UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Desenvolvimento de um sistema de comando para uma máquina *pick and place* baseado em visão computacional

Autor: Marcos Verdini Rosa

Orientador: Prof. Dr. Evandro L. L. Rodrigues

São Carlos

2014

MARCOS VERDINI ROSA

Desenvolvimento de um sistema de comando para uma máquina *pick and place* baseado em visão computacional

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

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Dr. Evandro L. L. Rodrigues

São Carlos
2014

Verdini Rosa, Marcos
V484d Desenvolvimento de um sistema de comando para uma máquina pick and place baseado em visão computacional / Marcos Verdini Rosa; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2014.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Automação. 2. Pick and Place. 3. Visão Computacional. I. Título.

FOLHA DE APROVAÇÃO

Nome: Marcos Verdini Rosa

Título: "Desenvolvimento de um sistema de comando para uma máquina *pick and place* baseado em visão computacional"

Trabalho de Conclusão de Curso defendido e aprovado em 20 / 11 / 2014,

com NOTA 9,0 (NOVE, ZERO), **pela Comissão Julgadora:**

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Associado Adilson Gonzaga - (SEL/EESC/USP)

Mestre Raissa Tavares Vieira - (Doutoranda - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Dedicatória

Mais do que este trabalho, dedico todo meu empenho diário, desde a formação básica até o título de engenheiro, a minha família, em especial à minha mãe Francisca e aos meus irmãos Marcio e Matheus. Através do cuidado, do apoio, do amor, do carinho, das críticas e dos conselhos que vocês me proporcionam, sempre tenho forças para traçar e atingir minhas metas e objetivos.

Marcos Verdini Rosa.

Agradecimentos

Agradeço meus familiares cuja educação, críticas e conversas francas resultaram na construção consciente de meu caráter e mentalidade como pessoa, e em especial à minha mãe, Francisca, por nunca ter economizado esforços em proporcionar uma forte base de ensino, educação e confiança, possibilitando-me ingressar numa universidade de qualidade.

Agradeço aos meus amigos Enzo Bertini Vieira, João Antônio Martins Adorno e em especial ao Gustavo Lahr e ao Itamar Santini que por toda ajuda me propiciaram a capacidade necessária para tornar possível a execução do projeto.

Sou muito grato aos meus professores, especialmente ao Evandro Rodrigues e ao Adilson Gonzaga que durante meus anos na USP me ensinaram e aconselharam da melhor forma possível.

Marcos Verdini Rosa

"Education is our passport to the future, for tomorrow belongs to the people who prepare for it today."

Malcolm X

Resumo

O presente trabalho visa uma abordagem para um sistema automático de seleção de pequenos objetos em formato retangular. O sistema trata da aplicação de técnicas de visão computacional para o reconhecimento e localização de objetos com o uso de uma câmera, assim como o desenvolvimento de um sistema de controle de movimentos para uma máquina do tipo *Pick and Place*. Estas máquinas são usadas para capturar e posicionar componentes utilizando alta velocidade e precisão. Foram feitos testes avaliando o desempenho do sistema aplicados à um protótipo que comprovaram a robustez do sistema em termos de velocidade e precisão.

Palavras-Chave: Visão Computacional, Automação, *Pick and Place*

Abstract

This paper presents a solution for a process of automatic selection of small objects in a rectangular shape. The system uses computer vision techniques for recognition and localization of objects using a camera as well as the development of a motion control system for a pick and place machine. These kind of machines are used for capturing and positioning of parts with high speed and accuracy. To avaluate the performance of the system tests were performed using a propotype to prove robustness with speed and accuracy.

