UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

"Implementação de sistema robótico autônomo movimentado de acordo com informações visuais"

ANDRÉ FELIPE NUNES TROFINO

São Carlos
Outubro/2014

ANDRÉ FELIPE NUNES TROFINO

IMPLEMENTAÇÃO DE SISTEMA ROBÓTICO AUTÔNOMO MOVIMENTADO DE ACORDO COM INFORMAÇÕES VISUAIS

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

Curso de Engenharia de Computação

Orientador: Evandro L. L. Rodrigues

São Carlos
Outubro/2014

Trofino, André Felipe
T843i Implementação de sistema robótico autônomo movimentado de acordo com informações visuais / André Felipe Trofino; orientador Evandro L. L. Rodrigues. São Carlos, 2014.

Monografia (Graduação em Engenharia de Computação)
-- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Raspberry Pi. 2. Sistemas Embarcados. 3. Processamento de Imagens. I. Título.

FOLHA DE APROVAÇÃO

Nome: André Felipe Nunes Trofino

Título: "Implementação de sistema robótico autônomo movimentado de acordo com informações visuais"

Trabalho de Conclusão de Curso defendido em 18 / 11 / 2014.

Comissão Julgadora:	**Resultado:**
Prof. Associado Evandro Luís Linhari Rodrigues (Orientador) - SEL/EESC/USP	Aprovado
Profa. Associada Simone do Rocio Senger de Souza SSC/ICMC/USP	Aprovado
Prof. Dr. Marcelo Andrade da Costa Vieira SEL/EESC/USP	APROVADO

Coordenador do Curso Interunidades - Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

Dedicatória

Dedico este trabalho aos meus pais, Julio e Cristina, que sempre me apoiaram, sempre acreditaram em mim e me mostraram o caminho correto; à minha irmã, Carol, minha companheira em todas as horas e minha melhor metade, sempre pronta a me aturar, e à minha namorada Ana Carolina, por ser meu suporte, minha ajudante e minha inspiração nesta etapa final.

Agradecimentos

Agradeço a todos meus amigos, que sempre estiveram comigo para compartilhar os momentos bons e os momentos ruins, e me sempre me deram uma razão pra sorrir e meus pais por toda a força e apoio que me proveram durante todos os anos.

Agradeço a minha primeira professora de programação, Simone Senger Souza, cuja bondade e paciência me incentivaram a continuar nesse caminho, e que continuou a me ajudar mesmo depois de não termos mais aulas.

Agradeço também meu professor e orientador Evandro L. L. Rodrigues, que sempre teve tempo para mim, e seus conselhos eu busco até hoje.

“A felicidade está antes na jornada que no destino”

- Stephen King, *A Torre Negra*

Resumo

Os sistemas embarcados comerciais evoluíram muito nos últimos anos, abrindo as possibilidades de aplicações embarcadas. Uma dessas possíveis aplicações é processamento de imagens e visão computacional. Neste trabalho é proposto um sistema de *fetch*; um robô autônomo que detecta a imagem de um círculo e se movimenta seguindo essa imagem, utilizando uma câmera para obter imagens e processamento de imagens para obter as informações necessárias. O objetivo deste trabalho é estudar o desempenho de um sistema embarcado barato e limitado computacionalmente no contexto de processamento de imagens. Foi obtida uma boa precisão em relação à detecção do círculo e movimento do sistema, mostrando que o sistema embarcado em questão é capaz de realizar aplicações de visão computacional, entretanto, a plataforma escolhida não é a melhor escolha devido às suas limitações.

Palavras-Chave: Raspberry Pi, Sistemas Embarcados, Processamento de Imagens.

Abstract

There has been a growth in embedded systems in last years, making them faster and more robust, allowing new possibilities for embedded applications. One of these possible applications is computer vision and image processing. This work proposes a fetch system, an autonomous robot that detects an image of a circle and moves following that image, utilizing a camera to obtain images and image processing to obtain the necessary information. This work's objective is to study a cheap and limited embedded system's performance when used for image processing. A good detection accuracy and movement was obtained, proving that image processing applications are possible using embedded systems, however, the chosen platform is not the best choice, due to its limitations.

Keywords: Raspberry Pi, Embedded Systems, Image Processing.

Sumário

Lista de Abreviaturas:

RAM = Random Access Memory
BIOS = Basic Input/Output System
RGB = Red Green Blue
HSV = Hue Saturation Value
USB = Universal Serial Bus
HD = High Definition
VGA = Video Graphics Array
HDMI = High-Definition Multimedia Interface
GPU = Graphics Processor Unit
GFLOPS = Giga Floating-point Operations Per Second
HDTV = High-Definition Television
SD = Secure Digital
MMC = Multimedia Card
SDIO = Secure Digital Input Output
PID = Proportional-integral-derivative
SDRAM = Synchronous Dynamic Random Access Memory
SSH = Secure Shell

Lista de Figuras

Lista de tabelas

1. Introdução

1.1. Motivação

O processamento de imagens e a visão computacional são duas áreas intimamente ligadas que apresentam uma gama muito grande de aplicações; mas retirar informações de dados visuais (fotos, vídeos) requer bastante poder computacional. Devido a esse requerimento de poder computacional, aplicações de processamento de imagens em sistemas embarcados eram muito limitadas, pois um sistema embarcado tem uma capacidade de processamento muito reduzida se comparado a um computador convencional. Mas com o crescimento dos hardwares embarcados, tornando-se mais poderosos e robustos, o processamento de imagens se torna possível também em sistemas embarcados.

A motivação desse projeto é estudar o poder de um sistema embarcado em relação a processamento de imagens, utilizando um sistema de detecção de imagens, e obter um sistema que pode ser expandido e utilizado em vários contextos e aplicações.

O sistema em questão é um sistema de "*fetch*" (busca), cujo objetivo é identificar um objeto pré-definido e movimentar-se até sua posição. O sistema seria totalmente autônomo, utilizando uma plataforma robótica móvel, um sistema embarcado em conjunto com uma câmera para obter as imagens e controlar todo o processo. As figuras 1 e 2 abaixo representam o sistema proposto.

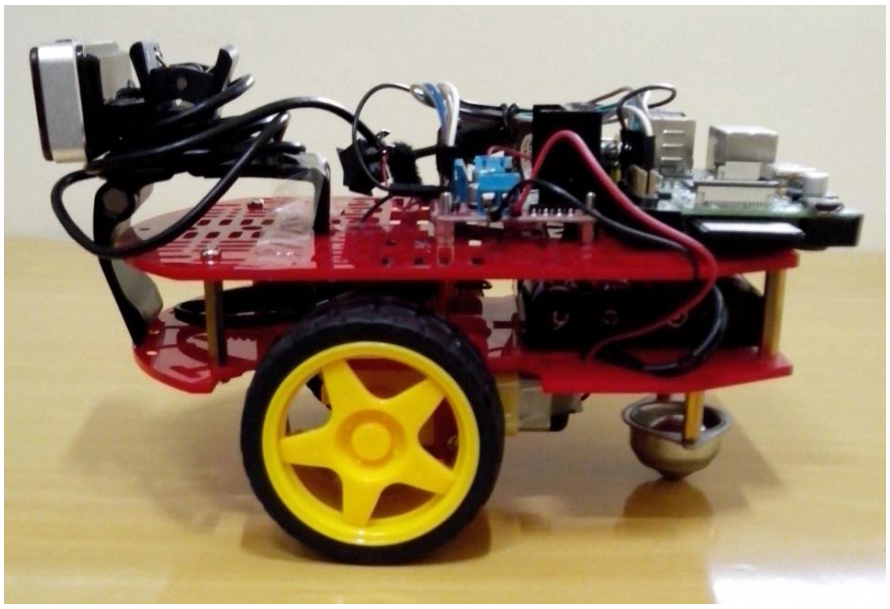

Figura 1: Sistema Proposto.

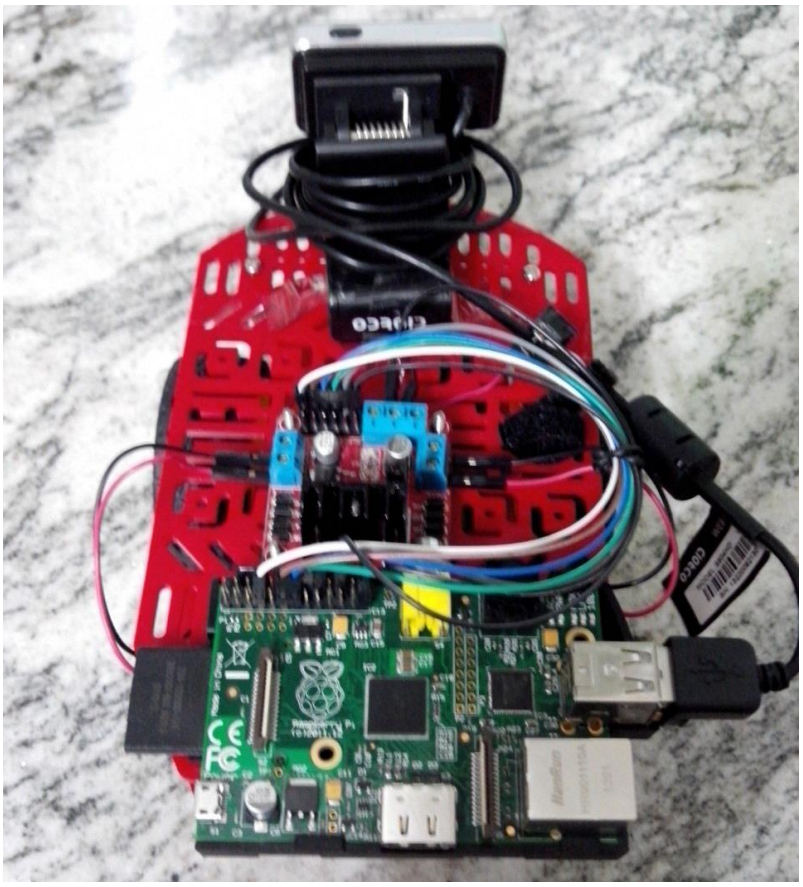

Figura 2: Visão superior do sistema.

Como é possível observar nas imagens acima, o sistema consiste em uma câmera na frente da plataforma móvel, responsável por alimentar o sistema embarcado com imagens do ambiente, uma plataforma móvel responsável pelo movimento do sistema e o sistema embarcado em si, responsável por comandar os outros elementos.

A ideia de se movimentar seguindo um objeto de interesse pode ser usada em várias situações nas quais não seria possível empregar uma pessoa, como por exemplo, verificação de defeitos em tubulações, desativação de minas terrestres (a mina seria um objeto de interesse, então bastaria reconhece-la para descobrir sua posição e realizar os procedimentos necessários para desativação), e extrapolando a ideia, seria possível o reconhecimento de pessoas em meio a multidões.

O sistema embarcado entra com a função de controle múltiplo, administrando todas as partes envolvidas no projeto e aplicando os métodos necessários para extrair as informações das imagens recebidas e os movimentos necessários em cada situação. Cada parte e processo do sistema serão explicados neste documento.

2. Objetivos

2.1. Objetivo Geral

Estudar e propor uma arquitetura embarcada eficiente (composta por hardware e software) para processamento de imagens e controle de periféricos eletrônicos sobre uma plataforma limitada computacionalmente, utilizando um sistema operacional que não tem suporte para aplicações em tempo real.

2.2. Objetivos Específicos

Os seguintes objetivos específicos foram definidos a fim de atingir o objetivo principal:

- Estudar a precisão do método escolhido para detecção de círculos.

- Analisar formas de melhorar o método escolhido para obter uma resposta satisfatória.

- Aplicar os resultados obtidos acima na plataforma embarcada e estudar o melhor compromisso entre precisão e velocidade de execução em tempo real.

- Discutir os resultados obtidos e propor melhorias futuras.

3. Contextualização

O campo de visão computacional pode ser caracterizado como imaturo e diverso. Apesar de existirem trabalhos já reconhecidos, somente após o final da década de 1970 que começaram estudos aprofundados, quando os computadores já podiam processar grandes conjuntos de dados como imagens. Entretanto, tais estudos foram geralmente originados de outros campos de pesquisa, e, consequentemente, não existe uma formulação padrão para o problema de visão computacional, assim como não existe uma formulação padrão de como os problemas de visão computacionais devem ser resolvidos. O que existe atualmente são diversos métodos para resolver várias tarefas bem definidas, no qual os métodos são bastante especializados e raramente podem ser generalizados para várias aplicações. Na maioria das aplicações de visão computacional, os computadores são pré-programados para resolver uma tarefa particular, mas métodos baseados em aprendizagem estão se tornando cada vez mais comuns.

Como uma imagem apresenta uma quantidade de dados muito grande, sua analise demanda muito processamento também. Os computadores mais avançados já conseguem resultados incríveis devido ao seu alto poder computacional, entretanto, eles possuem a desvantagem da necessidade de permanecerem estacionários. Esse problema é resolvido utilizando um sistema embarcado, que, apesar de apresentar um poder computacional menor que um computador desktop, permite tornar o sistema móvel e dedicado.

Um sistema embarcado é um sistema computacional dedicado a uma função específica dentro de um sistema elétrico ou mecânico maior [1]. Ele é embarcado como de um dispositivo, que geralmente inclui hardware e componentes mecânicos. Diferentemente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Sistemas embarcados existem hoje desde dispositivos móveis, como relógios digitais e tocadores MP3, até sistemas fixos, como semáforos, controladores de fábricas e sistemas complexos como veículos híbridos e aparelhos de ressonância magnética [1].

4. Embasamento Teórico

A seguir será descrita de forma sucinta os principais conceitos envolvidos neste projeto.

4.1. Sistemas Operacionais

Sistema operacional é um conjunto de programas que gerenciam os recursos do sistema (definir qual programa recebe atenção do processador, gerenciar memória, criar um sistema de arquivos, etc.), sendo a interface entre programas e recursos, e fornecem uma interface entre o computador e o usuário [2].

Um sistema operacional é composto por vários componentes, interligados para que as diferentes partes de um computador trabalhem em conjunto. Todas as aplicações devem passar pelo sistema operacional para que possa usar qualquer hardware do sistema.

Segue uma lista dos principais (mas não todos) componentes de um sistema operacional.

4.1.1. Kernel

O *kernel* provê o mais básico nível de controle sobre os dispositivos de hardware do computador. Ele administra acesso de memória dos programas a, determina qual programa têm acesso a qual recurso de hardware e organiza os dados para armazenamento não volátil utilizando sistemas de arquivos em mídias de armazenamento tais como fitas eletromagnéticas, disco rígidos, memórias flash, etc [2].

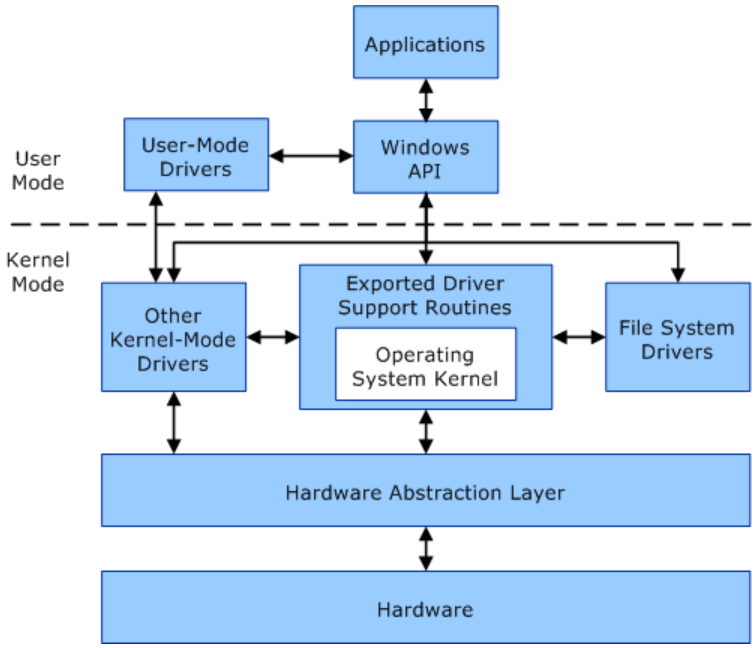

Figura 3: Papel do Kernel [3].

4.1.2. Execução de Processos

O sistema operacional é a interface entre um programa sendo executado e o hardware do computador, como já foi dito. Ele também dita regras e procedimentos para a utilização segura dos recursos utilizados por cada programa. Executar um programa envolve o sistema operacional criar um processo pelo seu *kernel*, o qual lhe atribui memória e outros recursos, estabelece uma prioridade para o processo em sistemas multitarefas, carrega o código binário do programa na memória do sistema e inicia a execução do programa, que só então interage com o usuário e os dispositivos de hardware [2].

4.1.3. Interrupções

Interrupções são uma parte central de um sistema operacional, pois é por meio delas que ele pode interagir com e reagir ao seu ambiente. Elas permitem que um computador salve automaticamente seu estado atual para que um código ou um programa específico seja executado em resposta a eventos, ou para que alguma operação de leitura ou escrita seja feito, e depois, retornar para o estado salvo anterior à resposta do evento [2]. Em sistemas operacionais modernos, as interrupções são controladas pelo *kernel* e podem surgir do hardware do computador ou de um programa sendo executado.

Quando um dispositivo de hardware aciona uma interrupção, o *kernel* decide como lidar com esse evento, geralmente executando algum código em resposta. O processamento de interrupções de hardware geralmente é delegado para um software chamado de "driver", que pode fazer parte do *kernel*, de outro programa, ou de ambos [2].

Uma interrupção acionada por um programa geralmente indica que o programa deseja acessar algum recurso de hardware. Por exemplo, se um programa necessita ler dados do disco rígido, ele aciona uma interrupção para o *kernel*, o que causa o controle a ser passado para o *kernel*, que então processa a requisição e executa os procedimentos necessários para atender a requisição [2].

4.1.4. Modos de Operação

Processadores modernos possuem vários modos de operação. Geralmente utilizam-se dois modos: protegido e supervisor. O modo supervisor é utilizado pelo *kernel* para tarefas de "baixo-nível" que necessitam acesso irrestrito ao hardware, como controle de escrita em memória e comunicação com dispositivos como a placa de vídeo. Em contraste, o modo protegido é utilizado para todo o resto. Programas são executados em modo protegido, e só podem utilizar recursos do hardware se comunicando com o *kernel*, que controla tudo em modo supervisor [2].

Os primeiros programas a serem executados em um computador (como BIOS e *bootloader*) e o sistema operacional têm acesso ilimitado ao hardware. Isso é necessário, pois por definição, iniciar um ambiente protegido só é possível de fora de um ambiente protegido [2]. Entretanto, quando o sistema operacional passa o controle para outro programa, ele pode colocar o processador em modo protegido.

4.1.5. Gestão de Memória

O *kernel* de um sistema operacional multitarefas precisa ser responsável por gerenciar toda a memória do sistema que está em uso. Isso garante que um programa não interfira com a memória sendo usado por outro programa. Cada programa deve ter acesso independente à memória [2].

A gestão da memória também envolve proteção de memória, que permite o *kernel* limitar o acesso de um programa a memória. Existem vários métodos de proteção de memória, incluindo segmentação (divisão da memória principal em seções) e paginação (utilização da memória secundária como armazenamento para dados utilizados na memória primária) [2]. Todos os métodos necessitam de certo nível de suporte do hardware, o qual não existe em todos os computadores.

Em ambos os métodos de proteção citados acima, o modo protegido específica quais endereços de memória são permitidos a quais programas. Tentativas de acessar um endereço de memória a outros endereços irão acionar uma interrupção, colocando o *kernel* no comando. Isso é chamado de violação de segmentação, e geralmente o *kernel* termina o processo do programa violador e reporta o erro ao sistema operacional [2].

4.1.6. Memória Virtual

A utilização de endereçamento de memória virtual (como paginação e segmentação) significa que o *kernel* pode escolher qual memoria cada programa poderá usar em dado momento, permitindo o sistema operacional utilize a mesma memória, para tarefas múltiplas.

Em sistemas operacionais modernos, endereços de memória que não são acessados frequentemente podem ser armazenados temporariamente em disco para liberar espaço na memória primária para outros programas. Isso é chamado de swapping, pois uma área de memória pode ser utilizada por vários programas, e seu conteúdo pode ser trocado ("*swapped*") por demanda. A memória virtual provê a percepção que há mais memória RAM no computador do que existe fisicamente [2].

4.1.7. Multitarefa

Multitarefa se refere a execução de múltiplos programas independentes no mesmo computador, dando a impressão que ele está executando as tarefas ao mesmo tempo. Como a maioria dos computadores consegue fazer no máximo duas tarefas ao mesmo tempo, isso é feito geralmente através de "time-sharing", que significa que cada programa utiliza uma parte do tempo do computador para ser executado.

