paulo
MIT	Instituto de Tecnologia de Massachusetts
RCX	Robotic Command Explorer
NXT	Segunda geração do LEGO MINDSTORMS
EV3	Terceira geração do LEGO MINDSTORMS
MCL	Localização de Monte Carlo
MSRS	Microsoft Robotics Studio
HOG	Histograma de Gradientes Orientados
AMDF	Função da Média de Diferenças de Amplitudes
I2C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface
I2S	Integrated Inter-IC Sound
PWM	Pulse-Width Modulation
OTG	On The Go
UVC	USB Video Class
GPS	Global Positioning System
PCB	Placa de Circuito Impresso
SI	Sistema Internacional de Medidas

1 INTRODUÇÃO

1.1 Tema

O projeto visa o prospecto, o desenvolvimento e a construção de um módulo de visão reprogramável integrado ao kit LEGO MINDSTORMS EV3[1].

1.2 Justificativa da Escolha do Tema

Um dos pilares da atuação profissional em mecatrônica é a programação. O processo de aprendizado dos conceitos iniciais da programação é complexo e marcado pela presença de inúmeras dificuldades. Alguns dos pontos mais destacados são: a baixa capacidade de resolução de problemas aliada a equívocos na formulação de modelos mentais adequados; a falta de motivação para executar tarefas; a dificuldade para tratar abstração, ferramentas e linguagens não adaptadas pedagogicamente; entre outros.

A LEGO apresenta um kit de desenvolvimento otimizado que segue os mais recentes desenvolvimentos de software intuitivo de fácil utilização para iniciantes. É neste contexto que o produto começou a ser utilizado em salas de aula de cursos de graduação, especialmente nos domínios da computação, eletrônica, mecatrônica e robótica. A Escola Politécnica da Universidade de São Paulo (EPUSP) não foi exceção: em 2009, a escola começou a oferecer cursos extra-curriculares para os calouros interessados em aplicar os conceitos de programação aprendidos no curso regular utilizando o kit(PET. . . , 2015).

Pessoalmente para os alunos envolvidos neste projeto, que participaram do curso como calouros para, em seguida, se tornarem monitores do mesmo, a motivação para o desenvolvimento deste projeto provem da proximidade dos mesmos com o kit. Apesar das ferramentas existentes no produto serem, por si só, muito potentes, a adição de um módulo de visão ao produto seria um avanço importante para as criações robóticas.

1.3 Estrutura do Trabalho

Este relatório será estruturado da seguinte forma: primeiramente, será apresentado o estado da arte do domínio de visão computacional em robótica e, mais especificamente, das tecnologias integradas a todas as gerações do kit da LEGO MINDSTORMS.

[1] Os nomes LEGO®, MINDSTORMS®, NXT e EV3 são marcas registradas do *LEGO Group*. Seu uso neste texto não implica em aval por parte do *LEGO Group* ao seu conteúdo.

Em seguida, os requisitos funcionais e não funcionais do projeto serão detalhadamente especificados. A posteriori, a arquitetura será definida e os seus componentes principais serão apresentados.

Finalmente, a metodologia utilizada será descrita de forma a exemplificar os processos necessários para a realização do projeto para, então, os resultados serem apresentados seguidos por uma conclusão sobre os mesmos com sugestões para trabalhos futuros.

2 ESTADO DA ARTE

2.1 LEGO MINDSTORMS

O Grupo LEGO é uma companhia privada, cuja matriz encontra-se em Billund, na Dinamarca. Fundado em 1932 por Ole Kirk Kristiansen, o grupo é um dos líderes mundiais em fabricação de jogos infantis. O produto mais importante do grupo é o bloco LEGO, e o princípio de encaixe com tubos o torna único, oferecendo infinitas possibilidades de construção (MORTENSEN, 2012).

Há 35 anos, o grupo criou uma divisão especial voltada à educação: a LEGO Education. A companhia trabalha em conjunto com professores e especialistas em educação para proporcionar soluções e recursos que serão utilizados dentro da sala de aula para fazer o aprendizado mais divertido. A mesma inspira interesse em diversas áreas, tais quais Ciência, Tecnologia, Engenharia, Ciência da Computação, Matemática e Ciências Humanas (EDUCATION, 2015).

Em 1988, a partir de uma colaboração entre o Grupo LEGO e o Instituto de Tecnologia de Massachusetts (MIT), desenvolveu-se um "bloco inteligente", o qual seria capaz de trazer as criações em blocos LEGO à vida via programação computacional. O kit, conhecido como LEGO MINDSTORMS , foi oficialmente introduzido no mercado em 1998 e consiste de um conjunto de peças da linha tradicional acrescido de atuadores, sensores e de um processador programável: o módulo RCX (Robotic Command Explorer). Até os dias atuais, existem três gerações de kits: o RCX, o NXT e o EV3 (LEGO, 2013c).

Todos os módulos programáveis contêm: portas de saída, identificadas por letras, que servem para conectar os atuadores do robô; portas de entrada, identificadas por números, que servem para conectar os diversos sensores; tela LCD com informações variadas como valores de leitura dos sensores e programas selecionados; e botões do painel utilizados para seleção do programa, ligar/desligar o bloco, executar a programação, entre outros.

Figura 1: As três gerações dos blocos inteligentes do kit. Reproduzido de (VALK, 2013)

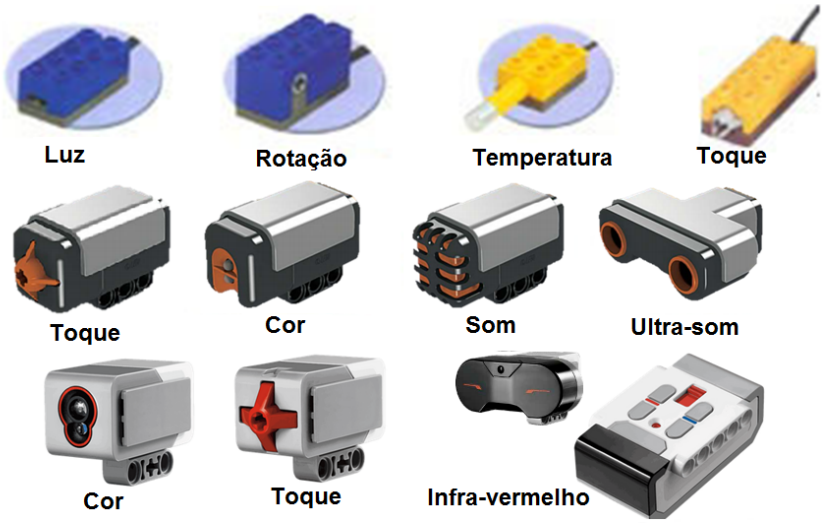


Figura 2: Evolução dos sensores da primeira, segunda e terceira gerações, respectivamente[2].

A cada geração, o kit passou por diversas alterações. No que diz respeito ao bloco programável, enquanto o RCX poderia conter no máximo 5 programas e comunicar-se via torre infravermelha (CAPRANI, 2006), o NXT permitia o armazenamento de diversos arquivos em sua memória além de comunicação Bluetooth e USB (LEGO, 2006). Já na terceira geração, o EV3 possui, além dos recursos da geração anterior, comunicação wi-fi, memória interna de 16MB e uma entrada para cartões micro-SD (LEGO, 2013f).

De maneira análoga, os sensores evoluíram muito com o passar do tempo: a primeira geração possuía sensores de luz, rotação, temperatura e toque (CAPRANI, 2006); a segunda toque, cor, som e ultrassom (seus motores já são equipados com sensores de rotação) (LEGO, 2006); e a terceira possui sensores de cor, toque e infravermelho (LEGO, 2013f). A última geração do kit permite, também, a utilização de todos os sensores fabricados para a geração anterior a ele.

Atualmente, além dos sensores oficiais comercializados pela LEGO , existem diversas empresas cujo foco é o desenvolvimento de sensores adicionais para os kits da LEGO MINDSTORMS NXT e EV3. Pode-se destacar, por exemplo, as empresas americanas HiTechnic (HITECHNIC, 2001-2012), Dexter Industries (INDUSTRIES, 2015) e Mindsensors.com (MINDSENSORS.COM, 2005-2015), que fabricam sensores de pressão, de aceleração, de voltagem e de corrente, pneumáticos, magnéticos, entre outros, todos compatíveis com ambos blocos NXT e EV3.

Em estudos recentes, os kits da LEGO MINDSTORMS vêm sendo usados com êxito em cursos de graduação em engenharia mecatrônica, elétrica e computação em vista de sua plenitude operacional: além de possuir uma programação simples e completa, o mesmo possui quase todos os sensores fundamentais de robótica. O conjunto fornece uma

[2] Adaptado de (CAPRANI, 2006), (LEGO, 2006) e (LEGO, 2013f).

poderosa ferramenta para os alunos de graduação na área, que podem facilmente prototipar uma solução robótica de qualidade para a materialização dos seus projetos.

2.2 Visão Computacional na Robótica

Um dos pilares da robótica baseia-se na localização de objetos no espaço e no referenciamento destes aos robôs. Assim, o desenvolvimento de algoritmos de localização são de extrema importância na área. É o caso da Localização de Monte Carlo (MCL), algoritmo probabilístico de localização global baseado nos conceitos de cadeia de Markov que representa as distribuições posteriores da posição do robô através de uma coleção aleatória de partículas ponderadas que se aproximam da distribuição desejada (AD; BURGARDC; DELLAERTA, 2000). Existe ainda a dependência em relação à visão, que apresenta um papel importante para a localização no espaço de um objeto. O processamento de imagem fornece os parâmetros necessários com os quais o robô pode se basear para tomar suas decisões, assim a identificação e o rastreamento de objetos se tornam um objetivo para a robótica. Um método simples, desenvolvido por DAS et al., utiliza-se da segmentação de imagens com o objetivo de capturá-las e determinar a cor de um alvo base. Neste trabalho, o alvo é escolhido e pré-determinado assim como sua cor, não tendo, portanto, inteligência ou liberdade para identificação de objetos aleatórios.

A localização de objetos pode ser feita de maneira mais eficaz utilizando-se da captura de imagem 3D (ŠULIGOJ et al., 2013). Inicia-se o sistema de visão capturando uma imagem 2D. A imagem é processada com o objetivo de encontrar os marcadores (referências). As coordenadas em 2D dos pixels destes marcadores são usadas para extrair uma imagem 3D, permitindo assim uma maior flexibilidade ao robô. Outros métodos para navegação foram desenvolvidos, como o descrito em (DAVISONA; KITAB, 2001), que realiza o mapeamento sequencial (utilizado na localização e mapeamento rápido e sucessivo) para aplicações em tempo real.

Paralelamente a estes estudos, desenvolveram-se também novas ferramentas que facilitam a captura e interpretação dos dados de uma câmera. Assim não é necessário domínio no campo da visão computacional ou da robótica para desenvolver um projeto. É o caso do Eyepatch (MAYNES-AMINZADE; WINOGRAD, 2007), uma ferramenta simplificada para extrair informações úteis de uma imagem voltada para programadores iniciantes, ou também do Microsoft Robotics Studio (MSRS) (TRUNG; AFZULPURKAR; BODHALE, 2009) assim como dos toolkits AForge.NET, MATLAB e OpenCV. Um exemplo de um robô inteligente seria o RHINO (BUHMANN et al., 1995), uma plataforma robótica móvel equipada com 24 sensores de distância do tipo sonar, uma câmera dual-color e 2 computadores embarcados. Ele opera com autonomia e é capaz de aprender com o ambiente, devido a um software inteligente e em tempo real, gerando caminhos com custo

mínimo.

As diversas pesquisas feitas no campo da robótica envolvendo visão computacional servem de base nesse projeto para futuras releituras, podendo ser então aplicadas como parte da solução do desenvolvimento do módulo de visão proposto.

2.3 Visão Computacional e o kit da LEGO MINDSTORMS

Desde a primeira geração do LEGO MINDSTORMS , a possibilidade de criar um módulo de visão que permitisse aos robôs ver o ambiente no qual eles se situam já era uma preocupação. De fato, já na primeira geração, a LEGO desenvolveu um sistema de visão integrável ao kit da LEGO MINDSTORMS : o Mindstorms Vision Command (GASPARI, 2001).

O sistema é composto por uma câmera Logiteck QuickCam embalada por um bloco especial equipado com encaixes compatíveis com os da LEGO . O módulo em si não necessita do RCX para funcionar: seu software de reconhecimento pode ser utilizado para ativar o PC ligado a este e produzir sons, capturar fotos e até mesmo vídeos. A comunicação com o RCX é feita através da torre infravermelha.

O programa do Vision Command envia ao RCX um pequeno programa que contém todas as instruções para cada evento. A partir daí, o sistema envia apenas o número referente ao evento ocorrido. Desta forma, as entradas para sensores do RCX não são usadas pelo Vision Command e o RCX não pode ser programado independentemente do Vision Command de maneira simples. Pesquisas começaram a serem feitas de modo a aperfeiçoar esse módulo de visão ou desenvolver um novo conceito.

A integração do kit com uma torre infravermelha, uma webcam e um computador permitiu o processamento de imagens visando a solução dos problemas de ruídos dos dados (pré-processamento com filtros Gaussianos e detectores de arestas de Canny) antes da aplicação dos algoritmos de tratamento da imagem (representados por uma máquina de estados) para controle da visão do robô (STEVENSON; SCHWARZMEIER, 2007), uma solução ainda primitiva que foi se renovando e se desenvolvendo com o crescimento da popularidade dos kits e a evolução dos mesmos.

Já na segunda geração do kit, as pesquisas nesta área se intensificaram e diversas soluções para o problema foram desenvolvidas. Demirci et al. (2013) exploram o processamento de imagens de uma câmera por um computador e o envio, através do Módulo de Comunicação por Bluetooth, das informações obtidas pelo PC para o NXT. Estas informações serão utilizadas então pelo robô na escolha do evento a ser realizado.

Neste, o processamento das imagens é feito em duas etapas: primeiro implementa-se algoritmo do Histograma de Gradientes Orientados (HOG - Histogram of Oriented

Gradients) para fazer a detecção de formas; em seguida é aplicada a Função da Média de Diferenças de Amplitudes (AMDF - Average Magnitude Difference Function) a fim de classificar os resultados obtidos anteriormente. Estes módulos combinados fornecem um método invariante de escala e orientação. Sua implementação é feita no ambiente MATLAB para então ser integrado ao RWTH, ferramenta utilizada para a programação e o controle remoto dos dispositivos da LEGO NXT por Bluetooth.

TRUNG; AFZULPURKAR; BODHALE propõem a utilização de uma webcam para fazer o reconhecimento de sinais de trânsito para controlar um robô LEGO MINDSTORMS . A solução utiliza o MSRS, um ambiente de criação de aplicações robóticas para Windows que contribui na confiabilidade e no paralelismo dos componentes em um sistema distribuído. Desta forma, coordenam-se três sistemas diferentes: a webcam para o reconhecimento das placas; o sensor de luz para a navegação do tipo "seguidor de linha"; e o sensor de ultrassom para a detecção dos sinais de trânsito.

Em Zhenjun, Nisar e Malik (2014), a visão computacional é aplicada ao problema de navegação interior. O sistema proposto é composto não mais por uma câmera, mas por um dispositivo Android conectado via wi-fi a um computador, o qual se conecta ao LEGO MINDSTORMS NXT via Bluetooth.

Desta forma, o computador age como o centro de comando para receber as informações do Android e do NXT para, em seguida, enviar-lhes instruções processadas a partir dos dados recebidos. Os algoritmos implementados pelo PC são a Localização pelo Método de Monte Carlo e o Algoritmo A* do Menor Caminho. Já o dispositivo Android implementa o algoritmo ORB de detecção de objetos no qual uma referência pré-definida é disponível no banco de dados do dispositivo. O NXT reage, portanto, passivamente aos comandos recebidos do computador.

Kirillov (2008) desenvolveu uma câmera pan-tilt (ou seja com dois graus de liberdade: `pitch` e `yaw`) com uma webcam comum, um PC e peças de LEGO . Utiliza-se o framework AForge.NET (toolkit usado principalmente para o reconhecimento de imagens básicas (KIRILLOV, 2010)) para o processamento de imagens. Neste, foram também desenvolvidos algoritmos de movimentação manual e automático dos ângulos da câmera para a tarefa de rastreamento de objetos.

Todas as soluções listadas até então possuem uma característica em comum: todas se servem de um computador, o qual se encarrega de fazer o processamento das imagens. Em geral, esta solução tende a oferecer algumas limitações ligadas, por exemplo, ao sistema de comunicação entre a câmera e o computador (no caso da utilização de cabos USB) e à compatibilidade com os outros sensores.

É neste cenário que nasce a NXTCAM (MORAL, 2008, pag. 47): um sistema de visão próprio para os robôs da LEGO MINDSTORMS NXT e EV3. Baseado na AVRCam

(ORLANDO, 2004), o sistema tem a capacidade de processar imagens em tempo real e de detectar e seguir até oito objetos coloridos. O módulo se conecta diretamente à porta de sensores do bloco inteligente. Uma vez conectada, a NXTCam é completamente autônoma e, portanto, não necessita se conectar a um computador. As informações pós-processadas que são enviadas ao bloco da LEGO contém estatísticas do objeto: quantidade, cor, coordenadas dos limites do objeto ou de segmentos (MORAL, 2008, pag. 59).

O sistema pode ser conectado a um PC através de um cabo USB e, após a instalação do software de visualização e de configuração, é possível visualizar a imagem no computador. Este software serve também para configurar os Mapas de Cores para o processamento a bordo. Os objetos de interesse são reconhecidos através da comparação dos valores de cor armazenados com a imagem capturada, o que significa que estes objetos precisam ser gravados na memória do sistema. O módulo pode, portanto, guardar 8 Mapas de Cores e prover as informações processadas referentes aos objetos correspondentes a estes mapas.

Figura 3: NXTCam[3].

Existem diversas aplicações já desenvolvidas que utilizam a NXTCam: desde as tarefas mais básicas como reconhecimento e seguimento de objetos até robôs que jogam ping-pong, connect four e de navegação autônoma.

[3] Reproduzido de (MINDSENSORS, 2014)

3 REQUISITOS DO PROJETO

Os requisitos do sistema a ser desenvolvido devem levar em consideração tanto a estrutura física (*hardware*) quanto a estrutura computacional (*software*) necessárias para que o bloco programável EV3 possa receber e interpretar as informações da câmera pré-tratadas pelo módulo de visão proposto.

3.1 Requisitos funcionais

O usuário, ao adquirir o módulo, deve ser capaz de:

- Conectar o produto diretamente às portas de entrada do EV3 utilizando o mesmo cabeamento dos sensores oficiais da LEGO;
- Utilizar o software de programação em blocos da LEGO (ICON-BASED SOFTWARE) para importar o bloco de funcionalidades do módulo criado neste projeto;
- Acessar informações fornecidas pelas diversas funções do bloco do módulo de visão para programar uma criação robótica que reaja aos dados obtidos pela funcionalidade escolhida;
- Reprogramar o módulo, criando funções diferenciadas daquelas criadas neste projeto.

Já o módulo deve, de forma automática, ser capaz de:

- Autoidentificar-se quando conectado ao EV3;
- Conectar-se a uma câmera embarcada e recolher informações da mesma;
- Fazer o tratamento das imagens obtidas pela câmera em tempo real de forma a sintetizar as informações contidas na mesma;
- Enviar ao EV3 as informações concernentes à funcionalidade selecionada pelo usuário;
- Identificar a troca de função requisitada pelo usuário e reagir de maneira apropriada à demanda.

As funcionalidades de base do módulo, ou seja, as funções desenvolvidas no âmbito deste projeto, são listadas abaixo:

1. Cores: identificação de objetos nas cores vermelha, azul e verde, posicionamento e tamanho do maior objeto nas respectivas cores;

2. Formas bases: identificação de objetos nas formas circulares, retangulares e triangulares, posicionamento e tamanho do maior objeto nas respectivas formas;

3. Faces: reconhecimento de faces, posicionamento e tamanho da maior face encontrada;

3.2 Requisitos não funcionais

3.2.1 Requisitos do produto

No que diz respeito aos aspectos não funcionais do módulo, o mesmo deve possuir as seguintes características:

- Por se tratar de um módulo a ser embarcado no robô, ele deve ser o menor e o mais leve possível a fim de não interferir na movimentação do mesmo;
- Como os usuários finais dos produtos da LEGO MINDSTORMS são majoritariamente crianças e adolescentes, a sua concepção deve ser pensada de modo a ser resistente, robusta e simplificada;
- Como o mesmo deve ser integrado às criações robóticas em peças LEGO, o invólucro do módulo deve seguir os padrões dos demais sensores.

