Felipe Fontes de Almeida
Luiz Felipe França Amadio

Projeto IPP - Interceptador de projéteis em um plano

São Paulo – SP
2013

Felipe Fontes de Almeida
Luiz Felipe França Amadio

Projeto IPP - Interceptador de projéteis em um plano

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a Conclusão do Curso de Engenharia Elétrica com Ênfase em Sistemas Eletrônicos.

Área de concentração: Engenharia de Sistemas Eletrônicos

Orientador:
Prof. Dr. Felipe Pait

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
DEPARTAMENTO DE ENGENHARIA DE SISTEMAS ELETRÔNICOS

São Paulo - SP
2013

"If I have seen farther than others,
it is because I stood on the shoulders of giants."
— SIR ISAAC NEWTON

AGRADECIMENTOS

Primeiramente à nossa família pelo apoio que nos deram e nos têm dado durante todo o período da elaboração e da execução deste projeto e ao longo do curso de engenharia de sistemas eletrônicos.

Um obrigado em especial aos nossos colegas de curso, e professores desta e de outras instituições de ensino que, sem os respectivos apoios, seria impossível a realização deste trabalho.

Ao Prof. Dr. Felipe Pait, nosso orientador, que foi de suma importância em todas as etapas do projeto, nos acompanhando e sempre oferecendo os melhores conselhos e orientações.

Agradecemos todas as pessoas que fizeram com que este trabalho caminhasse em direção ao sucesso! Em especial aos professores coordenadores do curso de conclusão, Prof. Dr. Marcelo Zuffo e Profª. Drª. Ramona Straube.

Felipe Fontes de Almeida

Luiz Felipe França Amadio

Resumo

O presente projeto propõe desenvolver um equipamento mecatrônico capaz de interceptar projéteis que se movimentam em um plano através de um projétil atirado. A execução se dará através de um sistema que contará com um canhão alimentado por ar comprimido, câmeras que reconhecem um objeto em movimento, servo motores que serão responsáveis pela movimentação do dispositivo, da utilização de uma placa que se utiliza da tecnologia Arduino e da elaboração e aperfeiçoamento de algoritmos de interceptação.

Palavras-chave: análise de imagens, rastreamento de objetos, microcontroladores, algoritmos de detecção.

Abstract

This project proposes to develop a mechatronic equipment capable of intercepting projectiles moving in a plane through a thrown projectile. The implementation will be through a system that will feature a pneumatic cannon, a camera that recognize and filters an object in motion, servo motors that will be responsible for handling the device, an IC board that uses Arduino technology and the development and refinement of interception algorithms.

Keywords: tracking loops, image motion analysis, object detection, microcontrollers, detection algorithms.

Lista de Figuras

Sumário

1 INTRODUÇÃO

1.1 IMPORTÂNCIA

A capacidade de interceptação de objetos lançados tem grande aplicação prática em diversos ramos de estudos. É fato que atualmente, com um sistema correto de captação de imagens, pode-se precisamente calcular e prever qual será a trajetória de um objeto para um futuro próximo, e a partir daí, interceptar o objeto com um projétil.

Este projeto tem certamente uma aplicação imediata na área militar, tendo sido extraído e tendo como base o sistema israelense de defesa contra mísseis, artilharia e morteiros, Iron Dome. Podemos ir além e pensar que poderíamos interceptar meteoritos, ou asteroides.

Com base nesse aspecto principalmente, consideramos que um interceptador, pode ser infinitamente útil quando se trata da segurança e defesa, possuir um alto poder mercadológico na área de entretenimento, além de potencialmente salvar inúmeras vidas se aplicado com este propósito.

Além das aplicações imediatas, a ideia por trás do projeto, nos permite fazer pensar e refletir sobre algoritmos que preveem movimentos, amplamente utilizados nas áreas de eletrônica através de sensores. Os conceitos matemáticos envolvidos, embora relativamente simples, são suficientemente adaptáveis para os mais diversos tipos de projeto por meio de bibliotecas de código já existentes e disponíveis na literatura.

A robótica, que é tratada neste projeto abordando seus conceitos de forma simplificada, é objeto de constante estudo nos dias atuais, potencializada pelas bem desenvolvidas técnicas de controle e sistemas. Conseguimos enxergar seu rápido desenvolvimento independentemente de vocação científica, visto que está presente no nosso cotidiano, tanto quanto em todos os setores da economia.

1.2 *Objetivos*

Este projeto tem como objetivo aperfeiçoar um software capaz de prever e identificar a trajetória de um alvo lançado, e que acoplado a um dispositivo disparador de projétil, será capaz de interceptar este alvo. Uma vez com informações sobre a trajetória de um alvo, deveremos calcular a região de impacto do mesmo através de uma média móvel e delimitar se ele deverá ou não ser interceptado. Para isso iremos utilizar um filtro de imagens que nos possibilitará enxergar o alvo e seus entornos em tempo real e efetuar a tradução para o circuito digital por meio de algoritmos a fim de executar a predição levando em consideração uma margem de erro pré-determinada.

Através de controladores, integração entre software e hardware, além da parte mecânica, o objetivo será reconhecer a posição ou trajetória de um projétil em um determinado tempo e intercepta-lo.

1.3 *Motivação*

O projeto teve sua principal motivação em viabilizar um robusto sensor de movimento que pode detectar com alta precisão a posição de seu alvo e prever uma ação através de controladores específicos.

Com a crescente exploração de tecnologias que aliam eletrônica e movimento (jogos eletrônicos, aparatos militares não-tripulados, quadricópteros), o mercado está demandando e constantemente procurando simplificar algoritmos e produtos que possibilitam esta aliança. Esta situação vigente nos leva a acreditar que existe uma lacuna inexplorada na questão, tornando o projeto em sua essência de extremo interesse para os realizadores.

A ideia do projeto teve sua origem e intuito em aplicações que preveem e analisam movimentos, sendo amplamente utilizado na área de esportes, jogos interativos, segurança domiciliar e empresarial além da militar.

2 PROBLEMA

Os interceptadores de projéteis assim como seus algoritmos de localização e interpretação de um objeto no espaço são tratados com sigilo absoluto por seus fabricantes devido ao cunho predominantemente militar, prejudicando o material de pesquisa e desenvolvimento disponível para acadêmicos e o público em geral.

A realidade desta situação levou ao interesse de se estudar e colocar em prática a solução de um problema tão pouco difundido na literatura e torna-lo viável para aplicações não triviais como o caso da interceptação de mísseis utilizados para a defesa de uma certa região no espaço.

Portanto, partiremos de um problema complexo envolvendo um algoritmo tridimensional para trabalharmos com trajetórias em um plano 2-D, simplificando assim a solução para um fim acadêmico.

Uma abordagem do problema é a análise da eficiência do processamento de imagens em tempo real. Este processamento possui um compromisso entre a velocidade de ação, a sensibilidade e robustez do projeto: para ser capaz de identificar objetos de qualquer formato, teríamos que ter limitações com relação a sua velocidade e área de "proteção"; para um objeto em qualquer velocidade, provavelmente precisaríamos limitar o tamanho/cor/forma do objeto. Nesse contexto, foi escolhida a utilização de um objeto padrão de fácil identificação por cor, para que o processamento de imagens fosse reduzido e que pudéssemos realizar um sistema mais robusto com relação às velocidades e direções possíveis do alvo a ser interceptado.

Dessa forma, iremos supor um cenário com maior grau de complexidade e com o mínimo de aproximações, para só fazê-las na medida em que for necessário, seja por falta de capacidade de processamento, seja por necessidades físicas do nosso sistema.

Portanto, abordaremos o problema da seguinte forma: teremos câmeras posicionadas adequadamente de forma a captar o posicionamento de uma bola de tênis e sua trajetória.

De posse desses dados, calcularemos se a região de impacto é uma região protegida, e interceptaremos o alvo, em caso afirmativo.

Será objeto de nosso estudo diferentes formas de melhorar o processamento das imagens e também diferentes formas de obter a trajetória do alvo, utilizando uma ou duas

câmeras, bem como melhorar o posicionamento do disparador do projétil, posicionando-o com apenas um servo motor, apenas modificando a direção horizontal do disparo, ou utilizando dois servo motores, controlando também o ângulo vertical do mesmo.

3 *REFERENCIAL TEÓRICO*

Assim como o já citado Iron Dome, de Israel, a tecnologia de interceptadores de projéteis (ABM em inglês) vem sendo desenvolvida desde 1940 por vários países do mundo como EUA, Rússia e mais recentemente Israel com propósitos puramente militares.

As técnicas utilizadas por militares são significantemente mais robustas e eficazes, pois lidam com lançamento de misseis interceptadores que por sua vez têm de ser lançados de uma estação remota. O nosso projeto consiste em reconhecer a trajetória de um objeto em um plano, simplificando o problema.

Dentre as técnicas pesquisadas, encontramos projetos de interceptação de projéteis baseados na quantificação eletrônica da visão humana através de câmeras, redes neurais e servo motores baseados em imagem.

A pesquisa nos levou a um trabalho de conclusão de curso com ideia similar, e que se tornará base do projeto apresentado por este documento. Tal projeto utilizava um par de câmeras estéreo. Em seu conceito, foram elaborados algoritmos de aproximação em MatLab que interceptavam uma bola de tênis através de uma decomposição de imagens captadas pelas câmeras apontadas para um plano.

3.1 *EQUAÇÕES DE MOVIMENTO PARABÓLICO*

As equações básicas encontradas na literatura para interceptar um objeto percorrendo uma trajetória parabólica é definido a seguir:

Partindo de princípios básicos da cinemática, temos, para um lançamento oblíquo

$$\frac{dx}{dt} = v \qquad (3.1)$$

$$\frac{dv}{dt} = \frac{d^2x}{dt^2} = a \qquad (3.2)$$

$$v = v_0 + a\Delta t \qquad (3.3)$$

$$vdt = \int_0^{\Delta t} (v_0 + a\Delta t)\, dt$$

Se

$$y - y_0 = v_0\Delta t + \frac{1}{2}at^2 \qquad (3.4)$$

Chegamos em:

$$v^2 = v_0^2 + 2a(x - x_0) \qquad (3.5)$$

A partir da equação (3.4) chegam-se às seguintes conclusões para um lançamento em uma direção:

$$a_x = 0$$
$$a_Y = 9.8m/s^2$$
$$v_x = v_0 cos\theta$$
$$v_Y = v_0 sen\theta + a_Y t$$
$$x = x_0 + v_0 t \qquad (3.6)$$
$$y = y_0 + v_0 t + + \frac{1}{2}a_Y t^2 \qquad (3.7)$$

Portanto, levando em consideração que o projétil atirado pelo atuador pode ser aproximado por uma reta, conseguimos interceptar no ponto x e y desejado a partir de um canhão posicionado a uma distância d (em z).

3.2 *MÉDIA MÓVEL SIMPLES*

A média móvel simples (MMS) é um parâmetro que retiramos de uma série de dados, calculando a média (no caso, aritmética) de valores recentes ao longo do tempo. Seu uso nos permite realizar a previsão do valor médio dos dados em um futuro determinado levando-se em consideração o erro agregado.

O cálculo é feito por períodos: A média móvel simples de período$N$, ao final de um período$T$, pode ser usada como uma previsão para um período$T + 1$se a série de dados for estacionária (ou convergente, que flutua em torno de uma mesma média ao longo do tempo).

A equação de média móvel simples pode ser interpretada da seguinte forma:

$$M_{T+1} = \frac{\sum_{i=T+1-N}^{T} X_i}{N} \qquad (3.8)$$

onde:

M_{T+1}: Previsão para o período T+1

N: Número de períodos para calculados para cada média

T: Período que se deseja calcular

X_i: Média calculada para o período atual (i)

O erro calculado EPR (Erro Padrão Residual) é calculado como:

$$EPR = \sqrt{\frac{\sum e_i^2}{N-k}} \qquad (3.9)$$

onde:

$\sum e_t^2$:Somatória dos erros simples de cada amostra ao quadrado

N: Número de períodos para calculados para cada média

k: Número de coeficientes que se deseja estimar (no caso, média móvel = 1)

A escolha pela média móvel simples e pelo EPR se deu principalmente pela relativa estabilidade e convergência da série de dados recolhidos e pelos bons resultados de previsibilidade que a MMS apresenta quando lidamos com curtos períodos, para n = 2, ou seja, apenas a última medida e a medida atual utilizada no cálculo das médias.

O EPR, diferentemente do desvio padrão convencional, assume que o erro médio é igual a zero ou seja, que a previsão é exata.

Assim sendo, ao calcularmos os valores separados por uma iteração de $\Delta x = x - x$, e$\Delta y = y - y_o$, provenientes das equações (3.6) e (3.7) respectivamente, calculamos a próxima média, levando em consideração o erro e então podemos ajustar o nosso dispositivo de rastreamento.

3.3 *ESTADO DA ARTE*

Nosso projeto possui diversos elementos que já foram amplamente estudados de maneira individual. Uma grande quantidade de material (sejam projetos caseiros ou trabalhos de graduação/pós-graduação) foi encontrada na literatura/internet, muitos deles possuindo apenas algumas características em comum com nosso projeto, como por exemplo, a identificação de objetos por câmera, e outros possuindo um maior número de características em comum, como por exemplo a identificação e interceptação de objetos em movimento. A seguir, alguns exemplos que foram encontrados, bem como o grau de similaridade dos mesmos com o nosso projeto

- Identificação de ervas daninhas por um robô autônomo:

  Este projeto tem como objetivo integrar em um robô autônomo com capacidade de percorrer campos de milho em busca de ervas daninhas. Ao encontrar uma erva

indesejada, o robô tem a capacidade de identifica-la pela sua cor, e a partir daí executar a remoção desta erva para manter a plantação de milho saudável. Como apenas envolve a identificação das ervas, podemos dizer que tem baixo nível de similaridade com nosso projeto.

- Um robô autônomo que joga sinuca

  Aqui, utiliza-se o artifício de identificação de cores para identificar a posição de uma bola de sinuca. O robô autônomo tem a capacidade de se deslocar sobre uma suposta mesa de sinuca, identifica uma bola e bate ela em direção a uma caçapa. Aqui, utilizaram a identificação por cores também para saber o posicionamento da caçapa onde a bola deverá entrar. Como, além da identificação da bola/caçapa por meio de câmeras, temos que o robô precisa fazer uma mira para acertar a caçapa, e, portanto, possui um maior grau de similaridade com nosso projeto.

- Disparador de discos em alvos fixos

  Um projeto da feira de mecatrônica de 2013 da universidade de Carnegie Mellon, dos Estados Unidos em que o objetivo era atingir alvos fixos. O sistema deveria ser capaz de identificar a cor do disco a ser lançado, e a partir daí, acertar um alvo de mesma cor, sendo que estão à disposição 3 alvos com cores diferentes. A altura, ordem e posição dos alvos não deveriam ser parâmetros fixos, e o sistema deveria ser capaz de identificar e acertar todos os alvos. Assim como o projeto anterior, podemos dizer que este possui um maior grau de similaridade com nosso projeto, por envolver a identificação por cor de um alvo e um disparo contra o mesmo.

- Disparador de bolas em alvos móveis

  Este projeto, um dos mais similares com o nosso dos encontrados em nossa pesquisa, é um projeto similar ao anterior, também desenvolvido na feira da universidade de Carnegie Mellon, porém do ano anterior, de 2012. Neste caso, os alvos são discos que se movimentam na horizontal, e o objetivo é acertar estes alvos. Para isso, é necessário prever qual será a posição do alvo no momento em que a bola atingir o plano que contém os alvos, e esta predição de trajetória é o que faz com que este projeto tenha o maior grau de similaridade com o nosso projeto, dentre os projetos encontrados.

- Projeto ASYS Defense

Utilizado como base ideológica e teórica para nosso projeto, o projeto ASYS Defense, do Instituto Mauá de Tecnologia, é um projeto com mesmos objetivos que o IPP – Interceptador de Projéteis em um Plano. Os alunos que o desenvolveram utilizaram um par de câmeras convencionais e programação em MatLab para criar um sistema capas de identificar a trajetória de um objeto e posicionar um disparador para a trajetória do mesmo, com o intuito de interceptação. Foi realizada uma troca extensa de informações com este grupo, que acabaram sendo condensadas e utilizadas para identificarmos as dificuldades já solucionada por eles, bem como possíveis otimizações e aprimoramentos que poderiam ser feitos de forma a melhorar o desempenho e a robustez do sistemas desenvolvido.