O *kernel* contém um programa chamado de "escalonador", que determina quanto tempo um programa será executado e qual a ordem de execução dos programas. Sistemas operacionais modernos estendem o conceito de preempção de programas para drivers e códigos do *kernel*, para que o sistema operacional tenha controle preemptivo também sobre os tempos de execução internos [2].

A filosofia do sistema multitarefa preemptivo é garantir que todos os programas recebam tempo regular no processador. Isso implica que todos os programas têm seu tempo de processamento limitado. Para realizar isso, o *kernel* faz uso de interrupções cronometradas.

4.1.8. Acesso ao Disco Rígido e Sistemas de Arquivos

Computadores armazenam dados em discos rígidos utilizando arquivos, que são estruturados de formas específicas para permitir acesso mais rápido, maior confiabilidade e maximizar o uso do espaço livre do disco. A forma específica como os arquivos são armazenados no disco é chamada de sistema de arquivos, e permite que arquivos tenham nomes e atributos e sejam armazenados em uma hierarquia de diretórios e pastas arranjados em uma árvore de diretórios [2].

Um dispositivo de armazenagem conectado, como um disco rígido, é acessado utilizando um driver. O driver entende a linguagem específica do disco e a traduz para uma linguagem padrão usada pelo sistema operacional para acessar todos os dispositivos de disco [2].

O *kernel* acessa o conteúdo do disco em um formato binário, o qual pode conter um ou mais sistemas de arquivos. Um driver de sistema de arquivo é utilizado para traduzir os comandos usados para acessar cada sistema de arquivo específico em um conjunto de comandos padrão que o sistema operacional pode usar para acessar todos os sistemas de arquivo. Programas interagem com esses arquivos por meio de nome de arquivo e diretórios e pastas.

4.1.9. Drivers de Dispositivos

Um driver de dispositivo é um tipo específico de software desenvolvido para permitir interação com dispositivos de hardware. Tipicamente constitui uma interface para comunicação com o dispositivo, provendo comandos para e/ou recebendo dados do dispositivo e interfaces para o sistema operacional e softwares. O driver é dependente do hardware e é específico ao sistema operacional.

4.1.10. Rede

Atualmente, a maioria dos sistemas operacionais suporta uma variedade de protocolos de rede, hardware e aplicações. Isso significa que computadores utilizando sistemas operacionais possam participar em uma rede comum para compartilhar recursos como arquivos, impressoras e tempo de processamento. Redes permitem que o sistema operacional de um computador possa acessar recursos de um computador remotamente como se esses recursos estivessem conectados diretamente no computador local [2].

Redes do tipo Cliente/Servidor permitem que o programa cliente se conecte ao programa servidor, o qual oferece vários serviços, como processamento, armazenamento, compartilhamento, entre outros.

4.1.11. Interface de Usuário

Todo computador operado por um indivíduo necessita de uma interface. A interface de usuário enxerga a estrutura de diretórios, requisita serviços do sistema operacional que adquirirão dados dos dispositivos de hardware de entrada, como o teclado e o mouse, e requisita serviços do sistema operacional para mostrar informações em dispositivos de hardware de saída, como o monitor [2]. As duas interfaces de usuário mais comuns são a interface de linha de comando, na qual aparecem apenas informações textuais e o usuário interage com o sistema através de comandos enviados linha por linha, e a interface gráfica de usuário, que apresenta um ambiente visual.

4.1.12. Tipos de Sistemas Operacionais

Real-Time

Um sistema operacional em tempo real visa executar aplicações em tempo real, utilizando geralmente um escalonador especializado para que se obtenha um comportamento determinístico. O principal objetivo do sistema operacional em tempo real é responder eventos de forma rápida e previsível. Ele possui um sistema de multitarefas focado em eventos ("*event-driven*"), isso significa que o escalonador alterna entre processos baseados em suas prioridades ou eventos externos, em contraste a um sistema de "*time sharing*", que alterna entre processos baseado no tempo de execução de cada processo.

Multiusuário

Um sistema operacional multiusuário permite que vários usuários acessem um computador ao mesmo tempo. Sistemas "*Time Sharing*" e servidores de internet são exemplos de sistemas multiusuários, pois permitem o acesso a vários usuários dividindo o tempo entre eles.

Distribuído

Um sistema operacional distribuído administra um conjunto independente de computadores e faz parecer ao usuário que são apenas um. Isso permite ao usuário utilizar recursos de computadores diferentes, fazendo-os trabalhar em cooperação. É uma das bases da programação distribuída.

Embarcados

Sistemas operacionais embarcados são feitos para serem usados em sistemas embarcados. Isso significa que eles são mais compactos e mais eficientes, podendo ser executados com menos recursos, mas também apresentam menos recursos ao usuário, e não atinge a velocidade de um sistema embarcado normal devido ao hardware.

4.2. Processamento de Imagens

Processamento de imagens é uma forma de processamento de dados no qual a entrada é uma imagem e a saída do processamento é uma imagem ou um conjunto de parâmetros relacionados à imagem, diferente do tratamento de imagens, cuja preocupação é a manipulação de figuras para sua representação final. A maioria das técnicas de processamento de imagens envolve tratar a imagem como um sinal bidimensional, e aplicar técnicas padrão de processamento de sinais [4].

O processamento de imagens permite o uso de algoritmos complexos e é a forma mais prática de realizar certas tarefas (mas não limitadas a estas), explicadas a seguir.

4.2.1. Classificação:

Classificação é o problema de identificar a qual conjunto de categorias pertence uma nova observação, levando em conta um conjunto de dados de treinamento contendo observações e informação sobre qual categoria pertence cada observação. Um exemplo seria classificar qual *email* é *spam* e qual não é.

4.2.2. Extração de Características:

Extração de características é uma forma especial de redução de dimensão. Quando se suspeita que um conjunto de dados seja muito redundante (como por exemplo, a repetitividade de imagens representadas por pixels) então o conjunto será transformado em um conjunto reduzido de características. Existe uma quantidade muito grande de características que podem ser extraídas, cada uma pertinente a um tipo de aplicação [4].

Dentro dessas características encontram-se as detecções de borda, canto, direção da borda, de movimento e as detecções baseadas em formas, como a extração de linhas, círculos, elipses e formas arbitrárias. Para esses três últimos, é utilizada uma transformada a ser explicada neste capítulo.

4.2.3. Reconhecimento de Padrões

Reconhecimento de Padrões é uma subcategoria de aprendizado de máquinas que foca no reconhecimento de similaridades e regularidades em dados [4]. Sistemas de reconhecimento de padrões geralmente utilizam um conjunto de dados classificados para treinamento, são testados utilizando dados novos que não estavam no primeiro conjunto. Esse método é chamado de aprendizado supervisionado. Mas existe também o chamado aprendizado não-supervisionado, quando não está disponível um conjunto de dados classificados e utiliza-se um algoritmo para a descoberta de padrões previamente desconhecidos.

4.3. Transformada Hough

A transformada Hough é uma técnica de extração de características usada em processamento de imagens, originalmente focada em encontrar linhas na imagem, que depois foi expandida para encontrar a posição de formas arbitrárias, principalmente círculos e elipses [5].

4.3.1. Teoria

O caso mais simples de uma transformada Hough é a transformada linear para detecção de retas. Uma reta pode ser definida pela equação $y = mx + b$, na qual m é a inclinação da reta e b é a intersecção da reta com o eixo y. A ideia principal da transformada de Hough é considerar as características da reta de acordo com a equação que a define (ou seja, de acordo com m e b), e não como pontos discretos na imagem. Geralmente, uma reta pode ser representada como um ponto (b, m) no espaço paramétrico, entretanto, retas verticais apresentam um problema para essa representação, já que elas são naturalmente descritas como $x = a$, o que geraria valores sem limites do parâmetro m. Para contornar isso, Duda e Hart propuseram o uso de coordenadas polares para linhas na transformada de Hough.

A coordenada polar é definida por r, a distância algébrica entre a reta e o ponto de origem do plano (0,0); e por θ, o angulo do vetor ortogonal a reta em direção ao plano superior direito. Usando essa parametrização, a equação da reta pode ser escrita como:

$$y = (-\frac{cos(\theta)}{sin(\theta)}) * x + (\frac{r}{sin(\theta)})$$

Que pode ser rearranjada para:

$$r = x * cos(\theta) + y * sin(\theta)$$

Portanto, é possível associar cada reta da imagem a um par (r, θ) que é único se $\theta \in [0, \pi)$ e $r \in \boldsymbol{R}$ ou $\theta \in [0, 2\pi)$ e $r \geq 0$. O plano das coordenadas polares (r, θ) é chamado de Espaço de Hough para um conjunto de linhas retas em duas dimensões.

Para um ponto arbitrário (x_0, y_0) no espaço da imagem, as retas que passam por esse ponto são os pares (r, θ) tal que:

$$r(\theta) = x_0 cos\theta + y_0 sin\theta$$

Onde r é determinado por $\theta \epsilon [0, \pi)$. Como r deve ser positivo, as retas que passam pelo ponto (x_0, y_0) são $r(\theta) = |x_0 cos\theta + y_0 sin\theta|$.

Essas representações correspondem a uma senóide no plano (r, θ), que é única para o ponto (x_0, y_0). Se as curvas correspondentes a dois pontos são sobrepostas, o lugar onde elas se cruzam (no Espaço de Hough) corresponde a uma reta (no plano original da imagem) que passa por ambos os pontos. De forma mais genérica, um conjunto de pontos que formam uma reta produzirá senóides que se cruzam nos parâmetros dessa reta. Portanto, o problema de encontrar pontos colineares se torna um problema de encontrar curvas concorrentes.

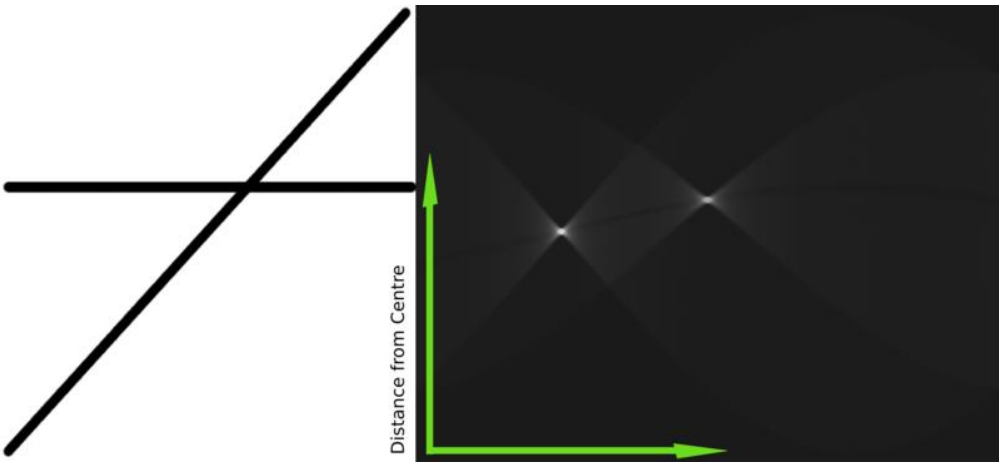

Figura 4: Representação da Transformada de Hough em duas linhas[6].

Utilizando essa base da transformada, é possível expandi-la para encontrar qualquer forma que possa ser descrita como um conjunto de parâmetros. Um círculo, por exemplo, pode ser descrito por três parâmetros, representando suas coordenadas do centro e seu raio, por tanto o espaço de Hough se torna tridimensional.

O processo de identificar objetos circulares no espaço de Hough é relativamente simples. Primeiramente é criado um espaço acumulador, feito por uma célula para cada pixel, com valor inicial de zero. Para cada ponto de borda na imagem (i, j), incrementam-se todas as células que, de acordo com a equação de um círculo $(i-a)^2 + (j-b)^2 = r^2$ poderiam ser o centro desse círculo, essas células são representadas pela letra 'a' na equação. Para cada valor encontrado no passo anterior, encontram-se todos os possíveis valores de 'b' que satisfazem a equação. Por último, procuram-se as células cujo valor é maior que qualquer outra célula em sua vizinhança. Essas células possuem a maior probabilidade de pertencerem ao circulo que estamos tentando encontrar. Como não é conhecido o valor do raio de antemão, utiliza-se um acumulador de três dimensões para tal.

Entretanto, há limitações, a transformada de Hough só é eficiente se um número alto de votos caírem na célula correta, de forma que ela possa ser detectada facilmente entre o ruído da imagem. Quando o número de parâmetros é grande, o número médio de votos em uma célula é muito baixo, e as células que correspondem a uma figura real na imagem não necessariamente recebem mais votos que os vizinhos. A complexidade é exponencial, $O(A^{m-2})$, onde A é o tamanho do espaço da imagem e m é o número de parâmetros. E finalmente, a eficiência da transformada depende da qualidade da imagem, as bordas devem ser facilmente detectadas para que a transformada Hough seja eficiente. Por isso geralmente a imagem passa por um pré-processamento para que a transformada Hough seja aplicada [7], [8].

4.4. Filtro de Cor

A aplicação de um filtro de cor elimina todas as cores da imagem, exceto a cor desejada, transformando-as em preto. Esse filtro é muito útil quando o objeto desejado na imagem tem uma cor fixa e conhecida. O primeiro passo para aplicação do filtro é a conversão da imagem RGB em HSV. HSV é uma forma de representar a cor de um pixel da imagem utilizando coordenadas cilíndricas. Neste caso, utiliza-se matiz (*hue*), saturação (*saturation*) e valor (*value)*. Matiz é a representação numérica do comprimento de onda do espectro eletromagnético de cada cor, com valores entre 0 e 360. A saturação s, $s \in [0,1]$, mede o distânciamento de um valor de matiz do branco ou cinza. O valor v, $v \in [0,1]$, mede a distanciamento de um valor de matiz do preto, a cor com zero de energia [9].

4.5. Filtro de Gauss

O filtro de Gauss aplica um embaçamento na imagem, usado tipicamente para reduzir ruídos em imagens, reduzir detalhes e aprimorar as estruturas da imagem em escalas diferentes. Matematicamente, o filtro aplica uma convolução da imagem com a função Gaussiana. Como a transformada de Fourrier de uma função Gaussiana é outra função Gaussiana, aplicar o filtro Gauss tem o efeito de reduzir os componentes de alta frequência da imagem, portanto, é um filtro passa-baixa. Uma função Gaussiana é definida como [9]:

$$G(x,y) = \frac{1}{2\pi\sigma^2} * \ e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

Na qual x é a distância da origem no eixo horizontal, y é a distância da origem no eixo vertical, e delta é o desvio padrão da distribuição de Gauss.

4.6. Dilatação

Dilatação é uma das duas operações básicas da morfologia matemática. Ela opera utilizando um elemento estrutural para sondar e expandir as formas da imagem [10].

Na morfologia, imagens são funções mapeando um espaço Euclidiano E sobre o domínio real $\mathbb{R} \cup \{-\infty, +\infty\}$. Denotando uma imagem como $f(x)$ e o elemento estrutural por $b(x)$, a dilatação de f por b é dada por [10]:

$$(f \oplus b)(x) = sup_{y \in E}[f(y) + b(x - y)]$$

"*Sup*" denota a função *supremum*, que calculo o menor limitante superior.

É comum utilizar elementos estruturantes planos, na forma de:

$$b(x) = \begin{cases} 0, \\ -\infty, \end{cases} \quad x \in B$$

Sendo que $B \subseteq E$.

Neste caso, a dilatação é simplificada e dada por:

$$(f \oplus b)(x) = sup_{y \in E}[f(y) + b(x - y)] = sup_{z \in E}[f(x - z) + b(z)] = sup_{z \in B}[f(x - z)]$$

4.7. Erosão

Erosão é a segunda operação básica da morfologia matemática. Em contraste a dilatação, essa função contrai as estruturas da imagem.

Da mesma forma que foi definida a Dilatação, a Erosão é definida como [10]:

$$(f \ominus b)(x) = inf[f(x + y) - b(y)]$$

Na qual "*inf*" denota a função *infimum*, que calcula o maior limitante inferior.

4.8. Filtro de Canny

O Filtro de Canny é um algoritmo detector de bordas multiestágio. Os passos do filtro são os seguintes:

- Aplicação de um filtro Gaussiano para limpar a imagem de ruídos;
- Aplicação do Operador Gradiente para obter a intensidade e direção do gradiente;

- Aplicação de uma Supressão Não-Máxima para determinar se um pixel é melhor candidato para uma borda do que seu vizinho;

- Limiar de histerese para encontrar onde a borda começa e termina.

Limiar de Histerese

Gradientes de intensidade grandes são mais propensos a corresponder a bordas do que gradientes pequenos, e na maioria dos casos é impossível especificar um limiar em que um dado gradiente de intensidade deixa de corresponder a uma borda. Por isso o filtro de Canny utiliza limiar com histerese.

O limiar com histerese necessita de dois valores, um alto e um baixo. Assumindo que bordas importantes se encontram ao longo de uma curva é possível seguir seções apagas de uma linha e descartar pixels ruidosos que não constituem uma linha, mas que produzem grandes gradientes. Portanto o filtro começa aplicando o limiar alto, marcando as bordas que são possíveis determinar como verdadeiras. A partir dessas bordas, utilizam-se as informações de direção obtidas anteriormente para traçar as linhas ao longo da imagem. Durante esse processo, aplica-se o limiar baixo, permitindo traçar linhas apagas, desde que se encontre um ponto de partida.

Figura 5: Resultado do Filtro de Canny [11].

4.9. Controle Digital

Controle digital é um subconjunto da teoria de controle que utiliza computadores digitais ou sistemas digitais para atuarem como controladores de sistemas. Dependendo dos requerimentos necessários, um sistema de controle digital pode ser um microcontrolador, um circuito integrado até ou um computador convencional [12]. Como um computador digital é um sistema discreto, a transformada de Laplace, comumente utilizada em teoria de controle, é substituída por sua representação discreta, a transformada Z. Como o computador tem uma precisão limitada, é necessário um cuidado especial com aproximações.

4.10. Ponte H

Uma ponte H é um circuito eletrônico que permite aplicar tensão em uma carga (geralmente um motor elétrico) em ambas as direções. O termo ponte H é derivado da sua representação gráfica típica, encontrada abaixo:

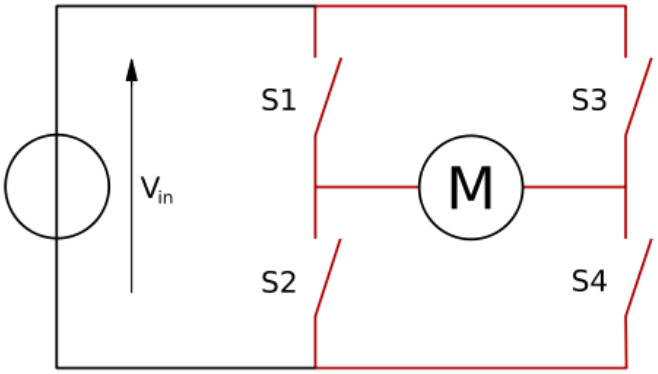

Figura 6: Representação Gráfica da Ponte H [13].

A ponte H é construída com quatro chaves, quando as chaves S1 e S4 estão fechadas (e S2 e S3 estão abertas) uma tensão positiva é aplicada no motor, quando as chaves S1 e S4 são abertas e S2 e S3 são fechadas, a tensão aplicada é reversa, permitindo a operação reversa do motor. Observa-se que os pares de chaves S1, S2 e S3, S4 nunca devem ser fechadas ao mesmo tempo, pois essa condição pode ocasionar curto-circuito. É possível também "brecar" os motores, fechando S1 e S3 ou S2 e S4, ou deixá-lo rodando livre, deixando todas as chaves abertas. A tabela abaixo resume as informações acima:

S1	S2	S3	S4	Resultado
1	0	0	1	O motor move-se para a direita
0	1	1	0	O motor move-se para a esquerda

0	0	0	0	O motor se movimenta livremente
0	1	0	1	O motor para
1	0	1	0	O motor para
0	0	1	1	Curto-Circuito
1	1	0	0	Curto-Circuito
1	1	1	1	Curto-Circuito

Tabela 1: Tabela de funcionamento da ponte H

Controlar o movimento do motor se torna uma simples questão de aplicar tensão nos pinos corretos para gerar o movimento desejado.