Keywords: Computer Vision, Automation, Pick and Place

Lista de Figuras

Lista de Tabelas

Siglas

MVC	*Model-View-Controller* - Modelo-Visão-Controlador
POO	Programação Orientada a Objetos
UI	*User Interface* - Interface do Usuário
UML	*Unified Modeling Language* - Linguagem de Modelagem Unificada

Sumário

Capítulo 1

Introdução

A automação industrial pode ser considerada como sendo a aplicação de técnicas e ferramentas necessárias em um determinado processo industrial, substituindo o gasto de bio-energia humana como esforço muscular e mental por elementos eletromecânicos computáveis. Com o objetivo de aumentar a sua eficiência, maximizar a produção reduzindo insumos como matérias primas, energia e garantindo melhores condições de segurança [1].

Não é tão fácil apontar o advento da Automação, no entanto, etimologicamente falando, para que haja automação industrial é, antes de tudo, preciso que haja indústria, e ainda processos automáticos autocontroláveis. Portanto, pode-se marcar como início da Automação Industrial o século XVIII, com a criação inglesa da máquina a vapor, aumentando a produção de artigos manufaturados, e estas foram as décadas da Revolução Industrial. No século seguinte a indústria cresceu e tomou forma, novas fontes de energia e a substituição do ferro pelo aço impulsionaram o desenvolvimento das indústrias na Europa e EUA. Neste contexto, nos anos que seguiram, foram criados dispositivos mecânicos chamados relés, que em breve tomariam as fabricas. A todos esses acontecimentos, e a outros que seguiram, foi dado o título de II Revolução Industrial.

No início do século XX, embora o conceito de indústria já estivesse bastante estabelecido, os ambientes fabris ainda não desfrutavam de processos de automação ainda muito rudimentares. Os mesmos pensamentos que fizeram com que surgisse a Revolução Industrial: aumento de produtividade, de lucro, de qualidade, etc.; surgiram nos industriais daquela época, e novos conceitos de produção em escala começaram a serem esboçados. Em 1909, Henry Ford teve a grande ideia que mudou o pensamento da indústria contemporânea, propagando-se até os dias de hoje. Henry Ford (1863-1947), da General Motors, idealizou algo que ele chamou de Linha de Montagem, e talvez esse seja o real gatilho para o grande desen-

volvimento industrial e ainda esta é uma boa marca de início pré-existencial da Automação industrial. A indústria da época foi revolucionada com a aplicação da idéia de Henry, novos conceitos surgiram na indústria: Produção em massa, pontos de montagem, estoques intermediários, etc. Em meados daquele século a GM já produzia automóveis em larga escala, e nos anos que seguiram a morte de Henry, a GM já possuía máquinas automatizadas por relés. No entanto a programação das máquinas era extremamente complexa, com a instalação de painéis e cabines de controle com centenas destes dispositivos mecânicos, o que exigia grande interconectividade e muita energia, isso sem mencionar outros problemas estruturais como cabeamento e vida útil dos relés [1].

Em 1968, a empresa BedFord Association, em BedFord - USA, foi contratada para desenvolver um dispositivo eletrônico que substituísse os relés. O MODICON (*Modular Digital Controller*) foi o primeiro Controlador Lógico Programável inventado, substituindo uma grande quantidade de componentes e tornando o sistema muito mais flexível, econômico e eficiente [1].

A Visão Computacional é uma área, relativamente recente, da ciência que se dedica a desenvolver teorias e métodos voltados à extração automática de informações úteis contidas em imagens, e pode ser usada para interpretar uma cena real no controle de máquinas e processos, ou seja, prover a uma máquina as capacidades de percepção do sistema visual humano em relação ao ambiente. A utilização de câmeras é uma prática cada vez mais comum na área de automação industrial, principalmente onde é preciso automatizar um processo em que os objetos de entrada não tem posição predeterminada.

Uma máquina *Pick and Place* constitui uma máquina robótica utilizada para transportar objetos de uma posição para outra específica com alta velocidade e precisão. Elas são usadas em diversas áreas para propósitos industriais (como na Figura 1.1), médicos, automotivos, militares e em telecomunicações, em etapas de produção que envolvem sistemas de alimentação e separação. Esses manipuladores devem ser suficientemente flexíveis para reagir rapidamente às mudanças na produção, sem que seja necessário substituir os componentes básicos.