3.2.2 Requisitos organizacionais

Os requisitos organizacionais do sistema consistem em:

- Entregar o protótipo no fim do ano letivo de 2015 para que este possa ser avaliado por uma banca como Tabalho de Conclusão de Curso de Engenharia Mecatrônica da Escola Politécnica a USP (EPUSP);
- O número de funções implementadas deve ser limitado devido ao período de dois semestres do trabalho;

4 ANÁLISE E DESIGN

4.1 Arquitetura

Seguinte à descrição detalhada dos objetivos e requisitos do projeto, propõe-se a seguinte arquitetura: uma câmera se conecta a uma unidade de processamento, responsável por efetuar o tratamento de imagem em tempo real e o envio das informações recolhidas para o EV3. Sabe-se de antemão que serão necessários diversos componentes eletrônicos adicionais para efetuar a comunicação entre os elementos do módulo.

Considerando que a unidade de processamento deve atender aos requisitos de tamanho, peso, processamento de dados em tempo real e reprogramabilidade, conclui-se que o mesmo deve ser tão potente quanto um computador, porém portátil. A tecnologia recomendada para este caso de utilização são os microcomputadores.

De maneira análoga, como a câmera deve atender aos mesmos requisitos físicos e temporais, pode-se utilizar um módulo de câmera embarcado, do tipo industrial, que além de pequena é capaz de fornecer imagens numa resolução razoável a uma velocidade compatível com sistemas em tempo real.

O esquema desta arquitetura é apresentado na figura 4.

Em resumo, os materiais necessários para o desenvolvimento do projeto são listados abaixo:

- Um kit LEGO MINDSTORMS EV3;
- Um microcomputador;

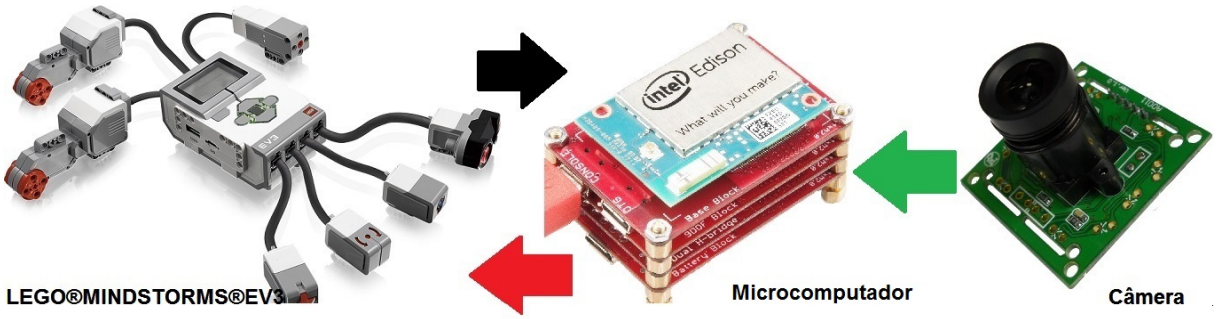


Figura 4: Arquitetura proposta do módulo de visão para LEGO MINDSTORMS EV3. Adaptado de (LEGO, 2013b), (CASEYTHEROBOT, 2014) e (ELECTRONICS123,).

- Circuitos eletrônicos para comunicação com o EV3;
- Um módulo de câmera embarcado.

A seguir serão apresentados cada um dos componentes do módulo de visão proposto.

4.2 LEGO MINDSTORMS EV3

O bloco programável EV3 contém (LEGO, 2013d):

- um processador ARM9 de 32 bits configurado com o sistema operacional Linux;
- 64MB de memória RAM e 16MB de memória FLASH;
- interface para cartão micro-SD;
- comunicação Bluetooth; interfaces client e host para comunicação USB;
- 4 portas de entrada com 6 fios de interface suportando tanto interfaces analógicas quanto digitais;
- 4 portas de saída com 6 fios suportando entrada de encoders para motor;
- um display de 178x128 pixels em preto e branco;
- um auto-falante;
- 6 botões de interface com o usuário;
- fonte de alimentação através de 6 pilhas AA ou uma bateria de lítio recarregável;
- conectores de 6 fios do tipo RJ-12 com ajuste do lado direito.

A programação do bloco inteligente é feita através de um software especializado fornecido pela LEGO. De forma intuitiva, cada elemento é caracterizado por um bloco que pode ser arrastado e colocado no ambiente de programação. As informações de saída de cada um dos blocos podem ser recuperadas e utilizadas como entrada nos demais blocos.

Os blocos são classificados em 5 categorias diferentes (LEGO, 2013e):

1. Ação: são os blocos que permitem o controle por parte do EV3;
2. Controle de fluxo: são os blocos de controle de tempo e loop de informações;
3. Sensores: são os blocos que fornecem informações ao EV3;
4. Operações: são os blocos de operações matemáticas, como declaração de variáveis, somas e comparações;

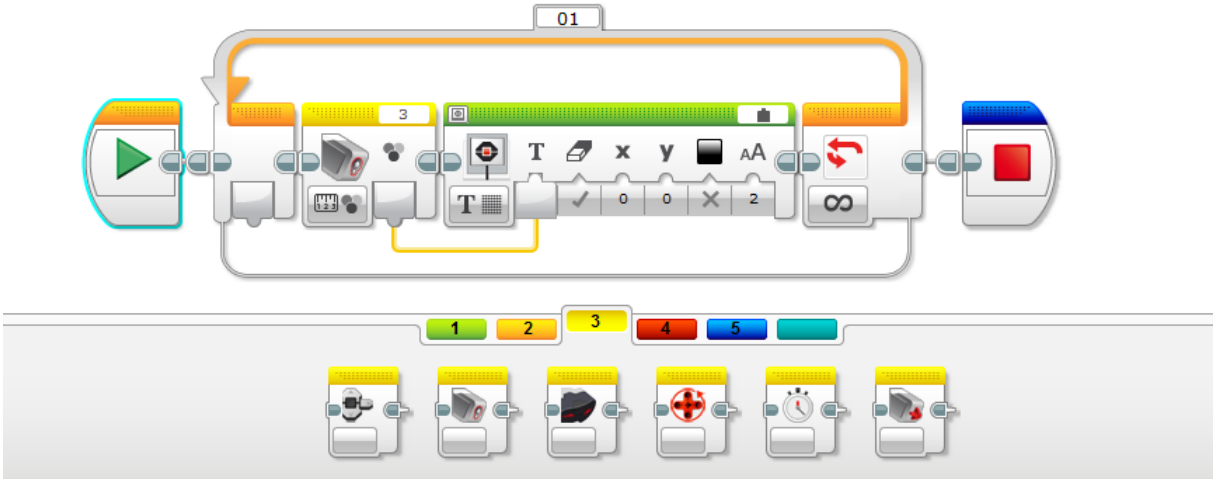


Figura 5: Ambiente de programação da LEGO MINDSTORMS EV3.

5. Avançado: são os blocos que não se encaixam em nenhuma das outras categorias, como os responsáveis pela comunicação Wi-Fi e Bluetooth, por exemplo.

O usuário é ainda capaz de criar blocos personalizados a partir dos blocos já existentes no programa. Estes são chamados de "My Blocks", e são utilizados para guardar segmentos de programas que são repetidamente utilizados em muitos projetos. Um exemplo de programação no software está apresentado na figura 5.

No âmbito do projeto, a criação de um novo sensor implica na criação de um bloco de programação correspondente a este novo sensor, que será classificado como um bloco do tipo "Sensores".

4.3 Modulo de Visão

4.3.1 Microcomputador

Atualmente, os microcomputadores mais conhecidos no mercado são o Raspberry Pi e o Intel Edison[5]. Oferecendo vantagens nos quesitos tamanho, peso e processamento, a solução da Intel se destaca como sendo a mais recomendada para o projeto.

Suas especificações são listadas abaixo (INTEL, 2015):

- Conector de 70 pinos;
- 35.5 x 25.0 x 3.9 mm de dimensões físicas;

[5] O nome Intel® é marca registrada da *Intel Corporation*. Seu uso neste texto não implica em aval por parte da *Intel Corporation* ao seu conteúdo.

- 40 GPIOs as quais podem ser configurados como interface para cartão SD, comunicação UART, comunicação I2C, comunicação SPI, comunicação I2S, controle de PWM, portas USB com controlador OTG e saída de clock;
- Intel SoC de 22nm que inclui uma CPU Intel Atom^{TM} dual-core e dual-thread a 500 MHz e um processador Intel Quark^{TM} de 32 bits a 100 MHz;
- 1GB de memória RAM e 4GB de memória FLASH;
- Wi-Fi dual-band integrado;
- Bluetooth 4.0;
- Suporte para Yocto Linux, Arduino, Python, Node.js e Wolfram.

A utilização deste microcomputador requer a conexão de placas de expansão ao mesmo. A Intel possui dois kits de expansão compatíveis com o Intel Edison: o Kit Intel Edison para Arduino e o Kit de Placa de Expansão Intel Edison. Já a Sparkfun apresenta diversos blocos de expansão para o Intel Edison (CASEYTHEROBOT, 2014). Cada um deles oferece uma funcionalidade diferente: alimentação, comunicação via console, acesso às entradas e saídas e funções especiais. Os blocos podem então ser empilhados devido à presença dos conectores de 70 pinos fêmea embaixo da placa.

Considerando que, com os blocos da Sparkfun, é possível obter uma solução fisicamente pequena e personalizada às necessidades do projeto, optou-se pelos mesmos. As funcionalidades necessárias ao projeto são: alimentação, comunicação via console, OTG (On The Go) e UART. A escolha da UART como protocolo de comunicação com o EV3 se encontra no capítulo 5.

Foram incorporados ao projeto, portanto, os Blocos Sparkfun para Intel Edison Base e UART. O primeiro possui 2 conectores micro AB USB: o Console e o OTG. Ambos permitem a energização do Edison. O Console utiliza o FT231x para fornecer uma interface USB-Serial para acessar o console do Edison. Já o OTG permite a conexão com webcams, dispositivos de armazenamento ou outros dispositivos USB.

O bloco UART possui uma interface de console simples via um cabo FTDI (com 6 pinos, sendo que os pinos 2 e 3 correspondem aos sinais Tx e Rx, respectivamente). Além disso, a placa possui uma chave para trocar entre a UART1 e a UART2, sendo a segunda especialmente configurada para acessar o console do Edison.

Ambas as placas são montadas uma em cima da outra de maneira a minimizar o espaço utilizado e o Intel Edison se encaixa no bloco superior.

4.3.2 Câmera

A escolha da câmera é essencial para o sucesso do projeto. Equipada com um sensor CMOS VGA para alta qualidade de imagem e baixo consumo de energia, o módulo de câmera embarcado escolhido provê até 30 fps em qualidade VGA (640x480 pixels) (ELECTRONICS123,).

A interface com o computador é feita através de um conector USB 2.0 de alta velocidade. A mesma possui driver UVC (USB Video Class) para uso em máquinas com sistemas operacionais Linux, Windows XP SP2 ou acima. As dimensões físicas da câmera são 32 x 32 mm. Sua lente possui distância focal de 3.6 mm e abertura de 1/2.0.

4.3.3 Cabeamento

Tão importante quanto os componentes em si são as conexões entre os mesmos. O esquema de cabeamento entre os elementos acima apresentados será como apresentado abaixo:

- Alimentação: as placas de expansão da Sparkfun utilizadas no módulo oferecem três opções para a alimentação do microcomputador, sendo elas através dos pinos da placa UART, da porta micro USB OTG ou ainda da porta micro USB Console. Como a primeira solução interfere no bom funcionamento dos outros pinos da UART e o VCC da porta micro USB OTG está conectado a um diodo zener, o qual provoca uma queda de tensão, a melhor opção é a alimentação pela porta micro USB Console. Sendo assim, é necessária a conexão dos fios de energia do conector fêmea da LEGO aos fios de energia da porta micro USB Console;

- Comunicação UART com o EV3: a comunicação UART utiliza uma linha única de transmissão de dados (Tx) e outra de recebimento de dados (Rx). O cabeamento necessário para a implementação dessa comunicação envolve a conexão dos pinos Tx, Rx e GND do conector fêmea da LEGO aos pinos Tx, Rx e GND da placa de expansão UART do Edison;

- Comunicação via Console: uma vez que a alimentação do Edison é implementada através da porta micro USB Console, a conexão direta do usuário ao microcomputador via console deve ser feita manualmente trocando-se o conector de alimentação pelo cabo desejado;

- Comunicação USB com a câmera: a conexão da câmera ao Edison será realizada de maneira direta, onde o conector de 5 pinos da mesma se conecta à porta micro USB OTG da placa de expansão do Edison.

4.3.4 Invólucro

Todos os sensores da terceira geração do LEGO MINDSTORMS possuem o mesmo design: uma estrutura de plástico, leve e compacta, nas cores cinza escuro e branca, e com encaixe compatível com as peças da LEGO no fundo. Assim, o invólucro do módulo de visão a ser desenvolvido planeja seguir as mesmas diretrizes desses sensores, uma vez que as suas características estão alinhadas aos requisitos não funcionais do projeto.

Como a porta micro USB Console é utilizada para a alimentação do microcomputador, o usuário deve ser capaz de abrir e fechar o invólucro de maneira fácil e prática a fim de se conectar ao mesmo. Desta forma, o invólucro foi desenvolvido em três partes separadas que se encaixam facilmente através de abas que devem ser parafusadas umas nas outras.

O esboço do invólucro do módulo de visão se encontra na figura 6, e o desenho de conjunto do mesmo se encontra no apêndice C.

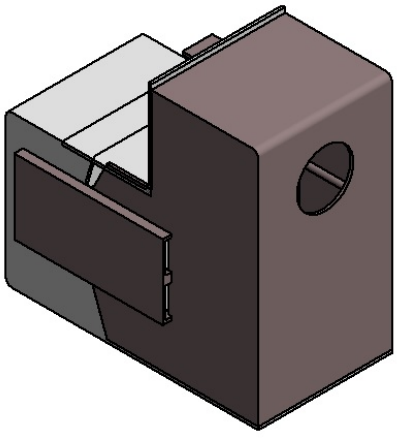


Figura 6: Esboço do invólucro.

5 METODOLOGIA

5.1 Bloco EV3

Como já mencionado anteriormente, a LEGO dispõe de um software de programação em blocos próprio para o desenvolvimento dos projetos com o EV3. Além de programar, o software disponibiliza ao usuário as informações físicas do bloco inteligente, diversas ferramentas de ajuda, atualização de software e de firmware, entre outros (LEGO, 2013e).

No âmbito deste projeto, a ferramenta mais importante disponível é a importação de blocos de programação. Através dela, o usuário é capaz de importar diferentes blocos criados por terceiros com as mais diversas finalidades. O arquivo a ser importado possui extensão *.ev3b*, e nada mais é do que uma pasta compactada possuindo todos os arquivos necessários para a utilização do bloco.

O módulo de visão a ser desenvolvido deve, portanto, estar associado a um novo bloco de programação, o qual deverá conter as funcionalidades propostas anteriormente. Foi neste cenário que a LEGO decidiu criar uma versão especial do software de programação direcionada ao desenvolvimento de novos sensores. O mesmo consiste em uma versão simplificada do software, porém com diferentes funcionalidades as quais permitem a análise, criação e modificação do conteúdo do arquivo a ser importado.

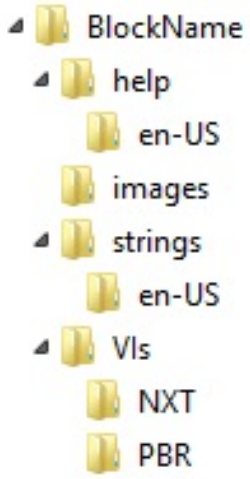


Figura 7: Árvore de diretórios de um bloco.

A estrutura completa do arquivo *.ev3b* pode ser encontrada em (LEGO, 2013a). Este é composto por diversos diretórios, como mostra o exemplo da figura 7. O diretório principal recebe o nome do bloco e possui um arquivo *blocks.xml*, um diretório **VIs**, um diretório **strings**, um diretório **images** e um diretório **help**. O diretório **VIs** é aquele que contém todo o código do bloco, implementado em arquivos *.vix*. Os códigos que independem do hardware pertencem ao diretório **VIs**, enquanto aqueles que pertencem aos blocos NXT e EV3 devem ficar agrupados nos subdiretórios de nomes **NXT** e **PBR**, respectivamente.

Já o diretório **strings** possui subdiretórios por país, nomeados com o código do idioma do mesmo (como "en-US"). Cada um desses subdiretórios contém um arquivo *blocks.xml* que provê os nomes a serem exibidos ao usuário, um texto descritivo e links de ajuda para os ítens programáticos definidos no arquivo *blocks.xml* principal no idioma do país.

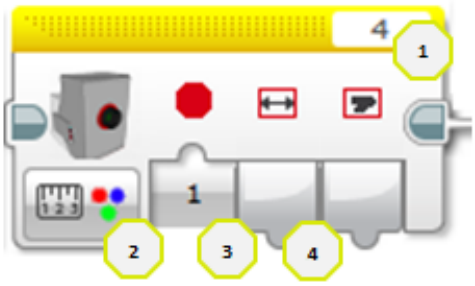


Figura 8: Representação em blocos do módulo de visão.

Todas as imagens usadas na palheta e as configurações dos parâmetros do bloco se agrupam no diretório **images**. Todas estas imagens possuem nomes e tamanhos específicos para serem reconhecidas pelo software.

Finalmente, o diretório **help** também possui subdiretórios por país, nomeados com o código do respectivo idioma, e contém um arquivo *.html* com informações de apoio à utilização do bloco. Todas as imagens relacionadas a este arquivo devem ser colocadas nesta mesma pasta.

Os arquivos *blocks.xml* acima mencionados são usados para definir quase tudo sobre o bloco, exceto o código *.vix* que é compilado e usado no programa. Aquele que se encontra no diretório principal define, por exemplo, qual código *.vix* usar em um determinado modo, os parâmetros de um determinado modo, e qual modo usar como padrão ao colocar o bloco no ambiente de programação pela primeira vez. Já aquele que se encontra no diretório **strings** mapeia alguns elementos programáticos no primeiro *blocks.xml* para exibir nomes e fornecer textos de ajuda.

Os arquivos *.vix*, responsáveis pelo funcionamento do bloco, só podem ser criados no ambiente de programação para desenvolvimento de novos sensores. Os mesmos podem ser programados através de uma combinação de blocos mais simples (como funções matemáticas, lógicas, de controle de fluxo e estruturas de dados) e de funções de mais baixo nível, chamadas "gray blobs"(LEGO, 2013a). Estas permitem, por exemplo, o acesso aos dados recebidos pelo bloco inteligente através das portas de entrada.

No desenvolvimento do módulo de visão, criou-se o bloco *EVision.ev3b*, que pode ser diretamente importado no software de programação em blocos da LEGO. Os arquivos intrínsecos ao mesmo se encontram no apêndice B.

A figura 8 apresenta o bloco quando colocado no ambiente de programação. Os principais elementos, indicados na figura, representam:

1. Porta de entrada padrão do módulo de visão (4);
2. Botão para seleção dos modos a serem utilizados;

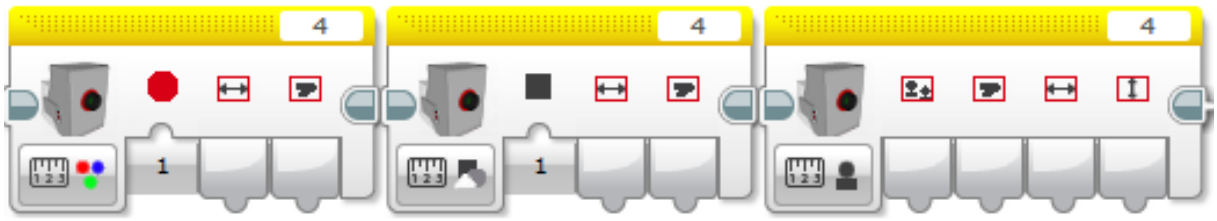


Figura 9: Representação em blocos dos modos de identificação de cores, formas e faces, respectivamente.

3. Parâmetro de entrada do modo selecionado;

4. Parâmetros de saída do modo selecionado.

O EVision implementa, em alinhamento com os requisitos do projeto, três modos: identificação de cores, de formas e de faces. Todos os modos implementados são categorizados como funções de medição. Os blocos correspondentes a cada um dos modos são representados na figura 9.

O modo de identificação de cores possui um parâmetro de entrada, o qual permite o usuário escolher a identificação da cor vermelha (1), azul (2) ou verde (3), e dois parâmetros de saída, que fornecem a posição no eixo horizontal (Posição X) e o tamanho (Área) do maior objeto da cor selecionada. Os valores são fornecidos como uma porcentagem do tamanho da imagem capturada pela câmera. Caso nenhum objeto da cor selecionada seja detectado, os valores dos parâmetros de saída são todos iguais a zero.

Analogamente, o modo de identificação de formas possui um parâmetro de entrada para a seleção das formas retangular(1), circular(2) e triangular(3), e dois parâmetros de saída com a posição horizontal (Posição X) e o tamanho (Área) do maior objeto da forma selecionada. Caso nenhum objeto da forma selecionada seja detectado, os valores dos parâmetros de saída são iguais a zero.