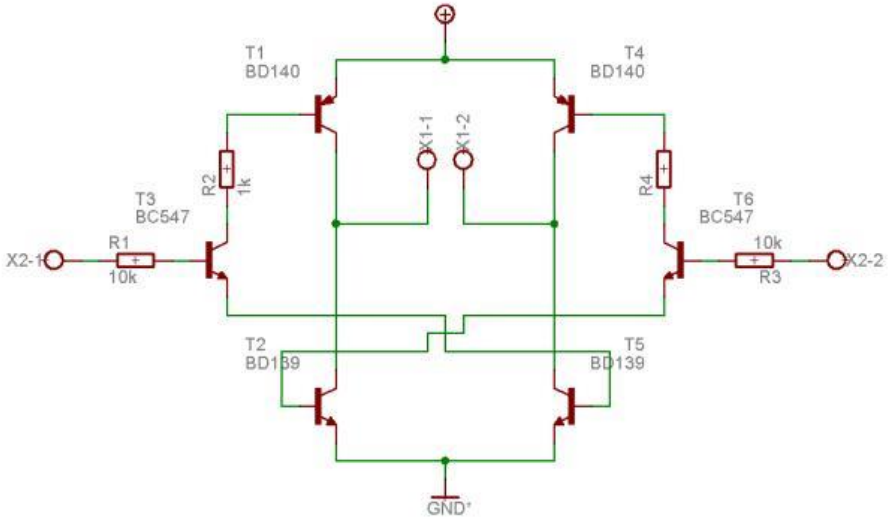

Figura 7: Circuito de uma ponte H [14].

4.11. PWM

PWM (*Pulse-Width Modulation*) é uma técnica de modulação que controla a largura do pulso de uma onda constante. Seu propósito principal é controlar a potência fornecida a uma carga. A proporção que a onda está em "alto" para o tempo que ela está em "baixo", ou seja, a quantidade de trabalho realizada pela onda, chama-se *duty cycle*, ou seja, um PWM de 50% significa que a onda fica metade do tempo em "alto" e metade do tempo em "baixo".

O princípio do PWM é a utilização de uma onda quadrada, cuja largura do pulso é modulada, resultando em uma variação do valor médio da forma de onda. Considerando uma forma de onda de pulso $f(t)$, com período T, valor mínimo ymin, valor máximo ymax e um duty cycle D, o valor médio da forma de onda é dado por [15]:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)dt$$

Como $f(t)$ é uma onda quadrada, seu valor atinge y_{max} quando $0 < t < D$ e seu valor mínimo quando $D * T < t < T$. Então, expressão acima pode ser escrita como:

$$\bar{y} = \frac{1}{T}\left(\int_0^{DT} y_{max}dt + \int_{DT}^{T} y_{min}dt\right)$$
$$= \frac{D \cdot T \cdot y_{max} + T(1-D)y_{min}}{T}$$
$$= D \cdot y_{max} + (1-D)y_{min}$$

Essa equação pode ser simplificada muitas vezes quando $y_{min} = 0$ e $y_{max} = D \cdot y_{max}$. Deduz-se facilmente que o valor médio da onda depende no *duty cycle*.

Como a velocidade de rotação do motor é diretamente proporcional à tensão aplicada, é possível controlar essa velocidade definindo um *duty cycle*. Um *duty cycle* de 50% produzirá uma tensão média igual a metade da tensão aplicada, e por tanto, metade da velocidade máxima para aquela tensão.

5. Desenvolvimento do Trabalho

A seguir será descrito os materiais utilizados deste trabalho e todo o desenvolvimento feito pelo aluno.

5.1. Materiais

Foram utilizadas duas câmeras USB para obter as imagens do ambiente para que seja possível realizar o processamento desejado. Uma das câmeras utilizadas é uma câmera genérica de 2.0 Megapixels, a outra é uma câmera HD com resolução de 720p, que possui maior nível de detalhe que a câmera genérica. Foi necessário fixar uma resolução de 640x320 pixels, pois esse é o máximo que a câmera genérica consegue atingir. Foi constatado também que uma resolução maior acarreta um maior tempo de processamento devido a maior quantidade, por tanto uma resolução menor é vantajoso para o projeto.

O sistema embarcado escolhido foi a Raspberry Pi Modelo B. A Raspberry pode ser chamada de "um computador do tamanho de um cartão de crédito", pois ela apresenta todas as funções de um computador: estradas de comunicação padrão (USB, Ethernet), saídas de Áudio e Vídeo (VGA e HDMI) e apresenta um sistema operacional Linux customizado, sendo mais leve para ser executado de forma eficiente sobre a plataforma, além de também apresentar 24 pinos de *General Purpose Input/Ouput*, além de ser necessário apenas 5V para alimentar todo o sistema.

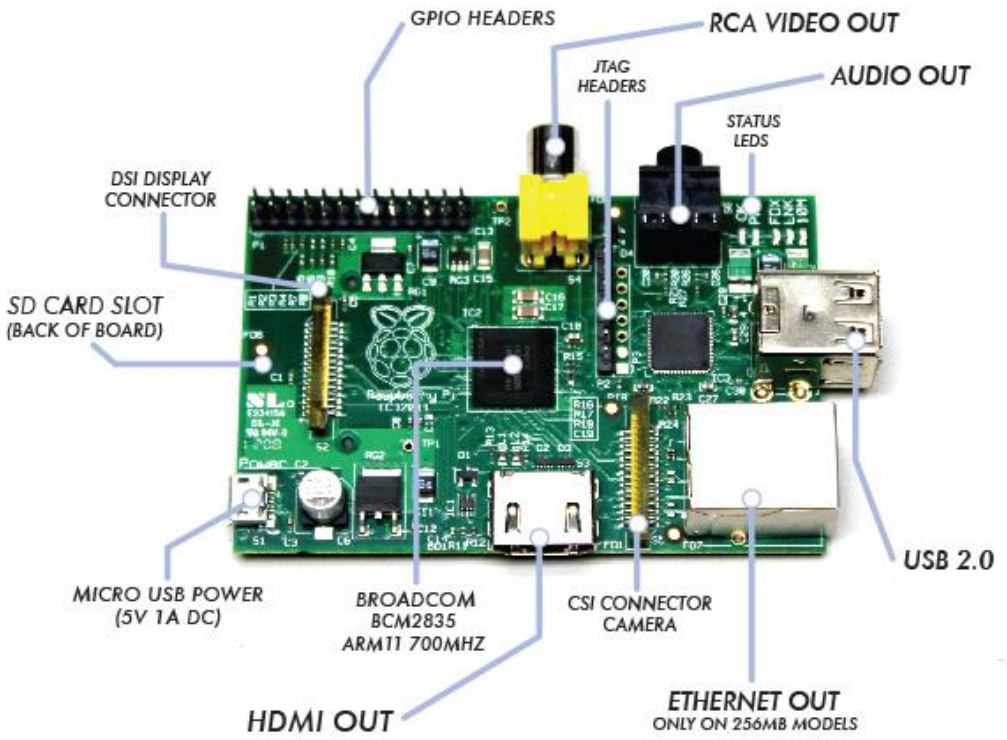

Figura 8: Raspberry Pi [16].

A escolha dessa plataforma se deve principalmente devido a sua simplicidade, preço e características. Por rodar um sistema operacional completo, não é necessário se preocupar com o gerenciamento de fatores não envolvidos nesse projeto (como por exemplo, drives para USB e Ethernet, etc.), facilitando o uso da webcam com um simples *plug-and-play*. Por se tratar de um computador completo, capaz de executar um sistema operacional, seu preço de $35,00 [17] é muito baixo, tornando-a muito atrativa. Como foi dito acima, ela também possui 24 pinos GPIO, o que permite que o controle de dispositivos elétricos (como motores de corrente contínua) e eletrônicos (como circuitos integrados). Com essas características, a Raspberry é um misto de microcontrolador e computador, sendo perfeito para este projeto [18].

As especificações de hardware da Raspberry são as seguintes :

- Processador arquitetura ARM1176JZF-S em um *System-on-Chip* Broadcom BCM2835, *clock* de 700 MHz, capaz de executar 0.041 GFLOPS e 128 kb de memória cache [19];
- 512 Mb de Memória RAM;

- GPU Broadcom VideoCore IV com *clock* de 250 MHz, utilizando OpenGL ES 2.0, capaz de executar 24 GFLOPS (equivalente a 1 GPixel/s) [19];
- Vídeo com resolução de 1920x1080, HDTV de 1080p;
- Armazenamento persistente utilizando um cartão de memória SD/MMC/SDIO;
- Potência de 700 mA e 3.5 W, alimentada através de uma entrada MicroUSB de 5V.

Apesar das vantagens apresentadas pela Raspberry, ela possui algumas desvantagens. Seu poder de processamento é pequeno se comparado a computadores modernos, se tornando assim o limitante da aplicação. Como o reconhecimento de imagens demanda muita carga computacional da Raspberry, algumas melhorias foram feitas no algoritmo de reconhecimento, bem como algumas concessões em relação à precisão e qualidade.

INSERIR IMAGEM

Como pode ser observado, a plataforma é composta por um chassi robótico com 3 rodas, duas na parte traseira ligadas a motores que podem ser controlados, e uma roda na parte dianteira que gira em todas as direções, uma Raspberry PI Model B, uma ponte H utilizando o CI L298N e uma webcam USB genérica.

O controle do movimento da plataforma será feito pelo sistema, mas a execução do movimento será feita por uma Ponte H ligada a Raspberry (da qual receberá ordens de movimento) e a dois motores (que irão movimentar efetivamente a plataforma). A ponte H em questão é uma placa baseada no circuito integrado L293D [20], um circuito robusto capaz controlar dois motores ao mesmo tempo e sustentar correntes de até 600 mA e tensões até 36 V.

5.2. Planejamento

O projeto envolve diversas partes diferentes e interligadas, por tanto foi feito uma divisão e uma ordem de prioridade para cada parte a ser feita. As principais tarefas definidas para o desenvolvimento do projeto foram:

- Escolha do método de detecção e símbolo detectado;
- Implementação e estudo da eficiência do método;
- Refinamento;

- Montagem da plataforma e clculo das distâncias;
- Integração do sistema de visão com o sistema de movimento.

Cada tarefa principal envolve diversas tarefas menores, que serão descritas a seguir.

5.3. Escolha do método de detecção e símbolo detectado

Primeiramente, a ideia seria detectar um símbolo de acordo com sua forma, por exemplo, uma cruz. Entretanto, os métodos para tal detecção são geralmente baseados em extração de características e aprendizado [21], [22]. Característica é qualquer informação relevante para resolver o problema computacional em questão, podendo ser estruturas específicas na imagem, como pontos, bordas ou objetos; ou o resultado de uma operação na vizinhança, como a média da vizinha de um pixel escolhido. Aprendizado refere-se aos sistemas que alteram seu comportamento de acordo com dados apresentados. O sistema irá extrair características da imagem e multiplicará seus valores por pesos pré-estabelecidos. O resultado é comparado ao limiar do sistema, se for maior que o limiar, a imagem apresenta o símbolo desejado, caso contrário, a imagem não apresenta o símbolo. Os pesos das características e o limiar são ajustados apresentando ao sistema um conjunto de imagens contendo o símbolo e um conjunto de imagens que não o contém. O sistema aplica seu processo a cada imagem, e quando ele erra a detecção (resultando em um falso positivo ou falso negativo) seus pesos e limiar são ajustados seguindo uma equação de correção.

Esse método necessita de muito poder de processamento, pois a extração das características é muito custosa. Uma detecção com erro baixo demanda um número grande de características, além da necessidade de ajuste manual de vários parâmetros envolvidos na fase de aprendizado.

Por esses motivos foi escolhido um método que não necessita de muito poder computacional e já foi bastante estudado e desenvolvido; a transformada de Hough para detecção de círculos. Cogitou-se a utilização "*Blob Detection*", um método que detecta regiões contínuas que diferem da imagem em alguma propriedade, como brilho ou cor, e modifica-lo para que detecte círculos. Entretanto, testes iniciais demostraram que esse método é muito dispendioso, levando mais tempo que a transformada de Hough para executar a detecção, obtendo resultados parecidos.

A transformada de Hough é precisa e permite extrair da imagem exatamente as informações necessárias do símbolo detectado, posição e raio. A transformada é um filtro adaptativo, ou seja, é necessário ajustar uma série de parâmetros para que a transformada se adapte a uma aplicação específica e obter resultados mais precisos, o que é muito útil neste projeto e será mais discutido abaixo.

5.4. Implementação

O desenvolvimento foi feito em um notebook comum e embarcado na Raspberry Pi utilizando acesso remoto (via SSH). Isso é possível, pois o código em Python é executado sobre uma máquina virtual Python, ou seja, não é necessária uma compilação do código para cada tipo de plataforma e sistema operacional, basta que exista uma máquina virtual Python instalada.

Para a implementação do método escolheu-se a biblioteca gráfica OpenCV, uma biblioteca *open source* que possui diversas ferramentas para visão computacional, concebida com foco em eficiência computacional e escrita em C/C++ otimizado, possui interface para Python, o que definitivamente é um ponto positivo, já que pode-se aproveitar da facilidade de desenvolvimento do Python e da velocidade e eficiência do C/C++. Também se garante que os algoritmos usados sejam os mais eficientes, para que se possa analisar o sistema como um todo, e não a implementação do algoritmo.

Um pré-processamento da imagem antes de aplicar a transformada de Hough gera resultados melhores e mais apurados [8], [23]. A aplicação de um detector de bordas antes da transformada aumenta a precisão da detecção. Como existem vários métodos de detecção de borda, foi feito uma análise das vantagens e desvantagens de cada um.

Os métodos considerados foram: o filtro de Sobel, filtro de Prewitt e filtro de Canny. O filtro de Prewitt é apenas uma variante do filtro de Sobel [24], apresentando apenas uma mudança nos valores usados nos *kernels* de convolução de cada método, então foi feita uma comparação entre o filtro de Canny e o filtro de Sobel.

Sobel é um filtro simples, capaz de detectar bordas e suas orientações, entretanto, muitas vezes ele é impreciso e muito sensível a ruídos, não sendo um método muito robusto [24]. O filtro de Canny tem a desvantagem de consumir mais tempo, por ser mais complexo, e depender fortemente de seus parâmetros; mas é um método robusto, não sensível a ruídos e apresenta uma precisão muito boa. As vantagens do filtro de Canny se alinham mais com o foco deste trabalho, já que é preciso obter uma detecção mais precisa, mesmo perdendo um pouco da eficiência. É provado também que o filtro de Canny tem um desempenho melhor na maioria dos possíveis cenários [24], [25], o que novamente encaixa com o objetivo do projeto, pois o cenário da detecção não é controlado (por exemplo, exigir que o fundo seja branco). Por essas razões, foi escolhido o filtro de Canny.

A ideia inicial do projeto era que o sistema detectasse qualquer círculo na imagem e priorizasse o maior encontrado. Essa ideia se mostrou não ser boa, pois apresentava muita instabilidade na detecção (grande variação da posição e raio do círculo detectado); como o cálculo do movimento depende dessas informações, alta instabilidade na detecção reflete baixa confiança no calculo da distância. Isso significa que a chance do sistema calcular a distância real com a precisão desejada não é satisfatória. Havia também o problema do algoritmo detectar mais de um círculo, ou existir mais de um circulo e o algoritmo não detectar o circulo desejado. Por essas razões definiu-se que era necessário existir alguma característica única que possa ser extraída e o diferencie do fundo e/ou outros círculos na imagem.

A característica escolhida foi a cor. Um filtro de cores é facilmente implementado e apresenta pouco impacto na eficiência do método. Uma cor incomum facilita a extração do fundo e a remoção de ruídos. A cor escolhida foi a amarela, pois não é uma cor facilmente encontrada. Para aplicar o filtro de cor, transforma-se a imagem em HSV, dessa forma é possível comparar o valor de um pixel com um intervalo de valores representando a cor amarela, algo que não é possível fazer utilizando RGB. O próximo passo é filtrar a imagem pela cor amarela, de forma que todos os pixels dentro de um intervalo pré-definido se tornam brancos e o resto se torna preto. Obtém-se então uma imagem binária, contendo apenas o objeto de interesse. O intervalo definido para o tom de amarelo foi de [20, 70, 70] até [70, 255, 255] dentro do espaço HSV, encontrado empiricamente.

Aplicado o filtro, obtém-se uma imagem na qual idealmente há apenas o objeto de interesse em branco e o fundo em preto. Isso reduz a possibilidade de falsas detecções e ruídos provenientes do fundo da imagem. As imagens 5 e 6 representam uma foto capturada transformada para HSV e depois aplicada o filtro de cor, respectivamente.

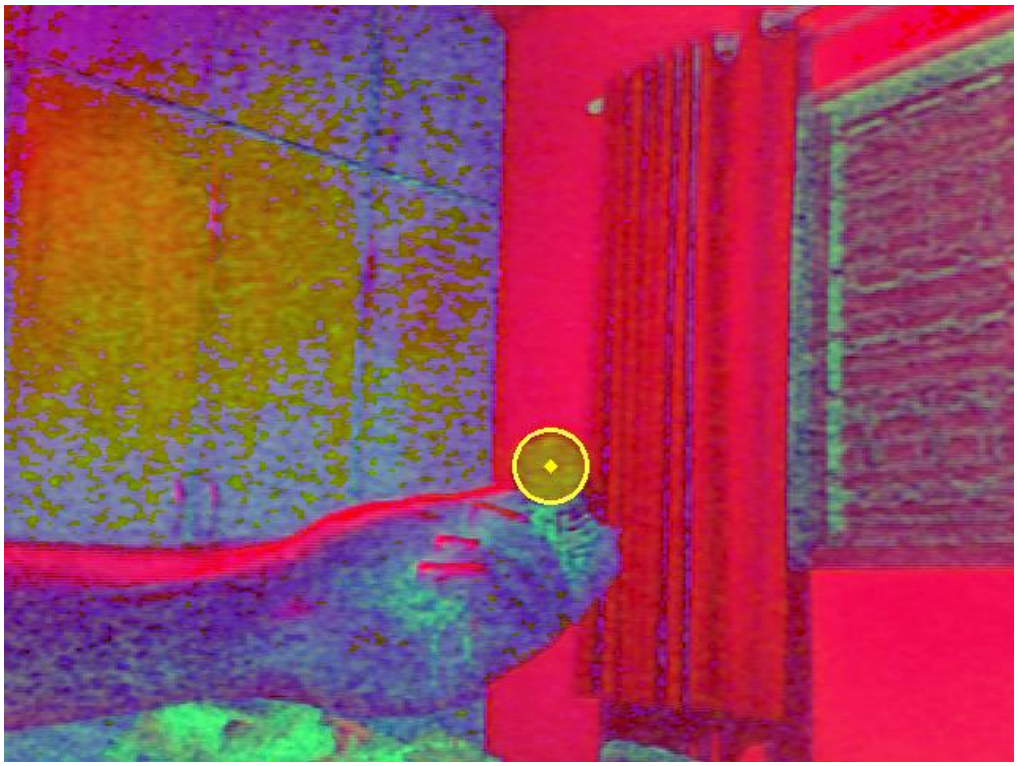

Figura 9: Foto transformada em HSV.

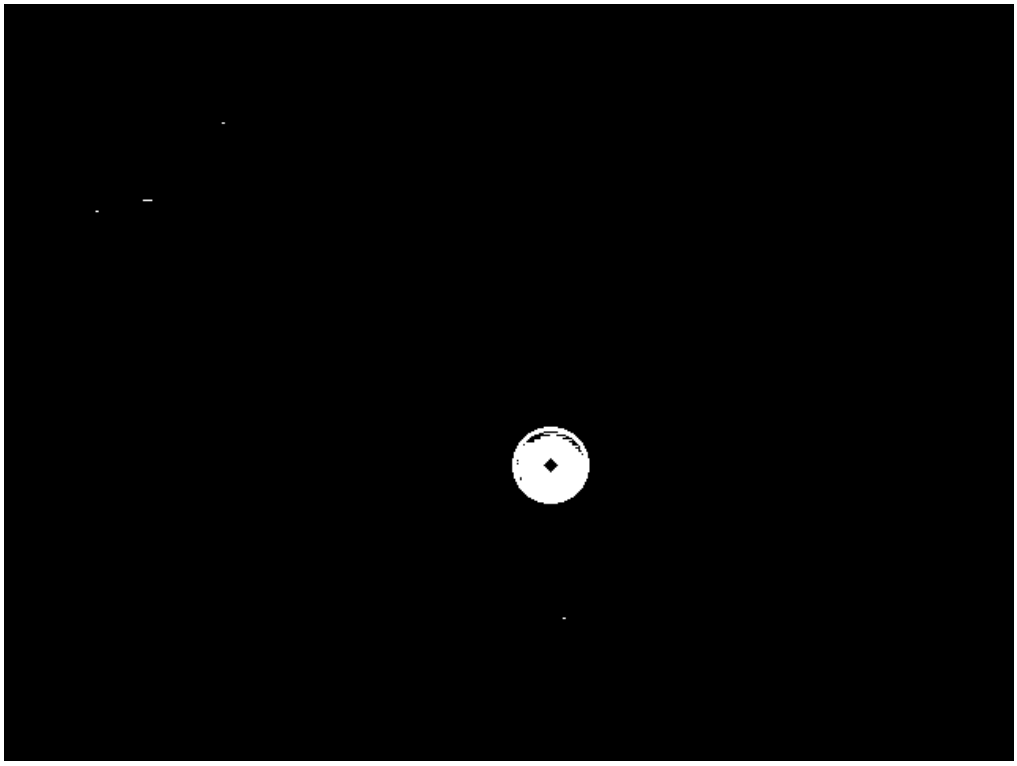

Figura 10: Foto após aplicação do filtro de cor.