4 *JUSTIFICATIVA*

Diversas são as técnicas que podem ser utilizadas para se identificar um objeto em uma câmera. Dentre elas, estão as que identificam contornos, que buscam formas específicas ou as que utilizam cores. É claro que todas elas possuem seus pontos fracos, que seriam a necessidade de um fundo sem elementos, dificuldade na programação e a intolerância a objetos de uma cor específica, respectivamente. Não estamos colocando nosso foco no processamento de imagens necessário para a identificação do corpo em questão, e sim no controle necessário para identificar a posição do objeto e a resposta dos atuadores para posicionar e atirar contra o alvo. Dessa forma, fizemos buscas e verificamos que se tem um bom resultado, com processamento razoavelmente rápido, utiliza-se a identificação do objeto por cor, pois temos um objeto muito comum para ser utilizado como alvo e com uma cor bem característica: a bola de tênis.

Uma vez identificado o objeto na imagem capturada pelas câmeras, precisamos determinar com precisão a trajetória do alvo, e para isso será necessário termos uma visualização em três dimensões do cenário em que ele se encontra. Foram encontradas na literatura algumas técnicas para se capturar cenas/objetos em 3D, a grande maioria delas utilizando duas câmeras capturando uma mesma região, e com o processamento de imagens adequado, se consegue o posicionamento exato do alvo no espaço (3D), para que depois se saiba a rota do mesmo e a partir daí tome as decisões necessárias.

Uma forma alternativa seria utilizar duas câmeras filmando em direções ortogonais. Dessa forma, estaríamos, literalmente, decompondo o vetor da velocidade da bola em duas componentes ortogonais (e a terceira componente – altura – é a mesma para as duas câmeras) e assim poderíamos precisar a trajetória da bola.

Para o projeto em questão a ser realizado, realizaremos simplificações em relação a um modelo real: utilizaremos apenas uma câmera para a captura do deslocamento e da trajetória da bola, e para isso, devemos considerar modelos práticos adequados nos quais:

- A velocidade da bola terá sua componente de maior contribuição na direção Z, ortogonal ao plano XY (plano de captura da imagem), isto é, a velocidade na componente Z é tão maior que as componentes X e Y que poderemos supor essa

velocidade como sendo infinita, de modo que iremos considerar que o tempo que o objeto disparado demora para percorrer a distância Z entre a câmera e o alvo é zero, ou aproximadamente zero.

- A velocidade do alvo terá sua componente de menor contribuição da direção Z, ortogonal ao plano XY (plano de captura da imagem), isto é, o deslocamento da bola na direção Z é praticamente zero, em face aos deslocamentos em X e Y.

4.1 *SOFTWARE EMBARCADO*

O Software embarcado é cada vez mais utilizado em aplicações eletrônicas onde se exige uma rápida resposta do hardware, diminuindo o risco de atrasos eventuais de software no processamento em tempo real.

São conhecidos alguns tipos de microcontroladores, métodos de descrição de hardware e kits de desenvolvimento para software embarcado que levamos em consideração ao analisar a proposta do projeto, tendo em mente a disponibilidade no mercado, custos e a facilidade de programação.

4.1.1 PIC

Os microcontroladores da família PIC são bastante utilizados em diversos projetos didáticos de engenharia pelo seu custo, robustez e rápida implementação no circuito de testes. Em uma breve análise, foram levados em consideração principalmente fatores diretamente relacionados ao projeto tais como implementação de servo motores que servirão para posicionar o disparador de projéteis.

Notoriamente, estes microcontroladores possuem uma rápida fase de depuração e substituição em caso de mau funcionamento ou para uma tomada de decisão em caso de urgência. A linguagem de programação utilizada é C/C++ bastante difundida e com vasta

literatura para eventuais consultas o que torna ainda mais rápido o processo de produção do algoritmo de controle.

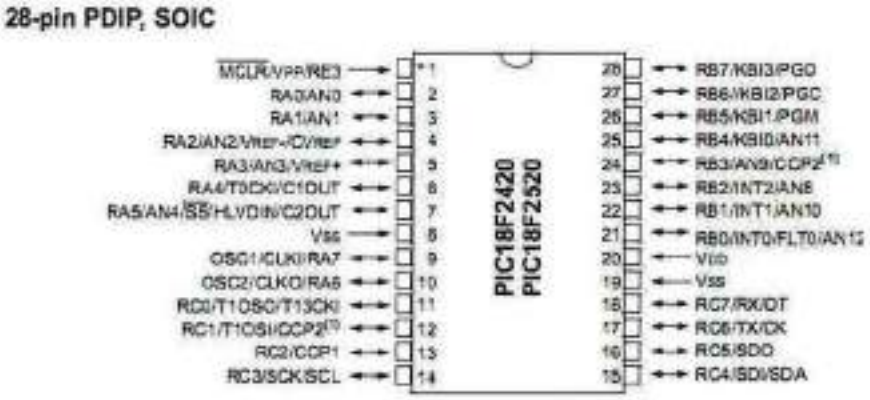

Figura 1: PIC18F240/250

4.1.2 FIELD-PROGRAMMABLE GATEARRAY (FPGA)

O método de descrição de hardware específico para o uso de FPGAs foi considerado por existir uma facilidade e um prévio conhecimento adquirido durante o curso de engenharia, além do laboratório da universidade possuir algumas placas didáticas, o que facilitaria a aquisição e possivelmente o custo do projeto.

Escolhemos a placa DE2 da Altera como uma possível solução, que possui o chip Cyclone II e é facilmente depurável pelo software Quartus II da própria fabricante. A linguagem de descrição de hardware utilizada é o VHDL que possui uma modesta literatura disponível em meios abertos como internet, porém contém uma considerável quantidade de livros que abordam o assunto em inglês.

Figura 2: Placa DE2 da Altera

4.1.3 ARDUINO

O Arduino é um microprocessador embarcado em uma placa, desenvolvido e fabricado na Itália, que possui uma extensa variação de aplicações variando desde um robusto suporte para I/O assim como banco de memória e entradas específicas para dispositivos.

O que nos chamou a atenção neste particular microprocessador é a existência de uma versão no mercado (DFRduino Romeo AIO) que possui entrada para dois servos motores independentes, o que o aproxima de nosso projeto, facilitando a construção e a integração hardware/software.

A linguagem de programação utilizada, assim como o PIC, é C/C++ que faz do Arduino um forte candidato a fazer parte do projeto.

Figura 3: Placa DFRduino Romeo AIO

4.2 MATLAB

O Matlab é uma ferramenta de modelamento de projeto multidisciplinar e altamente difundida pelo mundo quando se trata de simulações matemáticas e até mesmo capaz de inserir sinais e processa-los em tempo real com a ajuda de um PC.

Este software é considerado em nossas possíveis soluções pela alta facilidade em depurar o código em tempo real, além de ter um baixíssimo custo, já que as licenças do software são providenciadas pela universidade. Além disso, temos uma excelente referência para o código localizador, previamente elaborado e executado pelos alunos Fabio Bobrow, Bruno Muniz,Gihed Nassif e Kaique Grossmanndo Instituto de Tecnologia Mauá.

Foi feito uma simulação utilizando este código localizador, que recebe dados de uma trajetória de um objeto, previamente obtida das informações recebidas pelas câmeras, e determina se o ponto de impacto deste alvo está numa área protegida. Caso esteja, de posse de características do lançador responsável pela interceptação, ele calcula quais devem ser os ângulos horizontal e vertical para o lançamento, e efetua a interceptação do alvo.

Abaixo temos o resultado bidimensional desta simulação, na forma de um gráfico (neste caso, de y) em função da altura.

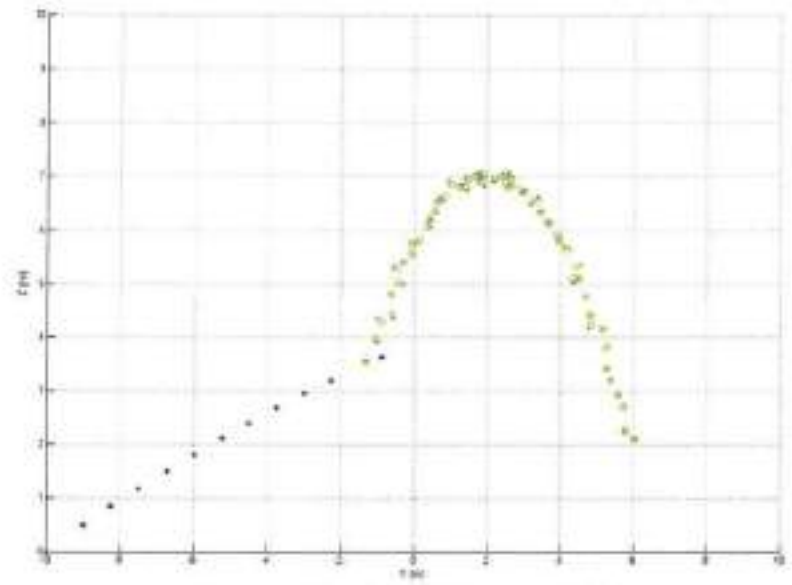

Figura 4: Simulação feita em MatLab

5 *METODOLOGIA E SOLUÇÃO*

5.1 *ESCOLHA DE UMA SOLUÇÃO*

5.1.1 O MÉTODO AHP

Para a tomada de decisão, optamos por utilizar o software Expert Choice que se utiliza do processo Analytic Hierarchy Process (AHP) para realizar os cálculos de objetivos, critérios e alternativas.

O método AHP consiste em, basicamente, decompor um problema de decisão em uma hierarquia de subproblemas onde as decisões se tornam mais simples e que possam ser analisadas independentemente. Os critérios de decisão são selecionados de acordo com a necessidade ou escolha de quem está decidindo, podendo variar desde matematicamente complexo ou estimado grosseiramente, ou seja, qualquer coisa que o faça decidir.

O próximo passo, já montada a hierarquia, inicia-se a fase comparativa, onde avaliam-se os subproblemas e seus vários elementos, comparando-os um ao outro de forma a formar pares (*Pairwise Comparison*) atribuindo pesos e critérios próprios para a tomada de decisão diante do par formado.

O AHP antes de tudo é um método de tomada de decisão humano, ou seja, o julgamento final provém de uma escolha humana sobre o que está sendo decidido.

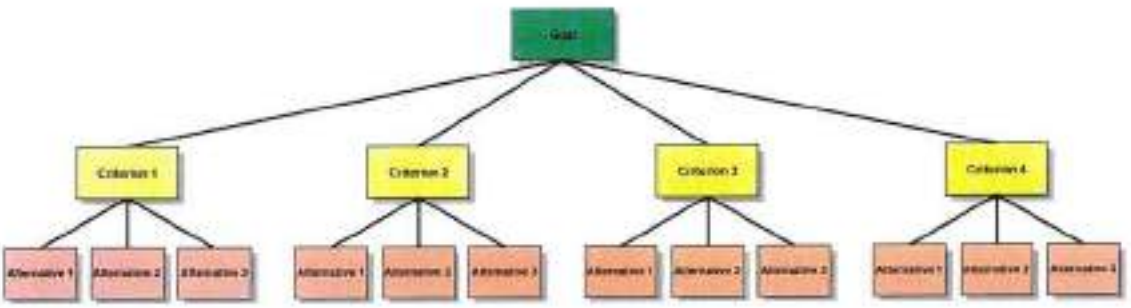

Figura 5: Exemplo esquemático do método AHP

5.1.2 CRITÉRIOS E SUBCRITÉRIOS ADOTADOS

- Portabilidade
    - Dispositivo leve e compacto
- Segurança
    - Dispositivo leve
    - Dispositivo seguro
- Robustez
    - Software com poucos ou sem *bugs*
    - Probabilidade de erro em menos de 1%
- Rapidez
    - Tempo de resposta do servo motor
    - Tempo de resposta do software
- Depurabilidade
    - Linguagem de software
    - Facilidade de depuração

5.1.3 ALTERNATIVAS E SOLUÇÃO

- MATLab
- Arduino
- FPGA
- PIC

Adotamos o método *Pairwise Comparison*, como descrito anteriormente, presente no software para atribuir pesos para critérios e alternativas. Este método compara todos os

critérios e subcritérios dois a dois, atribuindo valores aos pesos da decisão tomada pelo usuário, totalizando 87 comparações no caso deste projeto.

Primeiramente, o software confia ao usuário a função de atribuir pesos aos critérios primários (Portabilidade, Segurança, Robustez, Rapidez, Depurabilidade) para então iniciar as comparações entre todos os subcritérios.

> Ex: O que é mais importante? Um ***dispositivo Leve*** ou ***Facilidade na depuração***? (Comparando subcritérios que estão abaixo de portabilidade e depurabilidade)

> As alternativas também são comparadas à todos os critérios e subcritérios.

> Ex: Dê uma nota de 0 a 10 que correlacionam ***Tempo de resposta do servo motor*** com ***FPGA*** (Comparando agora um subcritério com uma alternativa)

Após a sequência de tomadas de decisão, o software calcula, baseado nos pesos atribuídos no início, e das notas dadas a cada comparação, qual a melhor solução para o problema, que pode ser visualizado na figura a seguir:

	Alternative Name	All Participants
4	PIC	21.83 %
3	FPGA	27.98 %
2	Arduino	32.01 %
1	MatLab	18.18 %

Figura 6: Resultados obtidos pelo software Expert Choice

Os pesos para todos os critérios principais são impressos neste demonstrativo de resultados oferecido pelo software:

Figura 7: Relação entre critérios e resultados

Como observado, portanto, nas figuras 5 e 6, concluímos que melhor solução a ser seguida é, então, a implementação do dispositivo utilizando a placa DFRduino Romeo AIO.

5.2 *WORK BREAKDOWN STRUCTURE (WBS)*

WBS é uma ferramenta muito utilizada em gerenciamento de projetos e engenharia em geral, que é, por definição, uma decomposição orientada à atividades do projeto em menores componentes.

Ela define e agrupa elementos menores de um projeto de uma forma que nos ajuda a organizar e definir o trabalho total do escopo do projeto.

Na figura a seguir, foi montada tal estrutura a partir do software WBS Chart Pro:

Figura 8: Diagrama WBS

5.3 PLACA DFRDUINO ROMEO AIO

O dispositivo DFRduino Romeo AIO é uma placa microcontroladora compatível com Arduino especialmente fabricada para aplicações robóticas. Esta placa se beneficia da plataforma de código aberto Arduino, portanto, uma redução de custos de software além de nos aproveitarmos diretamente das duas entradas para servos que se são controlados independentemente.

Além de sua importância em relação à alimentação independente de motores contínuos e servo-motores, a placa DFRduino Romeo AIO possui uma série de outras vantagens que serão aproveitadas no projeto.

5.3.1 CHIP ATMEGA32U4

O chip ATmega32u4, presente nesta placa, é um microcontrolador de baixa potência, produzido pela Atmel Corporation.

Sua arquitetura é baseada na tecnologia AVR RISC de 8 bits, desenvolvida pela própria Atmel, possui memória programável de 32KB, 2.5KB SRAM, 1KB EEPROM, 12 canais para conversores A/D de 10 bits. Alcança até 16MIPS a 16 MHz, operando de 2.7 até 5.5V.

Essa velocidade de processamento é possível pois o microcontrolador é capaz de realizar instruções complexas em um ciclo de *clock* chegando, como previamente dito, em aproximadamente 1 MIPS por MHz.