Já o modo de identificação de faces possui quatro parâmetros de saída: número de faces detectadas (# de faces), posição horizontal (Posição X), vertical (Posição Y) e tamanho (Área) da maior face detectada. Caso nenhuma face seja detectada, os valores dos quatro parâmetros de saída são iguais a zero.

Os nomes de exibição e textos de ajuda existem apenas nas línguas portuguesa e inglesa, ou seja, existem penas dois subdiretórios nos diretórios **strings** e **help**: o “en-US”e o “pt”.

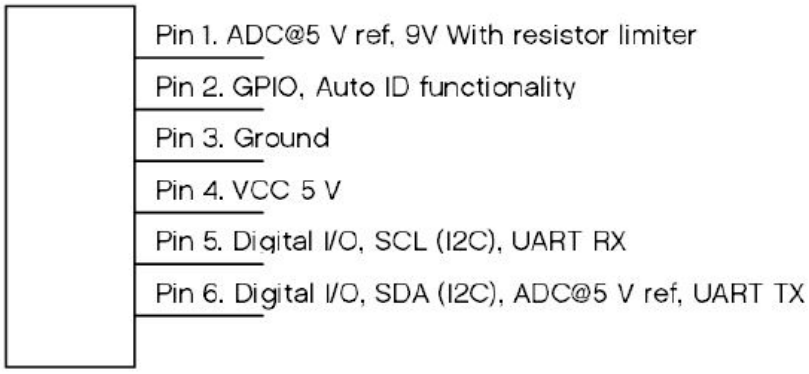


Figura 10: Configuração do conector implementado nas portas de entrada do EV3. Reproduzido de (LEGO, 2013d).

5.2 Protocolo de comunicação

O desenvolvimento de um novo sensor para o LEGO MINDSTORMS EV3 demanda um estudo aprofundado do bloco inteligente e, mais especificamente, das suas portas de entrada. A sua principal funcionalidade é permitir o sistema reagir ao seu entorno através do feedback dos sensores. Esta comunicação é implementada através de uma interface de 6 fios (LEGO, 2013d). O esquema detalhado dos fios atrás da porta 1 do EV3 se encontra na figura 10.

Neste caso, o pino 1 suporta a leitura de valores analógicos ou sensores que requerem um nível de tensão mais elevado. O pino 2 é usado durante a função de autoidentificação do sensor. Os pinos 3 e 4 fornecem os níveis de tensão 0 V (Ground) e 5 V (VCC), respectivamente. Quando o sistema identifica automaticamente o tipo do sensor atrelado à porta, o mesmo configura os pinos 5 e 6 para a funcionalidade apropriada.

O EV3 suporta a troca de informações de diferentes maneiras: valores analógicos, comunicação I2C ou UART. A comunicação I2C suporta uma taxa de transmissão máxima de 9600 bits/s e um tamanho máximo de 32 bytes de buffers de comunicação. Toda comunicação I2C é executada dentro de drivers de software, assim como todos os dispositivos externos devem incluir resistores pull-up em ambos os pinos 5 e 6.

A comunicação bi-direcional mais rápida que o EV3 suporta é a UART. Sendo uma comunicação assíncrona, ela suporta taxas de transmissão entre 2400 bits/s e 460k bits/s nas portas 1 e 2, enquanto as portas 3 e 4 suportam até 230k bits/s. A comunicação UART usa 1 bit de início, 8 bits de dados, nenhum bit de paridade e 1 bit de parada.

A partir de então, para se estabelecer a conexão desejada entre o sensor e o EV3, uma configuração de conexões específica deve ser seguida. A plataforma suporta a autodetecção dos elementos externos através da identificação dos níveis de tensão aos quais os pinos de 1 a 6 estão conectados.

Tabela 1: Sequência de autoidentificação nas conexões de entrada. Reproduzido de (LEGO, 2013d).

Dispositivos I2C	Nível no pino 2 é LOW Nível no pino 5 é HIGH Nível no pino 6 é HIGH Outras validações requerem comunicação
Sensor de Luz do NXT	Nível no pino 2 é LOW Nível no pino 5 é LOW
Sensor de Cor do NXT	Nível no pino 2 é LOW Valor no pino 1 é menor do que 100 mV
Sensor de Toque do NXT	Nível no pino 2 é LOW Valor no pino 1 é maior do que 4800 mV
Sensor de Toque do NXT	Nível no pino 2 é LOW Valor no pino 1 é entre 850 mV e 950 mV
Sensor de Toque do NXT	Nível no pino 2 é LOW Nenhum dos cenários acima são ativos
Sensor Digital do EV3	Nível no pino 2 é HIGH Valor no pino 1 é menor do que 100 mV
Sensor Simples do EV3	Nível no pino 2 é HIGH Valor no pino 1 é entre 100 mV e 3100 mV
Sensor de Temperatura do NXT	Nível no pino 2 é HIGH Valor no pino 1 é maior do que 4800 mV Nível no pino 6 é HIGH

Inicialmente, todas as portas de entrada são identificadas como “Porta Aberta”, estado correspondente aos seguintes níveis de tensão:

- Valor no pino 1 é maior do que 4800 mV (valor AD)
- Nível no pino 2 é HIGH (E/S digital)
- Nível no pino 5 é HIGH (E/S digital)
- Nível no pino 6 é LOW (E/S digital)
- Valor no pino 6 é menor do que 150 mV (valor AD)

A sequência de detecção utilizada, assim como os elementos identificáveis pelo atual firmware, são apresentados na tabela 1.

A arquitetura de comunicação implementada para os sensores digitais da LEGO MINDSTORMS EV3 (i.e.para os sensores que se comunicam via UART) requer que o dispositivo siga um protocolo específico. Este protocolo foi desvendado em (KOHLER, 2015). Existem 4 tipos de mensagem:

1. Mensagens de sistema
2. Mensagens de comando
3. Mensagens de informação
4. Mensagens de dados

Cada mensagem segue uma das seguintes estruturas:

- Byte de mensagem (mensagens do tipo 1)
- Byte de mensagem, byte de checksum[6] (mensagens dos tipos 2 e 4)
- Byte de mensagem, byte de informação, mensagem de payload[7] e byte de checksum (mensagens do tipo 3)

O byte de mensagem tem uma estrutura especial, ObXXLLLYYY, onde: XX indica o tipo de mensagem; LLL indica o tamanho da mensagem de payload (de fato, o tamanho do payload é exatamente 2^{0bLLL} bytes.); para mensagens do tipo 1 e 2, YYY indica o subtipo da mensagem , enquanto que para mensagens do tipo 3 e 4, 0bYYY é um número de modo do sensor.

As tabelas que decodificam todos os tipos de mensagens acima citados se encontram no anexo A.

O protocolo de comunicação com o EV3 funciona da seguinte maneira: inicialmente, o pino de transmissão Tx da UART (i.e. pino 6) deve ficar em LOW por, no mínimo, 500 ms. Esta condição indica ao EV3 uma condição de quebra, ou seja, a conexão de um novo sensor. A partir de então, ambos EV3 e sensor começam a mandar mensagens: o EV3 é capaz de enviar ao sensor as mensagens de sistema ACK e NACK e as mensagens de comando SELEÇÃO e ESCRITA; enquanto o sensor pode enviar ao EV3 as mensagens de sistema ACK e SYNC, as mensagens de comando TIPO, MODOS e VELOCIDADE, todas as mensagens de informação e de dados.

O diagrama de sequência da figura 11 apresenta uma visão geral do protocolo de comunicação entre o EV3 e um sensor digital.

Nele, o sensor retransmite o protocolo até que o mesmo receba uma mensagem de ACK do EV3 dentro de 80 ms após o final do envio do mesmo. A partir desta confirmação, o EV3 passa a enviar, a cada 300 ms, mensagens de NACK para o sensor. Se nenhuma

[6] Também conhecido por *soma de verificação*, é um código usado para verificar a integridade dos dados transmitidos.

[7] É a parte essencial da mensagem transmitida, ou seja, não inclui o "cabeçalho"da mensagem.

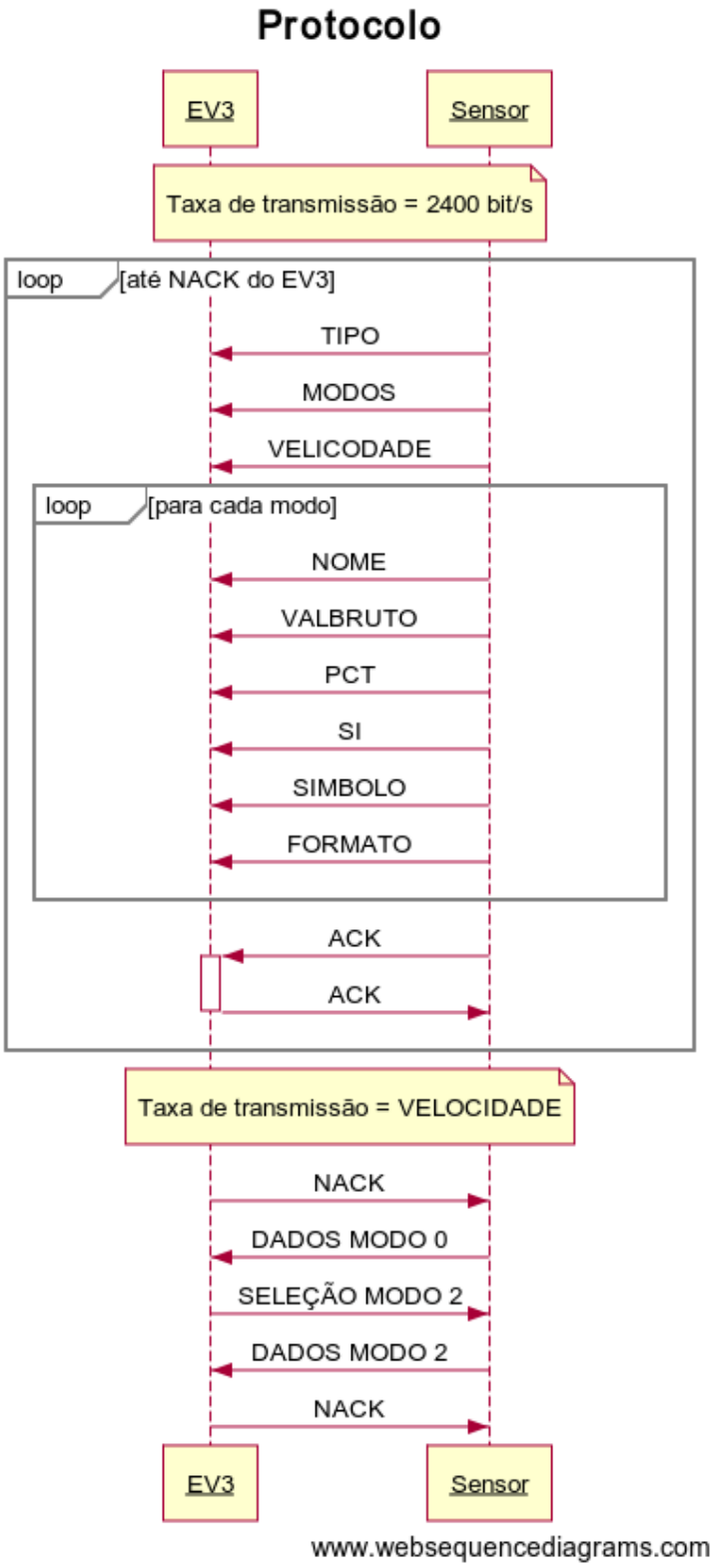


Figura 11: Protocolo de comunicação entre o EV3 e o sensor digital. Adaptado de (LEGO, 2013d).

mensagem de dados for recebida pelo EV3 dentro de 5 NACKs, a comunicação se encerra e o sensor deve ser reinicializado.

No presente projeto, a comunicação serial entre o módulo de visão e o EV3 escolhida foi a UART, devido à sua elevada taxa de transmissão de dados e simplicidade de implementação. Sendo assim, o módulo deverá se autoidentificar como sendo um **Sensor Digital do EV3**, o que significa que o nível de tensão no **pino 1** deverá ser **menor do que 100 mV** enquanto o **pino 2** deverá permanecer em **HIGH**. Os **pinos 5** e **6** serão, em seguida, configurados pelo EV3 com as funcionalidades **UART RX** e **UART TX**, respectivamente.

Uma vez implementado o bloco de programação do EV3, é possível definir as mensagens do protocolo de comunicação. A primeira mensagem é a mensagem de comando de **TIPO**, a qual define o identificador módulo de visão como sendo igual a **66**. Em seguida, tem-se a mensagem de comando de **MODOS**. Ela define dois parâmetros: o número de modos suportados e o número de modos a serem mostrados. No caso do projeto do sensor desenvolvido, esses números são iguais entre eles e iguais ao número de funções do módulo, ou seja, iguais a **3**. Na prática, envia-se o número de modos menos 1, ou seja, a informação a ser efetivamente enviada é igual a **2**. A última mensagem de comando é a **VELOCIDADE**. Nela define-se a velocidade máxima de transmissão de dados suportada pelo sensor, ou seja, a nova baudrate da comunicação serial entre o EV3 e o mesmo. A velocidade máxima do módulo de visão foi definida como sendo igual a **57600**.

As próximas mensagens a serem enviadas são as mensagens de informação de cada um dos 3 modos. Deve-se enviar os modos de trás para frente, ou seja, o último modo deve ser o primeiro a ser enviado enquanto o primeiro modo deve ser o último. Tanto a ordem quanto as informações de todos os modos foram estabelecidas no arquivo *blocks.xml* do diretório principal (vide apêndice B).

O terceiro modo, primeiro a ser enviado, é o de identificação de faces. A primeira mensagem de informação a ser enviada é a de **NOME**, ou seja, **EV-FACE**. Seguinte, deve-se enviar a mensagem de **VALBRUTO**, que define o intervalo de valores brutos dentro do qual as informações enviadas pelo sensor devem pertencer. Definiu-se, então, o intervalo entre **0** e **100** uma vez que todos os valores (exceto o número de faces detectadas) são dados em porcentagem da resolução da câmera.

Como o intervalo de valores em porcentagem seria igual ao intervalo padrão (**0-100**), não é necessário enviar a mensagem de **PCT**. A próxima mensagem a ser enviada é a mensagem de **SI**, que define o intervalo de valores no SI correspondente aos valores brutos. Este intervalo foi definido como sendo idêntico ao intervalo de valores brutos (**0-100**). A mensagem de **SIMBOLO** fornece o nome da unidade no SI. Como a maior parte das informações é enviada pelo módulo de visão em porcentagem, o símbolo definido foi **pct**.

Por fim, uma das mensagens mais importantes para o módulo de visão é a mensagem de **FORMATO**, a qual define o número e o tipo de dados enviados pelo sensor, além do número de dígitos e de decimais a mostrar. Para o modo de identificação de faces, são necessários **4** itens do tipo **inteiro de 8 bits** com até **3** dígitos e **0** decimais a mostrar.

O segundo modo é o de identificação de formas. Seu **NOME** é **EV-SHP** e os dados enviados pelo sensor em **VALBRUTO** estão dentro do intervalo de **0** a **100**, já que todos os valores são dados em porcentagem da resolução da câmera. O intervalo de valores no **SI** também é igual a **0-100** e o **SIMBOLO** é, portanto, igual a **pct**. Seu **FORMATO** é definido como **8** itens do tipo **inteiro de 8 bits** com até **3** dígitos e **0** decimais a mostrar.

Analogamente, o primeiro modo, e último a ser enviado, é o de identificação de cores. Seu **NOME** é **EV-COL**, os intervalos dos dados em **VALBRUTO** e no **SI** são iguais a **0-100** (todos os valores sendo portanto também dados em porcentagem da resolução da câmera), o **SIMBOLO** é **pct** e o **FORMATO** contém **8** itens do tipo **inteiro de 8 bits** com até **3** dígitos e **0** decimais a mostrar.

No final, uma mensagem de **ACK** é enviada. Espera-se que, durante os 80 ms que seguem o final do envio do protocolo, o EV3 responda com uma mensagem de **ACK**. Se esta resposta não for recebida, o protocolo completo deve ser re-enviado. Senão, o sensor pode começar a enviar as mensagens de **DADOS** referentes ao módo selecionado. Sendo o primeiro modo definido como padrão, o sensor deve começar qualquer comunicação enviando as informações referentes a este modo. Em seguida, o EV3 envia uma mensagem de **SELEÇÃO** indicando o modo que o sensor deve começar a enviar as informações a partir de então.

5.3 Programação do microcomputador

A biblioteca de tratamento de imagens mais recomendada para aplicações em tempo real é a **OpenCV** (Open Source Computer Vision). Ela possui interface para C++, C, Python e Java e suporta Windows, Linux, Mac OS, iOS e Android (OPENCV, 2015). Para manter a facilidade do projeto e do software aberto, foi escolhida a liguagem de programação mais "amigavel"dentre as listadas acima: o **Python**, cujas características estão ligadas à produtividade, legibilidade, qualidade, facilidade, portabilidade, interoperabilidade e customização. Em outras palavras, Python é uma linguagem que foi criada para programar de maneira rápida, suportando diversos paradigmas de programação. Sua característica mais marcante, comumente chamada de *baterias inclusas*, significa que quase tudo o que é necessário para lançar um programa em Python está presente na instalação básica.

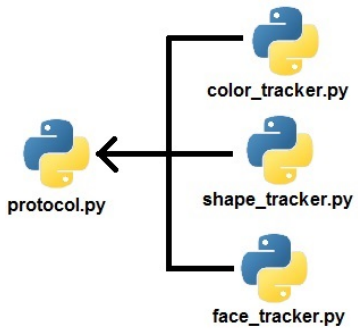


Figura 12: Esquema de arquivos em Python.

5.3.1 Estrutura do Programa

O programa é basicamente dividido em duas partes: o envio protocolar, responsável pela comunicação com o bloco inteligente e identificação do sensor; e o envio das mensagens de execução, que definem a função a ser utilizada pelo sensor e a trasmissão de informações do sensor ao bloco. Um esquema simplificado dos arquivos pode ser analisado na figura 12.