O próximo passo é o filtro de Gauss, que suaviza a imagem e limpa os ruídos restantes. A princípio ele pode parecer redundante dado que antes é executado o filtro de cor, mas série de testes foi executada, analisando quantas vezes o método detectava o círculo corretamente, para verificar se o filtro de Gauss era mesmo necessário. Foi constatado que quando o filtro de Gauss é aplicado à detecção é mais constante e na maioria das vezes, mais precisa. Optou-se então por utilizar o filtro de Gauss no pré-processamento. A imagem 9 representa a foto acima depois da aplicação do filtro de Gauss.

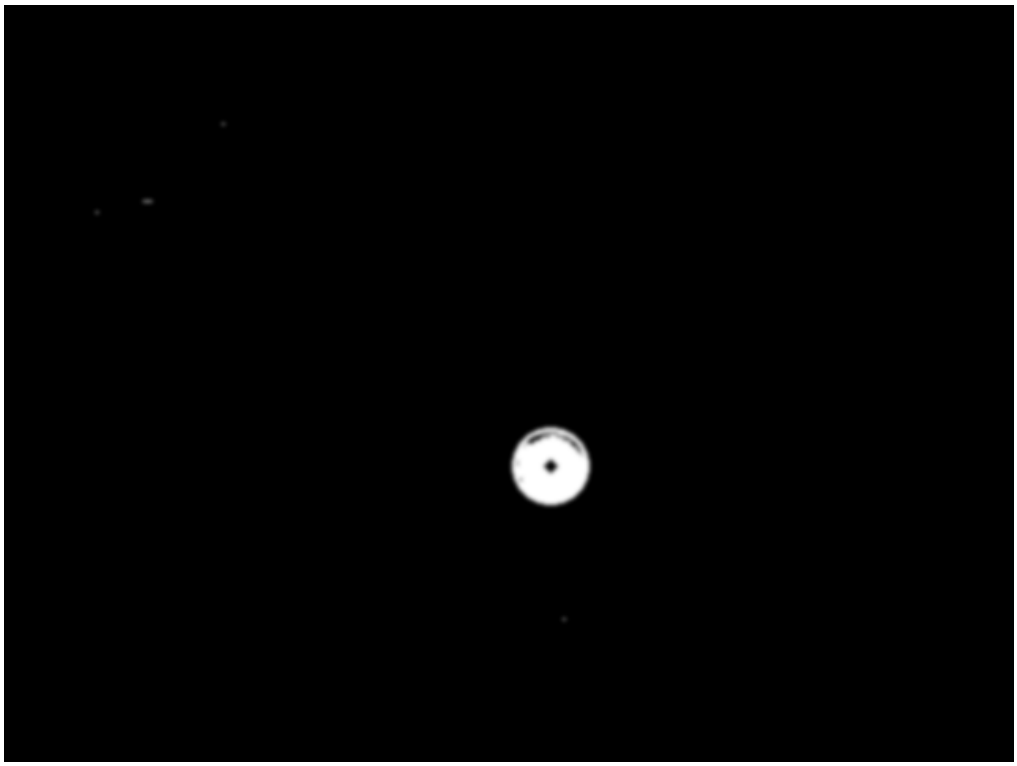

Figura 11: Foto após aplicação do filtro de Gauss.

Além do filtro de Gauss, usa-se outra medida é tomada para redução de ruídos, a chamada abertura da imagem. Abertura remove pequenos objetos no fundo da imagem, como os chamados ruídos "sal e pimenta", pequenos pixels corrompidos, causados por erro na transmissão de dados (da câmera para o sistema). A abertura da imagem consiste em uma operação de erosão, que efetivamente remove o ruído sal e pimenta, pois qualquer objeto que não tenha uma espessura expressiva. O problema com essa operação é que ela afeta a imagem inteira, corroendo também as bordas dos objetos de interesse, por isso realiza-se em seguida uma operação de dilatação, para que os objetos de interesse tenham suas bordas aumentadas, idealmente para o tamanho original. Essas operações também fazem um "polimento" do objeto, eliminando pixels da borda e tornando-a mais suave. As imagens 10 e 11 abaixo representam respectivamente o processo de erosão e o processo de dilatação.

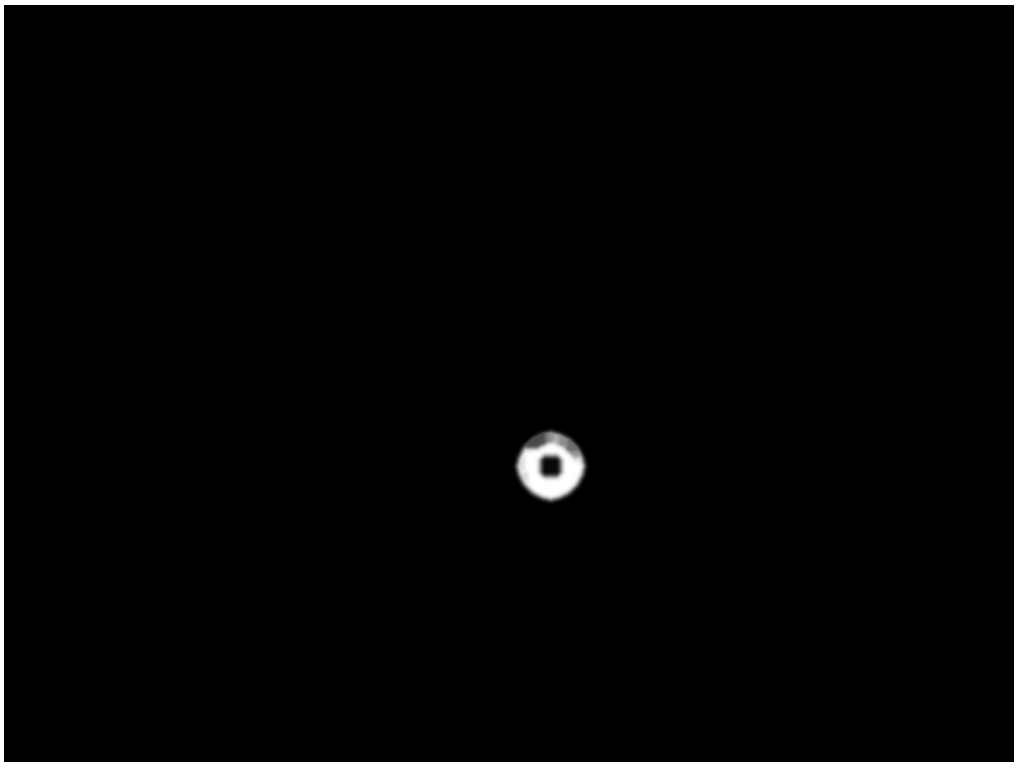

Figura 12: Foto após o processo de Erosão.

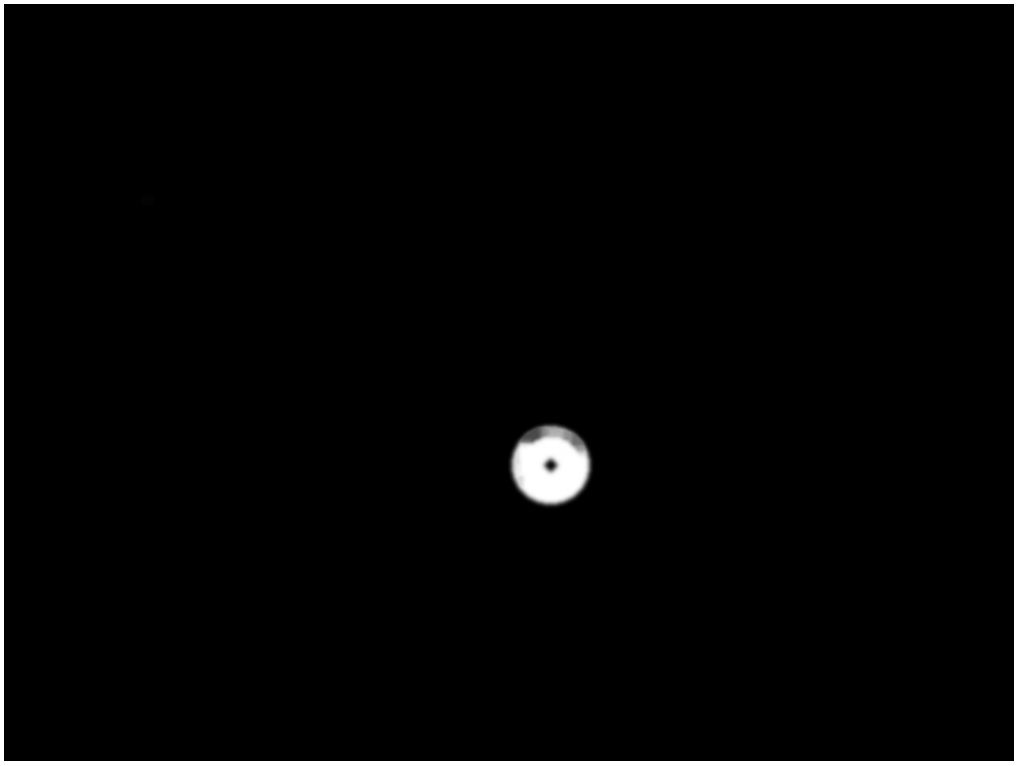

Figura 13: Foto após o processo de Dilatação.

Por último, aplica-se o filtro de Canny, que gera uma imagem apenas com as bordas detectadas (sem o preenchimento). Da mesma forma que o filtro acima, a execução de uma série de testes revelou que o filtro aumenta a precisão da transformada de Hough, melhorando o método de detecção. Esse filtro gera certo aumento no tempo de detecção, mas a melhora no método compensa em muito esse tempo aumentado, visto algumas vezes o método sem o filtro de Canny falhava e era necessário executá-lo mais de uma vez para extrair as informações do círculo. A imagem 12 abaixo mostra a aplicação do filtro de Canny e a imagem 13 mostra o círculo enfim detectado.

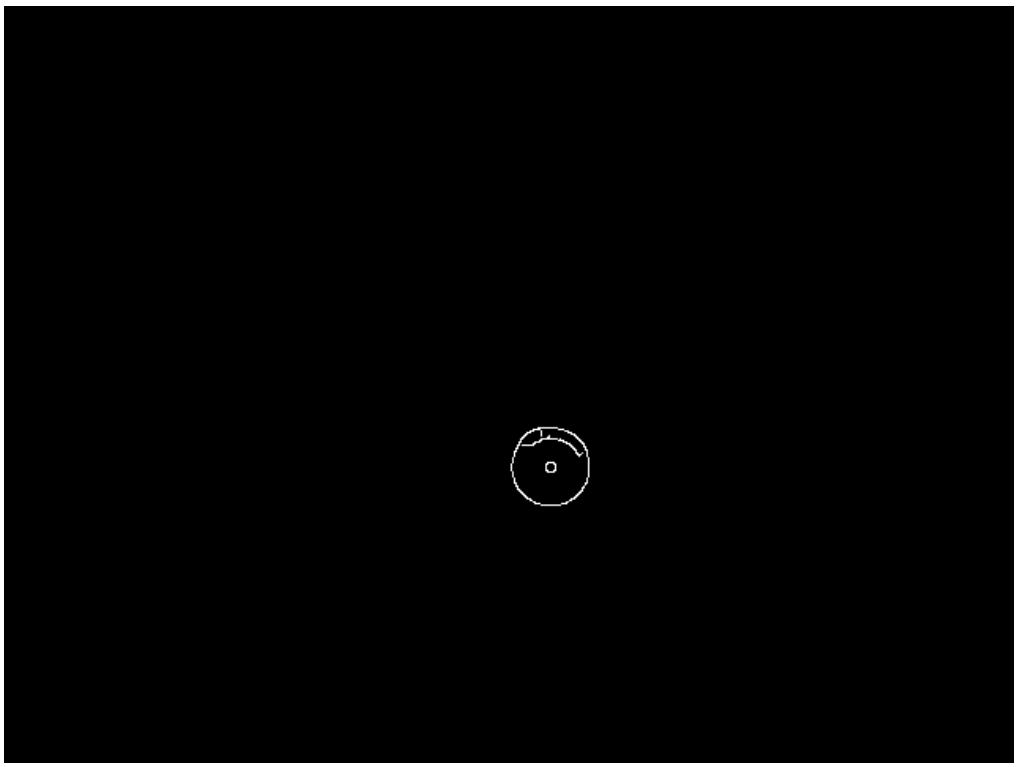

Figura 14: Foto após aplicação do filtro de Canny.

Figura 15: Círculo detectado pela transformada de Hough.

A imagem a seguir ilustra todo o processo de reconhecimento do círculo.

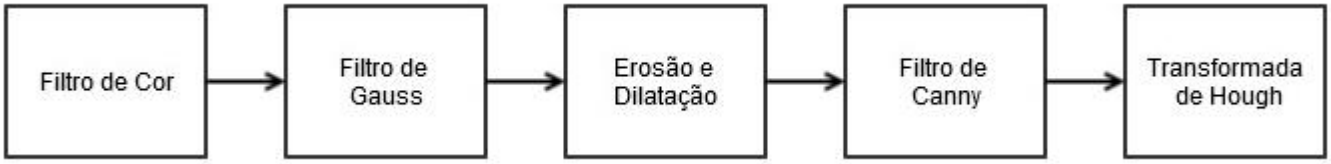

Figura 16: Processo de reconhecimento do círculo.

Foram feitas várias medidas do tempo de execução de cada etapa e do processo todo para identificar melhorias. Constatou-se que o tempo de execução melhorava consideravelmente se as imagens obtidas não forem mostradas. Como essa informação não é necessária para o sistema, essa parte retirada, deixando o sistema apenas com o essencial.

Depois disso, foi feita uma análise dos processos sendo executados no sistema operacional, para verificar se havia processos que poderiam ser parados para liberar processamento para o sistema. Uma das cargas do processador da Raspberry é o gerenciador de janelas, que não é otimizado para utilizar a unidade de processamento gráfico, todo o trabalho é feito pelo processador ARM [26]. Verificou-se que esse processo consumia grande parte do tempo de processamento. Como o acesso ao sistema embarcado sem a utilização de interface gráfica (utilizando SSH), esse processo não é necessário e, portanto foi terminado. Verificou-se também que é possível ajustar a prioridade do processo, para que ele seja executado acima dos outros processos, entretanto, o processo do sistema já é executado com prioridade máxima, portanto essa mudança não surtiu efeito.

5.5. Refinamento

Cada filtro apresenta uma série de parâmetros que precisam ser ajustados para que o funcionamento do filtro se adeque a esta aplicação. Abaixo há uma descrição dos parâmetros de cada filtro:

5.5.1. Filtro de Cor

- Variação do tom de Amarelo: é necessário definir um valor mínimo e um valor máximo, de forma que qualquer valor de um pixel compreendido entre o máximo e o mínimo é considerado como amarelo.

5.5.2. Filtro Gaussiano

- Tamanho do *Kernel*: Tamanho da região ao redor do pixel escolhido (ex: 7x7) no qual será aplicada a equação do filtro.

5.5.3. Detector de Borda de Canny

- Limiar alto de Histerese: Define quais bordas com certeza são genuínas. Aplica-se um limiar alto a imagem para marcas essas bordas. Começando com essas e usando as informações direcionais extraídas pelo filtro, as bordas podem ser traçadas.

- Limiar baixo de Histerese: Define quais bordas não são genuínas. Quando se está traçando uma borda, o limiar baixo permite encontrar seções mais fracas das bordas, desde que se encontre um ponto de começo.

5.5.4. Transformada de Hough

- Resolução do Acumulador: Relação entre o tamanho do acumulador utilizado na transformada e o tamanho da imagem
- Distância Mínima: A distância mínima entre o centro de dois círculos para que ambos sejam detectados
- Limiar Alto: O limiar alto passado para o filtro de Canny, que também define o limiar baixo, sendo que o limiar menor é duas vezes menor o que parâmetro passado.
- Limiar do Acumulador: Parâmetro utilizado na etapa de detecção para os centros dos círculos. Quanto menor esse valor, mais círculos falsos podem ser detectados.
- Raio Mínimo: Raio mínimo de um círculo que pode ser detectado
- Raio Máximo: Raio máximo de um círculo que pode ser detectado

Esses valores são escolhidos empiricamente, a partir de valores obtidos da literatura estudada, e então variados até que a detecção esteja perto do desejado.

Os primeiros valores definidos foram os limites do tom de amarelo. Em meio aos testes, foi descoberto que a câmera USB realiza um ajuste de histograma automático para melhorar o contraste da imagem. Isso pode parecer bom, mas distorce as cores em certos cenários (por exemplo, quando o círculo é muito grande). A biblioteca gráfica utilizada permite que esse ajuste automático seja configurado, entretanto a câmera não suporta esse ajuste. Para contornar esse problema, foi necessário ampliar o intervalo de valores considerados como amarelo. Foi implementada nos testes uma função que permitia observar os valores dos pixels da imagem, assim foi possível verificar quais valores dos pixels eram obtidos e ajustar o intervalo para obter a maior variação sem comprometer a detecção.

5.6. Montagem da Plataforma

A escolha da plataforma robótica representou uma parte importante do projeto. Era necessária uma plataforma capaz de acomodar a Raspberry, a câmera USB e a ponte H, além de ser possível controlar sua direção. Foi considerado uma plataforma automobilística, com 4 rodas e dois eixos, sendo que o eixo traseiro permanece imóvel e é conectado ao motor, e o eixo frontal controla a direção do chassi. Essa plataforma foi desconsiderada, pois apresentaria problemas com o controle do movimento. Para controlar a direção, seria necessário girar as rodas do eixo frontal para a posição desejada e movimentar a plataforma para frente até que todo o chassi estivesse virado para a direção desejada. Assim seria necessário controlar também o quanto a plataforma precisaria se movimentar para frente antes das rodas serem endireitadas e a distância percorrida nesse movimento, para que seja subtraída da distancia total calculada. Qualquer controle fora do principio do projeto (que é controlar apenas direção e distância) se torna desnecessário e apenas adiciona tempo computacional.

Figura 17: Plataforma automobilística [27].

A plataforma escolhida apresenta apenas duas rodas, cada uma conectada a um motor e girando independentes entre si, e uma "roda de apoio", que não pode ser controlada e gira em todas as direções. Essa configuração elimina o problema citado acima, já que para controlar sua direção, basta girar as rodas em sentidos opostos e a plataforma gira em seu centro, sem se deslocar.

A montagem do resto da parte física foi relativamente simples, bastou definir quais pinos da Raspberry seriam usados. A ponte H necessita de três pinos para o controle de cada motor, mais um pino para alimentação e outro para referência (pino que geralmente possui zero de tensão, comumente chamada de terra). Um dos três pinos de controle de motor tem a função de ligar ou desligar o motor. Os outros dois tem a função de controlar o sentido de rotação do motor, como foi explicado na sessão (por sessão). O pino de alimentação foi conectado a um conjunto de baterias de 6 volts separado da alimentação da Raspberry, por causa da quantidade de corrente envolvida na alimentação dos motores. O pino de terra é conectado ao conjunto de baterias e também a Raspberry, para fechar o circuito.

5.7. Cálculo da Distância e Ângulo e Movimento

O cálculo da distância e ângulo é o segundo desafio deste projeto, sendo o primeiro a detecção de objetos. Primeiramente considerou-se uma implementação envolvendo controladores PID. O sistema consegue extrair o raio do círculo e sua distância do centro. Essas informações seriam alimentadas para dois controladores PID separados, um que controlasse a distância e outro que controlasse o ângulo. Assim, seria necessário apenas ajustar os valores dos controladores, definir o erro do controlador de distância como sendo a diferença entre o valor do raio detectado e um valor de raio pré-definido e o erro do controlador de ângulo como sendo a distância lateral a partir do centro, e esperar o processo convergir para o erro próximo de zero. Assim seria possível implementar um sistema "online", que consegue seguir um circulo em movimento constante, uma vez que o controlador PID esteja configurado basta obter as informações necessárias e ele se encarrega de todo o trabalho.