5.3.2 Especificações

O dispositivo possui, como especificações técnicas:

- Entradas analógicas: A0 até A5 e A6 até A11 (nos pinos digitais 4, 6, 8, 9, 10 e 12)
- Pinos PWM: 3, 5, 6, 9, 10, 11 e 13 (PWM de 8 bits)
- Alimentação DC: Provida por USB ou externa de 6V–23V. Saída DC:5V(2A) / 3.3V DC
- Interface Serial Level TTL ou USB
- Soquetes de pinos macho e fêmea para todos os pinos
- Soquete integrado para APC220 e Módulo Bluetooth
- Soquete para módulo XBee
- Três pinos de interface I2C/TWI
- Sentido direto e inverso para 2 motores de até 2A

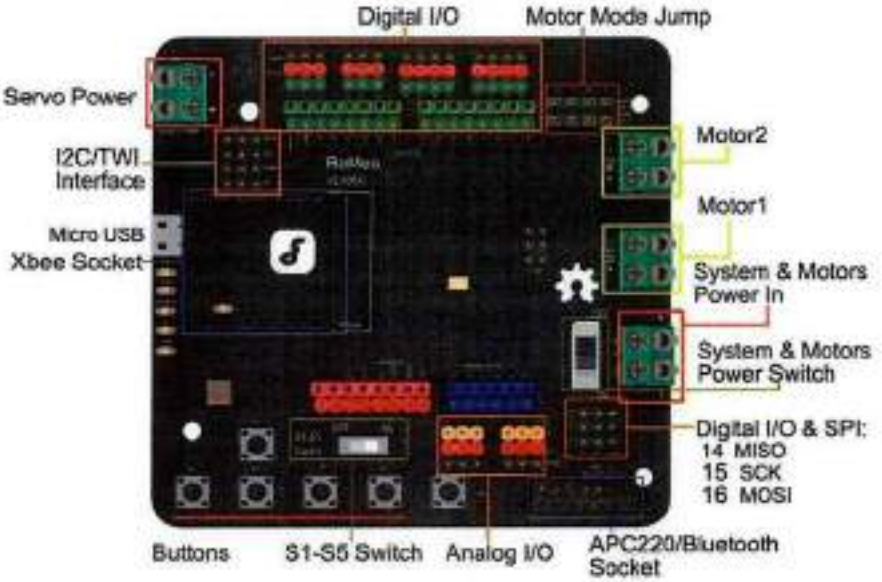

Figura 9: Pinout da placa DFRduino

5.4 SERVO MOTORES

Os requisitos de engenharia demandam servo motores de resposta rápida e que tenham um torque suficiente para mover a estrutura do disparador de projéteis. Em uma pesquisa por custo-benefício entre os servo motores existentes no mercado, encontramos um servo motor que excede os parâmetros requeridos e que não gera consideráveis custos adicionais. A preferência por um servo motor (no caso o BMS-631MG) que é mais rápido e forte do que o necessário nos serve para dar uma margem e uma garantia de que as especificações serão cumpridas.

Figura 10: Servo motor BMS-631MG

Especificações Técnicas:

Weight	47 grams / 1.66 oz.
Dimensions	40.5 x 20 x 38 mm / 1.59 x 0.79 x 1.5 inch
Torque At 4.8V	5 kg-cm , 6.9 oz-in
Torque At 6.0V	6.2 kg-cm , 8.6 oz-in
Speed At 4.8V	0.1 sec / 60° at no load
Speed At 6.0V	0.08 sec / 60° at no load

5.5 O DISPARADOR A AR COMPRIMIDO

Foi realizada uma extensa pesquisa para encontrar ou projetar um disparador adequado aos nossos requisitos. Num primeiro momento tivemos a ideia de utilização de projetos caseiros facilmente encontrados na internet para dispositivos como lançadores de foguetes a ar comprimido e armas de batata (ou outros projeteis) também a ar comprimido, mas algumas características destes projetos acabavam esbarrando em especificações técnicas do nosso disparador tais como peso, medidas, velocidade minima e poder de alcance. Para tanto, fizemos nosso próprio projeto de disparador, cuja lista de materiais e esquema simplificado está na figura 10.

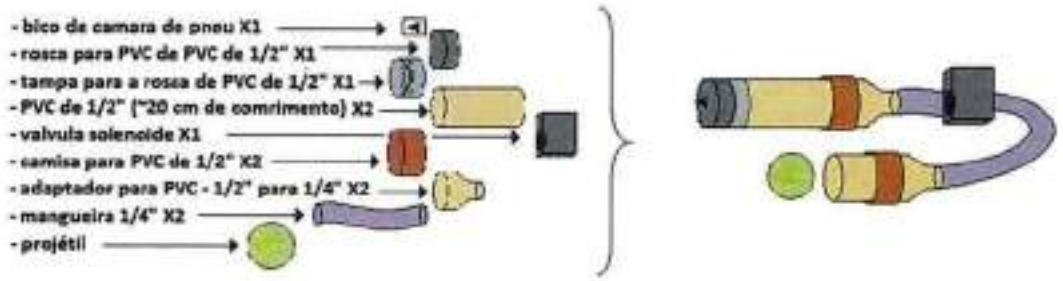

Figura 11: Esquemático do disparador de projéteis

5.6 DISPARADOR DE PROJÉTEIS E CALIBRAÇÃO

Utilizaremos um disparador de ar comprimido controlado por um sinal de tensão. Conceituamos que a direção e a velocidade inicial do projétil lançado será determinado pelo microcontrolador, logo, este disparador terá acoplado um conjunto de engrenagens que ligadas aos servos, darão a mobilidade necessária para que este atinja a bola de tênis.

O lançador será construido com tubos de PVC, uma válvula solenoide para a descompressão de ar e uma válvula comum para mantermos o ar comprimido.

Pensou-se, a princípio, que seriam necessários vários ensaios de calibração para permitir a determinação de todos os parâmetros para que consigamos determinar com

precisão a trajetória, o ponto de impacto e também prever a o resultado exato que um disparo com determinadas características irá provocar.

Porém, analisando melhor o problema, e levando em conta a simplificação do problema para interceptar o projeto em um plano (levando em conta apenas os deslocamentos X e Y do alvo e do projétil), podemos perceber que se determinarmos um quadriculado na imagem gerada pelas câmeras, precisamos somente associar a esses quadriculados a região que está sendo protegida, e caso as duas câmeras, ortogonais, identifiquem que o alvo cairá em uma área protegida, a intervenção ocorre.

5.7 Posicionamento das câmeras e a captura de imagens com câmeras convencionais

A primeira analise em todo o sistema se inicia com a identificação do posicionamento do alvo em uma imagem. Existem diversas técnicas exploradas em diferentes fontes bibliográficas, e como foi comentado anteriormente, não pusemos muito foco na obtenção e otimização do método que será utilizado para se determinar a posição de uma bola (ou seu centro) em uma imagem.

Já foi dito que, por uma questão de simplificação, iremos considerar o alvo como sendo uma bola de tênis: com formato e cor bem definido e distintos, utilizando assim a mesma escolha realizada no projeto Asys Defense, do Instituto Mauá de Tecnologia. Sabemos que tal escolha simplificou muito este projeto e facilitou a manipulação e processamento de dados e códigos, mas assim eles tiveram alguns problemas com a demora no processamento das imagens, visto que estavam rodando este processamento no MatLab, que não é um sistema embarcado e, portanto, é muito mais limitado neste quesito, como já foi mostrado anteriormente.

É fato que existe mais de uma técnica que pode ser utilizada para, a partir de duas câmeras, se conseguir o posicionamento exato de um corpo no espaço. Aqui, apesar de não ser uma técnica encontrada na literatura, pensamos em posicionar duas câmeras filmando em direções ortogonais. Dos cursos básicos de álgebra linear, temos que podemos decompor um vetor em elementos de qualquer base, e sendo essa base ortogonal, temos uma decomposição extremamente precisa, para recomposição e analise do vetor original. Com

base nesse raciocínio, a captura com duas câmeras ortogonais irá produzir um resultado extremamente satisfatório: cada câmera irá produzir uma mensagem em duas dimensões, sendo possível extrair duas componentes de velocidade de cada uma: x e z de uma câmera e y e z da outra. Dessa forma, com a amostra de 3 ou 4 imagens já se pode extrair os parâmetros em 3 dimensões da trajetória do lançamento.

Para efeito de adequação ao nosso modelo, será feita uma adaptação para que os cálculos e o processo de calibração sejam mais simples e menos exaustivos. Logo, no processo de calibração, precisamos apenas determinar uma relação entre os ângulos de posicionamento do servo e o ponto atingido no espaço, e no momento da interceptação do alvo, precisaremos apenas determinar as componentes X e Y da velocidade do alvo.

Uma vez com todos os parâmetros que definem a trajetória parabólica do alvo, se encerra a tarefa de análise de imagens, e utilizamos estes dados como entrada de um simulador e previsor de trajetórias, para poder se determinar o ponto de impacto do mesmo, e se ele necessita de intervenção. Se necessária, é feita a ativação do interceptador.

O projeto Asys Defense utilizou códigos que eram rodados em MatLab para determinar o centro da bola de tênis, utilizando técnicas de otimização, como a busca em imagens anterior de uma região de alta probabilidade de posicionamento da bola, visando otimizar os cálculos de processamentos apenas dessa região, visão uma melhor robustez do projeto, no que se diz a velocidade de lançamento da bola, capacidade de processamento e velocidade na ação de interceptação. Estes códigos servirão de base para os nossos cálculos utilizando a placa DFRduino.

5.8 Placa CMUcam4

Para realizar captura de imagens, consideramos o dispositivo *SparkFun CMUcam4 v10* que possui interface direta para a integração com o nosso hardware embarcado, o Arduino e realiza tal integração de diferentes formas, utilizando comunicação serial (Rx /Tx) ou acoplando-o de forma a reutilizar os pinos do DFRduino.

A placa CMUcam4 desenvolvida em cooperação entre a empresa SparkFun e a*Carnegie Mellon University* localizada em Pittsburgh nos Estados Unidos, é uma placa que possui um controlador para servo motor capaz de uma resolução de 1μs a 50Hz, capacidade de gerar imagens de 640 x 480 pixels por frame, dispositivo serial UART capaz

de transmitir 250.000 bps e uma câmera geradora de vídeo que suporta NTSC ou PAL.

Ela é um dispositivo embarcado completamente programável, contendo um processador principal Parallax P8X32A, acoplado com uma câmera-sensor OmniVision 9665 CMOS. É uma placa que permite total programação de suas funções, tendo sua melhor performance na identificação de objetos com alto contraste entre cores, como ocorre no nosso projeto.

Dentre as diversas funções desta placa, destacamos a sua capacidade de reconhecer em tempo real um objeto em um ambiente 3D, de acordo com sua cor que é previamente calibrada na câmera. Ou seja, podemos prever um movimento se analisarmos corretamente as coordenadas originadas pela câmera, através de um projétil que varia no tempo em uma trajetória conhecida.

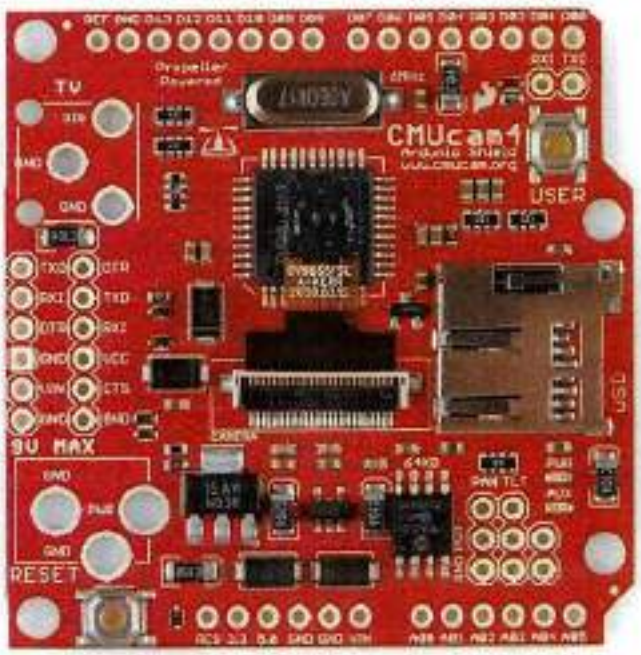

Figura 12: Placa CMUCam4 v10

Esta placa já possui uma programação inicial, que consiste numa calibração inicial para que a placa saiba qual é a cor do objeto que queremos obter informações. Depois

disso, simplesmente ao conectar a placa a uma saída de vídeo, teremos uma imagem filtrada e processada, que exibe uma posição aproximada mas bem precisa de onde está o centro (na realidade, o centroide, que é o centro geométrico) do objeto, bem como os limites verticais e horizontais do mesmo.

A obtenção de informações sobre a bola ocorre de maneira mais simples e rápida, se compararmos com a obtenção via duas câmeras comuns. Isso ocorre por que, estamos utilizando apenas uma câmera ao invés de duas, o que, teoricamente, já dobra nossa capacidade e velocidade de processamento de dados. Porém, agora nós temos apenas uma câmera que captura imagens de forma frontal, e não temos mais a câmera responsável pelas imagens que nos forneceriam as informações sobre a velocidade da bola na direção ortogonal a primeira: a profundidade.

Porém, graças ao processamento rápido e eficaz desta placa, não há a necessidade da segunda câmera. Como podemos verificar na imagem abaixo e como já foi dito, esta placa tem a capacidade de delimitação dos limites verticais e horizontais do objeto. Dessa forma, com a variação desses limites, teremos uma variação no tamanho aparente do nosso alvo, e dessa forma podemos estimar de maneira igualmente boa a taxa de variação do tamanho deste objeto, e, dessa forma, estamos também determinando a velocidade perpendicular a imagem obtida.

Claramente, a determinação das outras duas componentes da velocidade do lançamento é facilmente obtida ao se comparar a posição do centroide da bola de apenas 2 ou 3 imagens consecutivas (e sabemos que com um processamento veloz e adequado, teremos a condição de analisar um número maior de imagens, podendo realizar um cálculo mais preciso e apurado da trajetória do alvo).

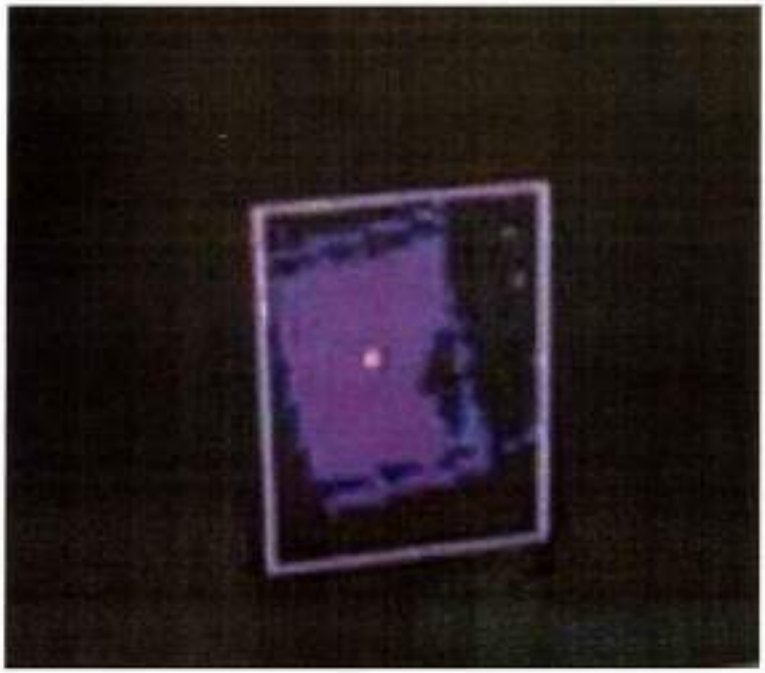

Figura 13: Imagem de uma caixa reconhecida e atualizada pela CMUCam4 v10

6 *VIABILIDADE E RISCOS*

Em um projeto desta magnitude, onde existem componentes de hardware, de software e uma integração de ambos, é importante definir passos de andamento do projeto e dividi-lo em módulos para que a depuração seja feita por etapas e de forma sistemática e para que se possa ter uma análise previa mais precisa a respeito da viabilidade de cada uma destas etapas, bem como quais os riscos de falhas poderemos ter nas mesmas.

O projeto como um todo foi dividido em 3 grandes etapas, completamente distintas e disjuntas, que serão testadas e depuradas individualmente e que não possuem correlação direta no sistema. São elas:

1. Elaboração do código em C/C++ que será embarcado na placa DFRduino Romeo AIO.
2. Construção de um disparador de ar comprimido.
3. Integração física de servo motores em ambos produtos produzidos pelas etapas 1 e 2.

As características das etapas do projetos, descritas acima, possuem grande importância, pois permite, caso seja implementado corretamente esta solução modular, a possibilidade de minimizar consideravelmente os riscos envolvidos no projeto final, pois estaremos tratando com subsistemas que podem ser planejados, analisados e depurados de forma independente. Durante todo o trabalho, contaremos com uma vasta porção de simuladores e *debuggers* que nos privarão de algum mau funcionamento do código, além de contarmos com uma grande quantidade de código aberto ao público disponível em rede online.

Mesmo tomando todas as precauções devidas e executando o projeto desta maneira, existem riscos de projeto e de execução que podem ocorrer:

- Defeitos de fabricação de componentes de hardware
- Erros de execução (Mecânicos, ligação de componentes)

  - Simplificação exagerada
  - Falhas no projeto do simulador
  - Dados insuficientes/incorretos de entrada do simulador

- Simulação não coerente com a realidade

- Varredura das câmeras não suficiente para detectar uma trajetória

- Equipamento não robusto o suficiente, não atingindo a mínima requisição de erros

- Disparador incapaz de alterar a rota do alvo

- Defeitos de equipamentos relacionados a mau uso

7 *REQUISITOS TÉCNICOS*

7.1 REQUISITOS DE MARKETING

Para uma própria abordagem sobre os requisitos qualitativos do projeto, devemos lembrar que o projeto como está apresentado neste documento faz parte de um conceito de um produto que potencialmente poderá ingressar no mercado de forma indireta, ou seja, através de transformações específicas para cada caso que queira aproveitar alguma de suas múltiplas funções.