O programa **protocol.py** é o responsável pelo gerenciamento da transmissão protocolar e de dados entre o sensor e o EV3. As funcionalidades do sensor são controladas pelos programas secundários **colortracker.py**, **shapetracker.py** e **facetracker.py**. Os

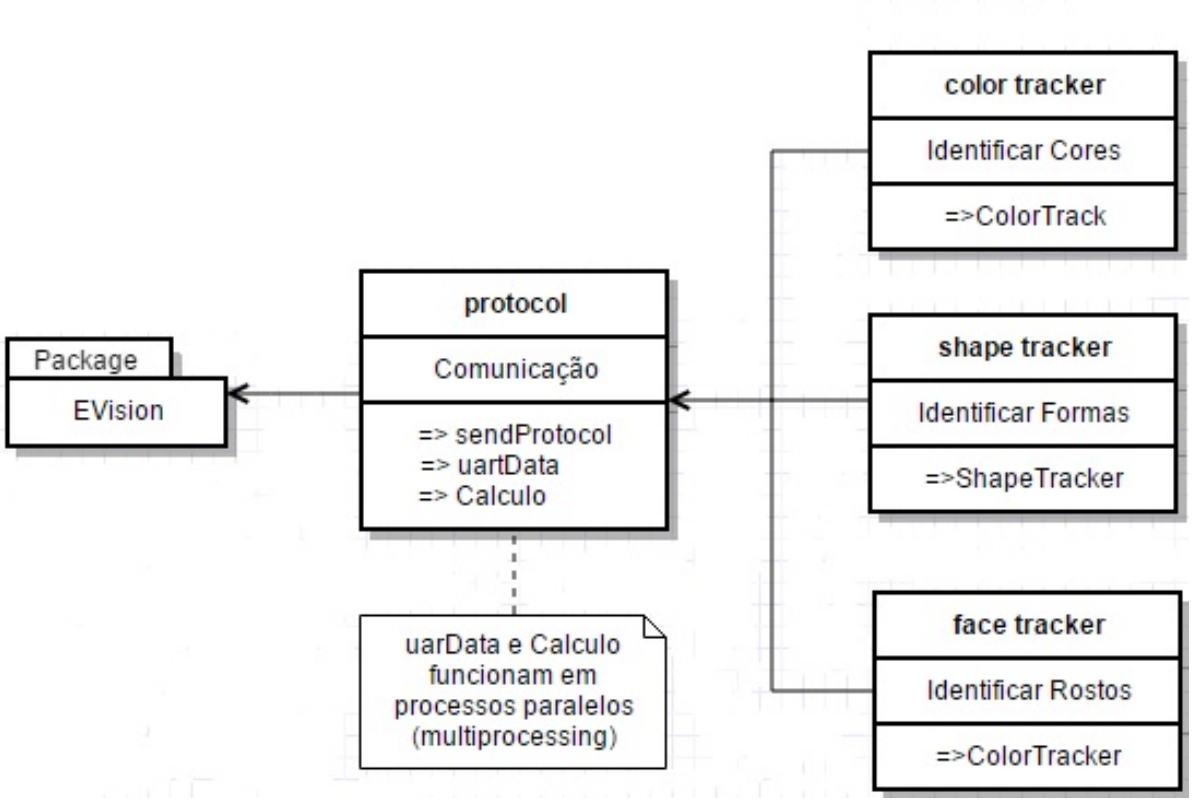


Figura 13: Esquema UML do programa.

arquivos relacionados aos programas se encontram no apêndice D. As classes e funções são descritas na figura 13.

sendProtocol é a função que estabelece todos os parâmetros de comunicação entre o sensor e o bloco inteligente. Esses parametros estão descritos dentro do protocolo inicial. Para essa comunicação foram utilizadas as bibliotecas *pyserial*, que encapsula o acesso para a porta serial, e a *mraa*, utilizadas em plataformas embarcadas, como o Edison ou Galileo. As ações realizadas pela função **sendProtocol** são:

- Configuração das portas da UART;
- Aplicação do nivel logico LOW durante 500 ms;
- Envio do protocolo de comunicação;
- Alteração da baudrate para o início da transmissão de dados.

```
#Transforma o pino 35 da UART (TX) em uma porta GPIO
x=mraa.Gpio(35)
#Define como pino de saida
x.dir(mraa.DIR_OUT)

#Forcar o TX > 500 ms com nivel logico baixo
x.mode(2)
time.sleep(0.505)
x.mode(0)

#Inicializacao dos pinos da UART
x = mraa.Uart(0)
#Definicao inicial da BAUD RATE (2400), padrao para o kit EV3
u = serial.Serial('/dev/ttyMFD1',2400, timeout = 1.4)

   #Limpa RX e o TX
   u.flushInput()
   u.flushOutput()

   #Inicio da mensagem protocolar
   msg = bytearray.fromhex('Protocolo a ser enviado')
   u.write(msg)

#Apos envio da mensagem, espera resposta afirmativa do bloco EV3 (ACK)
info=u.read()
```

```
#Testa o recebimento da mensagem
if info:
   #Se mensagem for um ACK saimos da funcao de envio do protocolo
   if (info.encode("hex")=='04'):
      break

#Redefine baudrate para uma de transmissao da dados (mais rapida)
u.baudrate=57600
```

As funções **uartData** e **Calculo** devem executadas em paralelo para permitir a simultaneidade de leitura e de envio de dados ao bloco inteligente. A arquitetura implementada deve permirir também a troca de informações entre as funções, como por exemplo a mudança de modo de operação. Para tanto, a biblioteca *multiprocessing*, que suporta processos através da utilização de uma API semelhante à biblioteca de *threading*, foi a solução escolhida. Mais especificamente, são utilizadas as funções *Process*, para gerar dois processos que trabalhariam simultaneamente e *Queue*, para criar uma "ponte"entre esses processos e permitindo, assim, a troca de informação entre eles.

```
from multiprocessing import Process, Queue

def uartData (q,t,v):
#Conteudo ...
def calculo (q,t):
#Conteudo ...

#Criamos um ponto de troca de informacoes entre os processos
q = Queue()
t = Queue()
#Criacao dos processos
p = Process(target=uartData, args=(q,t,u,))
k = Process(target=calculo, args=(q,t,))
p.join()
k.join()
```

uartData é a função responsável por "escutar"as mensagens enviadas pelo bloco inteligente pelo pino Rx e enviar as mensagens do sensor ao EV3 pelo pino Tx. Essencialmente, o EV3 envia ao sensor mensagens de **SELEÇÃO**, reagindo a uma demanda do usuário para a troca de modo no programa. Essa mensagem é composta por 3 bytes:

- **43:** indica a demanda de mudança de modo do sensor;
- **00/01/02:** *00* indica a troca para o modo de identificação de cores, *01* para formas

e *02* para faces;

- **Checksum:** byte de verificação.

```
#Espera alguma informacao enviada pela funcao Calculo
if not q.empty():
   #Recupera essa informacao
   msg = q.get()
#Envia essa informacao para a UART (para o Bloco Inteligente [Tx])
v.write(bytearray.fromhex(msg))
#Le constantemente a UART [RX]
info= v.read()

#Verifica a mensagem de modo
if info:
   #Se recebeu uma mensagem de selecao:
   if (info.encode("hex")=='43'):
      #Le o proximo byte (00,01 ou 02)
      info = v.read()

      if (info.encode("hex")=='01') :
         #Caso modo 01 envia modo 1 a funcao Calculo
         t.put(modo)

      elif (info.encode("hex")=='02') :
         #Caso modo 02 envia modo 2 a funcao Calculo
         t.put(modo)
```

Calculo tem como objetivos principais o tratamento das imagens capturadas pela câmera e o alinhamento do modo selecionado pelo usuário com a construção correta da mensagem que será enviada à função *uartData* e, posteriormente, ao bloco inteligente. A mensagem de dados construída por essa função apresenta a seguinte forma:

- **D8/D9/D2:** indica o modo do sensor (D8 = modo 0, D9 = modo 1 e D2 = modo 2);
- **Dados:** informação recuperada a partir do tratamento de imagens realizado. Seu tamanho varia em função do modo selecionado (8 bytes = modo 0 e 1 e 4 bytes = modo 2);
- **Checksum:** byte de verificação

```
#Verifica se recebeu alguma informacao do uartData (no caso o modo)
if not t.empty():
   #Recupera essa informacao
   modo = t.get()
#Se modo 0 (MODO PADRAO)
if modo == 0:
   #Color Tracker
   #Execucao da funcao de identificacao de cores
   res = col.ColourTrack(capture)
   #Aplicacao da funcao "checksum" para construir a mensagem
   msg = cs.cksum(res,modo)
elif modo == 1:
   #Form Tracker
   #Execucao da funcao de identificacao de formas
   res = shp.ShapeTracker(capture)
   msg = cs.cksum(res,modo)
elif modo == 2:
   #Face Tracker
   #Execucao da funcao de identificacao de rostos
   es = face.FaceTracker(capture,faceCascade)
   msg = cs.cksum(res,modo)
#Envia a mensagem para a funcao uartData
q.put(msg)
#Tempo de espera entre 2 mensagens enviadas
time.sleep(0.05)
```

Checksum é a função secundária, utilizada por **Cálculo**, responsável por gerar o byte de verificação da mensagem e montar a mensagem final a ser enviada. Sendo assim, essa função recebe os dados brutos gerados pelas funções de tratamento (*msg*) e o modo de execução (*modo*) e retorna a mensagem final completa, ou seja, com o byte de identificação de modos no início e o de verificação no final:

```
   #Numero de bytes contidos na mensagem
   nbytes=len(msg)
      M=''
      for i in range(0, nbytes):
         #Cria-se um vetor com a string msg
         M=M+"{:02x}".format(msg[i])
   #Incluimos o modo no inicio da mensagem
   if modo == 0:
      M ='D8'+M
```

```
        elif modo == 1:
           M ='D9'+M
        elif modo == 2:
           M ='D2'+M

        msg = bytearray.fromhex(M)
        nbytes=len(msg)
        R=0x00
        #Faz-se XOR entre cada byte da mensagem
        for x in range(0,nbytes):
           R=R^msg[x]
        #Finaliza-se com XOR ff
        CS=R^0xff
        #Monta a mensagem
        msgCS=M + "{:02x}".format(CS)
        return msgCS
```

As funções de tratamento de imagem seguem um padrão básico comum, diferenciando-se apenas em suas funcionalidades específicas. Elas são implementadas com a utilização da biblioteca *OpenCV.cv2*, que apresenta funções *standards* para a identificação de objetos e características específicas da imagem, como é o caso, por exemplo, da função *contours*, capaz de identificar curvas juntando todos os pontos contínuos (isto é, ao longo da fronteira) que tenham a mesma cor ou intensidade. Os "contornos"são uma ferramenta útil para a análise de formas e detecção de objetos.

```
#Captura um frame de imagem
ret, image = capture.read()
#Aplica um filtro sobre o frame para converter a imagem para escalas de
    cinza se flag igual a COLOR_BGR2GRAY ou HSV se COLOR_BGR2HSV
gray = cv2.cvtColor(image, cv2.flag)
#Implementa da funcao que encontra no frame as caracteristicas desejadas
    (seja cor, foma ou o numero de faces)
#Identifica o numero de objetos encontrados
for x in objs:
   #Dentre todos os objetos selecionados, apenas o maior sera identificado
   if area>max_area:
#Determina o vetor de informacao a ser enviado
res=np.int_(<VALORES>)
return res
```

ColorTrack é a função que implementa a identificação de cores. Nela são definidas

faixas de cores, no formato RGB, que são expressos em vetores na forma *[RED,GREEN,BLUE]*. Esses valores foram obtidos de forma experimental, e o resultado se encontra a seguir:

- **Vermelho:** max = [5, 255, 255] — min = [0, 150, 0]
- **Azul:** max = [130, 255, 255] — min = [100, 100, 100]
- **Verde:** max = [80, 255, 255] — min = [40, 100, 100]

Utiliza-se, então, a função *findContours* para identificar qualquer objeto com as características descritas pelas faixas de cores.

ShapeTracker é a função que implementa a identificação de formas simples. Nela, foram contabilizados o número de linhas que compõem os polígonos identificados na imagem, e cada uma das formas desejadas foi associada a um número específico de linhas, de maneira que:

- **Triangulo:** 3 linhas
- **Quadrado:** 4 linhas
- **Círculo:** mais de 15 linhas

Essa função utiliza o método *approxPolyDP*, o qual aproxima as formas detectadas por uma curva poligonal com a precisão especificada, para identificar o número de lados de todos os poligonos destacados.

FaceTracker é a função que implementa a identificação de faces. Utiliza-se um método específico, chamado *Haar Cascades* (VIOLA P. ; MITSUBISHI ELECTR. RES. LABS., 2001), o qual utiliza uma abordagem baseada na aprendizagem de máquina, onde uma função "cascata"é treinada a partir de varias imagens positivas (com o objeto) e negativas (sem o objeto), gerando um arquivo *.xml* com as características do objeto em análise. Esse arquivo é, em seguida, utilizado na detecção deste objeto em outras imagens.

5.3.2 Execução no Boot (Edison)

Um dos requisitos do projeto é a inicialização automática das funcionalidades do sensor, evitando assim a necessidade do usuário de acessar a porta serial do Edison a fim de inicializá-las manualmente. A solução escolhida foi o lançamento das funções durante o boot do microcomputador. Para isso, foi necessario escrever um programa para alterar a programação do *shell*. Mais especificamente, o código *shell* a ser alterado chama-se *Again Shell*, mais conhecido por *bash*. O diretorio que contém os arquivos *bash* e que é, então, responsável por iniciar e parar serviços durante a inicialização e/ou desligamento do

sistema, chama-se */etc/init.d/*. Adicionou-se um script *ev3.sh* a esse diretório que executa o programa do módulo de visão no boot do Edison e continua sua execução no *background* do sistema.

```
#! /bin/sh
### CONFIGURACOES INICIAIS DO SCRIPT
# Provides:           ev3
# Required-Start:     $all
# Required-Stop:
# Default-Start:      1 2 3 4 5
# Default-Stop:       0 6
# Short-Description:  Send ev3 protocol
# Description:        Send the module protocol to the ev3 brick.
### FIM DAS CONFIGURACOES
### Commandos para a Inicializacao do sistema
start(){
   ### Comando direcionado ao compilador
   export
      PYTHONPATH=$PYTHONPATH:/usr/local/lib/i386-linux-gnu/python2.7/s$
   ### Execucao do arquivo main.py (que chama a funcao sendProtocol)
   ### & : Esse comando permite que a execucao do programa aconteca no
      background
   python /home/edison/EVision/main.py &
}
### Comando para o desligamento do sistema
stop(){
   ### Ao fim do programa eliminamos qualquer processo remanescente do
      programa executado (main)
   pkill -9 -f main
}
case "$1" in
   start)
      start
      ;;
   stop)
      stop
      ;;
   *)
```

A interação entre os processos e funções acima descritos é apresentada na forma de diagrama de sequências na figura 14.

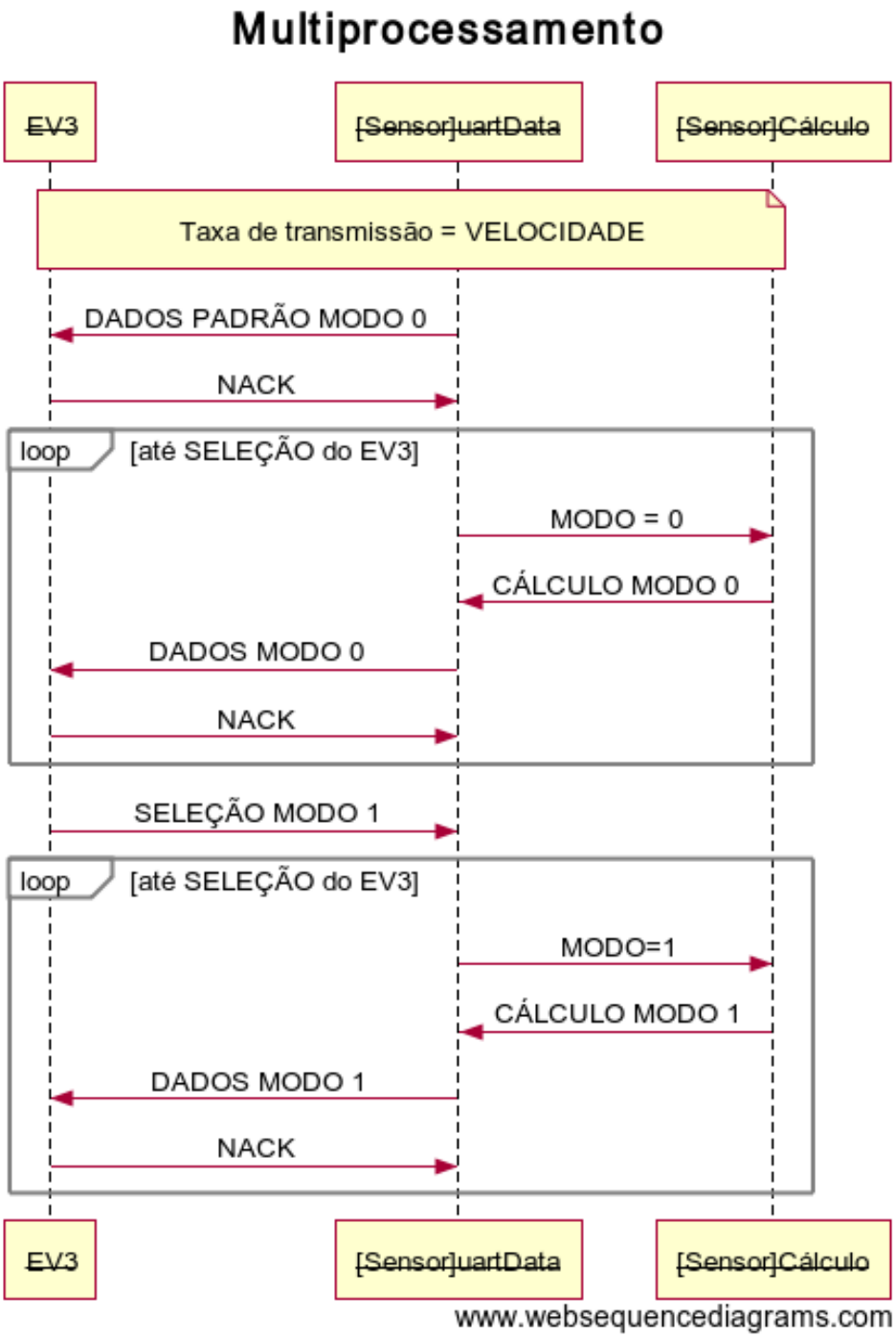


Figura 14: Troca de informações entre os processos.

5.4 Glue Logic

O processo de autoidentificação dos sensores digitais pelo EV3 possui um timeout de aproximadamente 3s dentro dos quais, depois que os níveis de tensão dos pinos 1 e 2 forem ajustados, o sensor deve enviar o protocolo. Sabe-se que, no Intel Edison, o tempo necessário para que o sistema operacional se inicie após a sua energização é de aproximadamente 30s. Levando em consideração que a função de implementação do módulo de visão será lançada no final do processo de inicialização do Edison e que o mesmo será alimentado pelo bloco inteligente, uma vez conectado ao EV3, o sensor deve demorar 10 vezes mais do que o tempo disponível para enviar o protocolo de comunicação após o estabelecimento dos níveis de tensão dos pinos 1 e 2, impedindo assim a autoidentificação do mesmo.

Faz-se necessária a adição de um circuito lógico que permita o controle dos níveis de tensão dos pinos 1 e 2. Como o o valor do pino 2 já é HIGH quando o mesmo está em aberto, resta apenas o controle do nível de tensão do pino 1. O objetivo deste circuito é, então, impor ao pino 1 uma tensão menor do que 100 mV somente quando o Edison começar a se comunicar com o EV3, ou seja, quando este começar a enviar o protocolo de comunicação. Este tipo de circuito é mais comumente chamado de "Glue Logic"(HILL, , pag. 537).

Durante a inicialização do Edison, o nível lógico do seu pino Tx é HIGH. Quando o protocolo começa a ser enviado, o mesmo muda para o nível LOW. A ideia desse circuito é se aproveitar desse comportamento do pino Tx para ajustar a tensão do pino 1. O nível de tensão esperado após a implementação do circuito de Glue Logic no pino 1 em função dos valores lógicos apresentados pelo Tx do Edison está representado na figura 15.

A Glue Logic utiliza portas lógicas NAND para a implementação do circuito digital conhecido como flip-flop (HILL, , pag. 504). A figura 16 apresenta esse circuito e a sua tabela da verdade.

Na Glue Logic, o Tx do Edison deve se conectar ao pino SET do flip-flop. Inicialmente, SET está em HIGH e CLEAR está em LOW. Nessa configuração, Q é LOW e $\overline{Q}$ é HIGH. Após aproximadamente 5 milisegundos, CLEAR deve passar de LOW para HIGH e se manter nesse nível durante todo o funcionamento do circuito. Faz-se necessária a utilização de um circuito temporizador. O mesmo pode ser obtido colocando um capacitor

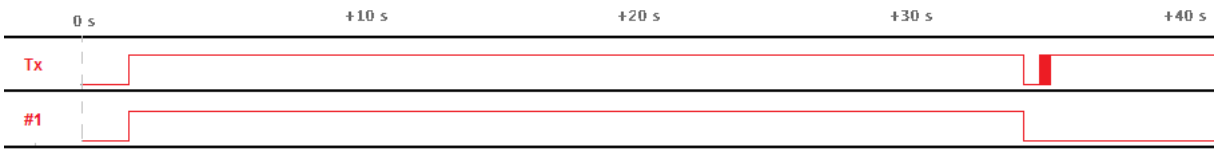
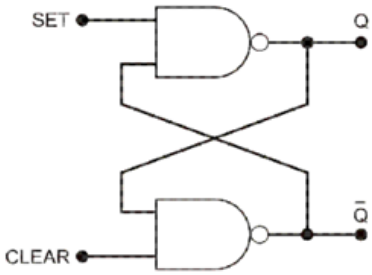


Figura 15: Resultado esperado com a adição do circuito de Glue Logic.



SET	CLEAR	Q	$\overline{Q}$
1	1	Mantêm	
0	1	1	0
1	0	0	1
0	0	Indeterminado	

Figura 16: Circuito e Tabela da Verdade do flip-flop de NAND.

de 470nF em série com uma resistência de 10 kΩ. Para a descarga do capacitor, coloca-se uma resistência em paralelo ao mesmo de valor 10 vezes maior do que a resistência em série. Nesse momento, SET e CLEAR estão ambos em HIGH, o que significa que os valores anteriores de Q e $\overline{Q}$ são mantidos. Dessa forma, garante-se que o estado inicial das saídas Q e $\overline{Q}$ será LOW e HIGH, respectivamente.

A partir desse momento, apenas o valor de SET, conectado ao Tx, pode mudar, e até que ele mude, o estado das saídas será mantido. Assim, quando o Tx muda para LOW pela primeira vez, o valor de Q e $\overline{Q}$ mudam para HIGH e LOW, respectivamente. Desse momento em diante, o valor das saídas Q e $\overline{Q}$ será HIGH e LOW, respectivamente, para quaisquer valores do Tx. A sequência dos valores lógicos dos pinos SET, CLEAR, Q e $\overline{Q}$ são graficamente apresentados na figura 17.

O circuito descrito fornece o comportamento esperado pelo pino 1 na saída $\overline{Q}$. Porém, o valor de tensão correspondente ao nível lógico LOW varia entre 190 e 140 mV. Para que o módulo seja reconhecido como um sensor digital, o valor de tensão no pino 1 deve ser inferior a 100 mV.