Entretanto, esse método é muito custoso e necessita alimentação constante de informações, caso contrário, o erro não irá convergir, e o processo será executado indefinidamente. Constatou-se que a plataforma se locomove a uma velocidade de 40 cm/s (utilizando baterias com carga completa). Então, para que o erro seja menor que 4 cm, é necessário que o controlador seja alimentado com novas informações a cada 0.1 segundo, caso contrário, não será possível corrigir o movimento a tempo. Uma solução seria diminuir a velocidade da plataforma para que a utilização do controle PID fosse possível, mas uma velocidade pequena não é interessante para o projeto.

Por essas razões, preferiu-se um método mais simples que também gera resultados muito bons. É possível encontrar uma relação entre o raio em pixels detectado e a distância real do objeto. Seguindo [29], uma observação geral é que um objeto que se distância da câmera tem um tamanho menor na imagem. Conforme o objeto se aproxima da câmera seu tamanho aumenta. Essa observação aponta para fato que a profundidade do objeto tem uma relação direta com seu tamanho na imagem. Essa relação é encontrada obtendo-se várias medidas dos raios detectados a várias distâncias e interpolando esses resultados. Assim é possível obter uma distância bem próxima da distância real com um impacto computacional muito pequeno.

O ângulo do objeto em relação ao centro da imagem é calculado de forma semelhante. A relação entre distância lateral em pixels e a distância lateral real mantém-se constante para certa distância. Logo, calculou-se a relação entre as distâncias laterais a várias profundidades, e interpolaram-se os resultados. Dado uma profundidade, é possível calcular a relação entre as distâncias laterais e, portanto, a distância lateral real. A partir dessa informação, basta utilizar a relação trigonométrica da tangente no triângulo retângulo (cateto oposto dividido pelo cateto adjacente) e assim obter o ângulo a ser corrigido. A imagem abaixo ilustra o procedimento.

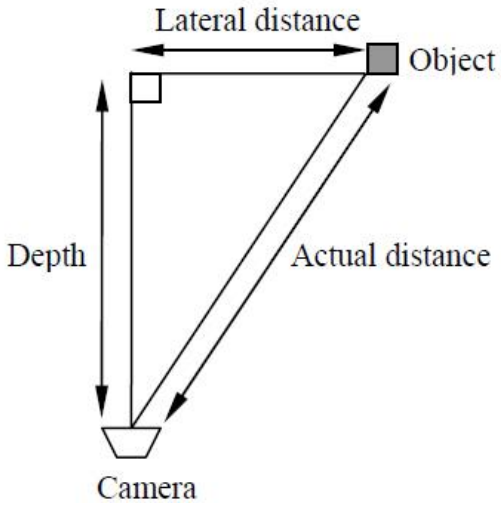

Figura 18: Procedimento de cálculo da distância real [29].

Abaixo se encontra o desvio padrão calculado para cada distância para ambas as câmeras.

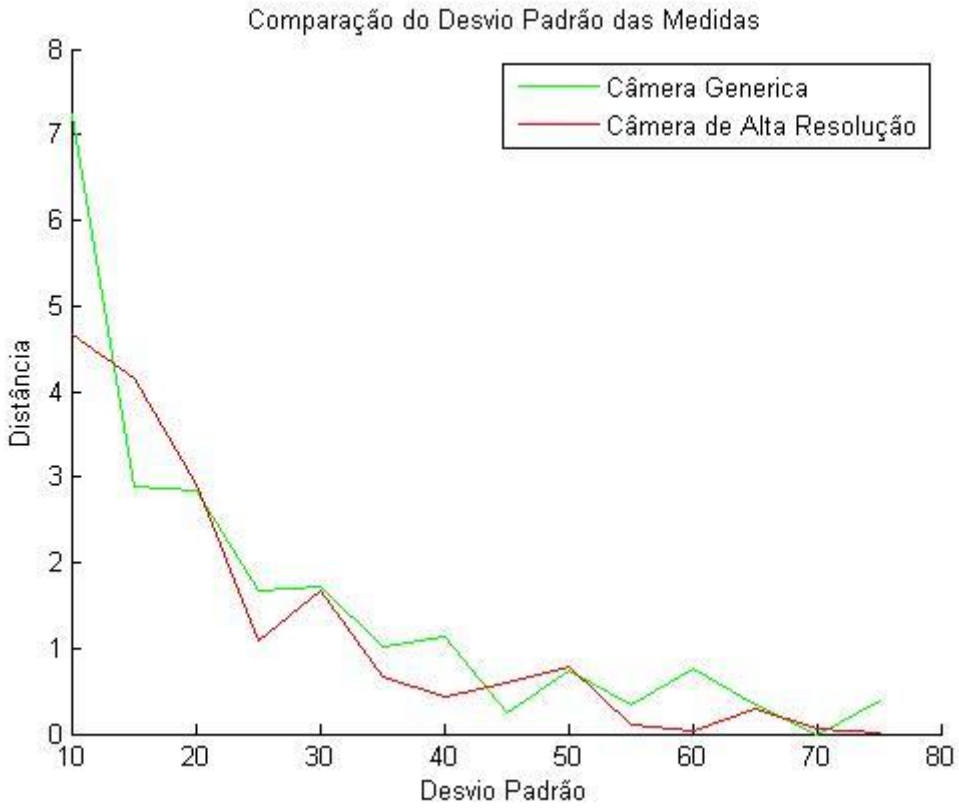

Figura 197: Desvio padrão das medidas das câmeras.

Observando o gráfico acima, é possível verificar que a câmera de alta resolução possui um desvio padrão menor que a câmera genérica na maioria das vezes. Um desvio padrão menor implica uma precisão maior, por tanto, resolveu-se utilizar a câmera de alta resolução no restante das medidas.

Como já foi citado, há um problema com a câmera utilizada no qual ela aplica uma equalização de histograma automaticamente, distorcendo as cores da imagem. Essa distorção ocorre quando o objeto a ser detectado se aproxima muito da câmera, em torno de 7 cm. Além disso, a distâncias menores que 10 cm a detecção começa a variar muito devido a essa distorção nas cores, o filtro de Canny começa a "quebrar" o círculo, dividindo-o em várias partes e confundindo a transformada de Hough, como pode ser visto na figura abaixo. Isso aumenta a variância da detecção e diminui sua estabilidade.

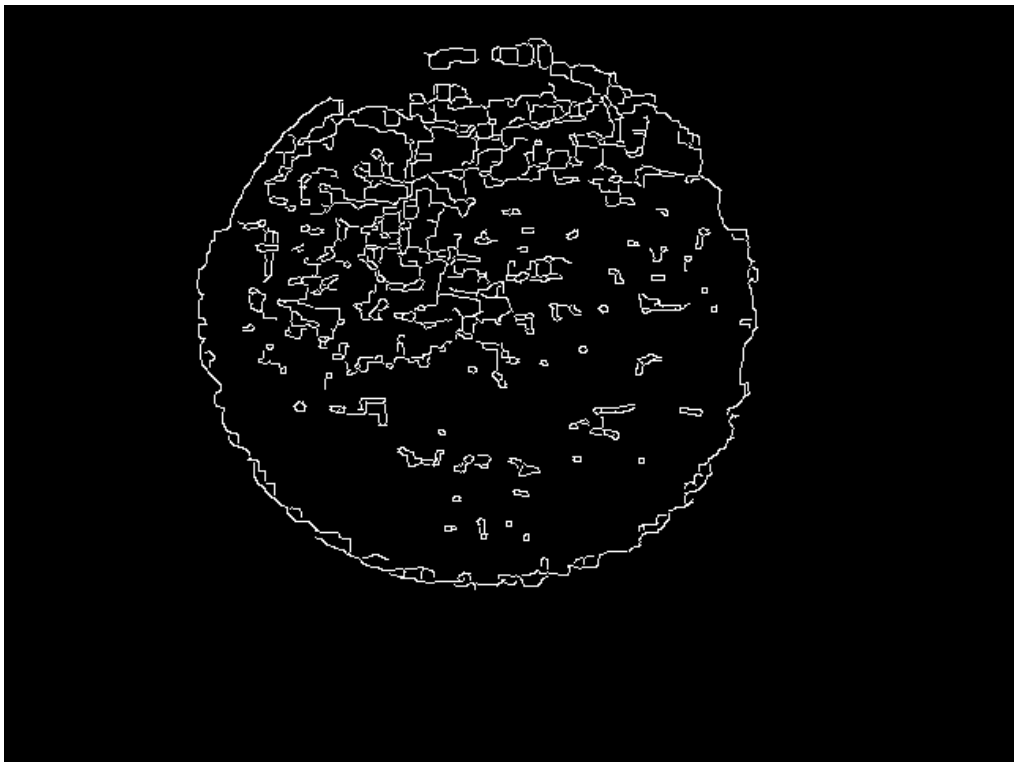

Figura 20: Círculo "quebrado" devido a distorções nas cores.

Uma pequena redução da velocidade de movimento da plataforma aumenta a precisão do movimento, possibilitando um melhor controle da distância percorrida. O único jeito de controlar a velocidade de um motor de corrente contínua é diminuindo a corrente que passa por ele. Isso é feito diminuindo a tensão aplicada. Entretanto, não é possível controlar a tensão de saída dos pinos da Raspberry, por tanto foi utilizado PWM com *duty cicle* de 80% nos pinos *Enable* da ponte H, obtendo uma tensão média de 4 V. A velocidade da plataforma dessa maneira foi em torno de 20 cm/s.

5.8. Integração

Por último, foi feita a integração das plataformas. Para tal, foi implementado um controlador geral que comanda ambos os sistemas, o Controle Mestre. Ele calcula e controla o movimento de acordo com as informações retiradas das imagens recebidas. A imagem abaixo mostra o funcionamento do sistema como um todo.

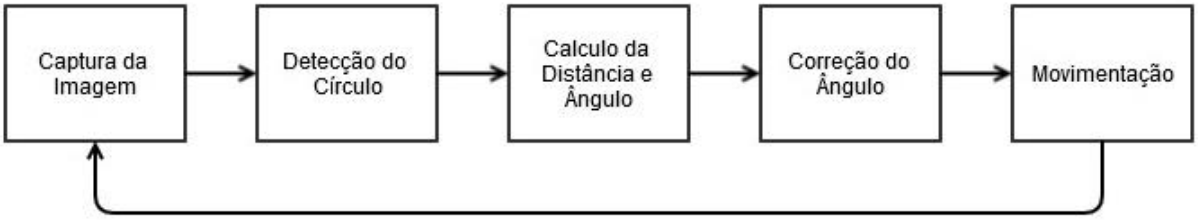

Figura 21: Ciclo completo do sistema.

De maneira simples, o Sistema de Visão captura uma imagem e extrai suas informações relevantes. O Controle Mestre recebe essas informações e calcula a distância a ser percorrida pelo sistema, e o angulo a ser corrigido. Ele então manda os comandos para o Sistema de Movimento, que corrige o angulo de acordo com o calculado e movimenta-se pela distância informada pelo Controle Mestre. Após o movimento, o ciclo se repete.

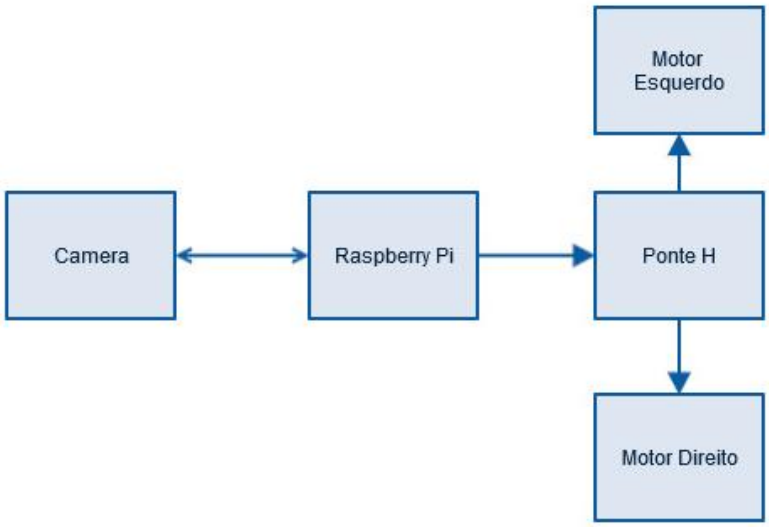

Figura 22: Integração dos dispositivos do sistema.

6. Resultados e Discussões

Os parâmetros do filtro de Gauss foram ajustados observando os resultados finais. Variou-se o valor do *kernel* até que os resultados fossem satisfatórios. Um kernel de tamanho pequeno não surte muito efeito na imagem, enquanto que um kernel de tamanho grande pode distorcê-la.

Os limiares do filtro de Canny foram ajustados observando o resultado obtido do filtro. Foi observado se as bordas detectadas condiziam com o círculo da imagem, ajustando os limiares. No caso desse filtro, levou-se em conta uma heurística que aconselha a usar um limiar alto três vezes maior que o limiar baixo [11].

Por fim, temos o método mais importante, a transformada de Hough. A resolução do acumulador influencia no raio mínimo detectado. Seu valor foi ajustado de acordo com a revisão bibliográfica e levando em conta o tempo de execução, já que uma resolução maior implica mais dados a serem processados. Manteve-se o tamanho do acumulador igual ao tamanho da imagem, pois um valor maior adicionaria mais tempo ao método.

Como deve ser detectado apenas um círculo na imagem, a distância mínima foi ajustada para ser o tamanho da imagem, assim garante-se que apenas um círculo será detectado por vez.

O valor do limiar passado ao filtro de Canny também foi ajustado com informações da revisão bibliográfica. O limiar do acumulador foi ajustado empiricamente, observando seu efeito nos resultados obtidos e encontrando o valor que produzisse uma detecção mais próxima do real.

Por fim, os valores do raio mínimo e máximo. O raio máximo foi definido de forma a não restringir o método, ajustando seu valor para um valor muito acima do que de fato seria possível detectar. O mesmo foi feito com o raio mínimo.

Para a interpolação da profundidade, a câmera foi fixada em um ponto a certa distância do círculo, e a distância foi aumentada a cada detecção. Foram feitas 50 detecções para cada distância. A maior profundidade foi definida como 75 cm, uma profundidade maior e o método não consegue mais detectar o círculo. A menor profundidade foi definida como 10 cm, pois uma profundidade menor causa um aumento da variância da detecção (uma diferença de 25 pixels entre duas detecções). O intervalo entre as distâncias foi de 5 cm. Esse processo foi feito para cada câmera, calculando-se a média e o desvio padrão. O resultado da interpolação está representado na figura abaixo.

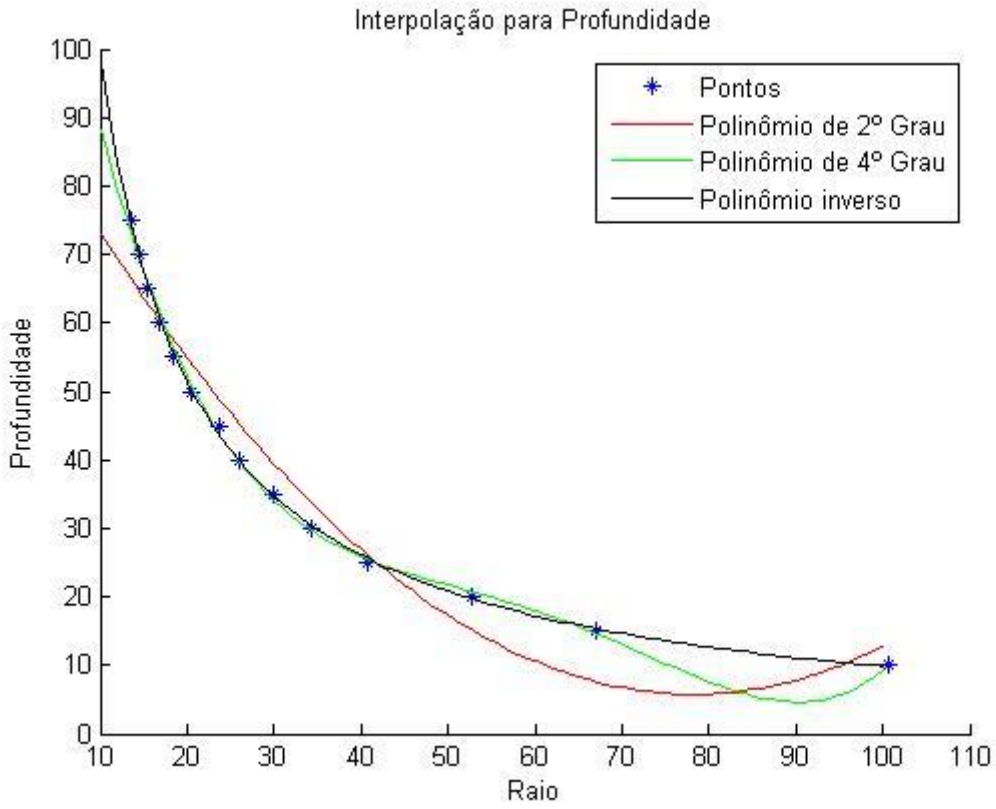

Figura 23: Interpolação dos dados de profundidade obtidos.

Observando os pontos obtidos escolheu-se interpolá-los utilizando três polinômios:

$$y = ax^2 + bx + c$$
$$y = ax^4 + bx^3 + cx^2 + dx + e$$
$$y = \frac{a}{x^b} + c$$

A interpolação encontra as constantes numéricas dos polinômios. Observando a imagem acima, é fácil perceber que o polinômio de quarto grau e o polinômio inverso apresentam resultados bem parecidos até $x = 40$, após esse valor, o polinômio inverso apresenta um erro menor e uma interpolação mais precisa. Esse polinômio foi utilizado no sistema para o calculo das distâncias.

A equação obtida foi a seguinte:

$$y = \frac{831.62}{x^{0.913}} - 2.556$$

Os resultados do cálculo da distância foram obtidos utilizando distâncias que não foram usadas para a interpolação (distâncias múltiplas de 5). Foram feitas 50 medidas e calculados a média e o desvio padrão de cada distância. A imagem abaixo ilustra o procedimento.

Figura 24: Resultado das medidas para profundidade de 62 cm.

O valor da média foi encontrado foi de 61,571 cm, com desvio padrão de 0,808. Abaixo se encontra o resultado para cada distância, em comparação com a distância real. Pode-se observar que apenas dois valores se distanciam da média. Abaixo se encontram os resultados para cada profundidade testada.

Distância	Média	Desvio Padrão	Diferença
13	13,49453188	0,887012529	-0,494531876
18	19,11083847	0,929734233	-1,11083847
22	23,46164263	1,418684932	-1,461642628
27	27,92465566	0,796496892	-0,924655664
33	32,9089354	0,911744933	0,091064597

38	37,23142046	1,248917101	0,76857954
42	41,43247953	1,030335953	0,567520468
47	47,00659117	2,317462392	-0,00659117
53	53,31217314	2,207595858	-0,31217314
58	58,06096776	1,67921114	-0,060967764
62	61,57138906	0,799481195	0,428610938
67	66,29145251	0,651249179	0,708547489
72	72,06994102	0,436145403	-0,069941024

Tabela 2: Resultado dos Cálculos de profundidade.

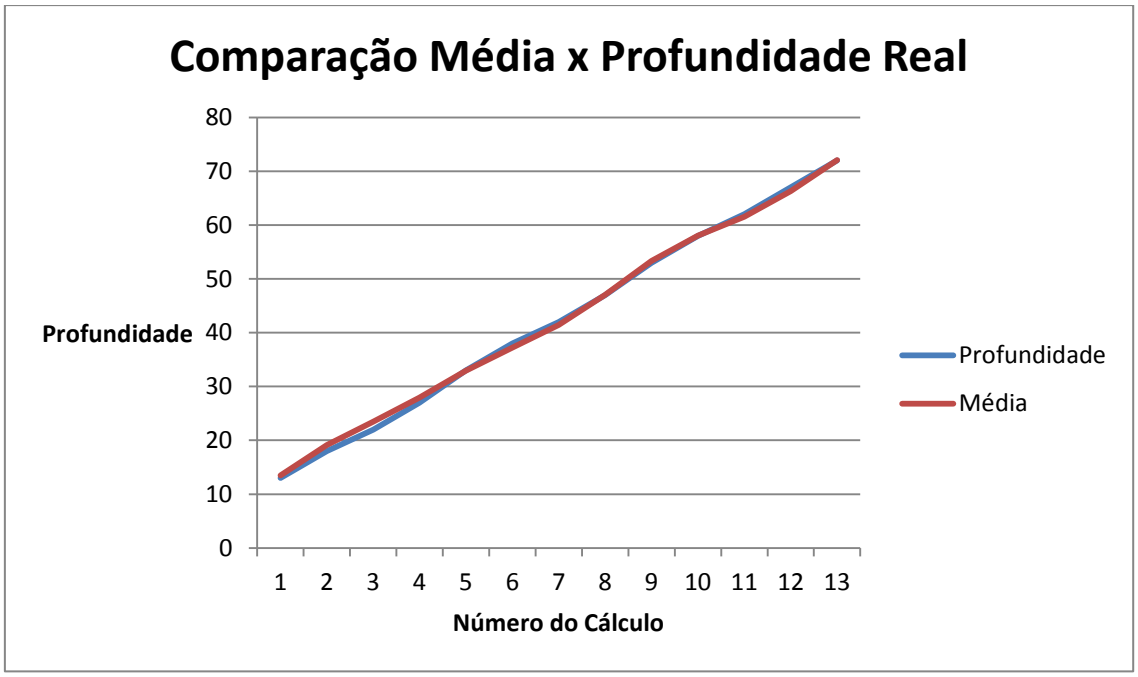

Figura 25: Comparação entre a média dos cálculos e a profundidade real.