Entretanto, este dispositivo deverá ser tratado como um produto acadêmico e para este fim:

- Deverá reconhecer e possivelmente interceptar fisicamente o projétil lançado em poucos segundos.
- Deverá ser operado facilmente, independendo de quem o fabricou.
- Produto físico deverá ser íntegro, sem partes de hardware retiráveis ou soltas, fechado em uma caixa com boa resistência.
- Produto deve ser portátil o suficiente para conseguir ser deslocado facilmente por uma pessoa.
- Por se tratar de um projeto que lançará projéteis interceptadores, deverá ser livre de erros quanto ao lançamento de tais projetos, focando apenas no objeto a ser interceptado.

7.2 REQUISITOS DE ENGENHARIA

Os requisitos de engenharia foram desenvolvidos a partir do conceito de que o produto em questão será uma peça de cunho didático para fins acadêmicos, ou seja, é um protótipo cujo propósito se estende a fim de concretizar os conhecimentos adquiridos além de servir como uma peça que, ao ser desenvolvida para determinado fim, teria um forte potencial mercadológico. Portanto, a necessidade do projeto servir a um usuário final (consumidor),

como produto industrializado não foi tomada como prioridade.

Para realizarmos simulações e desenvolver um produto final, foi decidido optar pela placa DFRduino Romeo AIO, que contém o microcontrolador *ATmega32u4*e se encontra disponível no mercado.

- **Unidade de controle**
  - Fácil desenvolvimento e depuração através do kit de desenvolvimento da placa.
  - Código C/C++ organizado e de fácil compreensão para rápidas modificações.
- **Servo motor**
  - Mínimo de velocidade angular (0.5s / 60º)
  - Mínimo de torque (1Kg-cm)
- **Lançador de projéteis**
  - Dimensões máximas (35cm altura, 2cm diâmetro)
  - Velocidade mínima do projétil (1m/s)
- **Área atingida pelo projétil**
  - Não deve ultrapassar $1m^2$ distando 1 metro do lançador de projéteis tomando este como referência central
- A parte de software deve ser integrada ao hardware de modo que consigamos depurar e solucionar os possíveis erros de forma rápida e eficaz
- O aparelho deve funcionar com os sensores (câmeras) apontados para um plano de uma única cor, sem a obstrução de nenhum objeto ou pessoas.

8 *CONSTRUÇÃO E DESENVOLVIMENTO*

8.1 *MODELO E FLUXO DE INFORMAÇÕES*

O processo se iniciará a partir de uma aquisição de dados da trajetória captados pela placa CMUCam v4. Os dados são compostos de 4 variáveis de borda que representam o ponto médio das arestas onde é calculada uma caixa *(box)* pelo próprio hardware embarcado. Além disso, a placa fornecerá um ponto central, também calculado em tempo real que nos servirá de guia para a movimentação dos servos.

Em posse dos dados, o modelo matemático começa a ser calculado, levando em consideração a distância entre o dispositivo e o plano onde o projétil será rastreado, a velocidade dos servo-motores e seus limites de angulação, assim como a aceleração dos dois projéteis. Este cálculo será mostrado em detalhes mais adiante neste documento.

O passo seguinte é a interpretação destes dados pelo algoritmo escrito em linguagem C e embarcado na placa DFRduino que permanecerá constantemente em busca de novos dados para calcular novas rotas e assim acionar os servo motores ligados diretamente à placa.

Este algoritmo terá como função principal direcionar os atuadores (servos) em uma direção pré-definida predita pela retroalimentação da placa através da câmera, e, assim que o cálculo estiver completo, acionar o disparador em um intervalo de tempo definido pelo cálculo da trajetória.

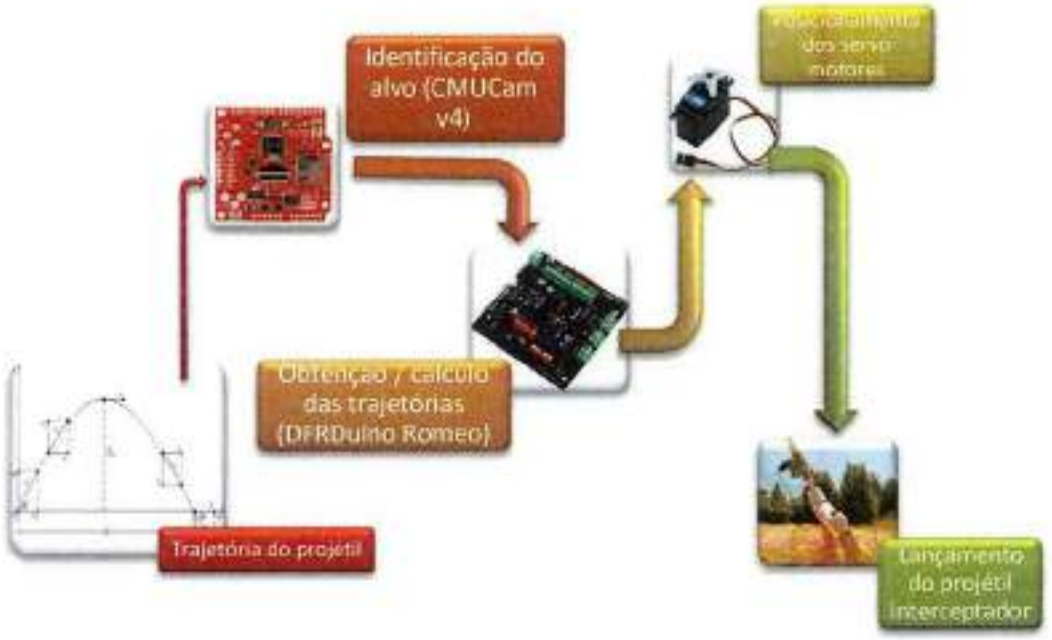

Figura 14 - Fluxo de Informações e principais funções de cada etapa

8.2 *MONTAGEM DOS COMPONENTES ELETRÔNICOS*

Em uma primeira etapa de testes, foi feita a ligação entre a placa de captura de vídeo e o circuito embarcado (placa DFR).

A comunicação entre as placas é feita através dos pinos RXI/TX0 (pinos 1 e 0 da placa DFR respectivamente) das duas placas, ou seja, a comunicação é feita de modo serial. Inicialmente esta configuração não seria satisfatória pois as placas que contém os microcontroladores preparados para atuar com a linguagem Arduino geralmente se comunicam com o programador, ou seja, o PC de forma serial através de um conversor Serial/USB diretamente implementado no circuito integrado. Porém, a escolha pela placa DFRduino baseada no Arduino Leonardo, possui uma interface serial de comunicação com o computador diferente daquela utilizada pelos pinos 0 e 1, fazendo com que conseguíssemos realizar a programação da placa e a leitura dos dados da câmera concomitantemente.

Estabelecida a conexão entre a placa DFR e a placa CMUcam4, inicia-se a fase de captura dos dados para análise de consistência. O circuito embarcado na CMUcam4 possui diversas variáveis que nos permite ajustar os níveis de cores as quais gostaríamos de filtrar, além da redução de ruídos externos e modos de captura.

Podemos requisitar 4 formas estruturais (pacotes) distintas de dados da CMUcam4. São eles:

1. *F packet (Image Data)*

   Recebemos da câmera pacotes do tipo F, que contém dados binários específicos sobre a imagem em formato Bitmap.

2. *S packet (Statistics Data)*

   Esta forma estrutural contém dados estatísticos sobre a média das cores captadas pela câmera em tempo real, dividida por canais R, G e B.

3. *H packet (Histogram Data)*

   Os pacotes que recebemos da forma H, nos trazem informações sobre a porcentagem de pixels por imagem que obtemos na captura, de forma que podemos calcular então a precisão e a confiabilidade da câmera em certas condições de luz e planos de fundo.

4. *T packet (Tracking Data)*

   Os pacotes do tipo T carregam informações sobre borda e centroide do objeto que está sendo rastreado no momento, além de também possuir informações sobre a porcentagem de pixels (*CMUcam4_tracking_data_t.pixels*) e a densidade da imagem ou a confiabilidade do rastreamento na captura da imagem (*CMUcam4_tracking_data_t.confidence*). São os dados que utilizaremos em nosso código em sua fase final (Apêndice B.2 e B.3), já que são esses os tipos de dados de interesse do projeto.

Segue-se então a uma análise detalhada sobre os atrasos do circuito parcial para estimarmos o tempo de resposta que os servo motores deverão ter para calcular uma previsão satisfatória. Os testes foram feitos a partir da observação do intervalo de tempo entre duas saídas no console serial. Levando em consideração o tempo em que um dado leva para ser capturado e ser mostrado no terminal (feito utilizando um código em um loop infinito (Apêndice B.2)), chegamos a uma taxa de amostragem de aproximadamente 25FPS ou seja, um período de 40ms por dado capturado. Para calcular o tempo total de atraso,

precisaríamos também calcular o tempo de resposta dos servo motores, o que será feito mais adiante nesta análise.

A próxima etapa consiste em adquirir dados através da placa de vídeo e transmitir por meio da placa de circuito embarcado parâmetros que movimentariam o servo motor em diferentes ângulos.

O servo motor é interligado ao conjunto DFRDuino + CMUcam4 através de um barramento PWM (*Pulse Width Modulation*) localizado na parte inferior da figura 14. Este barramento possui um conjunto de 42 pinos combinados 3 a 3, denominados PWR, GND e D que correspondem à VCC, Terra e pino de informação (largura do pulso) respectivamente.

A literatura existente para a movimentação de servos para circuitos embarcados que operam com a tecnologia Arduino é consideravelmente extensa, porém, as particularidades aparecem de acordo com o tipo de servo utilizado em projetos.

Em primeira instância, devemos considerar as limitações de nossos servos realizando testes operacionais a fim de obter os ângulos limites que dependem também da distância entre o dispositivo e o projétil a ser rastreado, estas limitações nos ditarão o caminho a ser traçado e calibrado na montagem da estrutura final. Este processo pode ser melhor visualizado como no esquema da figura a seguir:

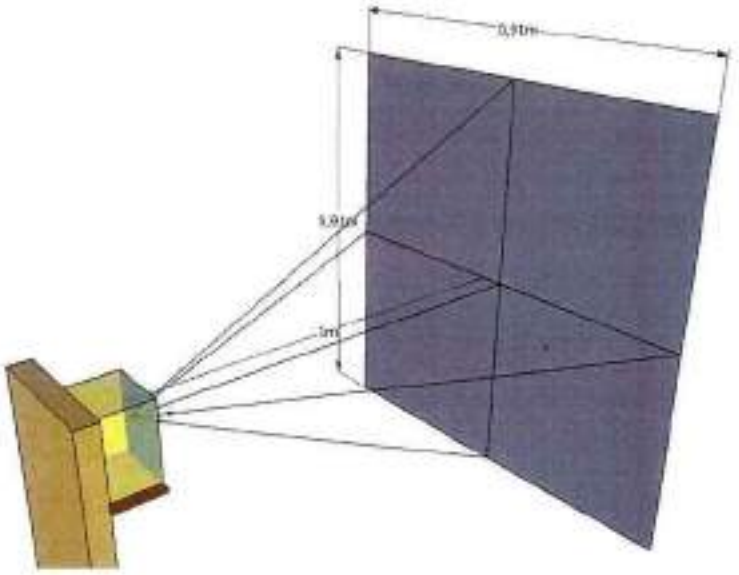

Figura 15 – Esquema apontando dimensões, angulação e visão do sistema

A montagem eletrônica do encaixe da placa CMUcam4 v10 com a placa DFRduino pode ser visualizada na figura 16:

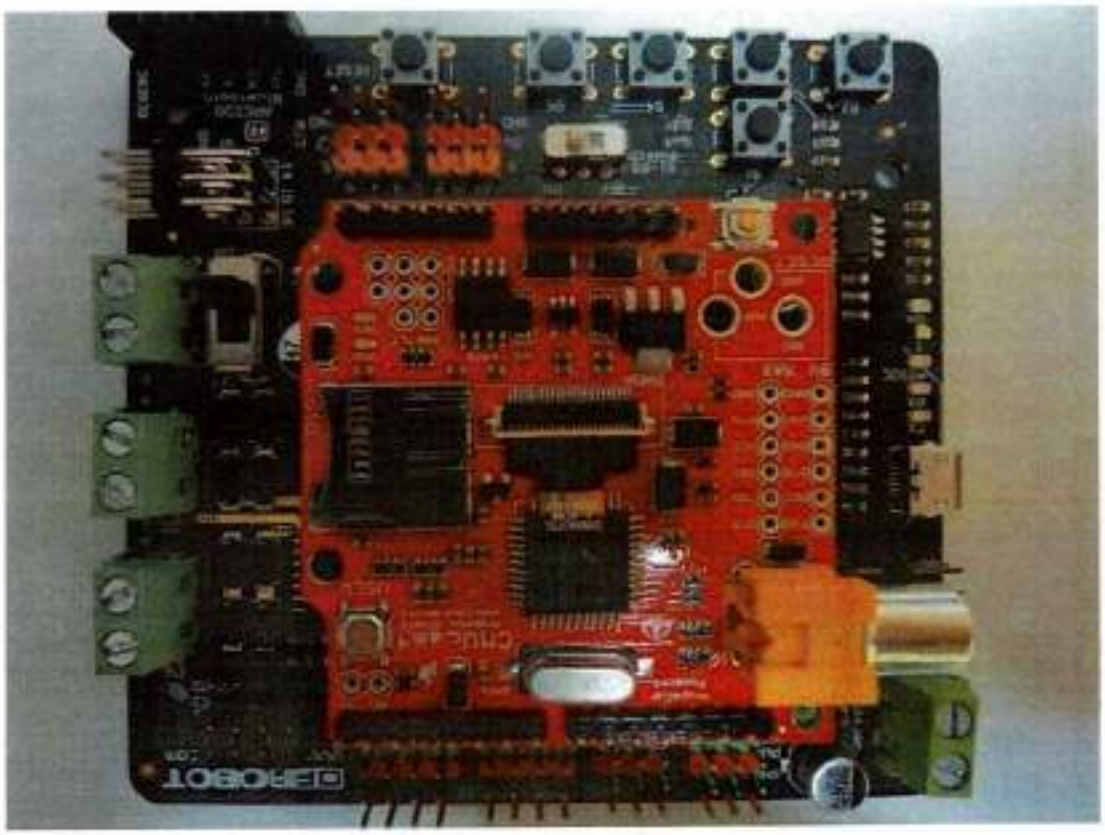

Figura 16: Placa CMUcam v4 interligada à placa DFRduino

8.3 *MONTAGEM DOS COMPONENTES MECÂNICOS E ESTRUTURAIS*

O conjunto descrito no item 6.2 será acoplado a um suporte fixo e nivelado, evitando assim erros de precisão. Ligado a esse conjunto, estará também os servo motores, montados em um dispositivo do tipo Pan/Tilt, posicionados de forma a suportar o disparador, provendo a ele os graus de liberdade necessários para o movimento.

O acoplamento se dará através de parafusos localizados de forma a limitar qualquer movimento inesperado do sistema. Sabemos que existirá torque suficientemente intenso, aplicado pelos servo motores que podem facilmente deslocar todo o sistema para uma direção, comprometendo a confiabilidade do projeto.

O dispositivo Pan/Tilt age como um suporte para os servo motores que transmitirá torque aplicado nas direções X e Y para o disparador em uma velocidade suficiente de interceptação, porém, evitando mudanças bruscas que poderiam comprometer a precisão do tiro.