A resolução implementada para esse problema utiliza um transistor, que realiza a amplificação da corrente (HILL, , pag. 62). Dessa forma, o nível lógico LOW passa a corresponder a um nível de tensão muito próximo de 0 V. No caso da utilização de um

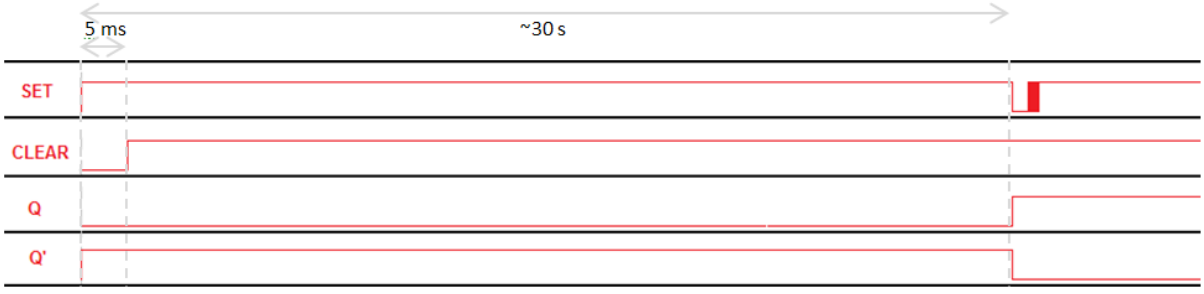


Figura 17: Valores lógicos do flip-flop ao longo do tempo.

transistor do tipo NPN, deve se conectar a saída Q, de comportamento inverso ao desejado no pino 1, ao pino de base do transistor. Assim, a saída com o comportamento desejado se encontra no pino coletor do transistor. Resistores devem ser colocados tanto na base quanto no coletor do componente para limitar a corrente no mesmo.

Uma vez consolidada a resolução do ajuste do nível de tensão do pino 1, o circuito de Glue Logic é também o responsável pela robustez dessa solução. Afim de estabilizar ruídos de baixa frequência provenientes dos pinos Tx e Rx do Edison, ambos devem estar conectados a filtros passa-altas implementados com resistores e capacitores em série.

Finalizando o circuito da Glue Logic, observou-se que o sinal do pino Tx não é potente o suficiente para ser identificado pelo EV3 de maneira constante.

A solução implementada utiliza duas portas lógicas NAND já disponíveis no circuito. Ambas possuem um dos pinos de entrada conectados ao VCC, configuração na qual o NAND passa a agir como uma porta lógica inversora (NOT) do sinal presente no outro pino de entrada. A ideia é, então, ligar o pino de saída de uma das portas ao pino de entrada da outra, resultando assim em uma dupla-inversão do sinal de entrada da primeira porta NAND. Esse circuito, conhecido como buffer, fortalece o sinal enviado sem alterar o seu valor lógico, resolvendo assim as instabilidades do pino Tx.

Os desenhos esquemáticos de fabricação eletrônica deste circuito podem ser encontrados no apêndice A.

6 RESULTADOS

O objetivo principal do projeto foi definido como sendo a construção de um módulo de visão integrado ao kit da LEGO MINDSTORMS EV3. O desenvolvimento desse sensor foi dividido em duas partes principais: a implementação da conexão entre o EV3 e o Edison; e a implementação do tratamento de imagens pelo Edison.

A implementação da conexão entre o bloco inteligente e o sensor pode ser subdividida em duas: o desenvolvimento do bloco de programação necessário para que o EV3 seja capaz de identificar o módulo de visão; e o envio, pelo Edison, das informações necessárias para o estabelecimento da conexão entre os dois elementos.

No que concerne o bloco de programação do EV3, os resultados obtidos foram satisfatórios: de maneira simples, o usuário é capaz de importar o arquivo *EVision.ev3b* no ambiente de programação em blocos da LEGO e integrá-lo aos outros blocos, permitindo assim a criação de programas mais complexos que se utilizam do módulo de visão.

As três funções que devem ser implementadas pelo módulo de visão são previstas neste bloco. No modo de identificação de cores, o bloco retorna a posição horizontal e a área do maior objeto da cor escolhida pelo usuário. De maneira análoga, o modo de identificação de formas retorna a posição horizontal e a área do maior objeto da forma filtrada. Já o modo de identificação de faces retorna o número de faces detectadas e as características de posição horizontal e vertical e tamanho da maior face encontrada.

Uma vez que o bloco de programação do EV3 esteja finalizado, é possível recuperar dele todas as informações necessárias para a implementação do protocolo de comunicação entre o sensor e o bloco inteligente. Utilizando as bibliotecas *piserial* e *mraa*, foi possível desenvolver o programa **sendProtocol.py** que permite ao Edison o envio do protocolo via UART para o EV3.

O mesmo, quando enviado em seguida ao ajuste dos níveis de tensão dos pinos responsáveis pela autoidentificação do módulo como sensor digital do EV3, é identificado com sucesso. Porém, essa conexão direta possui alguns defeitos graves: primeiramente, a elevada quantidade de ruídos nas portas Tx e Rx fazem com que a identificação do protocolo pelo EV3 fique extremamente instável. Além disso, para que o sensor cumpra a tarefa de autoidentificação de maneira independente, o protocolo deve ser enviado durante o processo de inicialização do sistema operacional do Edison. Esse processo restringe o momento em que os níveis lógicos dos pinos de autoidentificação devem ser ajustados, uma vez que o envio do protocolo deve ser feito aproximadamente 3 segundos depois desse ajuste.

Faz-se então necessária a adição do circuito de Glue Logic, o qual foi desenvolvido de maneira a resolver os problemas de comunicação encontrados. Utilizando-se de capacitores, resistores, transistores e portas lógicas NAND, a Glue Logic permitiu a integração do envio tardio do protocolo pelo Edison e a atenuação dos ruídos de forma que, uma vez ligado ao EV3, o módulo de visão é corretamente identificado dentro de um período de aproximadamente 30 s.

Já no que concerne a implementação do tratamento de imagens pelo Edison, a programação foi desenvolvida em Python utilizando a biblioteca OpenCV. A câmera, conectada à porta micro USB OTG, faz a captura de imagens VGA a 30 frames por segundo. Tanto a resolução quanto a taxa de transmissão de imagens mostraram-se suficientes para a aplicação em tempo real implementada. A cada captura de imagem, os filtros de cor, formas e face são implementados e as características desejadas dos objetos identificados são retornados pelas respectivas funções.

Para o função de identificação de cores, o algoritmo implementado é capaz de identificar de maneira satisfatória o maior objeto nas três cores desejadas (vermelho, azul e verde) e retornar suas posições horizontais e tamanhos relativos à resolução da iamagem. Considerando que o intervalo dos valores das cores foi definido de maneira experimental, a variação destes altera significantemente o resultado da função.

A função de identificação de formas foi implementada de forma similar à de cores, e o seu resultado foi tão satisfatório quanto o primeiro uma vez que o algoritmo é capaz de identificar o maior objeto das três formas desejadas (retangular, circular e triangular) e retornar as posições horizontais e tamanhos. Como a identificação das formas utiliza o número de arestas dos objetos como parâmetro de classificação, e circulos não tem arestas, foi determinado que qualquer objeto com mais de 15 arestas é classificado como um círculo, o que pode levar a resultados imprecisos para esta forma.

A implementação da identificação de faces, diferentemente das funções anteriores, exigiu a utilização do método *Haar Cascades*, que utiliza um arquivo *.xml* para a parametrização das características do rosto. Dessa forma, ela é a função que exige maior processamento. Ainda assim, o resultado obtido foi satisfatório, mesmo que o processo de identificação seja significantemente lento em relação aos outros filtros utilizados.

Uma vez desenvolvidas cada uma das partes do módulo, cria-se a necessidade de integração das mesmas. Este processo é realizado implementado através do conceito de multiprocessamento. Após o envio do protocolo de comunicação, dois processos paralelos são criados. O primeiro cuida das tarefas de leitura e escrita da UART do módulo, enquanto o segundo realiza os cálculos das funções. O processo de leitura controla o modo selecionado e envia esta informação ao processo de cálculo. Este, por sua vez, faz o cálculo da função do modo correspondente e envia o resultado deste para o primeiro processo, que se encarrega de enviá-lo ao EV3.

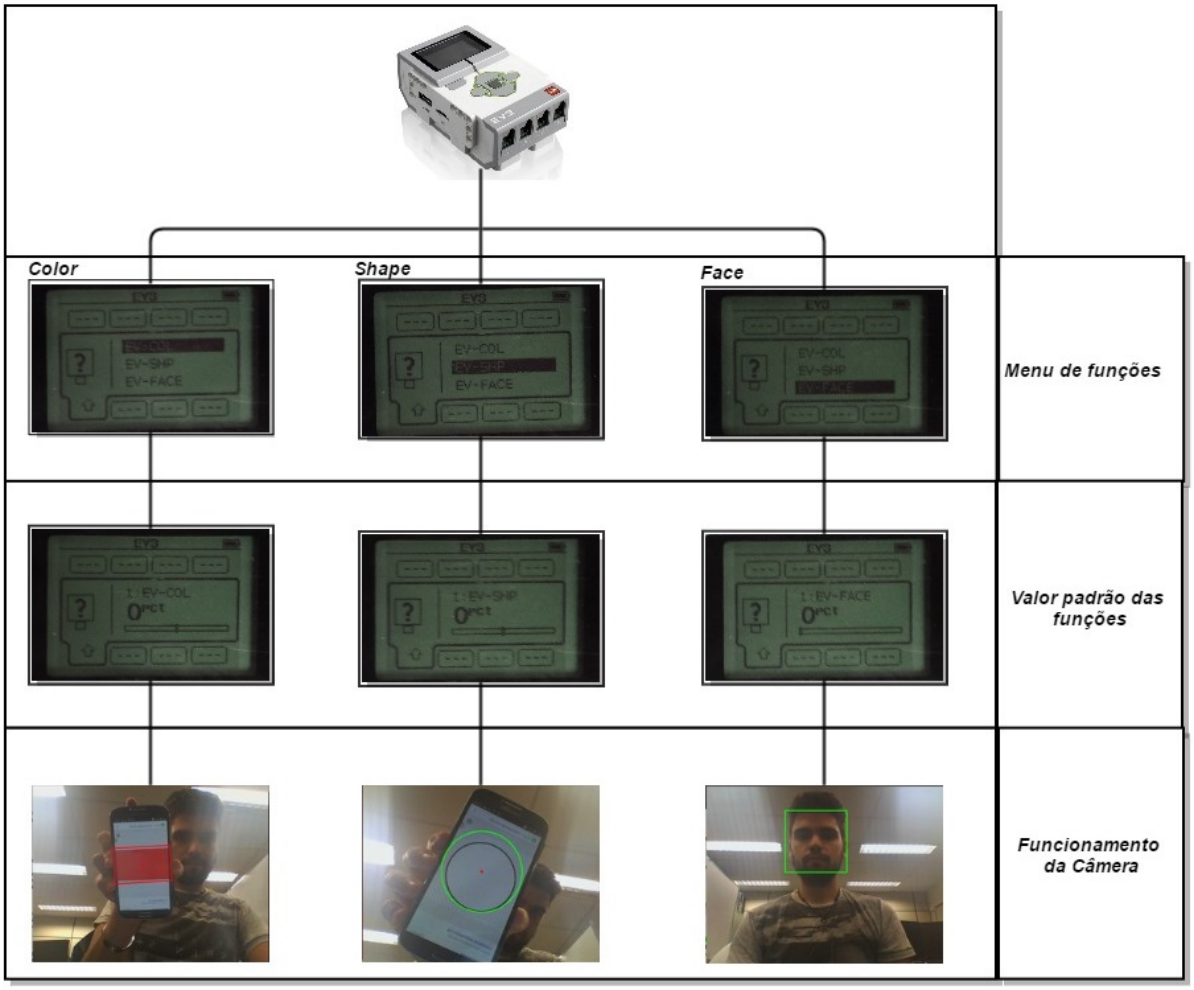


Figura 18: Validação da integração do módulo de visão.

A escolha desta solução para a integração entre o software e o hardware desenvolvidos mostrou-se muito eficaz. Enquanto o tratamento da imagem não está finalizado, o processo responsável pelo envio da informação continua enviando a última informação disponível. Essa informação é atualizada no momento em que o cálculo é finalizado e o processo responsável transmite o resultado.

No que concerne a reprogramabilidade do módulo, o mesmo foi projetado de maneira que o usuário seja capaz de abrir o envólucro, retirar o conector de alimentação da porta micro USB Console, e se conectar via serial diretamente ao Edison. Uma vez conectado, o mesmo é capaz de acessar o terminal, se conectar ao Edison, parar os processos que implementam o módulo de visão (uma vez que os mesmos são lançados em *background* no boot do sistema) e modificar o módulo. O usuário pode, também, configurar a rede wi-fi do Edison e se conectar a ele via ssh, evitando assim o trabalho de desmontar o invólucro todas as vezes que o mesmo for alterar a programação do sensor.

A maneira mais simples de validar o resultado obtido pela integração de todas as partes do projeto é atraves da utilização do modo de inspeção das portas de entrada do

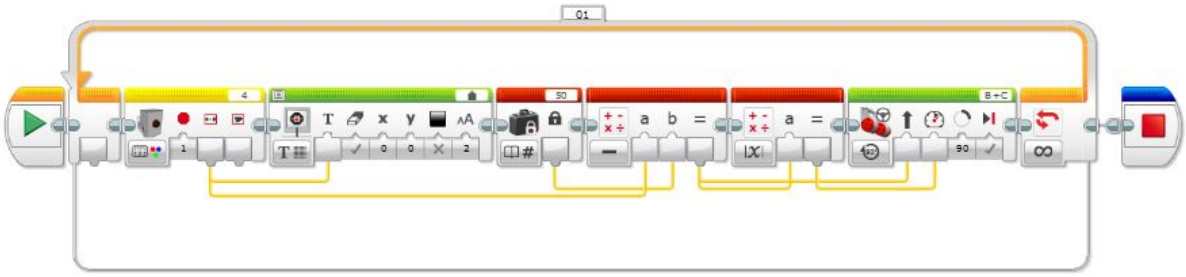


Figura 19: Exemplo de programação integrada aos outros blocos da LEGO.

EV3. Nele, o usuário é capaz de conectar os sensores às portas correspondentes e visualizar o primeiro parâmetro recebido. A figura 18 apresenta um esquema com o resultado obtido nessa validação após a integração de todos os elementos. Pode-se confirmar que, para quaisquer modos selecionados, o parâmetro é atualizado conforme o esperado. Vale notar que os valores que aparecem na figura são os valores padrão das funções, ou seja, eles não correspondem aos parâmetros encontrados nas figuras do funcionamento da câmera, que foram utilizadas somente para demonstrar visualmente o resultado do tratamento de imagens.

Após a validação da integração, é possível integrar o bloco a uma programação mais complexa. A figura 19 mostra um exemplo de programação na qual o robô tenta manter o maior objeto vermelho presente no seu campo de visão no centro do eixo horizontal da câmera. Nela, a distância entre o centro do objeto e o centro da imagem controlam a direção e a velocidade com que o robô precisa virar em torno do seu próprio eixo para colocar o objeto no centro. Caso este já esteja no centro, o robô não se move. Se este se encontra próximo ao centro, o robô vira suavimente no sentido de posicioná-lo no centro. Já se o objeto está muito distante do centro, o robô vira rápidamente com o objetivo de centralizá-lo.

O resultado da implementação deste algoritmo é conforme o esperado: o sensor é capaz de se autoidentificar dados os 30 s de inicialização do Edison e enviar os dados coletados da função selecionada. É possível coletar os valores recebidos de forma a utilizá-los como entradas para outros blocos da LEGO. Mesmo que este algoritmo não seja o mais eficaz para a solução deste problema (já que este não espera pelo tempo de autoidentificação do sensor e os resultados durante este período de tempo são imprevisíveis), o mesmo é capaz de consolidar a verificação da integração do módulo de visão desenvolvido ao kit da LEGO MINDSTORMS EV3.

7 CONCLUSÃO

No início do projeto, diversos requisitos funcionais e não funcionais foram estabelecidos a fim de guiar o desenvolvimento do módulo de visão integrado ao EV3. No final do projeto, é possível revisitá-los para realizar uma análise dos resultados obtidos na solução implementada.

Nela, o usuário é capaz de conectar o sensor às portas de entrada do EV3 utilizando os mesmos cabos dos sensores da LEGO, importar o bloco *EVision.ev3b* ao software de programação da LEGO, recuperar as informações deste bloco e utilizá-las em outros blocos da LEGO, além de reprogramar o módulo, mesmo que inicialmente seja necessário abrir o invólucro para realizar tal tarefa.

O módulo, por sua vez, consegue se autoidentificar de forma autônoma, conectar-se a uma câmera embarcada, realizar o tratamento das imagens dessa câmera e associar os resultados ao modo correto. Todas as funcionalidades de base planejadas foram implementadas: identificação de cores, formas e faces.

A solução é leve e pequena o suficiente para não atrapalhar a movimentação de um robô de LEGO ao ser embarcado ao mesmo, porém ela não se mostrou tão robusta quanto ela poderia ser, apresentando ainda alguns ruídos que quebram a conexão entre o sensor e o EV3.

De maneira geral, pode-se dizer que todos os requisitos, funcionais e não funcionais, foram cumpridos de maneira satisfatória, de onde conlui-se que o projeto é de possível implementação e que a escolha dos materiais e da arquitetura do módulo permitem a criação de um módulo de visão integrado ao kit da LEGO MINDSTORMS EV3, que era o objetivo incial do projeto.

7.1 Sugestões para trabalhos futuros

A visão computacional ainda é um domínio que está em constante desenvolvimento. No âmbito do tratamento de imagens, o projeto oferece infinitas oportunidades de desenvolvimento de novas funcionalidades para o módulo, como por exemplo streaming de imagens, identificação de formas mais complexas, entre outras. Com o avanço constante da tecnologia, pode-se também almejar a implementação de algoritmos de otimização das funcionalidades já implementadas nesse projeto.

Inspirada no software de blocos da LEGO MINDSTORMS, uma oportunidade latente que nasce da realização desse projeto é a criação de um software de programação

visual de tratamento de imagens, o qual implementaria as funções da biblioteca OpenCV.

No quesito compatibilidade, o módulo poderia ser aprimorado de forma a poder ser utilizado também com o kit da segunda geração da LEGO, o NXT.

No que concerne o bloco de programação do EV3, nota-se que todas as funções implementadas são do tipo "Medição", criando assim a oportunidade do desevolvimento de novos modos pertencentes a outras categorias. Pode-se, por exemplo, criar um modo do tipo "Comparação"que realiza a conversão do valor da posição horizontal do objeto em porcentagem recebido pelo módulo para a informação "direita"ou "esquerda", ou então o aprimoramento das dos modos de identificação de cores e formas no qual o usuário seja capaz de obter as informações de todas as cores simultaneamente.

REFERÊNCIAS

AD, D. F. S. T.; BURGARDC, W.; DELLAERTA, F. Robust monte carlo localization for mobile robots. In: *Artificial Intelligence Volume 128, Issues 1–2*. [S.l.: s.n.], 2000. p. 99–141. Citado na página 17.

BUHMANN, J. et al. The mobile robot rhino. *AI Magazine Volume 16 Number 2 (1995) (© AAAI)*, v. 16, n. 2, p. 31–38, 1995. Full text available. Citado na página 17.

CAPRANI, O. *RCX Manual*. [S.l.], 2006. Disponível em: ⟨http://legolab.daimi.au.dk/CSaEA/RCX/Manual.dir/RCXManual.html⟩. Acesso em: 02 março 2015. Citado na página 16.

CASEYTHEROBOT. General guide to sparkfun blocks for intel edison. SparkFun Electronics, dezembro 2014. Disponível em: ⟨https://learn.sparkfun.com/tutorials/general-guide-to-sparkfun-blocks-for-intel-edison⟩. Acesso em: 21 junho 2015. Citado 2 vezes nas páginas 23 e 26.

DAS, P. K. et al. Article: Vision based object tracking by mobile robot. *International Journal of Computer Applications*, v. 45, n. 8, p. 40–42, May 2012. Full text available. Citado na página 17.

DAVISONA, A. J.; KITAB, N. Sequential localisation and map-building for real-time computer vision and robotics. In: *Robotics and Autonomous Systems Volume 36, Issue 4*. [S.l.: s.n.], 2001. p. 171–183. Citado na página 17.

DEMIRCI, B. et al. Implementing hog amp; amdf based shape detection algorithm for computer vision amp; robotics education using lego mindstorms nxt. In: *Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 288–293. Citado na página 18.

EDUCATION, L. *LEGO Education*: Lego education worldwide. The LEGO Group, 2015. Institutional website. Disponível em: ⟨http://education.lego.com/fr-fr/about-us/lego-education-worldwide/our-company⟩. Acesso em: 25 setembro 2015. Citado na página 15.

ELECTRONICS123. *Electronics123.com,inc*: Sb101c usb cmos board camera. Electronics123.com,inc. Disponível em: ⟨http://www.electronics123.com/shop/product/sb101c-usb-cmos-board-camera-module-5204?search=SB101C+USB+CMOS+Board+Camera+Module⟩. Acesso em: 25 setembro 2015. Citado 2 vezes nas páginas 23 e 27.

GASPARI, M. Get started in robotic vision. *Robot Science and Technology Magazine*, n. 8, p. 51–52, Feb/Mar 2001. Citado na página 18.