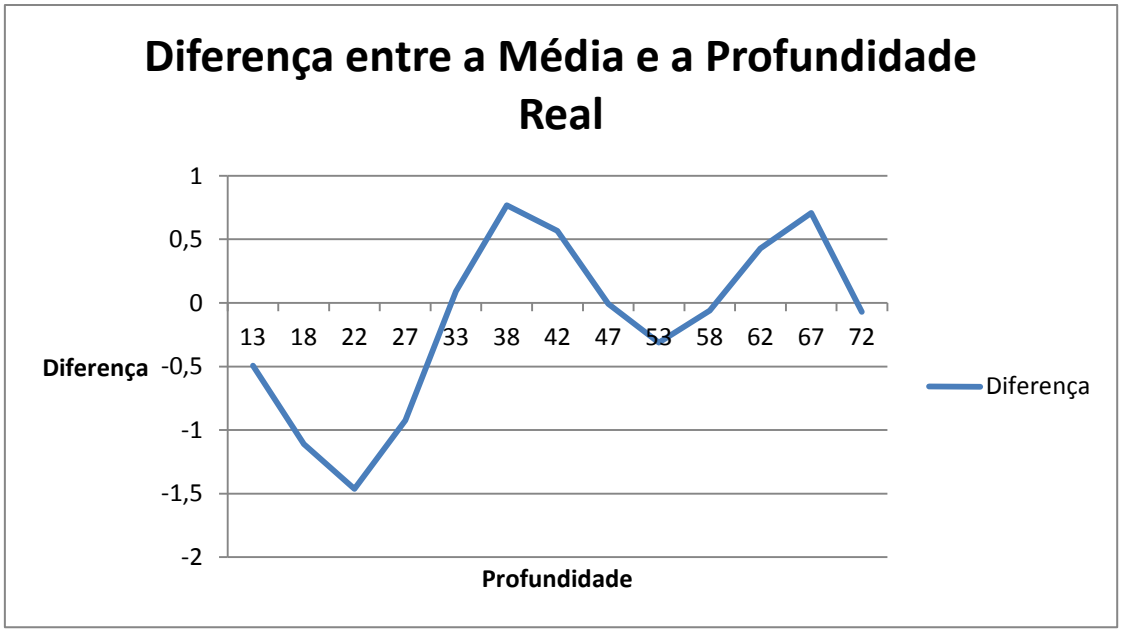

Figura 26: Diferença entre a média dos cálculos e a profundidade real.

Observando a figura 20, pode-se ver que o cálculo da profundidade está bem próximo da profundidade real. A figura 21 mostra a diferença entre esses dois valores, e pode-se ver que a maior diferença é -1,46. Entretanto, o desvio padrão de alguns cálculos ainda é grande, como por exemplo, para 47 cm de profundidade, obteve-se 2,2076. Com o objetivo de melhorar esses resultados, alterou-se o método de cálculo. Ao invés de fazer apenas uma detecção e utilizar as informações obtidas para realizar o cálculo, fizeram-se três detecções e utilizou-se a média das informações obtidas para realizar o cálculo. Dessa forma, se existir uma variação muito grande em uma das amostras, esse valor é balanceado na média com as outras amostras. Abaixo encontra-se os resultados obtidos dessa forma.

Profundidade	Média	Desvio Padrão	Diferença
13	13,504196	0,501200866	-0,504196001
18	19,16219591	0,512227847	-1,162195907
22	23,37774901	0,793824088	-1,377749009
27	27,77156975	0,257155579	-0,771569748
33	32,91569531	0,621757235	0,084304689
38	37,27201855	0,768392999	0,727981451
42	40,97086975	0,695311504	1,029130248

47	46,93949206	1,481547788	0,06050794
53	53,4675585	0,320578072	-0,467558499
58	58,37130093	0,948089761	-0,371300933
62	61,55406365	0,508445185	0,445936355
67	66,69297482	0,424181701	0,307025181
72	71,76284345	0,233504967	0,237156552

Tabela 3: Resultado dos cálculos de profundidade utilizando a média de três detecções para o cálculo da profundidade

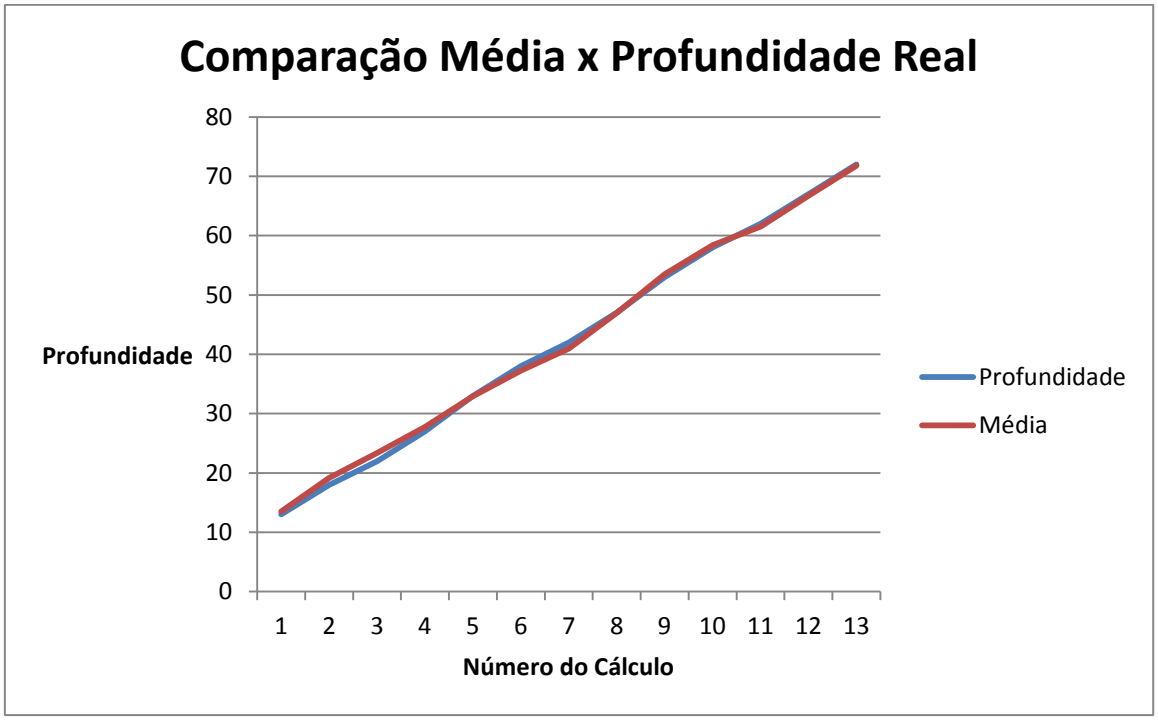

Figura 27: Comparação entre a média dos cálculos e a profundidade real utilizando a média de três detecções para o cálculo da profundidade

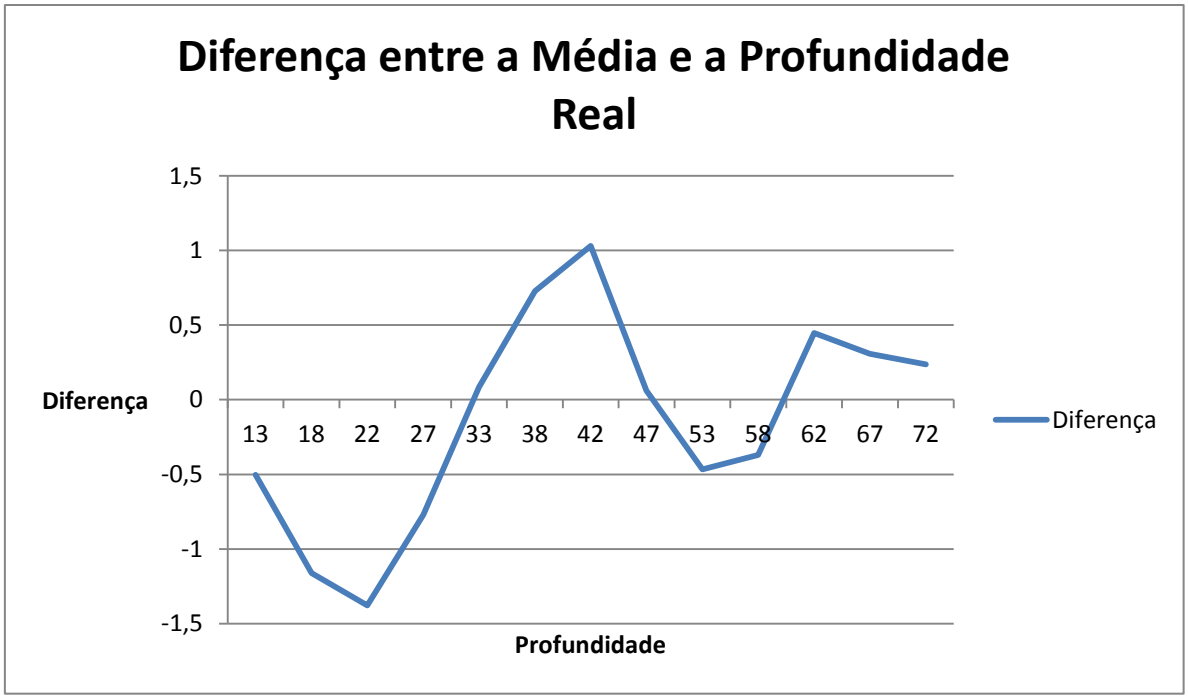

Figura 28: Diferença entre a média e a profundidade real utilizando a média de três detecções para o cálculo da profundidade.

Comparando os dados obtidos, vê-se que houve melhora nos cálculos da profundidade. O desvio padrão atinge um máximo de 1,4815, e os valores do desvio padrão são menores utilizando esse método. A diferença entre a profundidade real e a medida também diminuiu, atingindo um máximo de 1,029 cm. Portanto esse método gera resultados mais precisos. Essa melhoria tem um custo. Como são utilizadas três detecções para o cálculo da profundidade, o tempo necessário para obter um cálculo da profundidade triplica. Resolveu-se utilizar o método mais preciso, é uma decisão de projeto priorizar precisão a tempo.

Os resultados para o cálculo do ângulo foram obtidos da mesma forma que foram obtidos os resultados para o cálculo da profundidade. A câmera foi fixada a uma profundidade do círculo e a partir do centro, o círculo foi deslocado lateralmente, e feita uma detecção para o cálculo da distância lateral em pixels. Isso foi feito três vezes para cada profundidade, obtendo um razão entre deslocamento lateral real e em pixels. Esse processo foi feito 50 vezes para cada detecção, e feita a média. Abaixo encontram-se os pontos encontrados e a interpolação para distância lateral.

Esses resultados encontram-se abaixo.

O resultado da interpolação para distância lateral encontra-se abaixo:

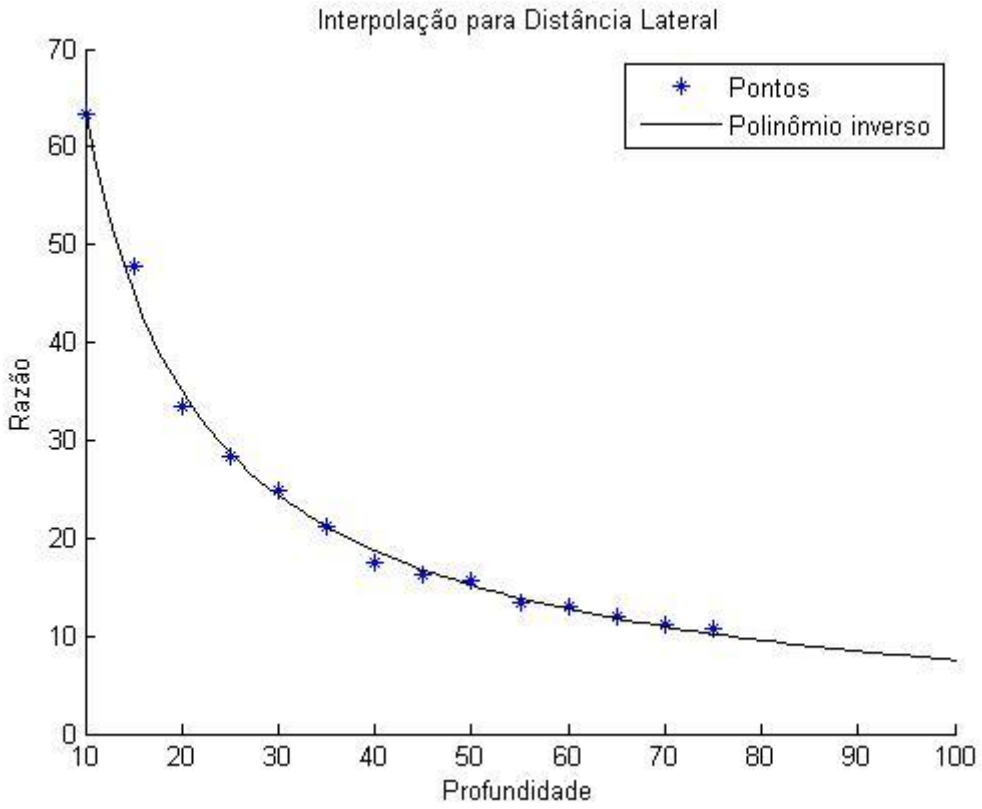

Figura 29: Resultado da interpolação para distância lateral.

A equação obtida foi a seguinte:

$$y = \frac{451.2049}{x^{0.8331}} - 2.1606$$

Com ambos os polinômios calculados, é possível calcular a profundidade e a distância lateral de círculos em posições arbitrárias em relação à câmera. Sendo assim, para confirmar a precisão do método, foram escolhidas alguns pontos aleatórios para que seja feito o cálculo e comparar o resultado com o valor real.

A tabela a seguir mostra o resultado de alguns cálculos feitos para testar o método utilizado.

Profundidade Real	Profundidade Calculada	Distância Lateral Real	Distância Lateral Calculada
36 cm	35.68 cm	7 cm	6.84 cm
53 cm	52.65 cm	12 cm	12.32 cm

46 cm	44.9 cm	17 cm	16.3 cm
62 cm	61.38 cm	20 cm	19.31 cm

Tabela 4: Comparação dos resultados obtidos para distância lateral.

Para a obtenção de tempo de execução, foram feitas 50 medidas detectando um círculo a média distância (54 cm), a pouca distância (10 cm), sem detectar um círculo e sem a execução do gerenciador de janelas. A tabela abaixo resume os resultados obtidos.

Detecção	Média	Desvio Padrão
Distância Média	0,602669573 s	0,152076521
Pouca Distância	0,614744344 s	0,102052241
Sem detecção	0,593784766 s	0,080612066
Sem "X"	0,597067962 s	0,135473583

Tabela 5: Resultados das medições dos tempos de execução.

Como pode ser observado, há pouca variação no tempo. Apesar de pequena, parando o processo do gerenciador de janelas obteve-se uma melhora no tempo de execução. Observa-se que há uma variação de aproximadamente 0,01 segundo dependo se o círculo está perto ou longe, ou se ele está presente.

Em relação ao tempo de execução, os resultados obtidos não foram suficientes para uma aplicação de tempo real. O tempo de 0,6 segundo para cada detecção é suficiente para a plataforma se movimentar em torno de 12 cm, o que pode levar a erros de detecção quando o círculo se movimenta e sai do campo de visão do sistema antes que ele possa detectá-lo. Entretanto, a proposta do projeto foi sucedida, o sistema se locomove seguindo um círculo detectado de forma precisa.

Conclui-se que é possível utilizar a Raspberry para processamento de imagens e visão computacional atingindo o objetivo proposto, entretanto, essa plataforma para não é a melhor escolha, considerando que há plataformas melhores com valores de mercado similares a Raspberry.

Como há um sistema operacional sendo executado sobre a Raspberry, muito poder de processamento é perdido atendendo a requisições irrelevantes ao processo. Conclui-se também que a câmera desempenha um papel fundamental nesse tipo de aplicação (processamento de imagem), visto que ruídos oriundos do hardware da câmera necessitam de filtragem e, portanto, perda de tempo computacional em relação a uma câmera que não apresenta esses ruídos.

O método utilizado para medir distâncias reais utilizando apenas informações visuais se provou bem preciso, obtendo um desvio máximo do valor real de apenas 1 cm, sendo ideal para aplicações em sistemas embarcados, devido a sua simplicidade e velocidade, basta apenas obter dados do sistema, e cálculos futuros são feitos com apenas um cálculo matemático.

Analisando o projeto durante o seu desenvolvimento, foi possível identificar várias possíveis melhoras. A primeira dela seria trocar o sistema embarcado. A Raspberry foi escolhida principalmente por seu baixo preço, mas a tecnologia avança tão rápido que já é possível encontrar sistemas melhores pelo mesmo preço, se não mais baratos, como por exemplo, a BeagleBone, que atualmente custa $55,00 [30]. Suas especificações para comparação seguem a baixo [31]:

- Processador Sitara AM33858BZCZ100, com *clock* de 1GHz, capaz de executar 2000 MIPS;
- GPU SGX530 3D, capaz de gerar 20M de polígonos por segundo;
- Memória SDRAM de 512 Mb DDR3L 606MHz;
- Memória Flash *onboard* de 4 Gb;
- Resolução de vídeo máxima de 1280x1024 HDMI.

Comparando as especificações de ambas, pode-se ver claramente que a BeagleBone é superior a Raspberry em todos os aspectos, exceto no saída de vídeo. Como a saída de vídeo não é relevante para este projeto, a BeagleBone seria uma escolha melhor de sistema embarcado.

Figura 30: Beaglebone [30]

Desconsiderando o fator preço, é fácil encontrar sistemas embarcados com muito mais poder computacional que ambas as placas citadas acima. Um exemplo é o ODROID-XU3, um sistema embarcado capaz de executar os sistemas operacionais Android 4.4 e XUbuntu 14.04. Atualmente custa US$179,00 [32]. Suas especificações seguem abaixo.

- Octa Core utilizando um processador Samsung Exynos5422 Cortex™-A15 2.0Ghz quad core e um processador Cortex™-A7 quad core CPUs;
- GPU Mali-T628 MP6 (OpenGL ES 3.0/2.0/1.1 e OpenCL 1.1 Full profile);
- Memória RAM de 2 Gbyte LPDDR3 RAM a 933MHz (14.9GB/s de banda de memória);
- Armazenamento de memória Flash utilizando soquete eMMC5.0 HS400;
- USB 3.0 Host x 1, USB 3.0 OTG x 1, USB 2.0 Host x 4;

Facilmente se observa que a ODROID-XU3 possui especificações melhores em todos os aspectos relevantes (processador, GPU, memória RAM, display).

Também é possível melhorar o desempenho utilizando um sistema operacional de tempo real. Dessa forma, as requisições do processo principal do projeto sempre serão atendidas com prioridade, cortando o tempo que o sistema operacional passa executando outros processos e disponibiliza todo o tempo para o processo principal.

Outra forma de melhorar o desempenho seria utilizar um sistema dedicado. Não são necessários todos os módulos oferecidos pelo sistema operacional, então um sistema operacional reduzido apenas ao essencial aloca todo o processador apenas para o projeto, aumentando significativamente a eficiência.

Em relação à precisão do movimento, seria possível substituir a plataforma utilizada por outra que possuísse os equipamentos necessários para controle de velocidade, e possivelmente uma câmera com uma resolução maior e mais liberdade para sua configuração (por exemplo, alterar o tempo de exposição, a abertura, etc) diminuiria a variação na detecção e, portanto, aumentaria a precisão do movimento.