A montagem e as dimensões do conjunto completo pode ser visualizado, por etapas nos diagramas a seguir:

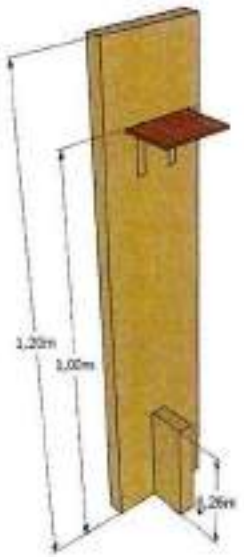

Figura 17: Projeto 3D do suporte estrutural do sistema

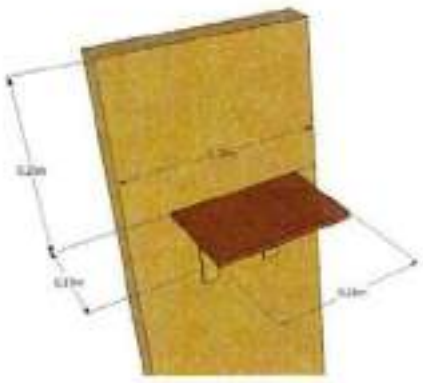

Figura 18: Projeto 3D do suporte estrutural do sistema com detalhe para o patamar de posição dos componentes eletrônicos

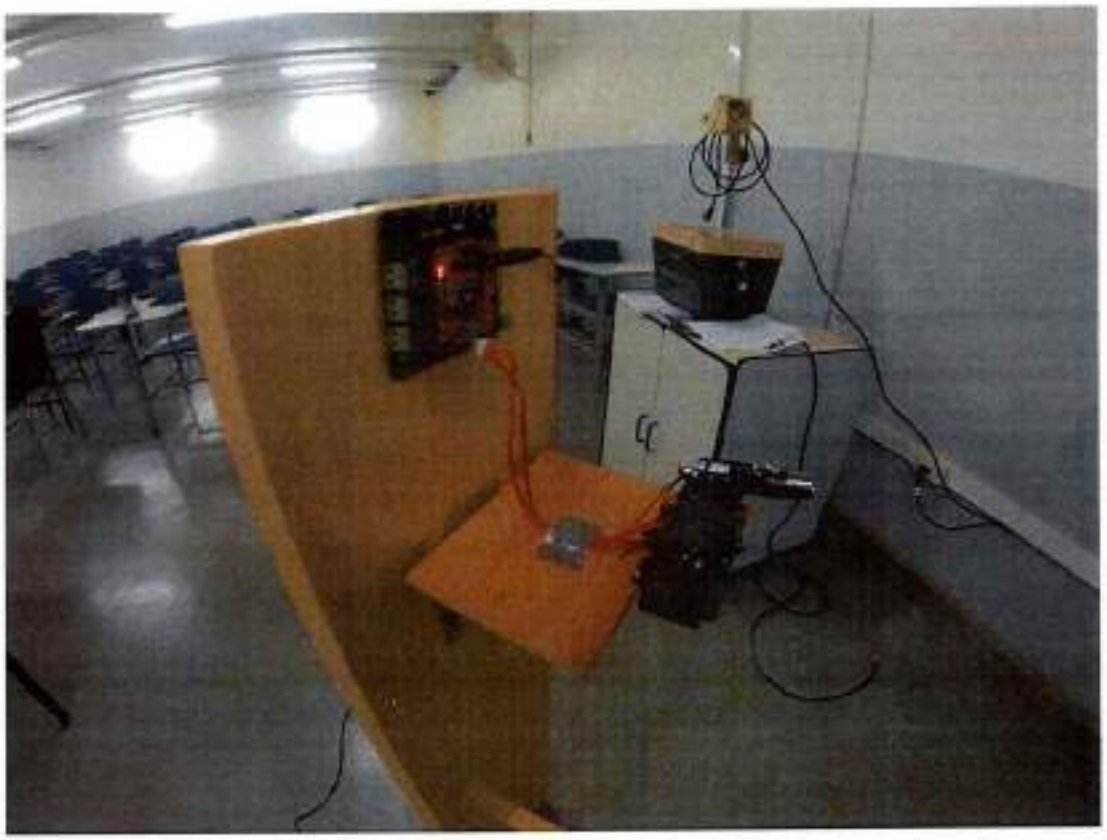

Figura 19: Construção do dispositivo final

8.4 *DESENVOLVIMENTO DO ALGORITMO*

8.4.1 A CALIBRAÇÃO DA CÂMERA

Como explicado previamente, o problema em questão se inicia tratando reconhecimento de uma trajetória no espaço bidimensional, ou seja, com componentes em x e y através de um dispositivo interpretador de imagens. Esta imagem deverá ser devidamente filtrada adequando-se às condições de luz ambiente, e consequentemente ajustando o ganho da câmera para uma captura mais específica.

Para ajustar o ambiente e passar os corretos valores para o algoritmo, utilizamos o software CMUcam4GUI que nos auxilia a determinar os valores limites para R, G e B

reconhecidos pela câmera assim como os modos de operação, taxa de atualização da tela e posicionamento da imagem, de modo estatístico.

O software também nos ajuda a perceber os limites da câmera em relação às distâncias e velocidades máximas dos projéteis a serem lançados pois nos fornece informações em tempo real da captura da câmera.

Nas figuras 17 e 18 a seguir, podemos claramente observar os limites de vermelho, verde e azul captados pela câmera. Parâmetros esses que teremos que definir como constantes ao escrever o algoritmo em C.

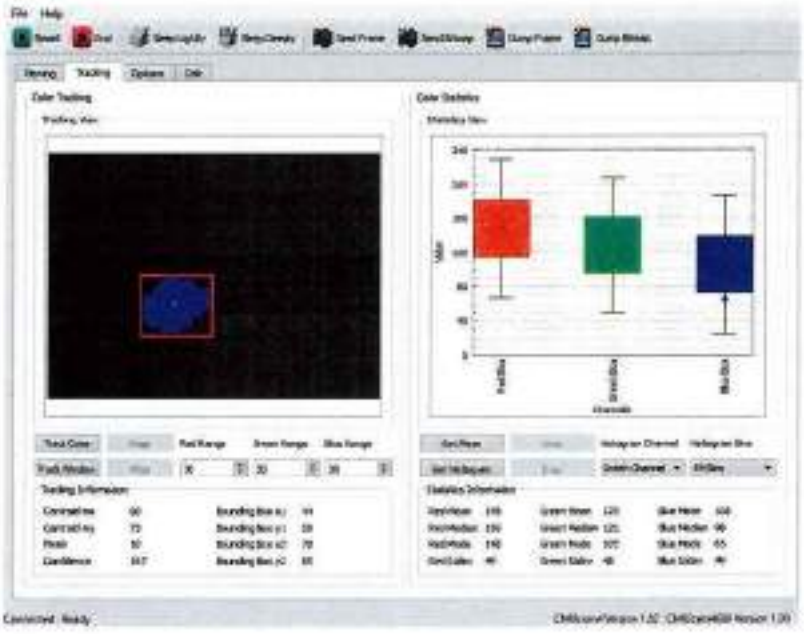

Figura 20: Imagem exemplo do software CMUcam4GUI

Figura 21: Imagem mostrando os parâmetros reconhecidos pela câmera no software CMUcam4GUI

8.4.2 O MODELO MATEMÁTICO

Como apontado no esquemático da figura 14, temos, uma distância D fixa que é o posicionamento do anteparo (plano XY em que o projétil executa sua trajetória), e o centro de massa do projétil rastreado, representado como CM_x e CM_y, componentes em x e y do centro de massa. Além disso, partindo de um ponto central de origem (0,0), possuímos as componentes R_x e R_y que representam as distancias da origem nos dois eixos, como na figura abaixo:

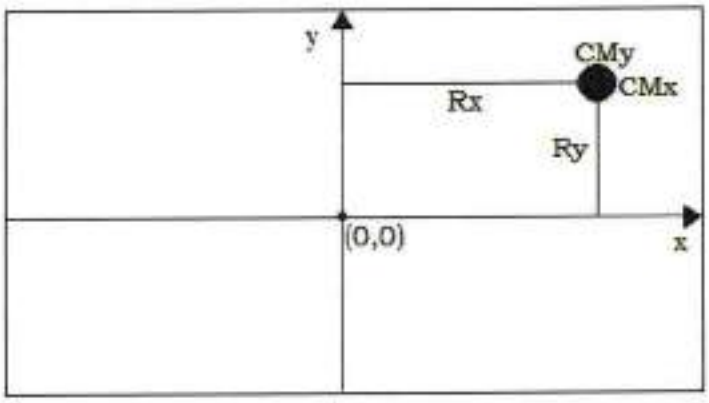

Figura 22: Esquema de visão da câmera mostrando as distancias à origem

Em um breve cálculo, e levando em consideração a quantidade máxima de pixels que a CMUcam4 tem capacidade de capturar por frame (representados por MAX_x, e MAX_y) e o seu FOV (*field of view*), podemos calcular a velocidade em que o servo se movimentará a fim de tornar o movimento o mais suave possível.

$$V_x = \frac{R_x . \theta_{FOV}}{MAX_x . \mathrm{T}}$$

$$V_y = \frac{R_y . \theta_{FOV}}{MAX_y . \mathrm{T}}$$

Onde T é o tempo em que planejamos atingir o alvo, previamente determinado, dependendo das dimensões de da distância D apresentada, dado em segundos

Transformamos então as constantes de nossa equação em uma única que chamaremos de k_x e k_y dadas por:

$$k_x = \frac{\theta_{FOV}}{MAX_x . \mathrm{T}}$$

$$k_y = \frac{\theta_{FOV}}{MAX_y . \mathrm{T}}$$

A partir de agora, para tornar o cálculo mais preciso, consideramos a velocidade máxima do servo e também o máximo sinal que podemos retirar das especificações, como descrito, o servo motor BMS-631MG possui velocidade angular de 60° / 0.1s ou 600° / s (M) e o sinal máximo que podemos mandar é de 1023 (I), o que nos dá:

$$\frac{I}{M} = \frac{1023}{600} = 1.705$$

Multiplicando este valor pelas constantes k_x e k_y temos então o valores finais dos sinais de controle que devemos mandar para os servos dado por:

$$C_x = k_x . \frac{I}{M} . R_x$$

$$C_y = k_y . \frac{I}{M} . R_y$$

Os sinais C_x e C_y serão mandados para os servos como sinais de controle, para o aproveitamento de suas velocidades, a partir de um ΔT previamente definido, o que será discutido no capítulo Testes e resultados.

8.4.3 FLUXOGRAMA DE DADOS

Após o modelamento matemático e seu prático entendimento, para dar início a escrita efetiva do código que futuramente será embarcado no Arduino, devemos executar um planejamento de software envolvendo todas as constantes de variáveis úteis ao projeto a fim de otimizar a performance como um todo.

Neste planejamento, constam todas as operações que o nosso algoritmo irá realizar assim como suas verificações. Em verde estão os caminhos que validam as verificações dadas e em vermelho as que negam.

O algoritmo final consiste de dois loops principais, onde o primeiro é encarregado de verificar se o sistema está pronto para receber dados, ou seja, se está preparado para inicializarmos o processo de rastreamento e disparo. O loop secundário é o loop de iteração que realimentará as variáveis necessárias para que os servo motores se movimentem em direção ao alvo. Além disso, este algoritmo também executa verificações sobre qualidade da imagem nos dando informações sobre a porcentagem de precisão da câmera, o que pode ser modificado, se alterarmos as variáveis iniciais.

(Escrever MAIS)

Os cálculos de média móvel simples estão inclusos no loop secundário, assim como o cálculo de velocidades, apresentado na seção 6.4.2.

O diagrama a seguir, ilustra visualmente o processo de fluxo de dados, que será transferido ao código fonte por meio da linguagem C.

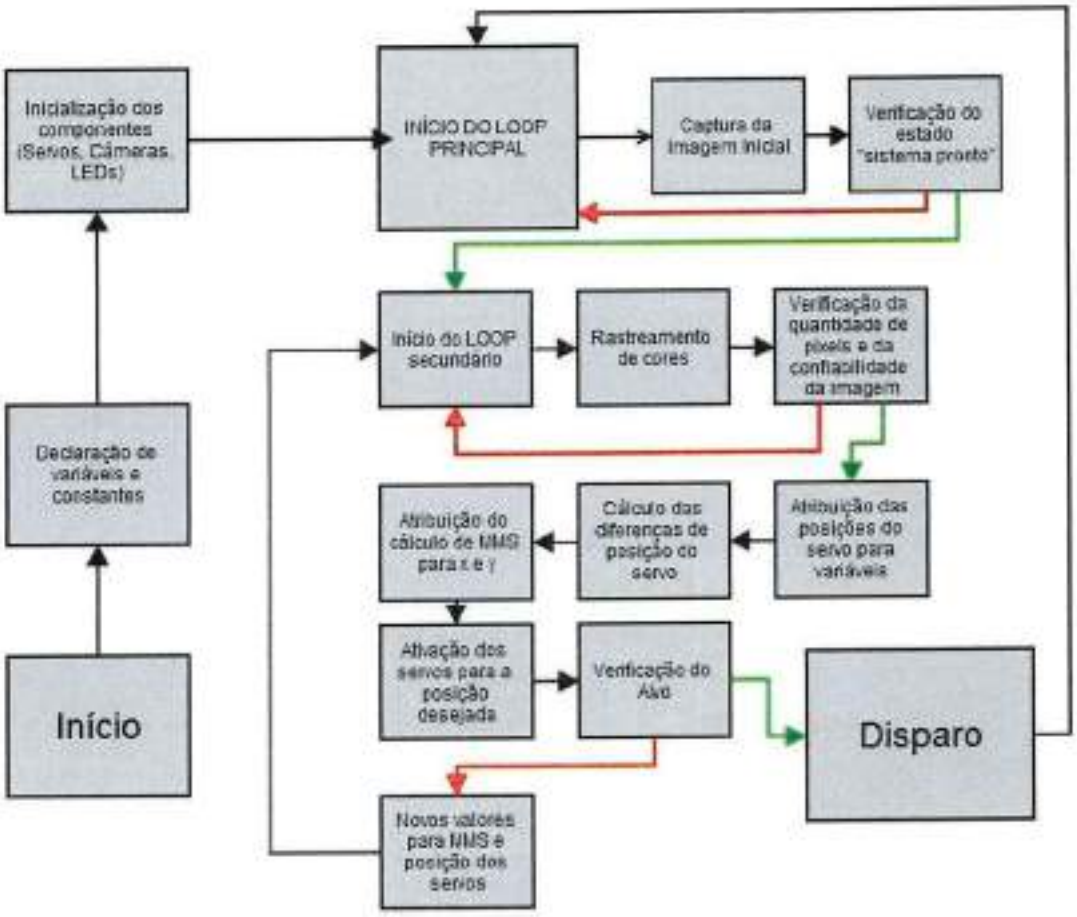

Figura 23: Diagrama do algoritmo de rastreamento e disparo

O código está impresso na íntegra ao final deste documento, no Apêndice B.3 e segue basicamente o descrito e apresentado visualmente neste capítulo.

9 *TESTES E RESULTADOS*

9.1 *PLANEJAMENTO E DESCRIÇÃO DOS TESTES*

A realização de testes sempre é necessária, não importa com qual tipo de sistemas estamos lidando nem tampouco o grau de complexidade e detalhamento teórico sobre o qual o projeto é desenvolvido.

É na etapa de testes em que podemos validar as proposições teóricas antes realizadas. A maioria dos projetos envolvem simplificações e abstrações de fatos e efeitos que ocorrem em situações reais, e é também neste momento de testes que podemos analisar quais os efeitos destas simplificações, e então verificar se tais simplificações foram feitas de maneira correta, ou se transformaram o problema em algo tão simples que se torna incorreto ou impreciso.

Nesta sessão iremos descrever quais e como foram realizados os testes, bem como o quais parâmetros e quais simplificações buscamos pôr a prova em cada um dos testes. Nós dividimos os testes em 3 grupos: testes de câmera, testes de servo e testes de captura de movimento, e a descrição e metodologia de cada um deles está descrita a seguir.

9.2 *TESTES DE CÂMERA*

Os testes com a câmera foram necessários para poder avaliar algumas características do ambiente em que será capturada as imagens, para o funcionamento correto nosso projeto. As características que estão sendo avaliadas são as seguintes:

- Alta capacidade de distinção da cor utilizada, em fundo branco;
- Tamanho do objeto a ser interceptado, na distância especificada, suficiente para boa captura do centro de massa do mesmo;
- Área capturada pela câmera, na distância determinada nos requisitos;

Além das características do nosso ambiente, também realizaremos testes com a finalidade de otimizar a captura do posicionamento do objeto. Dentro desta ideia, serão realizadas os seguintes testes, buscando a melhor forma de se realizar o tratamento prévio de imagens:

- Estabelecimento de pesos para as cores no sistema RGB
- Aplicação de filtros nas imagens
- Controle dos ganhos da câmera

Após a coleta de dados e calibração da CMUcam4 como descrito na seção 6.4.1 deste documento, iniciamos então os testes de imagem. O projétil a ser rastreado nos testes utilizado foi uma bola de tênis comum, que possui diâmetro entre 6,35cm e 6,67cm e também, possui uma cor amarelada forte, o que facilita no reconhecimento pela câmera.