HILL, P. H. W. *The Art of Electronics*. [S.l.]: Cambridge University Press. Citado 2 vezes nas páginas 47 e 48.

HITECHNIC (Ed.). Página Web Institucional, *HiTechnic Products*. 2001–2012. Disponível em: ⟨https://www.hitechnic.com/⟩. Acesso em: 19 junho 2015. Citado na página 16.

INDUSTRIES, D. (Ed.). Página Web Institucional, *Dexter Industries*. 2015. Disponível em: ⟨http://www.dexterindustries.com/site/⟩. Acesso em: 19 junho 2015. Citado na página 16.

INTEL. *Intel Edison Development Platform*. [S.l.], 2015. Disponível em: ⟨http://download.intel.com/support/edison/sb/edison_pb_331179002.pdf⟩. Acesso em: 30 setembro 2015. Citado na página 25.

KIRILLOV, A. *LEGO Pan Tilt Camera and Objects Tracking*. 2008. Disponível em: ⟨http://www.codeproject.com/Articles/31104/Lego-Pan-Tilt-Camera-and-Objects-Tracking⟩. Acesso em: 02 março 2015. Citado na página 19.

KIRILLOV, A. *AForge.NET framework*: Detecting some simple shapes in images. 2010. Disponível em: ⟨http://www.aforgenet.com/articles/shape_checker/⟩. Acesso em: 02 março 2015. Citado na página 19.

KOHLER, S. lejos ev3 wiki: Uart sensor protocol. Sourcefourge, fevereiro 2015. Disponível em: ⟨http://sourceforge.net/p/lejos/wiki/UART%20Sensor%20Protocol/⟩. Acesso em: 30 setembro 2015. Citado 3 vezes nas páginas 33, 92 e 93.

LEGO. *LEGO MINDSTORMS Education NXT User Guide*. [S.l.], 2006. Disponível em: ⟨http://cache.lego.com/downloads/education/9797_LME_UserGuide_US_low.pdf⟩. Acesso em: 02 março 2015. Citado na página 16.

LEGO. *Creating Blocks for LEGO Mindstorms EV3*. [S.l.], 2013. Disponível em: ⟨http://www.lego.com/en-us/mindstorms/downloads⟩. Acesso em: 25 setembro 2015. Citado 2 vezes nas páginas 29 e 30.

LEGO. *Inanimate Reason*: Lego® education evolves stem learning with the next generation lego mindstorms® education ev3 platform. The LEGO Group, 2013. Disponível em: ⟨http://inanimatereason.com/blog/2013/01/lego-education-evolves-stem-learning-with-the-next-generation-lego-mindstorms-education-ev3-platfo⟩. Acesso em: 22 abril 2015. Citado na página 23.

LEGO. *LEGO MINDSTORMS EV3*: History of lego robotics. The LEGO Group, 2013. Institutional website. Disponível em: ⟨http://www.lego.com/es-es/mindstorms/history⟩. Acesso em: 02 março 2015. Citado na página 15.

LEGO. *LEGO MINDSTORMS EV3 - Hardware Developer Kit*. [S.l.], 2013. Disponível em: ⟨http://www.lego.com/en-us/mindstorms/downloads⟩. Acesso em: 25 setembro 2015. Citado 4 vezes nas páginas 24, 32, 33 e 35.

LEGO. *LEGO MINDSTORMS EV3 - User Guide*. [S.l.], 2013. Disponível em: ⟨http://www.lego.com/en-us/mindstorms/downloads⟩. Acesso em: 25 setembro 2015. Citado 2 vezes nas páginas 24 e 29.

LEGO. *LEGO MINDSTORMS EV3 User Guide*. [S.l.], 2013. Disponível em: ⟨http://www.lego.com/en-us/mindstorms/products/31313-mindstorms-ev3⟩. Acesso em: 02 março 2015. Citado na página 16.

MAYNES-AMINZADE, D.; WINOGRAD, T. I. T. Eyepatch: prototyping camera-based interaction through examples. In: *UIST '07 Proceedings of the 20th annual ACM symposium on User interface software and technology*. [S.l.: s.n.], 2007. p. 33–42. Citado na página 17.

MINDSENSORS (Ed.). *NXTCam v4 User Guide*. [S.l.], 2014. Disponível em: ⟨www.mindsensors.com/index.php?module=documents&JAS_DocumentManager_op=downloadFile&JAS_File_id=1365⟩. Citado na página 20.

MINDSENSORS.COM (Ed.). Página Web Institucional, *Mindsensors.com*. 2005–2015. Disponível em: ⟨https://www.mindsensors.com/⟩. Acesso em: 25 setembro 2015. Citado na página 16.

MORAL, J. A. B. *Develop leJOS programs Step by Step*. [s.n.], 2008. Disponível em: ⟨http://www.juanantonio.info/lejos-ebook/⟩. Citado 2 vezes nas páginas 19 e 20.

MORTENSEN, T. F. The lego history: The lego group history. The LEGO Group, janeiro 2012. Disponível em: ⟨http://www.lego.com/en-us/aboutus/lego-group/the_lego_history⟩. Acesso em: 25 setembro 2015. Citado na página 15.

OPENCV (Ed.). Página Web Institucional, *OpenCV.org*. 2015. Disponível em: ⟨https://www.opencv.org/⟩. Acesso em: 07 outubro 2015. Citado na página 37.

ORLANDO, J. R. *AVRcam User's Manual*. [S.l.], 2004. Disponível em: ⟨http://www.jrobot.net/Download/AVRcam_Users_Manual_v1_4.pdf⟩. Acesso em: 02 março 2015. Citado na página 20.

PET Mecatrônica - Automação e Sistemas: Sobre o workshop de robótica para alunos do 1o ano da epusp. 2015. Disponível em: ⟨http://sites.poli.usp.br/pmr/pet/projetos_workshoprobo.asp⟩. Acesso em: 18 novembro 2015. Citado na página 13.

STEVENSON, D. E.; SCHWARZMEIER, J. D. Building an autonomous vehicle by integrating lego mindstorms and a web cam. In: *SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education*. [S.l.: s.n.], 2007. p. 165–169. Citado na página 18.

TRUNG, P.; AFZULPURKAR, N.; BODHALE, D. Development of vision service in robotics studio for road signs recognition and control of lego mindstorms robot. In: *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*. [S.l.: s.n.], 2009. p. 1176–1181. Citado 2 vezes nas páginas 17 e 19.

ŠULIGOJ, F. et al. Object tracking with a multiagent robot system and a stereo vision camera. In: *24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013*. [S.l.: s.n.], 2013. p. 968–973. Citado na página 17.

VALK, L. *Robot Square*: Ev3 and nxt: Differences and compatibility. 2013. Disponível em: ⟨http://robotsquare.com/2013/07/16/ev3-nxt-compatibility/⟩. Acesso em: 02 março 2015. Citado na página 15.

VIOLA P. ; MITSUBISHI ELECTR. RES. LABS., C. M. U. . J. M. Rapid object detection using a boosted cascade of simple features. Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on (Volume:1), 2001. Acesso em: 21 junho 2015. Citado na página 44.

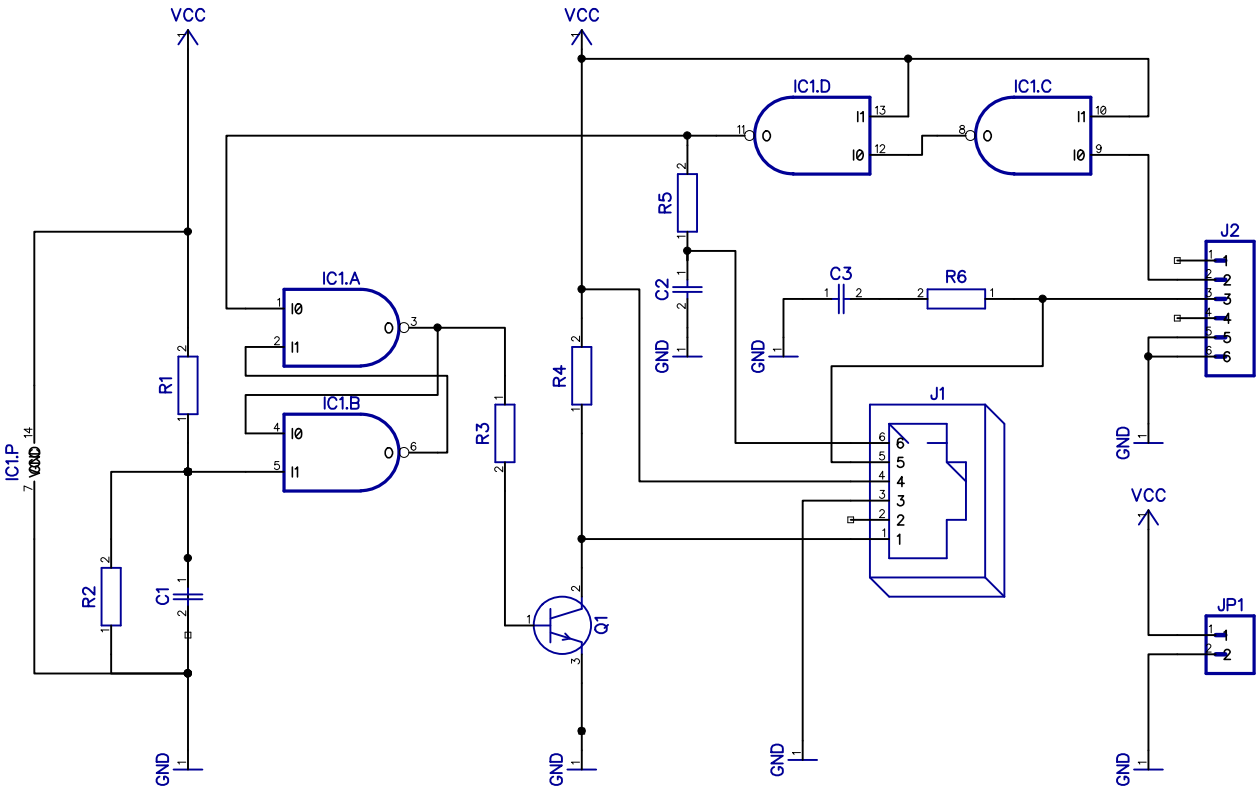
ZHENJUN, L.; NISAR, H.; MALIK, A. A framework for real time indoor robot navigation using monte carlo localization and orb feature detection. In: *Consumer Electronics (ISCE 2014), The 18th IEEE International Symposium on*. [S.l.: s.n.], 2014. p. 1–2. Citado na página 19.

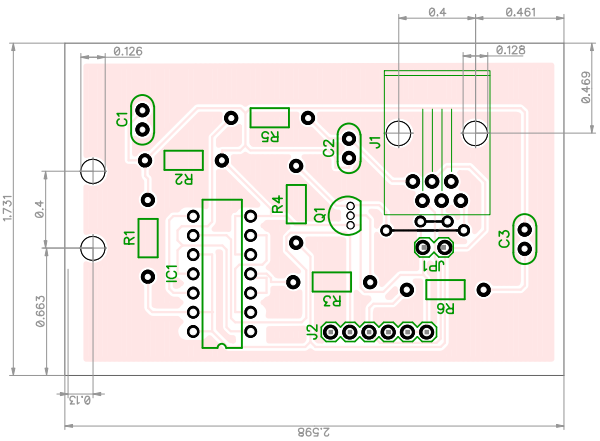
Apêndices

APÊNDICE A – DESENHOS DE FABRICAÇÃO ELETRÔNICA DA PLACA DE CIRCUITO GLUE LOGIC

Tabela 2: Lista de componentes da placa de circuito Glue Logic.

Componente	**Valor**	**Quantidade**	**Descrição**
IC1	7400	1	Quadriple 2-Input Positive-NAND Gates
Q1	2N3904	1	Transistor NPN
C1	470nF	1	Capacitor cerâmico
C2	330pF	1	Capacitor cerâmico
C3	10mF	1	Capacitor eletrolítico
R1	10KΩ	1	Resistor
R2,4	100KΩ	2	Resistor
R3	22KΩ	1	Resistor
R6	2.2KΩ	1	Resistor
R5	220Ω	1	Resistor
J1	RJ12	1	Conector fêmea RJ12 com ajuste direito
J2	-	1	Conector fêmea de 6 fios

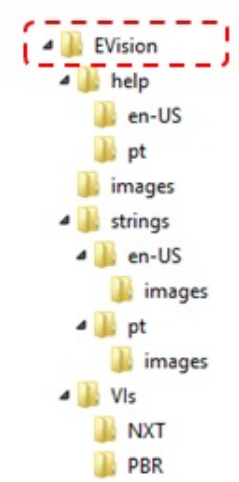

VCC
VCC
IC1.D
IC1.C
IC1.A
IC1.B
IC1.P
R1
R2
R3
R4
R5
R6
C1
C2
C3
Q1
J1
J2
JP1
GND
GND
GND
GND
GND
GND


0.4
0.461
0.126
0.128
0.469
1.731
0.4
0.663
0.13
2.598
C1
R5
C2
J1
R2
R4
Q1
R1
IC1
C3
JP1
R3
R6
J2

APÊNDICE B – ARQUIVOS DO BLOCO EVISION.EV3B

A seguir serão apresentados os arquivos presentes no bloco *EVision.ev3b*.

B.1 /EVision/



EVision é o diretório principal deste bloco. Nele constam todos os outros diretórios, além do arquivo principal *blocks.xml*.

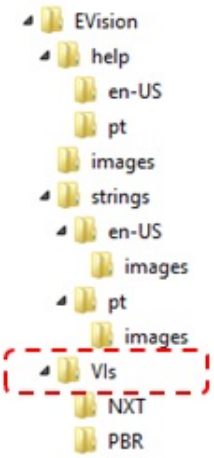
B.1.1 /EVision/blocks.xml

```
<?xml version="1.0" encoding="utf-8"?>
<EditorDefinitions>
  <PolyGroups ModuleName="EVision" ModuleVersion="1.00">
   <PolyGroup Name="EVision" BlockFamily="Sensor">
     <Parameter Name="Direction" Direction="Input" DataType="UInt32"
        DefaultValue="2" Configuration="Identification_WaitForChange.xml"
        Identification="Identification_WaitForChange.xml" />
     <Parameter Name="Port" CompilerDirectives="OneInputPort"
        Direction="Input" DefaultValue="1.4" />
     <Parameter Name="PositionX" DataType="Single" Direction="Output"
        Identification="Identification_PositionX.png" />
     <Parameter Name="AreaSize" DataType="Single" Direction="Output"
        Identification="Identification_AreaSize.png" />
```

```
<Parameter Name="Faces" DataType="Single" Direction="Output"
    Identification="Identification_Faces.png" />
<Parameter Name="PositionY" DataType="Single" Direction="Output"
    Identification="Identification_PositionY.png" />
<Parameter Name="Color" DataType="Single" Direction="Input"
    DefaultValue="1" Identification="Identification_SetOfColors.xml"
    Configuration="Identification_SetOfColors.xml" />
<Parameter Name="Shape" DataType="Single" Direction="Input"
    DefaultValue="1" Identification="Identification_SetOfShapes.xml"
    Configuration="Identification_SetOfShapes.xml" />
<Hardware>
 <EV3PlotColor>#ff5d5d5d</EV3PlotColor>
 <EV3AutoID>66</EV3AutoID>
 <Direction>Input</Direction>
 <DefaultPort>1.4</DefaultPort>
</Hardware>
<Block>
 <Mode>MeasureColorSeeker</Mode>
 <Reference Type="VILib" Name="EVColor.vix" />
 <PaletteInfo Weight="0.5" />
 <ParameterReference Name="Port" />
 <ParameterReference Name="Color" />
 <ParameterReference Name="PositionX" CompilerDirectives="Result"/>
 <ParameterReference Name="AreaSize" CompilerDirectives="Result"/>
 <BlockInterface>Selector</BlockInterface>
 <Flags>PBROnly</Flags>
 <Hardware>RudolphEV</Hardware>
 <HardwareModeInfo Name="EV-COL" ID="0" Range="0,100" Unit="%" />
</Block>
<Block>
 <Mode>MeasureShapeSeeker</Mode>
 <Reference Type="VILib" Name="EVShape.vix" />
 <ParameterReference Name="Port" />
 <ParameterReference Name="Shape" />
 <ParameterReference Name="PositionX" CompilerDirectives="Result"/>
 <ParameterReference Name="AreaSize" CompilerDirectives="Result"/>
 <BlockInterface>Selector</BlockInterface>
 <Flags>PBROnly</Flags>
 <Hardware>RudolphEV</Hardware>
 <HardwareModeInfo Name="EV-SHP" ID="1" Range="0,100" Unit="%" />
</Block>
<Block>
```

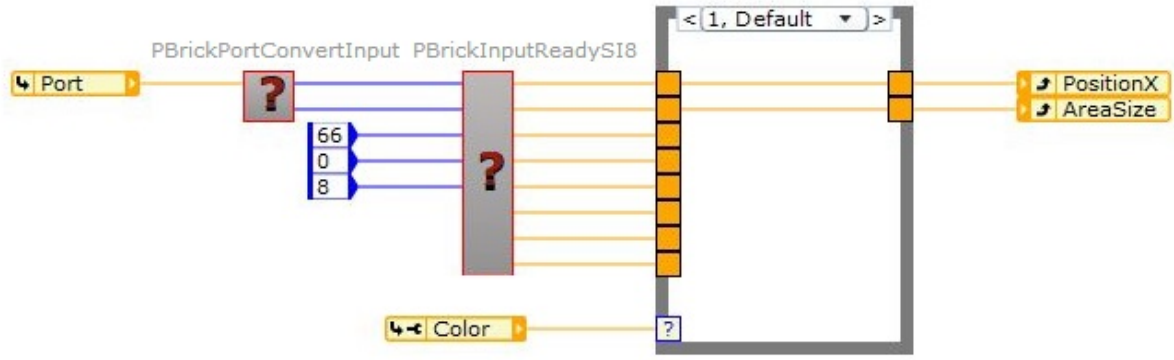
```
      <Mode>MeasureFaceSeeker</Mode>
      <Reference Type="VILib" Name="EVFace.vix" />
      <ParameterReference Name="Port" />
      <ParameterReference Name="Faces" CompilerDirectives="Result" />
      <ParameterReference Name="AreaSize" CompilerDirectives="Result" />
      <ParameterReference Name="PositionX" CompilerDirectives="Result"/>
      <ParameterReference Name="PositionY" CompilerDirectives="Result"/>
      <BlockInterface>Selector</BlockInterface>
      <Flags>PBROnly</Flags>
      <Hardware>RudolphEV</Hardware>
      <HardwareModeInfo Name="EV-FACE" ID="2" Range="0,10" Unit="%" />
     </Block>
   </PolyGroup>
  </PolyGroups>
  <Hardwares>
   <Hardware>
     <Name>RudolphEV</Name>
     <Label>EV</Label>
     <Target>PBR</Target>
     <Direction>Input</Direction>
     <AutoID>66</AutoID>
     <HardwareIcon Path="Hardware_PBR_EV.png" />
     <!-- Green -->
     <ColorHEX>#008000</ColorHEX>
     <RangeScale>0,1000</RangeScale>
     <Mode Name="EV-Col" ID="0" Unit="%" />
     <Mode Name="EV-Shp" ID="1" Unit="%" />
     <Mode Name="EV-Face" ID="2" Unit="%" />
   </Hardware>
  </Hardwares>
</EditorDefinitions>
```

B.2 /EVision/VIs/



VIs é o diretório que contém os códigos *.vix* que controlam o bloco. Nele encontram-se todos os códigos que são comuns tanto ao NXT quanto ao EV3, além dos diretórios **NXT** e **PBR** os quais contém os códigos específicos a cada um dos blocos inteligentes. Como a adaptção do bloco para o NXT não faz parte do escopo deste projeto, o diretório **NXT** se encontra vazio. O diretório **PBR**, portanto, contém todos os arquivos *.vix* referentes ao EV3.

B.2.1 /EVision/VIs/PBR/EVColor.vix



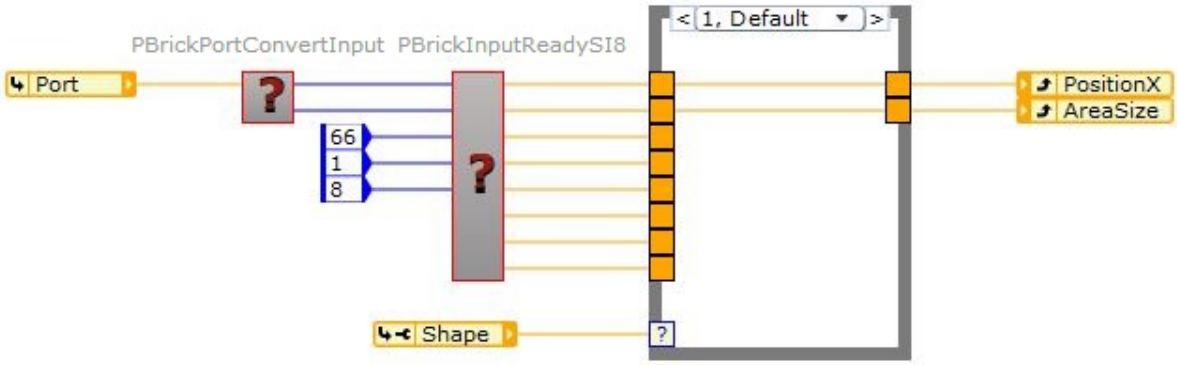
O arquivo *EVColor.vix* é o código responsável pelo modo de identificação de cores, chamado **MeasureColorSeeker**. Como dito anteriormente, sua programação é realizada com o auxílio de blocos de mais baixo nível disponíveis no ambiente de programação específico para a criação de novos blocos da LEGO. Os "gray blobs"são os blocos cinzas com o ponto de interrogação vermelho.