7. Trabalhos Futuros

A ideia desse projeto pode ser ampliada e aplicada a vários problemas existentes. Aumentando o poder de processamento do sistema (com as melhorias sugeridas na sessão acima) seria possível implementar um sistema que detecta e reage às informações obtidas em tempo real. Essa aplicação poderia ser usada para encontrar e neutralizar minas em antigos campos minados, lugar onde o acesso de pessoas é perigoso. Basta fornecer a imagem de uma mina a ser reconhecida e ajustar o procedimento de ao ser detectada uma mina.

Esse sistema pode ser usado também para identificação de defeitos em tubulações. Pensando em tubulações industriais (como por exemplo, de petrolíferas), onde uma pessoa não tem acesso, é possível localizar defeitos dentro dessas tubulações, de forma análoga a aplicação descrita acima.

Seguindo essa lógica, é possível trocar a plataforma terrestre do sistema por uma plataforma aérea e expandir ainda mais suas aplicações. Algumas ideias seriam busca de buracos e objetos perigosos em rodovias, busca em plantações por plantas danificadas, e até busca por pessoas em áreas pré-determinadas, utilizando reconhecimento de face.

Apesar das aplicações citadas acima necessitarem de hardwares melhores e métodos mais precisos, todas compartilham a ideia deste trabalho, um sistema autônomo que se movimenta de acordo com informações obtidas através de processamento gráfico em tempo real.

8. Bibliografia

[1] S. Heath, *Embedded Systems Design*. 2003.

[2] W. Stallings, *Operating Systems, Internals and Design PrinciplesNo Title*. Prentice Hall PTR Upper Saddle River, NJ, USA ©2005, 2005, p. 820.

[3] Microsoft, "User mode and kernel mode," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff554836%28v=vs.85%29.aspx. [Accessed: 26-Oct-2014].

[4] T. Solomon, ChrisBreckon, *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. 2011.

[5] G. Stockman and Linda G. Shapiro, *Computer Vision*. Prentice Hall PTR Upper Saddle River, NJ, USA ©2001, 201AD, p. 608.

[6] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Communications of the ACM*, 1972.

[7] MathWorks, "Hough transform - MATLAB hough," 2014. [Online]. Available: http://www.mathworks.com/help/images/ref/hough.html. [Accessed: 10-Oct-2014].

[8] OpenCV, "Hough Line Transform — OpenCV-Python Tutorials 1 documentation," 2013. [Online]. Available: http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html. [Accessed: 10-Oct-2014].

[9] M. K. Agoston, *Computer Graphics and Geometric Modeling: Implementation and Algorithms*. 2005, p. 306.

[10] E. R. Dougherty, *An Introduction to Morphological Image Processing*. 1992, p. 161.

[11] OpenCV, "Canny Edge Detection," 2013. [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html. [Accessed: 12-Oct-2014].

[12] P. Katz, *Digital control using microprocessor*. 1981.

[13] A. Williams, *Microcontroller Projects Using the Basic Stamp*. Taylor & Francis, 2002.

[14] L. Kneip, "H Bridge Circuit Schematic," 2010. [Online]. Available: http://www.laurentkneip.de/H_bridges.html. [Accessed: 12-Oct-2014].

[15] J. R. B. Alan V. Oppenheim, Ronald W. Schafer, *Discrete-Time Signal Processing*. 2009, p. 1120.

[16] M. Murray, "Raspberry Pi Review," 2012. [Online]. Available: http://www.pcmag.com/article2/0,2817,2407058,00.asp. [Accessed: 26-Oct-2014].

[17] Newark, “Newark Store Raspberry Pi,” 2013. [Online]. Available: http://www.newark.com/raspberry-pi/raspberry-modb-512m/raspberry-pi-model-b-board/dp/68X0155. [Accessed: 22-Oct-2014].

[18] Broadcom Corporation, “BCM2835 ARM Peripherals,” 2012.

[19] Raspberrypi.org, “RPi Performance,” 2013. .

[20] T. Instruments, “L293, l293d quadruple half-h drivers,” 2004.

[21] S. Escalera, A. Fornés, O. Pujol, P. Radeva, G. Sánchez, and J. Lladós, “Blurred Shape Model for binary and grey-level symbol recognition,” *Pattern Recognit. Lett.*, vol. 30, no. 15, pp. 1424–1433, Nov. 2009.

[22] Y. Freund, R. E. Schapire, and P. Avenue, “A Short Introduction to Boosting,” vol. 14, no. 5, pp. 771–780, 1999.

[23] P. Shetty, “Circle Detection In Images,” 2011.

[24] R. Maini and H. Aggarwal, “Study and Comparison of Various Image Edge Detection Techniques,” *Int. J. Image Process.*, vol. 3, no. 1, pp. 1–12, 2010.

[25] Z. Othman, M. Rafiq, and A. Kadir, “Comparison of Canny and Sobel Edge Detection in MRI Images,” pp. 133–136.

[26] E. Upton, “Wayland Preview: optimizing X using Raspberry Pi GPU,” 2013. [Online]. Available: http://www.raspberrypi.org/wayland-preview/. [Accessed: 21-Oct-2014].

[27] “Automodelismo de Competição.” [Online]. Available: http://automodelocba.esporteblog.com.br/. [Accessed: 26-Oct-2014].

[28] “Labaratório de Garagem,” 2014. [Online]. Available: www.labdegaragem.org/. [Accessed: 24-Oct-2014].

[29] A. Rahman, A. Salam, M. Islam, and P. Sarker, “An Image Based Approach to Compute Object Distance,” *Int. J. Comput. Intell. Syst.*, vol. 1, no. 4, pp. 304–312, 2008.

[30] Adafruit, “Adafruit Beaglebone,” 2014. [Online]. Available: https://www.adafruit.com/products/1876. [Accessed: 22-Oct-2014].

[31] G. Coley, “BeagleBone Black System Reference Manual,” 2013.

[32] Odroid, “Odroid-XU3 Store,” 2013. [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127. [Accessed: 22-Oct-2014].

Apêndica A – Código Fonte

A seguir é apresentado todo o código fonte do sistema, divido em módulos pertinentes.

"VideoModule.py"

```
__author__ = 'Andre'

import cv2
import numpy as np
import time
import math

class VideoModule:
    def __init__(self, opencamera=0):
        #Variaveis dos metodos de processamento de imagens
        self.gauss = 7
        self.lowThresCanny = 40
        self.cannyRatio = 3
        self.circDist = 1000
        self.upperThres = 300
        self.lowerThres = 80
        self.minRadius = 5
        self.maxRadius = 250

        #Cores
        self.min_yellow = np.array([20, 70, 70], np.uint8)
        self.max_yellow = np.array([70, 255, 255], np.uint8)

        #Outras variaveis
        self.nframes = 5
        self.detection = 0
        self.fps = 0
        self.circles = None
        self.center = (320, 240)
        self.opencamera = opencamera
        self.isOpen = False

        self.camera = None
        self.video = None

    def openCamera(self):
        self.isOpen = True
        self.camera = cv2.VideoCapture(self.opencamera)
        self.camera.set(3, 640)
        self.camera.set(4, 480)
        self.camera.set(14, 0.0)

    def warmUp(self, n=10):
        if self.isOpen:
            for i in range(n):
                self.camera.read()

    def cleanUp(self):
        self.circles = None
        cv2.destroyAllWindows()
        self.camera.release()
        self.isOpen = False
```

```
    def takeashot(self):
        if self.camera.isOpened():
            _, frame = self.camera.read()
            frame = cv2.flip(frame, 1)

            imgHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
            yellow = cv2.inRange(imgHSV, self.min_yellow,
self.max_yellow)

            gauss = cv2.GaussianBlur(yellow, (self.gauss, self.gauss),
0)

            element = cv2.getStructuringElement(cv2.MORPH_RECT, (7,
7))
            erode = cv2.erode(gauss, element)
            dilate = cv2.dilate(erode, element)

            canny = cv2.Canny(dilate, 100, 300)

            circles = cv2.HoughCircles(canny,
cv2.cv.CV_HOUGH_GRADIENT, 3, 600,
                                       param1=240, param2=80,
minRadius=10, maxRadius=300)

            return circles


    def toogleDetection(self):
        if self.detection == 1:
            self.detection = 0
            self.circles = None
            print 'Deteccao Desliada'
        else:
            self.detection = 1
            print 'Deteccao Ligada'

    def toogleFPS(self):
        if self.fps == 1:
            self.fps = 0
            print 'FPS Desligado'
        else:
            self.fps = 1
            print 'FPS Ligado'

    #circle[0] = x
    #circle[1] = y
    #circle[2] = raio
    def drawCircles(self, frame):
        if self.circles is not None:
            for circle in self.circles[0, :]:
                radius = np.round(circle[2])
                cv2.circle(frame, (circle[0], circle[1]), 3, (0, 255,
0), -1, 8, 0)
                cv2.circle(frame, (circle[0], circle[1]), radius, (0,
0, 255), 3, 8, 0)
                cv2.line(frame, self.center, (circle[0], circle[1]),
(0, 0, 0), 2)

    def detectCircles(self, frame):
```

```
        start = time.time()
        #HSV
        imgHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        yellow = cv2.inRange(imgHSV, self.min_yellow, self.max_yellow)

        #Filtro da media?
        gauss = cv2.GaussianBlur(yellow, (self.gauss, self.gauss), 0)

        element = cv2.getStructuringElement(cv2.MORPH_RECT, (7, 7))
        erode = cv2.erode(gauss, element)
        dilate = cv2.dilate(erode, element)

        #remover?
        canny = cv2.Canny(dilate, 100, 300)

        circles = cv2.HoughCircles(canny, cv2.cv.CV_HOUGH_GRADIENT, 2,
600,
                                   param1=240, param2=80,
minRadius=10, maxRadius=300)
        print time.time() - start
        return circles

    def nDetect(self, n=10):
        if (not self.isOpen):
            self.openCamera()

        shots = 0
        fname = "ndetect" + str(n) + ".txt"
        file = open(fname, "w")
        circles = []
        times = []
        x = []
        y = []
        r = []

        file.write(str(n) + '\n')
        print 'Starting'
        while shots < n:
            start = time.time()
            circle = self.takeashot()
            if circle is not None:
                shots +=1
                x.append(circle[0][0][0])
                y.append(circle[0][0][1])
                r.append(circle[0][0][2])
                times.append(time.time() - start)
                print time.time() - start
                print str(circle)
                time.sleep(0.1)


        xmean = 0
        ymean = 0
        rmean = 0
        print "Starting analytics: Mean"
        for i in range(len(x)):
            file.write(str(x[i]) + ' ')
            xmean += x[i]
            file.write(str(y[i]) + ' ')
            ymean += y[i]
            file.write(str(r[i]) + ' ')
```

```
            rmean += r[i]
            file.write(str(times[i]) + '\n')

        xmean = xmean/n
        ymean = ymean/n
        rmean = rmean/n

        xvar = 0
        yvar = 0
        rvar = 0

        print 'Starting analytics: Variance'

        for i in range(len(x)):
            xvar += (x[i] - xmean)*(x[i] - xmean)
            yvar += (y[i] - ymean)*(y[i] - ymean)
            rvar += (r[i] - rmean)*(r[i] - rmean)

        xvar = xvar/n
        yvar = yvar/n
        rvar = rvar/n

        xdev = math.sqrt(xvar)
        ydev = math.sqrt(yvar)
        rdev = math.sqrt(rvar)

        file.write('\n')
        file.write('Mean Deviation Variance\n')
        file.write('x: ' + str(xmean) + ' ' + str(xdev) + ' ' +
str(xvar) + '\n')
        file.write('y: ' + str(ymean) + ' ' + str(ydev) + ' ' +
str(yvar) + '\n')
        file.write('r: ' + str(rmean) + ' ' + str(rdev) + ' ' +
str(rvar) + '\n')
        self.cleanUp()
        file.close()
        print 'Finished'


    def lockAndDetect(self, n=10):
        print 'Abrindo a camera...'
        self.openCamera()
        print 'Pronto!'
        while self.camera.isOpened():
            _, frame = self.camera.read()
            frame = cv2.flip(frame, 1)

cv2.circle(frame,(self.center[0],self.center[1]),1,(0,0,255),2)
            cv2.imshow('WebCam', frame)
            #print time.time() - start
            key = cv2.waitKey(1)

            if key == 113:
                self.nDetect(n)
            elif key == 27:
                break

    def screenshot(self, name, frame):
        cv2.imwrite(name, frame)

    def calibrate(self, nframes=0):
```

```
        def getHSV(event,x,y,flags,param):
            if event == cv2.EVENT_LBUTTONDBLCLK:
                print "BGR" + str(frame[y,x])
                print "HSV" +
str(cv2.cvtColor(np.uint8([[frame[y,x]]]), cv2.COLOR_BGR2HSV))

        cv2.namedWindow('WebCam')
        cv2.setMouseCallback('WebCam', getHSV)

        show_mask = 0
        show_gauss = 0
        show_edges = 0
        show_erode = 0
        show_H = 0

        text = None
        n = 0

        avg = 0

        circles = None

        print 'Abrindo a camera...'
        self.camera = cv2.VideoCapture(self.opencamera)
        self.camera.set(15, 0.0)

        #HSV
        #Yellow 20, 100, 100
        #Yellow 70, 255, 255

        #Black 0, 0, 0
        #Black 0, 0, 75
        min_yellow = np.array([20, 70, 70], np.uint8)
        max_yellow = np.array([70, 255, 255], np.uint8)

        print 'Pronto!'
        while self.camera.isOpened():

            _, frame = self.camera.read()
            frame = cv2.flip(frame, 1)

            if nframes == 0:
                start = time.time()
                #HSV
                imgHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
                yellow = cv2.inRange(imgHSV, self.min_yellow,
self.max_yellow)

                gauss = cv2.GaussianBlur(yellow, (self.gauss,
self.gauss), 0)

                element = cv2.getStructuringElement(cv2.MORPH_RECT,
(7, 7))
                erode = cv2.erode(gauss, element)
                dilate = cv2.dilate(erode, element)

                canny = cv2.Canny(dilate, 100, 300)
                #print "Canny: " + str(time.time() - start)
                circles = cv2.HoughCircles(canny,
cv2.cv.CV_HOUGH_GRADIENT, 3, 600,
```

```
                                            param1=240, param2=80,
minRadius=10, maxRadius=300)
                #print time.time() - start
                if circles is not None:
                    circles = np.uint16(np.around(circles))
                    for i in circles[0,:]:
                       print i
                       # draw the outer circle
                       cv2.circle(frame,(i[0],i[1]),i[2],(0,255,0),2)
                       # draw the center of the circle
                       cv2.circle(frame,(i[0],i[1]),2,(0,0,255),3)

            elif n%nframes == 0:
                start = time.time()
                #HSV
                imgHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
                yellow = cv2.inRange(imgHSV, self.min_yellow,
self.max_yellow)

                gauss = cv2.GaussianBlur(yellow, (self.gauss,
self.gauss), 0)

                element = cv2.getStructuringElement(cv2.MORPH_RECT,
(7, 7))
                erode = cv2.erode(gauss, element)
                dilate = cv2.dilate(erode, element)

                #remover?
                canny = cv2.Canny(dilate, 100, 300)
                #print "Canny: " + str(time.time() - start)
                circles = cv2.HoughCircles(dilate,
cv2.cv.CV_HOUGH_GRADIENT, 3, 600,
                                            param1=240, param2=80,
minRadius=10, maxRadius=300)
                print time.time() - start
                if circles is not None:
                    circles = np.uint16(np.around(circles))
                    for i in circles[0,:]:
                       print i
                       # draw the outer circle
                       cv2.circle(frame,(i[0],i[1]),i[2],(0,255,0),2)
                       # draw the center of the circle
                       cv2.circle(frame,(i[0],i[1]),2,(0,0,255),3)



            n += 1
            end = time.time()
            #Media de fps
            #avg = ((n-1)*avg + (end-start))/n

            #if self.fps == 1:
                #text = 'FPS: {0:.3f}'.format(1/avg)

            #cv2.putText(frame, text, (500, 460),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 2)

            if show_gauss == 1:
                cv2.imshow('WebCam', yellow)
            elif show_mask == 1:
                cv2.imshow('WebCam', imgHSV)
```

```
            elif show_edges == 1:
                cv2.imshow('WebCam', canny)
            elif show_erode == 1:
                cv2.imshow('WebCam', erode)
            elif show_H == 1:
                cv2.imshow('WebCam', dilate)
            else:
                cv2.imshow('WebCam', frame)

            #Alterar esta parte para a rasp
            key = cv2.waitKey(1)
            #Sair
            if key == 27:        #27 ASCII = 'ESC'
                break
            #Ativar/Desativar FPS
            elif key == 102:     #102 ASCII = 'f'
                n = 0
                self.screenshot(frame)
                print frame
                #self.toogleFPS()
            elif key == 113:     #111 ASCII = 'q'
                show_gauss = not show_gauss
                show_mask = 0
                show_edges = 0
                show_erode = 0
                show_H = 0
            elif key == 119:      #77 ASCII = 'w'
                show_mask = not show_mask
                show_edges = 0
                show_gauss = 0
                show_erode = 0
                show_H = 0
            elif key == 101:     #101 ASCII = 'e'
                show_edges = not show_edges
                show_mask = 0
                show_gauss = 0
                show_erode = 0
                show_H = 0
            elif key == 114:     #114 ASCII = 'r'
                show_erode = not show_erode
                show_gauss = 0
                show_mask = 0
                show_edges = 0
                show_H = 0
            elif key == 104:     #104 ASCII = 'h'
                show_H = not show_H
                show_edges = 0
                show_gauss = 0
                show_erode = 0
                show_mask = 0
            elif key == 115: #'s'
                self.saveImages(frame)


        print 'Finalizando'
        self.circles = None
        cv2.destroyAllWindows()
        self.camera.release()


    def saveImages(self, frame):
```

```
        imgHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        self.screenshot("HSV.png", imgHSV)

        yellow = cv2.inRange(imgHSV, self.min_yellow, self.max_yellow)
        self.screenshot("YellowThreshold.png", yellow)

        gauss = cv2.GaussianBlur(yellow, (self.gauss, self.gauss), 0)
        self.screenshot("Gauss.png", gauss)

        element = cv2.getStructuringElement(cv2.MORPH_RECT, (7, 7))
        erode = cv2.erode(gauss, element)
        self.screenshot("Erode.png", erode)

        dilate = cv2.dilate(erode, element)
        self.screenshot("Dilate.png", dilate)

        canny = cv2.Canny(dilate, 100, 300)
        self.screenshot("Canny.png", canny)

        circles = cv2.HoughCircles(dilate, cv2.cv.CV_HOUGH_GRADIENT,
3, 600,
                                   param1=240, param2=80,
minRadius=10, maxRadius=300)

        if circles is not None:
            circles = np.uint16(np.around(circles))
            for i in circles[0,:]:
                print i
                # draw the outer circle
                cv2.circle(frame,(i[0],i[1]),i[2],(0,255,0),2)
                # draw the center of the circle
                cv2.circle(frame,(i[0],i[1]),2,(0,0,255),3)

        self.screenshot("Circle.png", frame)

    def nAllTimes(self, n=10):
        print 'Starting'
        self.openCamera()
        self.warmUp(30)

        shots = 0

        capture = []
        HSV = []
        color = []
        gaussfilter = []
        erodefilter = []
        dilatefilter = []
        cannyfilter = []
        hough = []
        total = []

        while shots < n:
            init = time.time()
            _, frame = self.camera.read()
            frame = cv2.flip(frame, 1)
            captureTime = time.time() - init

            start = time.time()
            imgHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
            hsvTime = time.time() - start
```

```
            start = time.time()
            yellow = cv2.inRange(imgHSV, self.min_yellow,
self.max_yellow)
            yellowTime = time.time() - start

            start = time.time()
            gauss = cv2.GaussianBlur(yellow, (self.gauss, self.gauss),
0)
            gaussTime = time.time() - start

            element = cv2.getStructuringElement(cv2.MORPH_RECT, (7,
7))

            start = time.time()
            erode = cv2.erode(gauss, element)
            erodeTime = time.time() - start

            start = time.time()
            dilate = cv2.dilate(erode, element)
            dilateTime = time.time() - start

            start = time.time()
            canny = cv2.Canny(dilate, 100, 300)
            cannyTime = time.time() - start


            start = time.time()
            circles = cv2.HoughCircles(canny,
cv2.cv.CV_HOUGH_GRADIENT, 3, 600,
                                       param1=240, param2=80,
minRadius=10, maxRadius=300)
            houghTime = time.time() - start

            totaltime = time.time() - init
            print totaltime

            if circles is not None:
                print circles

            shots += 1

            capture.append(captureTime)
            HSV.append(hsvTime)
            color.append(yellowTime)
            gaussfilter.append(gaussTime)
            erodefilter.append(erodeTime)
            dilatefilter.append(dilateTime)
            cannyfilter.append(cannyTime)
            hough.append(houghTime)
            total.append(totaltime)

        #Capture
        fname = "capture" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(capture[i]) + '\n')
            mean += capture[i]

        mean = mean/n
```

```
        var = 0

        for i in range(n):
            var += (capture[i] - mean)*(capture[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

        #HSV
        fname = "HSV" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(HSV[i]) + '\n')
            mean += HSV[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (HSV[i] - mean)*(HSV[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

        #Color
        fname = "color" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(color[i]) + '\n')
            mean += color[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (color[i] - mean)*(color[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

        #Gauss
        fname = "gauss" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(gaussfilter[i]) + '\n')
            mean += gaussfilter[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (gaussfilter[i] - mean)*(gaussfilter[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
```

```
        file.close()

        #Erode
        fname = "erode" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(erodefilter[i]) + '\n')
            mean += erodefilter[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (erodefilter[i] - mean)*(erodefilter[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

        #Dilate
        fname = "dilate" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(dilatefilter[i]) + '\n')
            mean += dilatefilter[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (dilatefilter[i] - mean)*(dilatefilter[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

        #Canny
        fname = "canny" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(cannyfilter[i]) + '\n')
            mean =+ cannyfilter[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (cannyfilter[i] - mean)*(cannyfilter[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

        #Hough
        fname = "hough" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
```

```
            file.write(str(hough[i]) + '\n')
            mean += hough[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (hough[i] - mean)*(hough[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

        #Total
        fname = "total" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(total[i]) + '\n')
            mean += total[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (total[i] - mean)*(total[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

    def nTotalTime(self, n=10):

        c = raw_input()

        print 'Starting'
        self.openCamera()
        self.warmUp(30)

        shots = 0

        total = []

        while shots < n:
            init = time.time()
            _, frame = self.camera.read()
            frame = cv2.flip(frame, 1)

            imgHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

            yellow = cv2.inRange(imgHSV, self.min_yellow,
self.max_yellow)

            gauss = cv2.GaussianBlur(yellow, (self.gauss, self.gauss),
0)

            element = cv2.getStructuringElement(cv2.MORPH_RECT, (7,
7))

            erode = cv2.erode(gauss, element)
```

```
            dilate = cv2.dilate(erode, element)

            canny = cv2.Canny(dilate, 100, 300)

            circles = cv2.HoughCircles(canny,
cv2.cv.CV_HOUGH_GRADIENT, 3, 600,
                                       param1=240, param2=80,
minRadius=10, maxRadius=300)

            totaltime = time.time() - init
            print totaltime

            if circles is not None:
                print circles

            shots += 1

            total.append(totaltime)

        #Total
        fname = "totalTime" + str(n) + ".txt"
        file = open(fname, "w")
        mean = 0
        for i in range(n):
            file.write(str(total[i]) + '\n')
            mean += total[i]

        mean = mean/n
        var = 0

        for i in range(n):
            var += (total[i] - mean)*(total[i] - mean)

        dev = math.sqrt(var)
        file.write(str(mean) + ' ' + str(dev) + ' ' + str(var) + '\n')
        file.close()

    def nDetectLateral(self, n=10):
        if (not self.isOpen):
            self.openCamera()

        shots = 0

        circles = []
        times = []
        x = []
        r = []

        print 'Starting'
        while shots < n:
            start = time.time()
            circle = self.takeashot()
            if circle is not None:
                shots +=1
                x.append(circle[0][0][0])
                r.append(circle[0][0][2])
                times.append(time.time() - start)
                print time.time() - start
                print str(circle)
                time.sleep(0.1)
```

```
        xmean = 0
        rmean = 0

        print "Starting analytics: Mean"
        for i in range(len(x)):
            xmean += x[i]
            rmean += r[i]

        xmean = xmean/n
        rmean = rmean/n

        xvar = 0
        rvar = 0

        for i in range(len(x)):
            xvar += (x[i] - xmean)*(x[i] - xmean)
            rvar += (r[i] - rmean)*(r[i] - rmean)

        xvar = xvar/n
        rvar = rvar/n

        xdev = math.sqrt(xvar)
        rdev = math.sqrt(rvar)

        print 'x: ' + str(xmean) + ' ' + str(xdev)
        print 'r: ' + str(rmean) + ' ' + str(rdev)

        print 'Finished'

        return [x, r, xmean, rmean, xdev, rdev]

    def lockAndDetectLateral(self, n=10):
        print 'Abrindo a camera...'
        self.openCamera()
        print 'Pronto!'

        fname = "nDetectLateral" + str(n) + ".txt"
        file = open(fname, "a")

        while self.camera.isOpened():
            _, frame = self.camera.read()
            frame = cv2.flip(frame, 1)

cv2.circle(frame,(self.center[0],self.center[1]),1,(0,0,255),2)
            cv2.imshow('WebCam', frame)
            #print time.time() - start
            key = cv2.waitKey(1)

            if key == 113:

                [x, r, xmean, rmean, xdev, rdev] =
self.nDetectLateral(n)

                print 'Distance: ' + str(xmean - self.center[0])

                file.write('Radius: ' + str(rmean) + ' ' +str(rdev) +
'\n')
                file.write('X: ' + str(xmean) + ' ' +str(xdev) + '\n')
                file.write('Distance: ' + str(xmean - self.center[0])
+ '\n\n')
```

```
            elif key == 27:
                break

        self.cleanUp()
        file.close()
```