O amarelo (ou amarelado), corresponde aos valores no entorno de (255,255,0) no sistema RGB o que nos aponta um caminho a seguir, consequentemente, para a câmera também, que ao não dar relevância significativa a um canal (BLUE), consegue capturar melhores imagens, mesmo com sombras e movimentos abruptos. Os testes que colocavam um peso maior para esse terceiro canal se mostraram significativamente menos satisfatórios ao rastrear a cor amarela, ou em torno do amarelo.

A CMUcam4 possui em seu firmware embarcado, algumas variáveis que podemos modificar a fim de adequar suas capacidades aos nossos requisitos. Além do já descrito software CMUcam4GUI que nos ajuda a reconhecer e calibrar a visão através das cores e luzes do ambiente, a biblioteca embarcada nos permite colocar filtros de ruído (como pode ser observado no Apêndice B.3 com o comando *cam.noiseFilter(NOISE_FILTER)*) além de controlar o ganho automático da câmera em sua tentativa de se adequar a luz ambiente.

Começamos então, obtendo imagens da bola de tênis pela câmera, conectada à saída RCA da CMUcam4 ligada a uma TV para obter uma melhor visualização. Neste momento, a bola de tênis ainda estava sendo erguida e comandada manualmente por trajetórias aleatórias.

Podemos observar que o contraste gerado pela bola de tênis em fundo branco é satisfatório, dado que os limites da bola, bem como seu centro de massa, podem ser determinados facilmente, mesmo nas regiões de borda da esfera, que por questões de iluminação, acabam modificando a tonalidade da cor da bola. Também pudemos observar, com a movimentação da bola, que o posicionamento do centro de massa e das bordas da imagem é estável, o que indica que o estabelecimento, em software, de pesos diferenciados com relação às componentes RGB, bem como a aplicação dos filtros corretos nos levou a um bom resultado.

Em seguida, posicionamos na distância D especificada nos requisitos de engenharia e marketing, e primeiramente verificamos que na distância especificadas, temos ainda uma boa definição das bordas e centro de massa do alvo, o que nos faz concluir que o alvo está com um tamanho satisfatório para as especificações.

Depois, ainda com o alvo a uma distância D da câmera, o movimentamos para as bordas de captura da câmera, a fim de registrar seus limites, e assim verificamos que o tamanho do plano a ser monitorado, de área de $1 m^2$, distando 1m da câmera, está de acordo com o que foi planejado e especificado.

9.3 *TESTES DE SERVO*

O servo mecanismo é o que representa a parte mecânica de nosso sistema, e foi imaginado que este seria o elemento que teria maior probabilidade de acrescentar atrasos ao nosso sistema.

Nesse cenário, precisamos realizar testes que primeiramente comprovem as especificações do fabricante na velocidade de movimentação do servo, e depois para verificar se esta velocidade e seus atrasos envolvidos estão de acordo com a especificação de velocidade máxima do projétil a ser interceptado.

Os testes de movimentação dos servos foram baseados nos testes encontrados na literatura quando se trata de servo motores funcionando concomitantemente com a placa arduino.

O software básico utilizado está disponível no SDK (*Software Development Kit*) do arduino e se encontra também impresso no Apêndice B.2. O código descreve uma simples

tarefa que posiciona o servo de acordo com a variação de um potenciômetro. Com estes testes, conseguimos verificar uma série de informações contidas no *datasheet* do nosso servo motor (BMS-631MG) além de retirarmos os nossos próprios dados pois, em se tratando de um dispositivo mecânico, existe um número considerável de variáveis suscetíveis à uma margem de erro às quais não podemos ter total confiança.

Segundo a documentação do servo motor, ele é capaz de se movimentar 60° em 0.1s dada a direção especificada. Em nossos testes, que foram feitos com o servo já conectado no arduino, ou seja, está acrescido um tempo de atraso entre o sinal e a efetiva movimentação do motor, detectamos que o movimento é significantemente menor chegando aos 60° em 0.2s ou seja, o atraso de comunicação entre o servo e o arduino, representa uma perda de 50% da velocidade do servo entre dois comandos dados pelo software. Os resultados por nós coletados foram medidos a partir da comunicação RX/TX entre o arduino e o PC, que possui atraso e também é contabilizado na perda de velocidade.

Para uma melhor visão sobre os limites de nossos componentes, devemos avaliar qual seria a velocidade máxima do projétil, em função apenas da resposta do servo. Para isso, precisamos determinar a velocidade tangencial partindo da velocidade angular.

Partindo, então, desta velocidade média angular, com atrasos, de 60° em 0.2s e estando a uma distância de 1m, temos então, como podemos observar pelo desenho em 3D gerado, ou por formulas trigonométricas simples:

$$Tg\ 60° = \frac{X}{D}$$

Sendo D a distância até o plano e X a continuação da projeção da velocidade do servo a esta distância (como mostra a figura 23).

Então, temos:

$$Tg\ 60° = \frac{X}{1}$$

$$X = \sqrt{3}$$

$$X \approx 1{,}732$$

Figura 24: Determinação da velocidade tangencial do servo

Então, concluímos, que o servo motor tem capacidade, na nossa aplicação, de se deslocar aproximadamente 1,73m em 0,2s.

$$\frac{1,73}{0,2} = 8,65\ m/s$$

Este valor é a maior velocidade com que o servo motor conseguiria acompanhar um objeto, estando este a uma distância de 1m. Dessa forma, podemos perceber que este valor é bastante elevado, confirmando a suposição de que a relação de frame por segundo da câmera é, de fato, o nosso fator limitante.

Com estes dados computados, podemos agora compensar o nosso sistema para ir de encontro com os resultados práticos, logo, necessitaríamos estudar a soma de todos os atrasos e soma-los com o atraso do disparador para sabermos o tempo que levaria para o objeto disparado atingir o projétil em movimento. Mas para tanto, temos que também estudar a velocidade com que a câmera consegue capturar uma imagem em movimento que é o tópico da seção seguinte.

9.4 *TESTES DE CAPTURA DE MOVIMENTO*

Com relação à captura de movimento, temos que o projétil deve se manter em uma velocidade abaixo do limite para o qual a câmera consegue capturar amostras sucessivas com precisão, e dessa forma, conseguindo alimentar o arduino com informações suficientes para a correta determinação da trajetória do alvo.

Iniciamos os testes de velocidade e captura de movimento da câmera em relação a variação da velocidade do projétil. Atravessamos o projétil de ponta a ponta dos limites da câmera em velocidades crescentes, a partir de 0.1m/s. A velocidade de testes foi calculada a partir do intervalo de tempo que levava o projétil entre os limites X e Y do plano de visualização.

Durante os testes de velocidade, tentamos aproximar os nossos movimentos de uma parábola, a fim de tornar os testes mais próximos da realidade do projeto. O projétil estaria em queda livre, descrevendo um movimento parabólico com velocidade em X assumida constante por hipótese (desprezando a resistência do ar) e a velocidade em Y como um movimento uniformemente variado, com aceleração constante de aproximadamente $9{,}8m/s^2$.

Os resultados dos testes, que indicam se a câmera consegue acompanhar o trajeto descrito pelo projétil, com passo constante de 0.1m/s por etapa aproximadamente, é descrito na tabela a seguir:

Velocidade (m/s)	Resultado
0.1	Rastreamento OK
0.2	Rastreamento OK
0.3	Rastreamento OK
0.4	Rastreamento OK

0.5	Rastreamento OK
0.6	Rastreamento OK
0.7	Rastreamento OK
0.8	Não rastreado
0.9	Não rastreado
1	Não rastreado
1.2	Não rastreado
1.3	Não rastreado

Tabela 1 – Testes de Câmera

Os testes foram feitos de forma aproximada, pois o projétil estava se locomovendo manualmente em alguns casos. Porém, fica claro pelos resultados que existe um momento em que a câmera não tem mais capacidade de seguir o objeto com a confiabilidade e a porcentagem de pixels necessária para passar os parâmetros de centro de massa e bordas do objeto.

Estão descritos na Tabela 1, testes de câmera que foram feitos após o ajuste de cores e ganho, assim como filtros de ruído já embarcados no firmware da câmera. Testes estes que foram aperfeiçoados através de uma série de outras tentativas de otimizar o desempenho da câmera que podem ser feitos modificando seus parâmetros. Temos plena consciência de que estes resultados podem ser melhorados com um ajuste minucioso levando em conta também a luz ambiente e as sombras do objeto.

9.5 *TESTE DO SISTEMA (DISPOSITIVO FINAL)*

Após a coleta de dados de todos os componentes individualmente, e agrupados em subsistemas porém não de maneira completa, iniciamos o teste final do dispositivo.

O dispositivo foi montado como no esquema das figuras 16, 17 e 18 e o plano de fundo foi colocado a 1m de distância como na figura 15. Os servo motores foram montados no esquema pan/tilt descrito anteriormente e em seu suporte, posicionado o disparador.

O software foi embarcado no DFRduino por meio de uma interface Serial/USB conectado a um PC e se manteve desta forma durante todo o teste pois este software sofria constantes modificações e ajustes.

Em um primeiro teste, usamos como plano de fundo uma folha A0 branca, que possui dimensões de 1189mm x 841mm, maior que o especificado, para evitar determinados ruídos de borda, que respeitam o FOV da CMUcam4 ($\theta_{FOV} = 66.5°$).

O atraso total foi determinado como a soma de todos os atrasos previstos anteriormente da seguinte forma:

$$\tau_{total} = \tau_{cam} + \tau_{servos} + \tau_{disparador} + \tau_{arduino}$$

Onde os atrasos dos subsistemas tratados de forma individual é dada por:

$$\tau_{cam} = \tau_{FPS} + \tau_{stream}$$
$$\tau_{servos} = 0$$
$$\tau_{disparador} = \tau_{solenoide}$$
$$\tau_{arduino} = \tau_{acCam} + \tau_{acServo} + \tau_{acDisparador} + (\tau_{RX/TX})$$

De todos estes atrasos, os que precisariam ser extraídos experimentalmente seriam todos aqueles relacionados com o arduino (com exceção de $\tau_{RX/TX}$ pois este seria eliminado após a fase de testes), além do $\tau_{solenoide}$.

Os parâmetros τ_{FPS} e τ_{stream} foram previamente calculados na seção 8.2 onde chegamos ao resultado de 25FPS ou 40ms/*frame*, porém não podemos considerar que a câmera acarreta um atraso de 40ms a cada frame o atraso é agregado apenas no resgate dos dados através do comando *cam.getTypeTDataPacket()*. Considerando que a câmera já esteja em modo *stream* (iniciado pelo comando *cam.trackColor()*) este atraso acarreta 4ms a cada frame rodado tomando como referência a chamada do comando uma vez a cada 10 frames.

Logo, teríamos que, por métodos iterativos, estimar o atraso do sistema total a fim de atingir o alvo com relativo sucesso.

Através dos softwares descritos na íntegra nos apêndices B.1 e B.2, fomos capazes de, mesmo com todo o circuito montado, eliminar variáveis indesejadas no momento, tornando o sistema sem solução viável. Com o software teste para servo (B.1), descontamos o atraso natural calculado do servo motor, 0.2s para 60° e, após 10 testes consecutivos,

estimamos uma média de 15ms a cada vez que interagirmos com os servos, ou seja, para cada frame que ele é acionado e, considerando que ele é acionado toda vez que a câmera é acionada, dividimos 15ms por 10 frames rodados fazendo com que $\tau_{acServo=1,5.frame}$.

Utilizamos o software descrito no Apêndice B.2 para estimar o tempo de atraso do acionamento da câmera, assim como nos servos. Após mais 10 testes consecutivos, percebemos que a câmera leva por volta de 33ms para entrar em modo *stream*, ou seja, iniciar um recebimento de dados contínuos, e, só acionamos a câmera desta forma uma única vez durante o loop secundário (seção 8.4.3), portanto, τ_{acCam}se torna desprezível para esta aplicação.

O parâmetro $\tau_{acDisparador}$ foi calculado como a diferença em milissegundos do acionamento de um pino digital da placa DFRromeo através do comando *digitalWrite(m_disparador, HIGH)* que direciona um sinal lógico '1' (5V) para o pino digital atribuído à variável m_disparador. Este tempo de atraso também foi imperceptível ao executarmos o teste em um código-exemplo, disponível na biblioteca arduino.

Nos restava então calcular o parâmetro $\tau_{solenoide}$ ou seja, o tempo entre a ativação do pino do disparador e o disparo efetivo. Por não possuirmos instrumentos precisos o suficiente para calcular este intervalo além de não termos o controle do momento exato no qual o disparo é feito pela saída serial do arduino, consideramos $\tau_{solenoide} = 20ms$ como descrito no manual do fabricante.

Portanto a equação 9.1 com os devidos parâmetros calculados, se torna:

$$\tau_{total} = 4.frame + 0 + 20ms + 1{,}5.frame$$

Nos restava então encontrar o número total de frames que seriam necessários a cada disparo. Além disso, devemos encontrar uma relação entre a quantidade de graus em que podemos movimentar os nossos servos e o número de frames necessários para que o projeto seja bem sucedido. O atraso total contém uma constante de 20ms (por que é calculada a cada disparo) e duas parcelas que multiplicam a quantidade de frames por disparo, dado por:

$$\tau_{total} = 4.frame + 20ms + 1.5.frame$$

$$\tau_{total} = (20 + 5{,}5.frame)ms$$

O que nos pareceu bastante plausível quando observamos a olho nu, de acordo com esta equação, após 2 segundos a 25 FPS, ou seja, 50 frames, temos um atraso de 295ms ou 1/3 de segundo.

Figura 25: Imagem de um ensaio bem sucedido do sistema

9.6 *RESULTADOS*

As previsões teóricas e os testes realizados proveram uma base sólida para os resultados que são apresentados nesta seção. Com os testes, verificamos principalmente a importância das condições ambientais para o ótimo funcionamento do sistema reconhecedor de imagens.

O sistema se comportou como previsto e projetado na maioria dos testes. Alguns fatores que não foram considerados inicialmente contribuíram com o funcionamento faltoso em várias das análises e, atribuímos a isso, primariamente às condições de calibração do sistema de rastreamento da CMUcam4.

Além disso, no processo de construção do disparador de projéteis não conseguimos a tempo uma válvula solenoide que pudesse ser acionada a baixa potência, que é o caso da placa DFRduino. Os testes com o disparador apresentados na seção 9.4 foram feitos independentemente do restante do sistema. A contribuição de 20ms para o atraso total não foi considerada nos resultados.

Para verificarmos se o sistema funcionava como previsto, acoplamos um laser do tipo apontador, encontrado em larga escala no mercado e, suficientemente os testes e resultados foram efetuados e analisados.

A posição inicial do projétil a ser rastreado e sua velocidade foram os fatores predominantes os quais tornaram o sistema faltoso dependendo do caso. A velocidade era um fator que estava previsto em nossos testes, porém a posição inicial nos surpreendeu em certo ponto.

A taxa de acerto do sistema, levando em consideração todos os casos foi de 77%. Se reduzirmos o espaço amostral dos testes para uma certa posição inicial (0.1m a partir do solo), este número cai para 25% dos ensaios funcionando perfeitamente. O gráfico a seguir demonstra a variação de acerto em função da posição inicial.