Nesta função, o parâmetro **Port** referencia a porta selecionada. Seus parâmetros são recuperados pelo bloco cinza **PBrickPortConvertInput**. Estes são usados como

entrada para o bloco cinza **PBrickInputReadySI8** que é o responsável pela recuperação das informações enviadas pelo sensor. Os demais parâmetros de entrada deste bloco são: o identificador do sensor (**66**); o modo ao qual as informações recuperadas pertencem (**0**); e o número de informações enviadas pelo sensor, que serão recuperadas nas saídas do bloco (**8**).

Sabe-se que, no modo de identificação de cores, o módulo de visão envia ao EV3 as posições horizontais e o tamanho dos maiores objetos nas cores vermelha, azul e verde. Cabe, então, ao bloco de casos (**Case**) a seleção da informação que estará disponível nos parâmetros de saída **XPosition** e **AreaSize**. A seleção é feita em função do valor do parâmetro de entrada **Color**. Desta forma, o usuário é capaz de filtrar a informação disponível nas saídas do bloco.

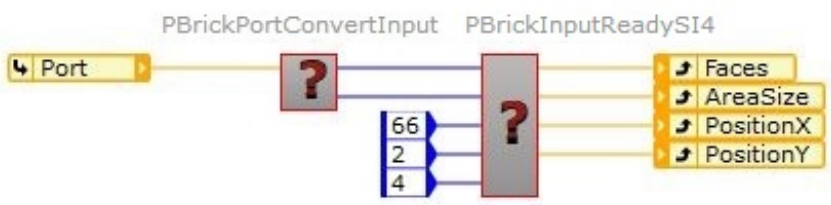
B.2.2 /EVision/VIs/PBR/EVShape.vix



O arquivo *EVShape.vix* é o código responsável pelo modo de identificação de formas, chamado **MeasureShapeSeeker**. A estrutura desta função é idêntica àquela apresentada em *EVColor.vix*. Isso porque, de maneira análoga ao modo de identificação de cores, o modo de identificação de formas é implentado de forma que o módulo de visão envia ao EV3 as posições horizontais e o tamanho dos maiores objetos nas formas retangular, circular e triangular.

Nota-se, porém, algumas diferenças nos seguintes parâmetros: no modo ao qual as informações recuperadas pertencem (**1**) e no parâmetro **Shape** usado na seleção das informações disponíveis nos parâmetros de saída do bloco.

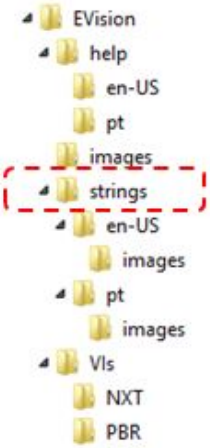
B.2.3 /EVision/VIs/PBR/EVFace.vix



O arquivo *EVFace.vix* é o código responsável pelo modo de identificação de faces, chamado **MeasureFaceSeeker**. Esta função um "gray blob"diferente para a recuperação doas informações enviadas pelo sensor: o **PBrickInputReadySI4**. Os parâmetros de entrada deste bloco são os mesmos: o identificador do sensor (**66**); o modo ao qual as informações recuperadas pertencem (**2**); e o número de informações enviadas pelo sensor(**4**).

Nesse modo, sabe-se que o módulo de visão envia ao EV3 o número de faces detectadas, a posição horizontal, vertical e o tamanho da maior face detectada. Diferentemente das outras funções, não é necessária a filtragem das informações enviadas, pois estas pertencem aos parâmetros de saída **Faces**, **XPosition**, **YPostion** e **AreSize**, respectivamente.

B.3 /EVision/strings/



strings é o diretório que contém os subdiretórios **en-US** e **pt** responsáveis pela exibição das informações dos parâmetros do bloco em inglês e em português, respectivamente. Cada um desses subdiretórios contém um arquivo *blocks.xml* e um diretório **images** com os arquivos *Identification_SetOfColors.xml* e *Identification_SetOfShapes.xml*.

B.3.1 /EVision/strings/en-US/blocks.xml

```
<?xml version="1.0" encoding="utf-8"?>
<EditorStrings>
  <PolyGroups ModuleName="EVision">
   <PolyGroup Name="EVision" DisplayName="EVision Sensor"
       DisplayNamePrefix="EVision">
     <Description><![CDATA[<p>Context help for PolyGroup <b>EVision
        Sensor</b></p>]]></Description>
     <Parameter Name="Direction" DisplayName="Direction"
        Link="page.html?Path=blocks%2FLEGO%2FWait.html#Direction">
      <Description><![CDATA[<p><b>Type:</b> Numeric<br/><b>Notes:</b>
         Direction for a Numeric sensor value to change.<br/>Used in Sensor
         Change Modes that have an Amount input.<br/>0 = Increase<br/>1 =
         Decrease<br/>2 = Any</p>]]></Description>
     </Parameter>
     <Parameter Name="Port" DisplayName="Port"
        Link="page.html?Path=editor%2FPortSelector.html#Port">
      <Description><![CDATA[Many programming blocks require that you select
         the ports on the EV3 Brick (A, B, C, D, 1, 2, 3, and 4) that these
         blocks will use. The Port Selectors are in the top right-hand corner
         of these blocks.]]></Description>
     </Parameter>
     <Parameter Name="PositionX" DisplayName="X Position">
      <Description><![CDATA[<p><b>Type:</b> Numeric<br/><b>Values:</b> 0 to
         100<br/><b>Notes:</b> The biggest object position on X axis. 0 means
         to the left, and 100 means to the right. The X Position will be 0 if
         the object is not detected at all.</p>]]></Description>
     </Parameter>
     <Parameter Name="AreaSize" DisplayName="Area">
      <Description><![CDATA[<p><b>Type:</b> Numeric<br/><b>Values:</b> 0 to
         100<br/><b>Notes:</b> The biggest object proximity. 0 means far away,
         and 100 means very close. The Area will be 0 if the object is not
         detected at all.</p>]]></Description>
     </Parameter>
     <Parameter Name="Color" DisplayName="Color">
      <Description><![CDATA[<p><b>Type:</b> Numeric<br/><b>Allowed Values:</b>
         1 - 3<br/><b>Notes:</b> The color parameter on the EVision Sensor to
         detect.</p>]]></Description>
     </Parameter>
     <Parameter Name="Shape" DisplayName="Shape">
      <Description><![CDATA[<p><b>Type:</b> Numeric<br/><b>Allowed Values:</b>
         1 - 3<br/><b>Notes:</b> The shape parameter on the EVision Sensor to
```

```
        detect.</p>]]></Description>
    </Parameter>
    <Parameter Name="Faces" DisplayName="# of faces">
     <Description><![CDATA[<p><b>Type:</b> Numeric<br/><b>Values:</b> 0 to
        infinite<br/><b>Notes:</b> The number of faces
        detected.</p>]]></Description>
    </Parameter>
    <Parameter Name="PositionY" DisplayName="Y Position">
     <Description><![CDATA[<p><b>Type:</b> Numeric<br/><b>Values:</b> 0 to
        100<br/><b>Notes:</b> The object position on Y axis. 0 means up, and
        100 means down. The Y Position will be 0 if the object is not
        detected at all.</p>]]></Description>
    </Parameter>
    <Block Mode="MeasureColorSeeker" DisplayName="Color Seeker">
     <Description><![CDATA[The Measure - Color mode uses the EVision Sensor
        in Color mode. Set the Channel to the color that you want to detect.
        The object parameters is output in PositionX and AreaSize. If the
        object is not detected, PositionX will be 0, and AreaSize will be
        0.]]></Description>
    </Block>
    <Block Mode="MeasureShapeSeeker" DisplayName="Shape Seeker">
     <Description><![CDATA[The Measure - Shape mode uses the EVision Sensor
        in Shape mode. Set the Channel to the shape that you want to detect.
        The object parameters is output in PositionX and AreaSize. If the
        object is not detected, PositionX will be 0, and AreaSize will be
        0.]]></Description>
    </Block>
    <Block Mode="MeasureFaceSeeker" DisplayName="Face Seeker">
     <Description><![CDATA[The Measure - Face mode uses the EVision Sensor in
        Face mode. The face parameters is output in Faces, AreaSize,
        PositionX and PositionY. If the object is not detected, Faces will be
        0, PositionX and PositionY will be 0, and AreaSize will be
        0.]]></Description>
    </Block>
   </PolyGroup>
  </PolyGroups>
</EditorStrings>
```

B.3.2 /EVision/strings/en-US/images/Identification_SetOfColors.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<Definition>
  <Entry Name="RedColor" DisplayName="Red" />
  <Entry Name="BlueColor" DisplayName="Blue" />
  <Entry Name="GreenColor" DisplayName="Green" />
</Definition>
```

B.3.3 /EVision/strings/en-US/images/Identification_SetOfShapes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Definition>
  <Entry Name="SquareShape" DisplayName="Square" />
  <Entry Name="CircleShape" DisplayName="Circle" />
  <Entry Name="TriangleShape" DisplayName="Triangle" />
</Definition>
```

B.3.4 /EVision/strings/pt/blocks.xml

```
<?xml version="1.0" encoding="utf-8"?>
<EditorStrings>
  <PolyGroups ModuleName="EVision">
   <PolyGroup Name="EVision" DisplayName="Sensor EVision"
       DisplayNamePrefix="EVision">
     <Description><![CDATA[<p>Context help for PolyGroup <b>EVision
        Sensor</b></p>]]></Description>
     <Parameter Name="Direction" DisplayName="Direcao"
        Link="page.html?Path=blocks%2FLEGO%2FWait.html#Direction">
      <Description><![CDATA[<p><b>Tipo:</b> Numerico<br/><b>Observacoes:</b>
          Direcao para um valor de sensor Numerico para mudar.<br/>Usado em
          Modos de alteracao do sensor que possuem uma entrada de
          Quantia.<br/>0 = Aumentar<br/>1 = Diminuir<br/>2 =
          Indiferente</p>]]></Description>
     </Parameter>
     <Parameter Name="Port" DisplayName="Porta"
        Link="page.html?Path=editor%2FPortSelector.html#Port">
      <Description><![CDATA[Muitos blocos de programacao exigem que voce
          selecione portas no Bloco EV3 (A, B, C, D, 1, 2, 3 e 4) que estes
          blocos usarao. Os seletores de porta estao no canto superior direito
          destes blocos.]]></Description>
     </Parameter>
     <Parameter Name="PositionX" DisplayName="Posicao X">
```

```
 <Description><![CDATA[<p><b>Tipo:</b> Numerico<br/><b>Valores:</b> 0 a
    100<br/><b>Observacoes:</b> A posicao no eixo horizontal (X) do maior
    objeto encontrado. 0 significa mais a esquerda e 100 significa mais a
    direita. A Posicao X sera 0 caso nenhum objeto seja
    identificado.</p>]]></Description>
</Parameter>
<Parameter Name="AreaSize" DisplayName="Area">
 <Description><![CDATA[<p><b>Tipo:</b> Numerico<br/><b>Valores:</b> 0 a
    100<br/><b>Observacoes:</b> A proximidade do maior objeto encontrado.
    0 significa mais afastado e 100 significa bem proximo. A Area sera 0
    caso nenhum objeto seja identificado.</p>]]></Description>
</Parameter>
<Parameter Name="Color" DisplayName="Cor">
 <Description><![CDATA[<p><b>Tipo:</b> Numerico<br/><b>Valores
    permitidos:</b> 1 - 3<br/><b>Observacoes:</b> Selecao da cor a ser
    identificada pelo Sensor EVIsion.</p>]]></Description>
</Parameter>
<Parameter Name="Shape" DisplayName="Forma">
 <Description><![CDATA[<p><b>Tipo:</b> Numerico<br/><b>Valores
    permitidos:</b> 1 - 3<br/><b>Observacoes:</b> Selecao da forma a ser
    identificada pelo Sensor EVision.</p>]]></Description>
</Parameter>
<Parameter Name="Faces" DisplayName="# de faces">
 <Description><![CDATA[<p><b>Tipo:</b> Numerico<br/><b>Valores:</b> 0 a
    infinito<br/><b>Observacoes:</b> O numero de faces
    detectadas.</p>]]></Description>
</Parameter>
<Parameter Name="PositionY" DisplayName="Posicao Y">
 <Description><![CDATA[<p><b>Tipo:</b> Numerico<br/><b>Valores:</b> 0 a
    100<br/><b>Observacoes:</b> A posicao no eixo vertical (Y) do maior
    objeto encontrado. 0 significa mais acima e 100 significa mais
    abaixo. A Posicao Y sera 0 caso nenhum objeto seja
    identificado.</p>]]></Description>
</Parameter>
<Block Mode="MeasureColorSeeker" DisplayName="Identificacao de Cores">
 <Description><![CDATA[O modo Medida - Identificacao de Cores usa o
    Sensor EVision no modo de cor. Defina o parametro Cor na cor que voce
    deseja identificar. Os parametros do objeto sao extraidos em Posicao
    X e Area. Se nenhum objeto e identificado, Posicao X sera 0 e Area
    sera 0.]]></Description>
</Block>
<Block Mode="MeasureShapeSeeker" DisplayName="Identificacao de Formas">
```

```
      <Description><![CDATA[O modo Medida - Identificacao de Formas usa o
          Sensor EVision no modo de forma. Defina o parametro Forma na form que
          voce deseja identificar. Os parametros do objeto sao extraidos em
          Posicao X e Area. Se nenhum objeto e identificado, Posicao X sera 0 e
          Area sera 0.]]></Description>
     </Block>
     <Block Mode="MeasureFaceSeeker" DisplayName="Identificacao de Faces">
      <Description><![CDATA[O modo Medida - Identificacao de Faces usa o
          Sensor EVision no modo de faces. Os parametros da face sao extraidos
          em Faces, Area, Posicao X e Posicao Y. Se nenhum objeto e
          identificado, Faces sera 0, Posicao X e Posicao Y serao 0, e Area
          sera 0.]]></Description>
     </Block>
    </PolyGroup>
   </PolyGroups>
</EditorStrings>
```

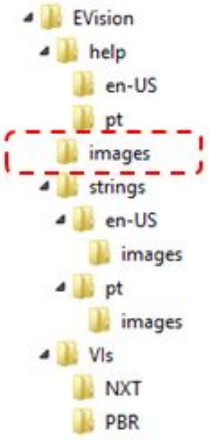
B.3.5 /EVision/strings/pt/images/Identification_SetOfColors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Definition>
  <Entry Name="RedColor" DisplayName="Vermelho" />
  <Entry Name="BlueColor" DisplayName="Azul" />
  <Entry Name="GreenColor" DisplayName="Verde" />
</Definition>
```

B.3.6 /EVision/strings/pt/images/Identification_SetOfShapes.xml

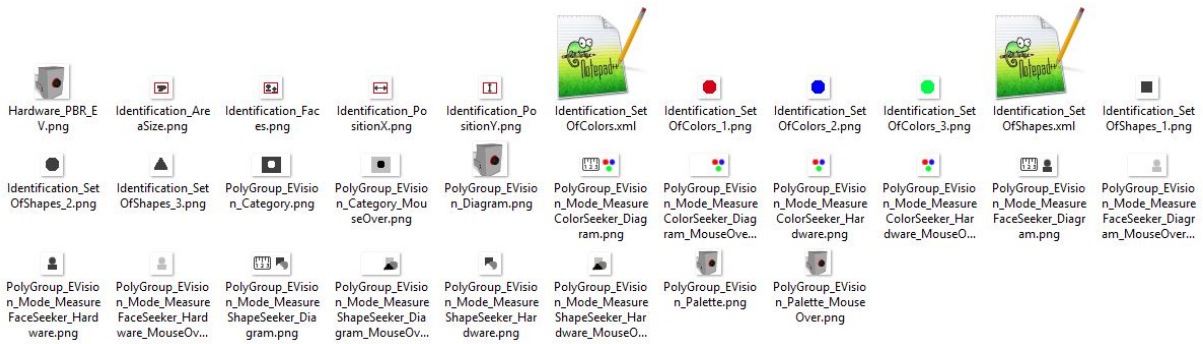
```
<?xml version="1.0" encoding="utf-8"?>
<Definition>
  <Entry Name="SquareShape" DisplayName="Retangulo" />
  <Entry Name="CircleShape" DisplayName="Circulo" />
  <Entry Name="TriangleShape" DisplayName="Triangulo" />
</Definition>
```

B.4 /EVision/images/



images é o diretório que contém todas as imagens referentes ao bloco. Cada imagens possui um tamanho e um nome específicos a serem seguidos, de tal forma que:

- a imagem da palheta deve ter **20x20** pixeus e seu nome deve ser **PolyGroup**_<*PolyGroup-Name*>_**Palette.png**;
- a imagem do diagrama deve ter **34x34** e seu nome deve ser **PolyGroup**_<*PolyGroup-Name*>_**Diagram.png**;
- as imagens dos modos devem ter **38x22** e seu nome deve ser **PolyGroup**_<*PolyGroup-Name*>_**Mode**_<*PolyGroup-Name*>_**Diagram.png**;
- as imagens de hardware devem ter **38x22** e **22x22** e seus nomes devem ser **PolyGroup**_<*PolyGroup-Name*>_**Category.png** e **PolyGroup**_<*PolyGroup-Name* >_**Mode**_<*PolyGroup-Name*>_**Hardware.png** para o bloco e os modos, respectivamente;
- as imagens de identificação dos parâmetros do bloco devem ter **22x22** e seu nome deve ser **Identification**_<*ParameterName*>**.png**.



Além das imagens, os arquivos *.xml* de identificação de parâmetros também devem estar neste diretório. Assim, no módulo de visão, **imagens** também possui os arquivos *Identification_SetOfColors.xml* e *Identification_SetOfShapes.xml*.

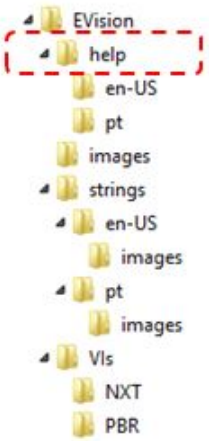
B.4.1 /EVision/images/Identification_SetOfColors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Definition Type="Integer">
  <Point Value="1" Name="RedColor" ImageSuffix="_1" />
  <Point Value="2" Name="BlueColor" ImageSuffix="_2" />
  <Point Value="3" Name="GreenColor" ImageSuffix="_3" />
</Definition>
```

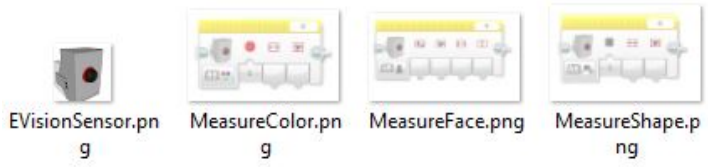
B.4.2 /EVision/images/Identification_SetOfShapes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Definition Type="Integer">
  <Point Value="1" Name="SquareShape" ImageSuffix="_1" />
  <Point Value="2" Name="CircleShape" ImageSuffix="_2" />
  <Point Value="3" Name="TriangleShape" ImageSuffix="_3" />
</Definition>
```

B.5 /EVision/help/



help é o diretório que contém os subdiretórios **en-US** e **pt** responsáveis pelo material de apoio à utilização do bloco em inglês e português, respectivamente. Cada um desses subdiretórios contém um arquivo *EVisionSensor.html* e as imagens utilizadas por pelo mesmo:



B.5.1 /EVision/help/en-US/EVisionSensor.html

```
<html>
<head>
<title>EVision Sensor Block</title>
</head>
<body BGCOLOR="FFFFFF">
<h1>EVision Sensor Block</h1>
<p>This block is the result of the final paper of the Mechatronics Engineer
    undergraduate course at Escola Polit&eacute;cnica da Universidade de
    S&atilde;o Paulo, written by Amanda Fernandes and Renan Marchetto and
    supervised by Prof. Dr. Thiago Martins.</p>
<img src="EVisionSensor.png" width="68" height="68" />
<p> At runtime, the EVision Sensor provides specific parameters of the
    selected image treatment.
<br><br>
<span style="color: rgb(255, 0, 0);"><b> The sensor takes about 30 seconds to
    initialize.</b></span>
<h2>Modes</h2>
<a name="Mode_MeasureColorSeeker"></a>
<h3>Color Seeker</h3>
<img src="MeasureColor.png" width="178" height="118" />
<p>
This mode implements a filter for red, blue and green objects.
<br><br>
It has a color selector as an input and horizontal position and size of the
    biggest object in the selected color detected as outputs.
<br><br>
</p>
<a name="Mode_MeasureShapeSeeker"></a>
<h3>Shape Seeker</h3>
<img src="MeasureShape.png" width="198" height="128" />
<p>
This mode implements a filter for rectangular, circular and triangular shaped
    objects.
<br><br>
```