"MotorControl.py"

```
__author__ = 'Andre'

from RPIO import PWM
import RPi.GPIO as gpio
import time


#Utilizar RPi.GPIO
#Frequencia para Motor DC 10kHz
#Exemplo no teste
#mc = MotorControl()
#print 'PWM:'
#duty = raw_input()
#mc.test(7, 5000, duty)


#0.5 seg = 90
#GND = 25

class MotorControl():

    def __init__(self, duty=80):

        self.frequency = 1000
        self.duty = duty

        self.motorREn = 22
        self.motorRF = 16
        self.motorRB = 18

        self.motorLEn = 7
        self.motorLF = 13
        self.motorLB = 11

        gpio.setmode(gpio.BOARD)

        gpio.setup(self.motorLEn, gpio.OUT)
        gpio.setup(self.motorLF, gpio.OUT)
        gpio.setup(self.motorLB, gpio.OUT)
        self.pwmL = gpio.PWM(self.motorLEn, self.frequency)

        gpio.setup(self.motorREn, gpio.OUT)
        gpio.setup(self.motorRF, gpio.OUT)
        gpio.setup(self.motorRB, gpio.OUT)
        self.pwmR = gpio.PWM(self.motorREn, self.frequency)

    def test(self, pin, freq, duty):
        gpio.setup(pin, gpio.OUT)
        p = gpio.PWM(pin, freq)
        p.start(float(duty))
```

```
        print 'Press any key to stop'
        value = raw_input()

        p.stop()
        gpio.cleanup()

    def start(self):
        self.startL()
        self.startR()

    def changeFreq(self, freq):
        self.frequency = freq
        self.pwmL.ChangeFrequency(freq)
        self.pwmR.ChangeFrequency(freq)

    def startL(self):
        self.pwmL.start(float(self.duty))

    def stopL(self):
        gpio.output(self.motorLF, False)
        gpio.output(self.motorLB, False)

    def setPWML(self, pwm):
        self.pwmL.ChangeDutyCycle(pwm)

    def forwardL(self):
        gpio.output(self.motorLB, False)
        gpio.output(self.motorLF, True)

    def backwardL(self):
        gpio.output(self.motorLF, False)
        gpio.output(self.motorLB, True)

    def startR(self):
        self.pwmR.start(float(self.duty))

    def stopR(self):
        gpio.output(self.motorRF, False)
        gpio.output(self.motorRB, False)

    def setPWMR(self, pwm):
        self.pwmR.ChangeDutyCycle(pwm)

    def forwardR(self):
        gpio.output(self.motorRB, False)
        gpio.output(self.motorRF, True)

    def backwardR(self):
        gpio.output(self.motorRF, False)
        gpio.output(self.motorRB, True)

    def clean(self):
        gpio.cleanup()

    def turnRight(self, delay=0):
        self.start()
        self.backwardL()
        self.forwardR()

        if delay != 0:
```

```
            time.sleep(delay)
            self.stopL()
            self.stopR()

        self.pwmL.stop()
        self.pwmR.stop()

    def turnLeft(self, delay=0):
        self.start()
        self.forwardL()
        self.backwardR()
        if delay != 0:
            time.sleep(delay)
            self.stopL()
            self.stopR()

        self.pwmL.stop()
        self.pwmR.stop()

    def goForward(self, delay=0):
        self.start()

        self.forwardL()
        time.sleep(0.1)
        self.forwardR()

        if delay != 0:
            time.sleep(delay)
            self.stopR()
            self.stopL()

        self.pwmL.stop()
        self.pwmR.stop()

    def goBackward(self, delay=0):
        self.start()
        self.backwardR()
        self.backwardL()

        if delay != 0:
            time.sleep(delay)
            self.stopR()
            self.stopL()


        self.pwmL.stop()
        self.pwmR.stop()
```

"MasterControl.py"

```
__author__ = 'Andre'
from VideoModule import *
from MotorControl import *

class MasterControl:

    def __init__(self, opencamera=0):
        self.depthinvpol = [831.6204780016491, 0.913031178353066, -
2.559894453263738]
        self.lateralinvpol = [451.2049, 0.8331, -2.1606]
```

```
        self.lateralDistRatio = 0

        self.pwm = 80
        self.velocity = 23 #23 cm por segundo
        self.turn = 180 #graus por seg

        self.vm = VideoModule(opencamera);
        self.mc = MotorControl();

    #y = c[0]/(x^c[1]) + c[2]
    def inverseval(self, coef, x):
        return coef[0]/(x**coef[1]) + coef[2]

    def polyval(self, poly, x):
        y = 0
        for i in range(len(poly)):
            value = poly[i]*pow(x, i)
            y += value

        return y


    def depth(self, radius):
        return self.inverseval(self.depthinvpol, radius)

    def lateralDist(self, lateralPixelDist, radius):
        dep = self.depth(radius)
        ratio = self.inverseval(self.lateralinvpol, dep)
        print "Ratio: " + str(ratio)

        return lateralPixelDist/ratio

    def start(self):
        print 'Starting:'
        self.vm.openCamera()
        while self.vm.camera.isOpened():
            _, frame = self.vm.camera.read()
            frame = cv2.flip(frame, 1)

cv2.circle(frame,(self.vm.center[0],self.vm.center[1]),1,(0,0,255),2)
            cv2.imshow('WebCam', frame)
            #print time.time() - start
            key = cv2.waitKey(1)
            if key == 113: #'q'
                circle = None
                while circle is None:
                    circle = self.vm.takeashot()

                x = circle[0][0][0]
                r = circle[0][0][2]
                center = self.vm.center[0]

                leftOrRight = 0 #left
                lateralPixel = 0
                if x > center:
                    leftOrRight = 1
                    lateralPixel = x - center


                distance = self.depth(r)
```

```
                #lateralDistance = self.lateralDist(lateralPixel,
distance)

                print distance
                #print lateralDistance


            elif key == 27:
                break

        print 'Finalizando'
        cv2.destroyAllWindows()
        self.vm.camera.release()

    def startN(self, n=3):

        print 'Warming Up'
        self.vm.warmUp(20)

        print 'Starting:'
        self.vm.openCamera()
        while self.vm.camera.isOpened():
            #_, frame = self.vm.camera.read()
            #frame = cv2.flip(frame, 1)

#cv2.circle(frame,(self.vm.center[0],self.vm.center[1]),1,(0,0,255),2)
            #cv2.imshow('WebCam', frame)
            #print time.time() - start
            key = raw_input()
            if key == 'q': #'q'
                circle = None
                x = 0
                r = 0
                shots = 0
                while shots < n:
                    circle = self.vm.takeashot()
                    if circle is not None:
                        shots += 1
                        x += circle[0][0][0]
                        r += circle[0][0][2]
                        print circle[0][0][0], circle[0][0][2]

                x = x/n;
                r = r/n;

                print x, r

                center = self.vm.center[0]
                Left = 1
                lateralPixel = 0
                lateralPixel = x - center
                if lateralPixel < 0:
                    Left = 0
                    lateralPixel *= -1

                distance = self.depth(r)

                print "Distance: " + str(distance)
                print "Pixel Distance: " + str(lateralPixel)
                lateralDistance = self.lateralDist(lateralPixel, r)
                print "Lateral Distance: " + str(lateralDistance)
```

```
                angle =
math.atan(lateralDistance/distance)*180/math.pi

                print "Angle: " + str(angle)

                distance *= 0.8

                tempo = distance/self.velocity

                print "Tempo: " + str(tempo)

                self.mc.goForward(tempo)

            elif key == 'c':
                break

        print 'Finalizando'
        cv2.destroyAllWindows()
        self.vm.camera.release()

    def startandmove(self):
        #self.mc.start(self.pwm)
        print 'Starting:'
        self.vm.openCamera()

        print 'Esquentando'
        for i in range(20):
            _, frame = self.vm.camera.read()
        print 'Pronto!'
        key = raw_input()
        if key == 'd':
            circle = None
            while circle is None:
                circle = self.vm.takeashot()
            x = circle[0][0][0]
            r = circle[0][0][2]
            print 'Raio: ' + str(r)
            center = self.vm.center[0]

            leftOrRight = 0 #left
            lateralPixel = x - center

            distance = self.depth(r)

            lateralDistance = self.lateralDist(lateralPixel, distance)

            print 'Distance: ' + str(distance)
            print 'Lateral Distance:' + str(lateralDistance)
            angle = atan(lateralDistance/distance)
            angle = angle*180/math.pi

            #time = distance/self.moveratio
            #print 'Time: ' + str(time)
            #self.mc.goForward(time)
            #print lateralDistance

        elif key == 'q':
            print 'Finalizando'
            cv2.destroyAllWindows()
            self.vm.camera.release()
```

```
    def nDistance(self, n=10):
        if (not self.vm.isOpen):
            self.openCamera()

        r = []
        dist = []

        shots = 0
        self.vm.openCamera()
        #self.vm.warmUp(20)

        print 'Starting'
        while shots < n:
            circle = self.vm.takeashot()
            if circle is not None:
                shots +=1
                print str(circle)
                r = circle[0][0][2]
                print self.depth(r)
                #time.sleep(0.1)

        print 'Finalizando'
        cv2.destroyAllWindows()
        self.vm.camera.release()

    def nDistanceMean(self, n=10, mean=1):
        if (not self.vm.isOpen):
            self.vm.openCamera()
            self.vm.warmUp(50)

        shots = 0
        fname = "ndistance" + str(n) + "Mean" + str(mean) + ".txt"
        file = open(fname, "w")


        distance = []
        times = []
        x = []
        r = []

        file.write(str(n) + '\n')
        print 'Starting'
        while shots < n:

            start = time.time()

            tmpx = 0
            tmpr = 0
            tmpmean = 0
            while tmpmean < mean:
                circle = self.vm.takeashot()
                if circle is not None:
                    tmpmean += 1
                    tmpx += circle[0][0][0]
                    tmpr += circle[0][0][2]

            shots +=1
            tmpx = tmpx/mean
            tmpr = tmpr/mean
            x.append(tmpx)
```

```
            r.append(tmpr)
            d = self.depth(tmpr)
            distance.append(d)
            times.append(time.time() - start)
            #print time.time() - start
            print tmpx, tmpr, d
            time.sleep(0.1)


        xmean = 0
        rmean = 0
        dmean = 0

        print "Starting analytics: Mean"
        for i in range(len(x)):
            file.write(str(x[i]) + ' ')
            xmean += x[i]
            file.write(str(r[i]) + ' ')
            rmean += r[i]
            file.write(str(distance[i]) + ' ')
            dmean += distance[i]
            file.write(str(times[i]) + '\n')

        xmean = xmean/n
        rmean = rmean/n
        dmean = dmean/n

        xvar = 0
        rvar = 0
        dvar = 0

        print 'Starting analytics: Variance'

        for i in range(len(x)):
            xvar += (x[i] - xmean)*(x[i] - xmean)
            rvar += (r[i] - rmean)*(r[i] - rmean)
            dvar += (distance[i] - dmean)*(distance[i] - dmean)

        xvar = xvar/n
        rvar = rvar/n
        dvar = dvar/n

        xdev = math.sqrt(xvar)
        rdev = math.sqrt(rvar)
        ddev = math.sqrt(dvar)

        file.write('\n')
        file.write('Mean Deviation Variance\n')
        file.write('x: ' + str(xmean) + ' ' + str(xdev) + ' ' +
str(xvar) + '\n')
        file.write('r: ' + str(rmean) + ' ' + str(rdev) + ' ' +
str(rvar) + '\n')
        file.write('d: ' + str(dmean) + ' ' + str(ddev) + ' ' +
str(dvar) + '\n')
        self.vm.cleanUp()
        file.close()
        print 'Finished'

    def lockAndDistance(self, n=10):
        print 'Abrindo a camera...'
        self.vm.openCamera()
```

```
        print 'Pronto!'
        while self.vm.camera.isOpened():
            _, frame = self.vm.camera.read()
            frame = cv2.flip(frame, 1)

cv2.circle(frame,(self.vm.center[0],self.vm.center[1]),1,(0,0,255),2)
            cv2.imshow('WebCam', frame)
            #print time.time() - start
            key = cv2.waitKey(1)

            if key == 113:
                self.nDistance(n)
            elif key == 27:
                break

    def nLateralDistanceMean(self, n=10, mean=1):
        if (not self.vm.isOpen):
            self.vm.openCamera()
            self.vm.warmUp(50)

        shots = 0
        fname = "ndistance" + str(n) + "Mean" + str(mean) + ".txt"
        file = open(fname, "w")


        distance = []
        dep = []
        times = []
        x = []
        r = []

        file.write(str(n) + '\n')
        print 'Starting'
        while shots < n:

            start = time.time()

            tmpx = 0
            tmpr = 0
            tmpmean = 0
            while tmpmean < mean:
                circle = self.vm.takeashot()
                if circle is not None:
                    tmpmean += 1
                    tmpx += circle[0][0][0]
                    tmpr += circle[0][0][2]

            shots +=1
            tmpx = tmpx/mean
            tmpr = tmpr/mean
            x.append(tmpx)
            r.append(tmpr)
            offset = tmpx - 320
            if offset < 0:
                offset = offset*-1

            d = self.lateralDist(offset, tmpr)
            distance.append(d)
            dep.append(self.depth(tmpr))
            times.append(time.time() - start)
            #print time.time() - start
```

```
            print tmpx, tmpr, d
            time.sleep(0.1)


        xmean = 0
        rmean = 0
        dmean = 0
        depmean = 0

        print "Starting analytics: Mean"
        for i in range(len(x)):
            file.write(str(x[i]) + ' ')
            xmean += x[i]
            file.write(str(r[i]) + ' ')
            rmean += r[i]
            file.write(str(distance[i]) + ' ')
            dmean += distance[i]
            file.write(str(times[i]) + '\n')
            depmean += dep[i]

        xmean = xmean/n
        rmean = rmean/n
        dmean = dmean/n
        depmean = depmean/n

        xvar = 0
        rvar = 0
        dvar = 0
        depvar = 0

        print 'Starting analytics: Variance'

        for i in range(len(x)):
            xvar += (x[i] - xmean)*(x[i] - xmean)
            rvar += (r[i] - rmean)*(r[i] - rmean)
            dvar += (distance[i] - dmean)*(distance[i] - dmean)
            depvar += (dep[i] - depmean)*(dep[i] - depmean)

        xvar = xvar/n
        rvar = rvar/n
        dvar = dvar/n
        depvar = depvar/n

        xdev = math.sqrt(xvar)
        rdev = math.sqrt(rvar)
        ddev = math.sqrt(dvar)
        depdev = math.sqrt(depvar)

        file.write('\n')
        file.write('Mean Deviation Variance\n')
        file.write('x: ' + str(xmean) + ' ' + str(xdev) + ' ' +
str(xvar) + '\n')
        file.write('r: ' + str(rmean) + ' ' + str(rdev) + ' ' +
str(rvar) + '\n')
        file.write('depth: ' + str(depmean) + ' ' + str(depdev) + ' '
+ str(depvar) + '\n')
        file.write('dl: ' + str(dmean) + ' ' + str(ddev) + ' ' +
str(dvar) + '\n')
        self.vm.cleanUp()
        file.close()
        print 'Finished'
```

“Run.py”

```
__author__ = 'Andre'
from MasterControl import *

mc = MasterControl(0)
mc.startN()
```