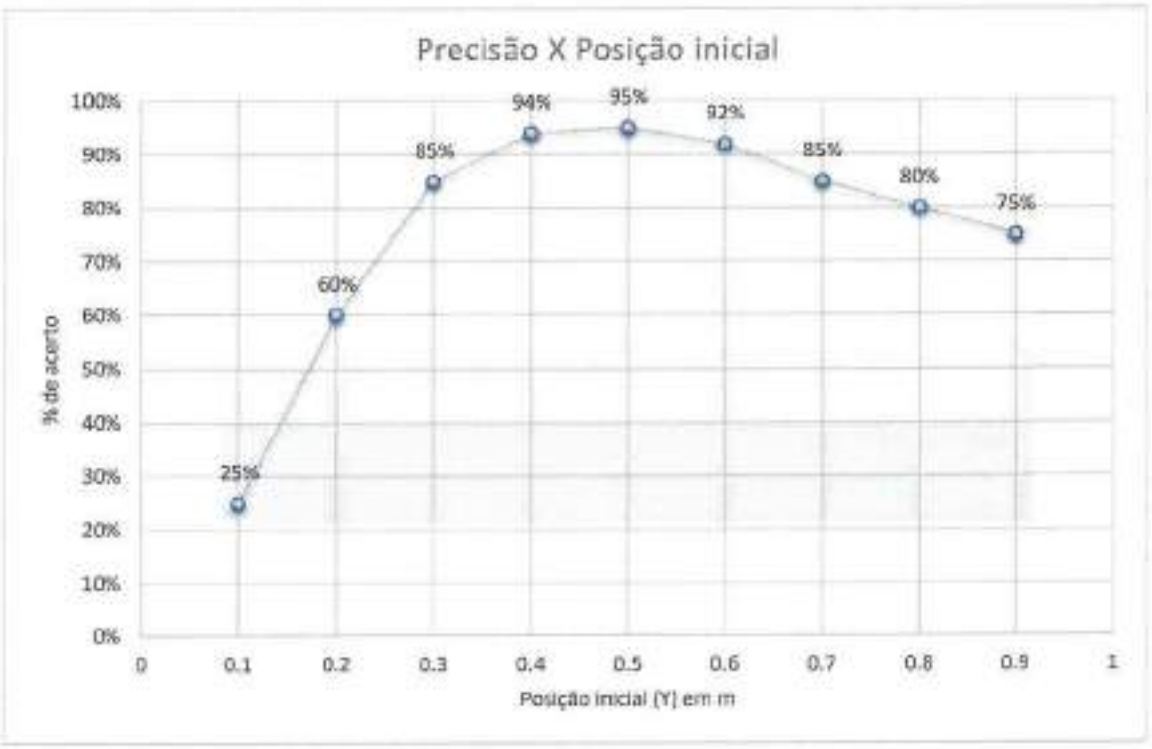

Figura 26: Gráfico de precisão X posição inicial

Os resultados nos mostram uma deficiência sensível no algoritmo final que não consegue lidar efetivamente com dados que recebe em determinadas regiões.

Em uma análise mais profunda, notamos certa discrepância nos dados recebidos pela câmera em pontos ao redor das bordas. Esta situação alia-se ao fato do sistema mecânico necessitar de um tempo para se ajustar aos novos dados. Isto é, se os dados recebidos ao redor da borda não são consistentes, não há tempo hábil para os servo motores se posicionarem corretamente, tornando o sistema instável em algumas situações.

10 CRONOGRAMA

O presente cronograma descrimina as atividades que serão realizadas ao longo do ano, de forma simplificada. Ele contém uma visão geral das tarefas realizadas de março de 2013 até novembro de 2013, totalizando o tempo de trabalho total de 9 meses.

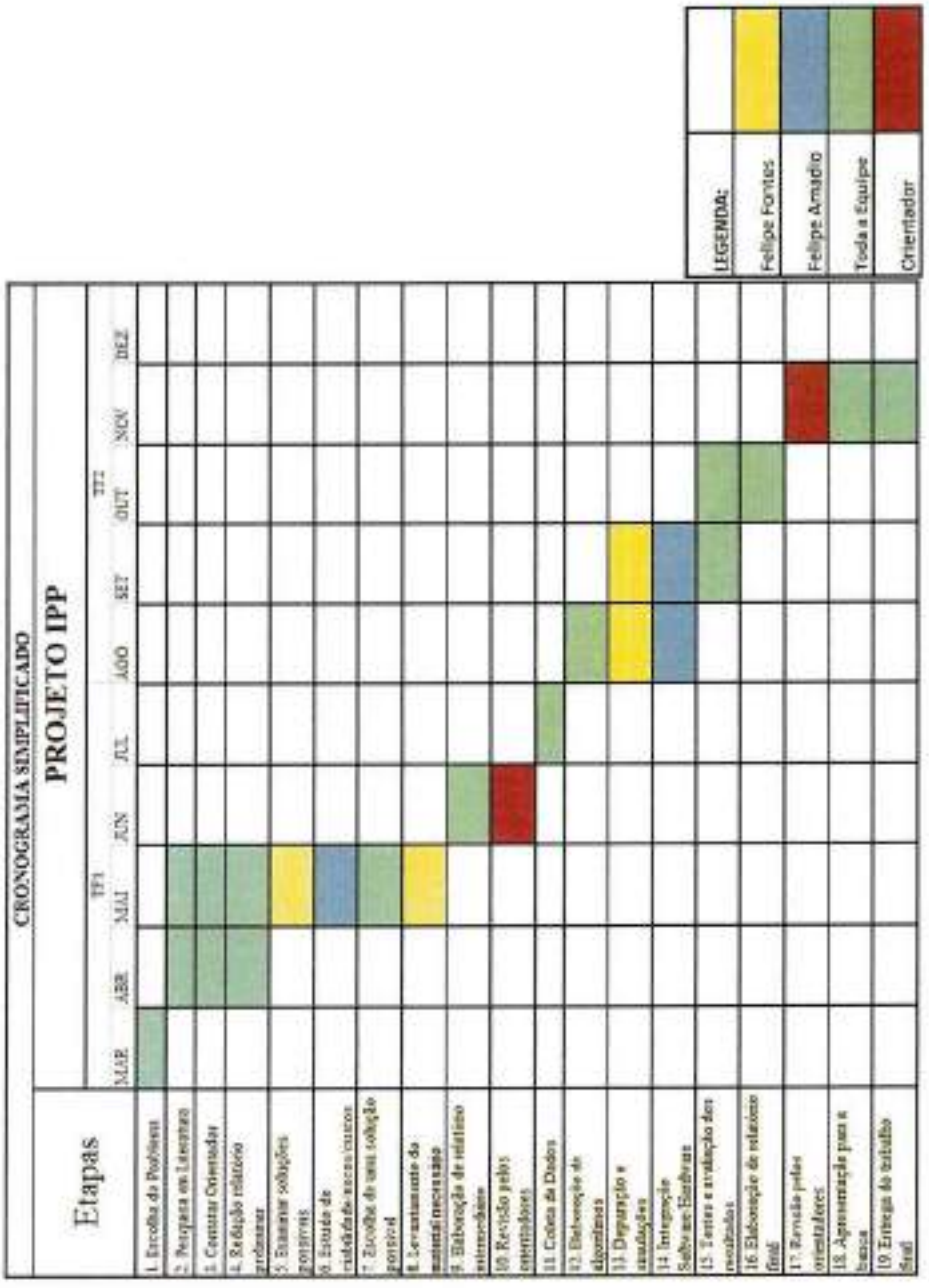

Figura 27: Cronograma Detalhado do projeto IPP

11 ORÇAMENTO

A análise de custos foi feita baseada na solução adotada, levando em consideração todos os componentes físicos e licenças de software.

- 2x BMS-631MG Super-Fast Servo .....US$ 30,00
- Pacote com tubos PVC pra montar o lançador..... R$80,00
- Placa CMUCam4.....US$99,00
- Arduino Romeo AIO (para duplo servo) .....R$159,00

Total.....R$499,00

12 CONCLUSÕES

O projeto IPP de uma forma geral foi bem sucedido em sua meta de desenvolver um sistema robusto e estável de rastreamento de um projétil em movimento em um plano. O desenvolvimento do sistema, atingiu aos requisitos de engenharia e de marketing no que diz respeito a confiabilidade e precisão de projeto atingindo 95% de precisão dadas as condições necessárias de luz, velocidade e posição inicial do projetil rastreado.

A interceptação se deu por meio de um *laser pointer* ao invés de um lançador de projéteis como descrito no projeto inicial dada uma situação onde ocorreu a falta de potência a ser transferida, não sendo possível efetuar uma mudança fundamental abrupta no desenvolvimento do sistema como um todo.

Com suas fundações absolutamente acadêmicas, o projeto IPP cumpre com a tarefa de se tornar um objeto que possa auxiliar futuros projetos que envolvam algoritmos de controle, assim como de movimentação de servo motores e reconhecimento de imagens.

Aos realizadores do projeto, se enfatiza o vasto aprendizado e um conhecimento aprofundado nas áreas de controle, sistemas eletrônicos e computação, além da conceituação, elaboração e execução de projetos, complementando o curso de graduação da Escola Politécnica de Engenharia da Universidade de São Paulo.

12.1 POSSÍVEIS APRIMORAMENTOS FUTUROS

- Calibragem minuciosa da câmera, ou tornar o sistema de reconhecimento mais robusto, modificando também as bibliotecas da CMUcam4 para arduino.

- Estudo cauteloso sobre as condições de contorno que permeiam as regiões afetadas pela falta de consistência de dados (regiões de borda).

- Projetar um sistema alternativo, ou complementar, que amplifique a potência de saída do sistema de uma forma abrangente, para, consequentemente, acionar uma válvula solenoide com 5V (sinal lógico '1')

- Desenvolver um sistema com mais pontos de recolhimento de dados (como uma câmera extra), possibilitando a inclusão de mais um eixo de rastreamento, fazendo com que o projeto tenha funcionalidade também em aplicações 3D.

- Aprimoramento do algoritmo de rastreamento, eliminando pontos de parada crítica e tratando situações pouco prováveis acrescentando robustez ao código.

- Substituição do DFRduino Romeo AIO que opera como um Arduino Leonardo para outro circuito embarcado que possui um microcontrolador mais rápido, minimizando ainda mais os atrasos.

13 REFERÊNCIAS BIBLIOGRÁFICAS

[1] RAFAEL ADVANCE DEFENSE SYSEM. *Iron Dome –Defense against Short Range Artillery Rockets*. Disponível em <http://www.rafael.co.il/Marketing/186-1530-en/Marketing.aspx>

[2] WIKIPEDIA. *Anti-ballistic Missile*. Disponível em <http://en.wikipedia.org/wiki/Anti-ballistic_missile>

[3] FABIO BOBROW, BRUNO MUNIZ, GIHED NASSIF E KAIQUE GROSSMANN. – AsysDefense (Instituto Mauá de Tecnologia) Disponível em <www.asysdefence.com>

[4] WIKIBOOKS. *Média móvel simples (MMS)*. Disponível em <http://pt.wikibooks.org/wiki/Logística/Técnicas_de_previsão/Decomposição_de_séries_temporais/Médias_móveis/Média_móvel_simples>

[5] KLOSE, R.; RUCKELSHAUSEN, A.; Thiel, M.; MARQUERING, J. (2008): *Weedy – a Sensor Fusion Based Autonomous Field Robot for Selective Weed Control*. Tagung LAND.TECHNIK 2008, VDI-Verlag, 25.–26.09.2008, Stuttgart- Hohenheim, pp. 167–172

[6] LARSON, JAMES: *Scratchy: An Autonomous Pool-Playing Robot*. University of Florida, Department of Electrical and Computer Engineering, Intelligent Machines Design Laboratory, EEL5666 Fall 2002

[7] ABCOUWER, N.; WASSERMAN, B.; WANG, Q.; CARLSON, J. (2013): *Mechatronic Design Final Report*, 16-778 / 18-578 / 24-778 Spring 2013 Carnegie Mellon University

[8] MUSGRAVE, R.; MYERS, W.; JANG, S.; JIA, D.; *Final Report*. 18-578 Mechatronic Design Spring 2012

[9] WIKIPEDIA. *Analytic Hierarchy Process*. Disponível em <http://en.wikipedia.org/wiki/Analytic_hierarchy_process>

[10] DELURGIO, Stephen A. *Forecasting principles and applications*. Singapura: McGraw-Hill, 1998.

[11] MICROCHIP. *Datasheet* PIC18F2520. Disponível em <http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010277#documentation>

[12] ALTERA CORPORATION. *FPGA CycloneII – DATASHEET*. Disponível em <http://www.altera.com/literature/lit-cyc2.jsp>

[13] DFROBOT. *Romeo All-In-One Controller*. Disponível em <http://www.dfrobot.com/wiki/index.php/Romeo_V2-All_in_one_Controller_(R3)_(SKU:DFR0225)>

[14] MATHWORKS. *MatLab –The Language of Technical Computing*. Disponível em <http://www.mathworks.com/products/matlab/>

[15] EXPERT CHOICE. *Software Comparion 5.0 Trial*. Disponível em <http://expertchoice.com/>

[16] BLUE BIRD. *Product Specifications - BMS-631MG*. Disponível em <http://www.blue-bird-model.com/en/servos/bms-631mg.html>

[17] MAKEZINE. *Compressed Air Rocket*. Disponível em <http://makezine.com/15/airrocket/>

[18] CMUCAM. *CMUCam4 v10*. Disponível em <http://cmucam.org/projects/cmucam4/wiki/SparkFun_Camera#SparkFun-CMUcam4-v10>

[19] WIKIPEDIA. *Work breakdown structure (WBS)*. Disponível em <http://en.wikipedia.org/wiki/Work_breakdown_structure>