```
It has a shape selector as an input and horizontal position and size of the
    biggest object in the selected shape detected as outputs.
<br><br>
</p>
<a name="Mode_MeasureFaceSeeker"></a>
<h3>Face Seeker</h3>
<img src="MeasureFace.png" width="198" height="128" />
<p>
This mode implements a filter for face detection.
<br><br>
It's outputs are: number of faces detected, horizontal and vertical position
    and size of the biggest face detected.
</p>

<h2>Parameters</h2>
<a name="Parameters"></a>
<h3>Direction, Port, PositionX, AreaSize, Faces, PositionY, Color, Shape</h3>
<p>
All parameters, with exception of "Faces", are given as a percentage of the
    camera resolution (640x480).
</p>
<p>
<b>PositionX</b>, <b>PositionY</b>, <b>AreaSize</b> and <b>Faces</b> are the
    object parameter returned by the sensor.<br>
<b>Color</b> and <b>Shape</b> are the selection parameters for the sensor
    functions.<br>
<b>Direction</b> and <b>Port</b> are standard parameters for a "Measure" block.
</p>
</body>
</html>
```

B.5.2 /EVision/help/pt/EVisionSensor.html

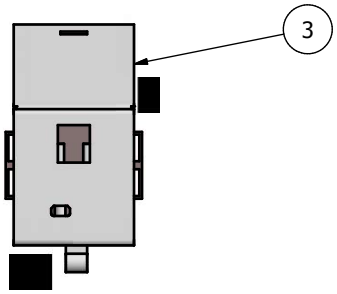
```
<html>
<head>
<title>Bloco do Sensor EVision</title>
</head>
<body BGCOLOR="FFFFFF">
<h1>Bloco do Sensor EVision</h1>
<p>Este bloco &eacute; o resultado do trabalho de conclus&atilde;o de curso de
    Engenharia Mecatr&ocirc;nica na Escola Polit&eacute;cnica da Universidade
```

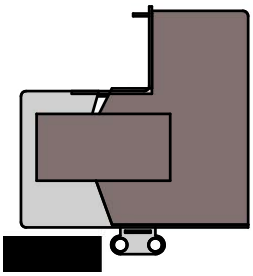
```
    de S&atilde;o Paulo, escrito por Amanda Fernandes e Renan Marchetto e
    orientado pelo Prof. Dr. Thiago Martins.</p>
<img src="EVisionSensor.png" width="68" height="68" />
<p> Durante o seu funcionamento, o Sensor EVision fornece par&acirc;metros
    espec&iacute;ficos do tratamento de imagens selecionado.
<br><br>
<span style="color: rgb(255, 0, 0);"><b> O sensor leva aproximadamente 30s
    para inicializar.</b></span>
<h2>Modos</h2>
<a name="Mode_MeasureColorSeeker"></a>
<h3>Identifica&ccedil;&atilde;o de Cores</h3>
<img src="MeasureColor.png" width="178" height="118" />
<p>
Este modo implementa um filtro para identifica&ccedil;&atilde;o de objetos nas
    cores vermelha, azul e verde.
<br><br>
Ele possui um seletor de cor como par&acirc;metro de entrada e a
    posi&ccedil;&atilde;o horizontal e o tamanho do maior objeto identificado
    na cor selecionada como par&acirc;metros de sa&iacute;da.
<br><br>
</p>
<a name="Mode_MeasureShapeSeeker"></a>
<h3>Identifica&ccedil;&atilde;o de Formas</h3>
<img src="MeasureShape.png" width="198" height="128" />
<p>
Este modo implement um filtro para identifica&ccedil;&atilde;o de objetos nas
    formas retangular, circular e triangular.
<br><br>
Ele possui um seletor de formas como par&acirc;metro de entrada e a
    posi&ccedil;&atilde;o horizontal e o tamanho do maior objeto identificado
    na forma selecionada como par&acirc;metros de sa&iacute;da.
<br><br>
</p>
<a name="Mode_MeasureFaceSeeker"></a>
<h3>Identifica&ccedil;&atilde;o de Faces</h3>
<img src="MeasureFace.png" width="198" height="128" />
<p>
Este modo implementa um filtro para identifica&ccedil;&atilde;o de faces.
<br><br>
Seus par&acirc;metros de sa&iacute;da s&atilde;o: o n&uacute;mero de faces
    identificadas, as posi&ccedil;&otilde;es horizontal e vertical, o tamanho
    da maior face identificada.
```

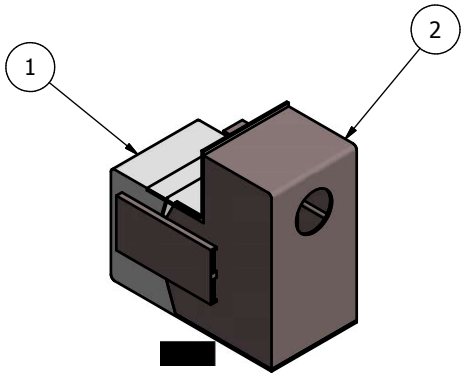
```
<br><br>
</p>

<h2>Par&acirc;metros</h2>
<a name="Parameters"></a>
<h3>Dire&ccedil;&atilde;o, Porta, Posi&ccedil;&atilde;o X, &Aacute;rea, Faces,
   Posi&ccedil;&atilde;o Y, Cor, Forma</h3>
<p>
Todos os par&acirc;metros, exceto "Faces", s&atilde;o dados como uma
   porcentagem da resolu&ccedil;&atilde;o da c&acirc;mera (460x480).
</p>
<p>
<b>Posi&ccedil;&atilde;o X</b>, <b>Posi&ccedil;&atilde;o Y</b>,
   <b>&Aacute;rea</b> e <b>Faces</b> s&atilde;o os par&acirc;metros do objeto
   retornado pelo sensor.<br>
<b>Cor</b> e <b>Forma</b> s&atilde;o os par&acirc;metros de
   sele&ccedil;&atilde;o das fun&ccedil;&otilde;es do sensor.<br>
<b>Dire&ccedil;&atilde;o</b> e <b>Porta</b> s&atilde;o os par&acirc;metros
   padr&atilde;o para o bloco "Medidas".
</p>
</body>
</html>
```

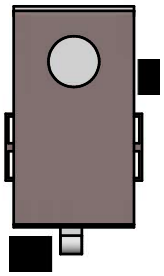
APÊNDICE C – DESENHOS DE FABRICAÇÃO DO INVOLUCRO

PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	Back	Material : ABS
2	1	Front	Material : ABS
3	1	Top	Material : ABS









DRAWN Amanda/Renan	19/11/2015		
CHECKED		TITLE	
QA		Involucro Sensor	
MFG			
APPROVED			
		SIZE A4	DWG NO Assembly1-1 REV
		SCALE 1 / 2	SHEET 1 OF 1

APÊNDICE D – PROGRAMAÇÃO (PYTHON)

D.0.3 protocol.py

```
import mraa
import checksum as cs
import time
import serial
import color_tracking as col
import shape_tracking as shp
import face_tracking as face
import cv2
import sys
from multiprocessing import Process, Queue

def uartData (q,t,v):
   msg = 'D808070605040302012F'
   modo = 0
   v.timeout = 1

   while True:
      if not q.empty():
         msg = q.get()

      v.write(bytearray.fromhex(msg))

   info= v.read()

   if info:
      if (info.encode("hex")=='43'):
         info = v.read()

      if (info.encode("hex")=='01') :
         modo = 1
         t.put(modo)

      elif (info.encode("hex")=='02') :
```

```
            modo = 2
            t.put(modo)

    def calculo (q,t):
        capture = cv2.VideoCapture(-1)
        faceCascade =
            cv2.CascadeClassifier('evision/haarcascade_frontalface_default.xml')
        modo = 0

        while True:
            if not t.empty():
                modo = t.get()

            if modo == 0:
                #Color Tracker
                res = col.ColourTrack(capture)
                msg = cs.cksum(res,modo)
            elif modo == 1:
                #Form Tracker'
                res = shp.ShapeTracker(capture)
                msg = cs.cksum(res,modo)
            elif modo == 2:
                #Face Tracker
                res = face.FaceTracker(capture,faceCascade)
                msg = cs.cksum(res,modo)

            q.put(msg)
            time.sleep(0.05)

    def sendprotocol():
        i = 0
        while i <= 8:

        #Turn TX pin into GPIO port
        x=mraa.Gpio(35)
        x.dir(mraa.DIR_OUT)

        #Keep TX > 500 ms as LOW
        x.mode(2)
        time.sleep(0.505)
        x.mode(0)
        #Initialize UART pins
```

```
#ATTENTION: timeout may change as the protocol changes
x = mraa.Uart(0)
u = serial.Serial('/dev/ttyMFD1',2400, timeout = 1.4)

   #Clean RX and TX
   u.flushInput()
   u.flushOutput()

#Start protocol
print "START PROTOCOL"
   msg = bytearray.fromhex('4042FD')
   u.write(msg)
   msg = bytearray.fromhex('490202B6')
   u.write(msg)
   msg = bytearray.fromhex('5200E100004C')
   u.write(msg)
   msg = bytearray.fromhex('A20045562D464143450000000000000000062')
   u.write(msg)
   msg = bytearray.fromhex('9A010000000000C842EE')
   u.write(msg)
   msg = bytearray.fromhex('9A030000000000C842EC')
   u.write(msg)
   msg = bytearray.fromhex('9A047063740000000006')
   u.write(msg)
   msg = bytearray.fromhex('928004000300EA')
   u.write(msg)
   msg = bytearray.fromhex('990045562D534850000013')
   u.write(msg)
   msg = bytearray.fromhex('990100000000000C842ED')
   u.write(msg)
   msg = bytearray.fromhex('990300000000000C842EF')
   u.write(msg)
   msg = bytearray.fromhex('990470637400000000005')
   u.write(msg)
   msg = bytearray.fromhex('918008000300E5')
   u.write(msg)
   msg = bytearray.fromhex('980045562D434F4C000019')
   u.write(msg)
   msg = bytearray.fromhex('980100000000000C842EC')
   u.write(msg)
   msg = bytearray.fromhex('980300000000000C842EE')
   u.write(msg)
```

```
        msg = bytearray.fromhex('9804706374000000000004')
        u.write(msg)
        msg = bytearray.fromhex('908008000300E4')
        u.write(msg)
        msg = bytearray.fromhex('04') #ACK
        u.write(msg)
    print "END PROTOCOL"

    #Wait for the first byte from EV3
    info=u.read()

    print info.encode("hex")

    #Test if ACK
    i = i + 1
    if info:
        if (info.encode("hex")=='04'):
            print "ACK 04"
        break

    #Redefine baudrate
    time.sleep(0.1)
    u.baudrate=57600

    q = Queue()
    t = Queue()

    p = Process(target=uartData, args=(q,t,u,))
    k = Process(target=calculo, args=(q,t,))

    p.start()
    k.start()
```

D.0.4 colortracking.py

```
import cv2, math
import numpy as np

def ColourTrack(capture):
    scale_down = 1
```

```
f, orig_img = capture.read()
orig_img = cv2.flip(orig_img,1)

img = cv2.GaussianBlur(orig_img, (5,5), 0)
img = cv2.cvtColor(orig_img,cv2.COLOR_BGR2HSV)
img = cv2.resize(img, (len(orig_img[0])/ scale_down, len(orig_img)/
    scale_down))

red_lower = np.array([0, 150, 0],np.uint8)
red_upper = np.array([5, 255, 255], np.uint8)
red_binary = cv2.inRange(img, red_lower, red_upper)
blue_lower = np.array([100, 100, 100],np.uint8)
blue_upper = np.array([130, 255, 255], np.uint8)
blue_binary = cv2.inRange(img, blue_lower, blue_upper)
green_lower = np.array([40, 100, 100],np.uint8)
green_upper = np.array([80, 255, 255], np.uint8)
green_binary = cv2.inRange(img, green_lower, green_upper)

dilation = np.ones((15,15),"uint8")
red_binary = cv2.dilate(red_binary, dilation)
blue_binary = cv2.dilate(blue_binary, dilation)
green_binary = cv2.dilate(green_binary, dilation)

contoursr, hierarchyr = cv2.findContours(red_binary, cv2.RETR_LIST,
    cv2.CHAIN_APPROX_SIMPLE)
contoursb, hierarchyb = cv2.findContours(blue_binary, cv2.RETR_LIST,
    cv2.CHAIN_APPROX_SIMPLE)
contoursg, hierarchyg = cv2.findContours(green_binary, cv2.RETR_LIST,
    cv2.CHAIN_APPROX_SIMPLE)

max_area_r = 0
max_area_b = 0
max_area_g = 0

largest_contour_r = None
largest_contour_b = None
largest_contour_g = None

cx_r=0
cx_b=0
cx_g=0
```

```
for idx, contour in enumerate(contoursr):
   area = cv2.contourArea(contour)
   if area>max_area_r:
      max_area_r = area
      largest_contour_r = contour

for idx, contour in enumerate(contoursb):
   area = cv2.contourArea(contour)
   if area>max_area_b:
      max_area_b = area
      largest_contour_b = contour

for idx, contour in enumerate(contoursg):
   area = cv2.contourArea(contour)
   if area>max_area_g:
      max_area_g = area
      largest_contour_g = contour

if not largest_contour_r == None:
   moment_r = cv2.moments(largest_contour_r)
   if moment_r["m00"]>1000/scale_down:
      cx_r=np.int0(moment_r["m10"]/moment_r["m00"])

if not largest_contour_b == None:
   moment_b = cv2.moments(largest_contour_b)
   if moment_b["m00"]>1000/scale_down:
      cx_b=np.int0(moment_b["m10"]/moment_b["m00"])
if not largest_contour_g == None:
   moment_g = cv2.moments(largest_contour_g)
   if moment_g["m00"]>1000/scale_down:
      cx_g=np.int0(moment_g["m10"]/moment_g["m00"])

res=np.int_([cx_r*100/capture.get(3),
    max_area_r*100/(capture.get(3)*capture.get(4)),cx_b*100/capture.get(3),
    max_area_b*100/(capture.get(3)*capture.get(4)),cx_g*100/capture.get(3),
    max_area_g*100/(capture.get(3)*capture.get(4)), 0 , 0])
return res
```

D.0.5 shapetracking.py

```
import cv2, math
import numpy as np

def ShapeTracker(capture):

   scale_down = 1

   f, orig_img = capture.read()
   orig_img = cv2.flip(orig_img,1)

   cimg = cv2.GaussianBlur(orig_img, (5,5), 0)
   cimg = cv2.cvtColor(cimg,cv2.COLOR_BGR2GRAY)
   cimg = cv2.resize(cimg, (len(orig_img[0])/ scale_down, len(orig_img)/
       scale_down))
   cimg = cv2.Canny(cimg,50,190)

   contours,h = cv2.findContours(cimg,cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

   contours_s=[]
   contours_c=[]
   contours_t=[]
   for cnt in contours:
      approx = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)
         if len(approx)==3:
            contours_t=contours_t+[cnt]
         elif len(approx)==4:
            contours_s=contours_s+[cnt]
         if len(approx) > 15:
            contours_c=contours_c+[cnt]

   max_area_s=0
   max_area_c=0
   max_area_t=0
   largest_contour_s=None
   largest_contour_c=None
   largest_contour_t=None
   cx_s=0
   cx_c=0
   cx_t=0

   for idx, contour in enumerate(contours_s):
      area = cv2.contourArea(contour)
```

```
    if area>max_area_s:
       max_area_s = area
       largest_contour_s = contour

 for idx, contour in enumerate(contours_c):
    area = cv2.contourArea(contour)
    if area>max_area_c:
       max_area_c = area
       largest_contour_c = contour

 for idx, contour in enumerate(contours_t):
    area = cv2.contourArea(contour)
    if area>max_area_t:
       max_area_t = area
       largest_contour_t = contour


 if not largest_contour_s == None:
    moment_s = cv2.moments(largest_contour_s)
    if moment_s["m00"]>1000/scale_down:
       cx_s=np.int0(moment_s["m10"]/moment_s["m00"])

 if not largest_contour_c == None:
    moment_c = cv2.moments(largest_contour_c)
    if moment_c["m00"]>1000/scale_down:
       cx_c=np.int0(moment_c["m10"]/moment_c["m00"])

 if not largest_contour_t == None:
 moment_t = cv2.moments(largest_contour_t)
    if moment_t["m00"]>1000/scale_down:
       cx_t=np.int0(moment_t["m10"]/moment_t["m00"])

 res=np.int_([cx_s*100/capture.get(3),
     max_area_s*100/(capture.get(3)*capture.get(4)),cx_c*100/capture.get(3),max_area_c*1
     max_area_t*100/(capture.get(3)*capture.get(4)),0,0])
 return res
```

D.0.6 facetracking.py

```
import cv2
import numpy as np
```

```
import sys

def FaceTracker(capture, faceCascade):

   # Capture frame-by-frame
   ret, image = capture.read()
   gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

   faces = faceCascade.detectMultiScale(
   gray,
   scaleFactor=1.1,
   minNeighbors=5,
   minSize=(30, 30),
   flags=cv2.cv.CV_HAAR_SCALE_IMAGE)

   max_area=0
   largest_x =0
   largest_y =0

   # Rectangle around the faces
   for (x, y, w, h) in faces:
      area = w*h
      if area>max_area:
         max_area = area
         largest_x = x + w/2
         largest_y = y + h/2

   res=np.int_([len(faces),max_area*100/(capture.get(3)*capture.get(4)),largest_x*100/capt
   return res
```

Anexos

ANEXO A – DESCRIÇÃO DAS MENSAGENS ENVIADAS E RECEBIDAS PELO LEGO MINDSTORMS EV3

Tabela 3: Descrição dos bits mais significativos do byte de mensagem. Reproduzido de (KOHLER, 2015).

XX	Descrição
00	Mensagem de sistema (tipo 1)
01	Mensagem de comando (tipo 2)
10	Mensagem de informação (tipo 3)
11	Mensagem de dados (tipo 4)

Tabela 4: Descrição das mensagens de sistema. Reproduzido de (KOHLER, 2015).

Byte	Descrição
0b00000000	SYNC
0b00000010	NACK
0b00000100	ACK
0b00LLL110	ESC

Tabela 5: Descrição das mensagens de comando. Reproduzido de (KOHLER, 2015).

Byte	Payload	Descrição
0b01000000	T	TIPO: tipo do sensor T é o tipo do sensor (número entre 0 e 255)
0b01001001	M,V	MODOS: modos do sensor M+1 e o número de modos suportados (entre 1 e 8) V+1 é o número de modos a serem mostrados (entre 1 e M)
0b01010010	SSSS	VELOCIDADE: máxima taxa de transmissão SSSS é a máxima taxa de transmissão suportada pelo sensor
0b01000011	M	SELEÇÃO: muda o modo do sensor M especifica o modo do sensor desejado
0b01LLL100	<dados>	ESCRITA: envio de dados para o sensor <dados> consiste de 2^{0bLLL} bytes

Tabela 6: Descrição das mensagens de informação. Reproduzido de (KOHLER, 2015).

Mensagem	**Info**	**Payload**	**Descrição**
0b10LLLMMM	0	<string>	NOME: nome do modo 0bMMM <string> é uma string ASCII de tamanho 2^{0bLLL}
0b10011MMM	1	LLLL HHHH	VALBRUTO: gama de leituras brutas do sensor LLLL é o menor valor bruto HHHH é o maior valor bruto
0b10011MMM	2	LLLL HHHH	PCT: gama de leituras em porcentagem LLLL é o valor % correspondente ao menor valor bruto HHHH é o valor % correspondente ao maior valor bruto
0b10011MMM	3	LLLL HHHH	SI: gama de leituras no SI LLLL é o valor SI correspondente ao menor valor bruto HHHH é o valor SI correspondente ao maior valor bruto
0b10LLLMMM	4	<string>	SIMBOLO: nome da unidade no SI <string> é uma string ASCII de tamanho 2^{0bLLL}
0b10010MMM	0x80	S,T,F,D	FORMATO: formato dos dados do sensor no modo 0bMMM S: número de itens (no mínimo 1) T: tipo de dado dos itens (8, 16 ou 32) F: número de dígitos a mostrar (0-15) D: número de decimais a mostrar (0-15)

Tabela 7: Descrição das mensagens de dados. Reproduzido de (KOHLER, 2015).

Mensagem	**Payload**	**Descrição**
0b11LLLMMM	<dados>	DADOS: dados do sensor <dados> contêm as leituras brutas do sensor no modo 0bMMM de tamanho 2^{0bLLL}