APÊNDICE A – HARDWARE

A.1 PLACA ROMEO RFDUINOV2

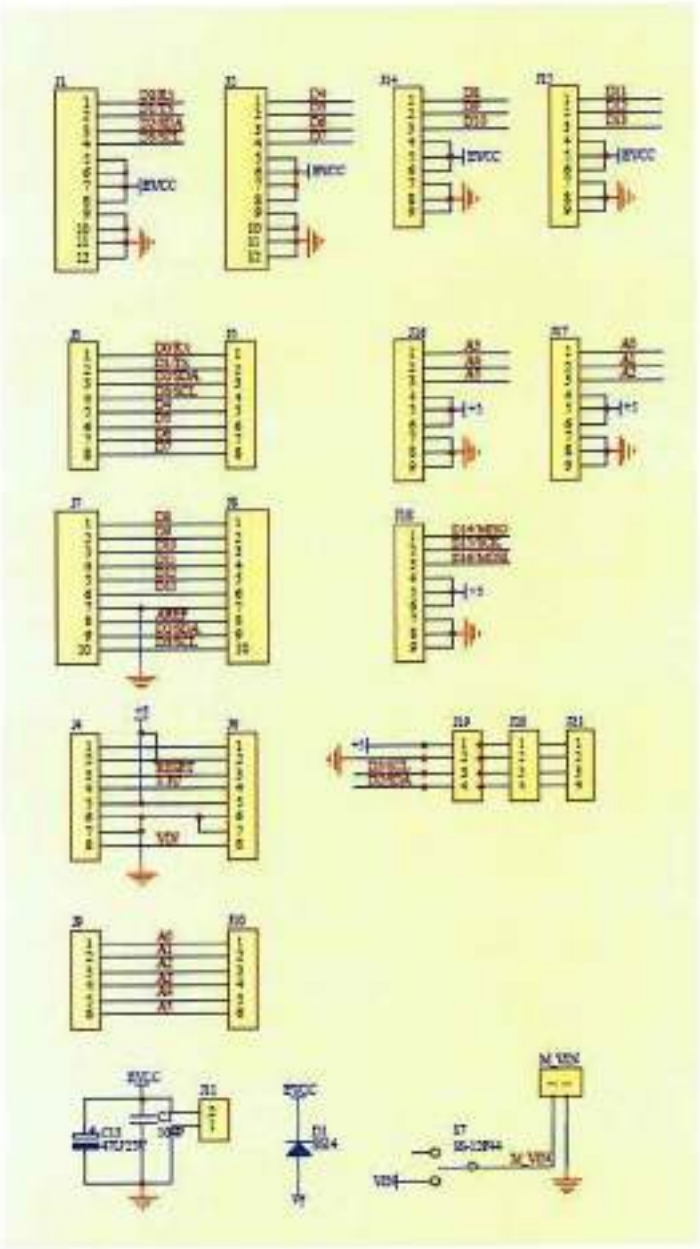

Figura 28: Jumpers de entrada DFRduinoV2

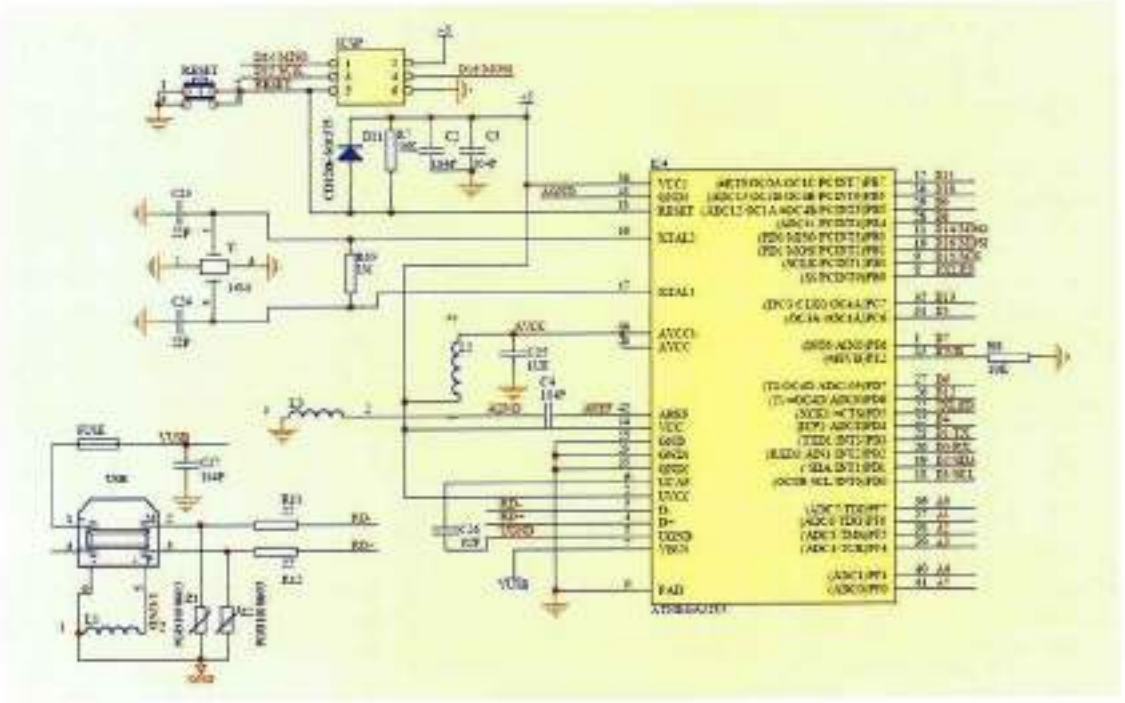

Figura 29: Esquemático de entradas e saídas do microcontrolador ATmega32u4

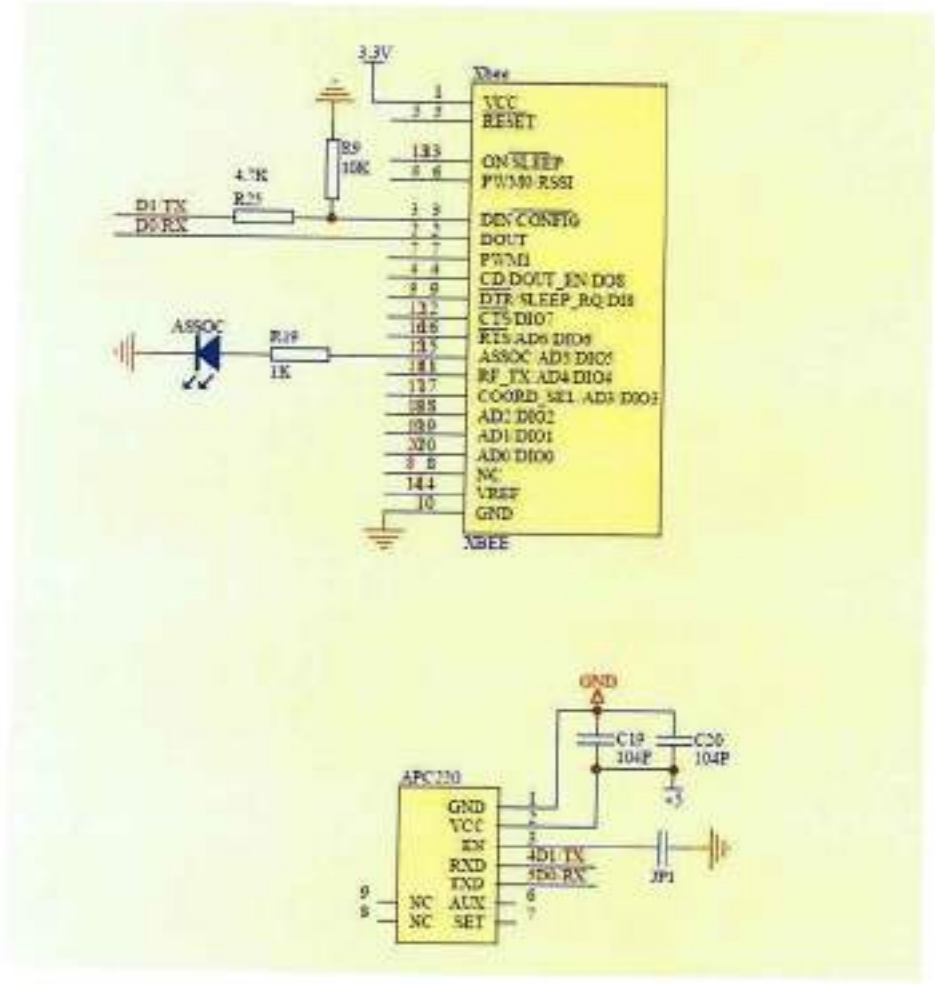

Figura 30: Controlador Xbee embarcado no DFRduinoV2

Figura 31: Pinos Analógicos e LEDs RX/TX

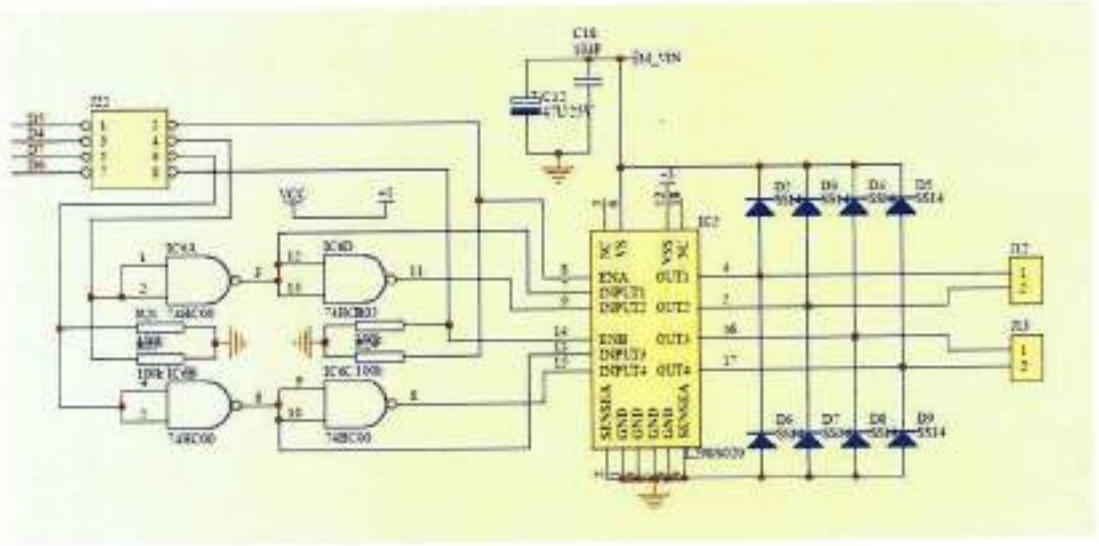

Figura 32: Esquemático de diagrama lógico da placa DFRduinov2

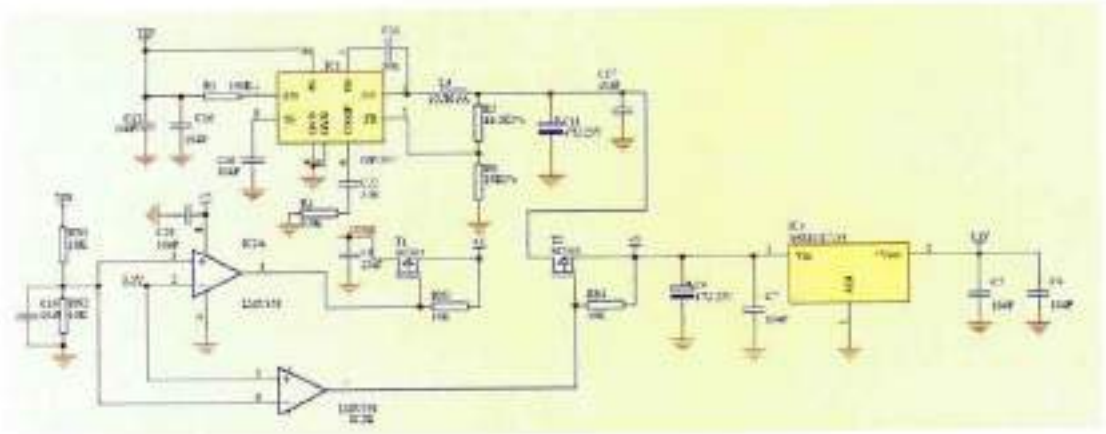

Figura 33: Regulador de tensão da placa DFRduinov2

A.2 PLACA CMUCAMV4

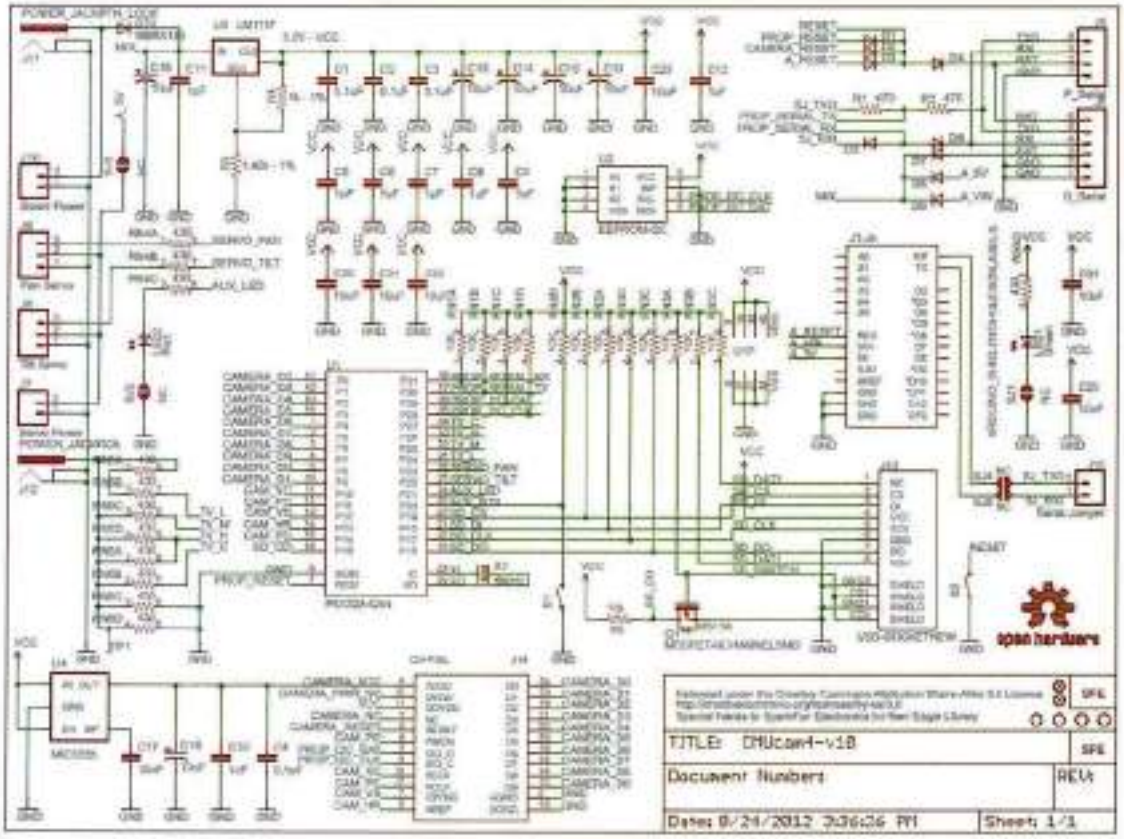

Figura 34: Esquemático completo da placa CMUcam4

APÊNDICE B – SOFTWARE

B.1 TRECHO DE CÓDIGO COMENTADO – EXEMPLO SERVO

```
//Código baseado na biblioteca Servo do Arduino

#include <Servo.h>

Servo myservo;

int potpin = 0;
int val;

void setup()
{
  myservo.attach(9);
}

void loop()
{
 val = analogRead(potpin);
 val = map(val, 0, 1023, 0, 179);
 myservo.write(val);
 delay(15);
}
```

B.2 TRECHO DE CÓDIGO COMENTADO – EXEMPLO CÂMERA

```
//Código baseado na biblioteca de rastreamento de cores

#include <CMUcam4.h>
#include <CMUcom4.h>

#define RED_MIN 230
#define RED_MAX 255
#define GREEN_MIN 230
#define GREEN_MAX 255
#define BLUE_MIN 230
#define BLUE_MAX 255
#define LED_BLINK 5
#define WAIT_TIME 5000

CMUcam4 cam(CMUCOM4_SERIAL);

void setup()
{
 cam.begin();



 cam.LEDOn(LED_BLINK);
 delay(WAIT_TIME);


 cam.autoGainControl(false);
 cam.autoWhiteBalance(false);

 cam.LEDOn(CMUCAM4_LED_ON);
}

void loop()
{
 CMUcam4_tracking_data_t data;

 cam.trackColor(RED_MIN, RED_MAX, GREEN_MIN, GREEN_MAX, BLUE_MIN,
BLUE_MAX);

 for(;;)
 {
  cam.getTypeTDataPacket(&data);
/*Adiquirindo um pacote de dados – informações sobre o centro de massa*/


 }

}
```

B.3 TRECHO DE CÓDIGO COMENTADO – ALGORITMO FINAL

```
//Código baseado na biblioteca de rastreamento de imagens da CMUcam4

#include <CMUcam4.h>
#include <CMUcom4.h>
#include <math.h>
#include <Servo.h>

#define RED_MIN 0
#define RED_MAX 70
#define GREEN_MIN 0
#define GREEN_MAX 70
#define BLUE_MIN 0
#define BLUE_MAX 70

#define LED_BLINK 5 // 5 Hz
#define WAIT_TIME 5000 // 5 seconds

#define NOISE_FILTER 2
#define ACCEL 3.75
#define PIXELS_THRESHOLD 1
#define CONFIDENCE_THRESHOLD 50

#define SERVOX_PIN 9
#define SERVOY_PIN 8

CMUcam4 cam(CMUCOM4_SERIAL);

Servo servoX;
int oldServoX;
Servo servoY;
int oldServoY;
int posAntX;
int posAntY;
```

```
int Velx;
int Vely;
int VelyAnt;
int count;
int amostras;
double dx;
boolean Foi;

void setup()
{

 delay(500);

 delay(2000);
 cam.begin();


 cam.LEDOn(LED_BLINK);
 delay(WAIT_TIME);

 count = 0;
 cam.autoGainControl(false);
 cam.autoWhiteBalance(false);
 posAntX = 0;
 posAntY = 0;
 Velx = 0;
 Vely = 0;
 amostras = 0;
 VelyAnt = 0;
 cam.noiseFilter(3);
 oldServoX = 0;
 oldServoY = 0;
  servoY.attach(SERVOY_PIN);
 servoX.attach(SERVOX_PIN);
```

```
  //Opção para Calibrar os limites da tela
  //Calibrate();

  }

  void Calibrate()
  {

   delay(1000);
   servoX.write(54);
   servoY.write(28);
   delay(1000);

   servoX.write(54);
   servoY.write(65);
   delay(1000);

   servoX.write(114);
   servoY.write(65);
   delay(1000);

   servoX.write(114);
   servoY.write(28);
   delay(1000);

   servoX.write(84);
   servoY.write(40);
}

void loop()
{
 CMUcam4_tracking_data_t data;
```

```
 cam.trackColor(RED_MIN, RED_MAX, GREEN_MIN, GREEN_MAX, BLUE_MIN,
BLUE_MAX);


 for(;;)
 {
cam.getTypeTDataPacket(&data); // Get a tracking packet.

    Velx = data.mx - posAntX;
    Vely = data.my - posAntY;

    posAntX = data.mx;
    posAntY = data.my;



  if((data.pixels >= PIXELS_THRESHOLD) &&
    (data.confidence >= CONFIDENCE_THRESHOLD))
 {
     servoY.write(40);

   if(VelyAnt <= 0 && Vely >= 0 && count >= 2)
   {
     //Início dos cálculos

     double t = sqrt((97 - posAntY)*3/ACCEL);
     dx = posAntX + Velx*t;

     count = 0;


     int posFinalX = dx;
     int servoM = round(posFinalX*0.377 + 56);
```

```
        if(servoM > 56 && servoM <= 116){
         servoX.write(servoM);
        }
         delay(3000);
         servoX.write(84);

    }
    if(servoY.read() != 30)
     servoY.write(30);

    //delay(300);
    if(Vely != 0)
     count++;
    else
     count = 0;


   }
   else
   {
    // Inicializando variáveis
    amostras = 0;
    count = 0;
    Velx = 0;
    Vely = 0;
    VelyAnt = 0;

   }
   VelyAnt = Vely;
 }

}
```