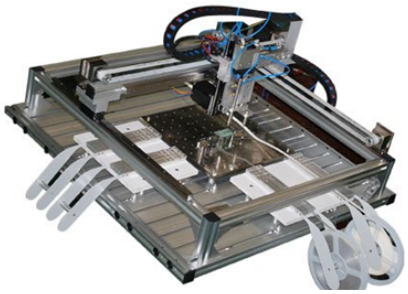

Figura 1.1: Exemplo de máquina usada na montagem de circuitos impressos. (Fonte da imagem: *website Alibaba*)

1.1 Objetivos

O objetivo deste trabalho consiste no planejamento, desenvolvimento de um sistema de comando para uma máquina *pick and place* e na avaliação do desempenho para rastreamento de objetos em tempo real usando processamento de imagens em um ambiente controlado.

1.2 Justificativa

A aplicação de técnicas de visão computacional em problemas de automação industrial permitem obter resultados que até então não poderiam ser alcançados ou não seriam viáveis sem uma ferramenta capaz de substituir o sentido visual humano e a capacidade de julgamento a partir da informação adquirida por este meio. Alternativamente o conhecimento exigido para cumprir os objetivos propostos é base para diversas outras aplicações e finalidades e, desta forma, uma vez cumpridos podem servir de base para outros trabalhos envolvendo aplicações em robótica, controle e automação.

1.3 Organização do trabalho

A organização deste trabalho apresenta o embasamento teórico no Capítulo 3 para o desenvolvimento do projeto, a caracterização dos componentes utilizados, tanto eletrônicos quanto mecânicos, os métodos usados para construção e teste do protótipo no Capítulo 4 e, por fim, a performance do equipamento analisando-se os testes realizados no Capítulo 5.

Capítulo 2

Especificação do Projeto

Este capítulo visa esclarecer e descrever o funcionamento do sistema envolvido, transmitindo assim o escopo do projeto. Todo o embasamento teórico, descrição dos materiais utilizados e os métodos serão tema dos capítulos seguintes.

2.1 Casos de uso

Como já descrito anteriormente, o projeto consiste no desenvolvimento de um sistema de comandos para uma máquina do tipo *pick and place*. Esta máquina tem como objetivo varrer uma área definida por seu alcance de movimentos, procurar e identificar objetos com geometria retangular utilizando algoritmos de processamento de imagens após a aquisição com uma câmera acoplada.

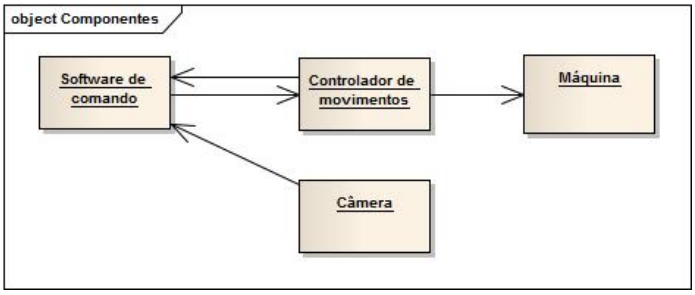

Figura 2.1: Fluxo de comunicação entre os principais componentes.

O sistema baseia-se em um aplicativo que opera a máquina por meio de um dispositivo de controle de movimentos, simplificando a lógica de controle dos motores com comandos simples. A câmera envia informações ao software referentes a posição atual e

O funcionamento do trabalho pode ser visto na figura 2.2 em que se apresenta o diagrama

de casos de uso do sistema. Este diagrama descreve um cenário mostrando a interação entre o usuário e o sistema, ilustrando suas principais funcionalidades. Mais detalhes a respeito destas funções são descritos na seção a seguir.

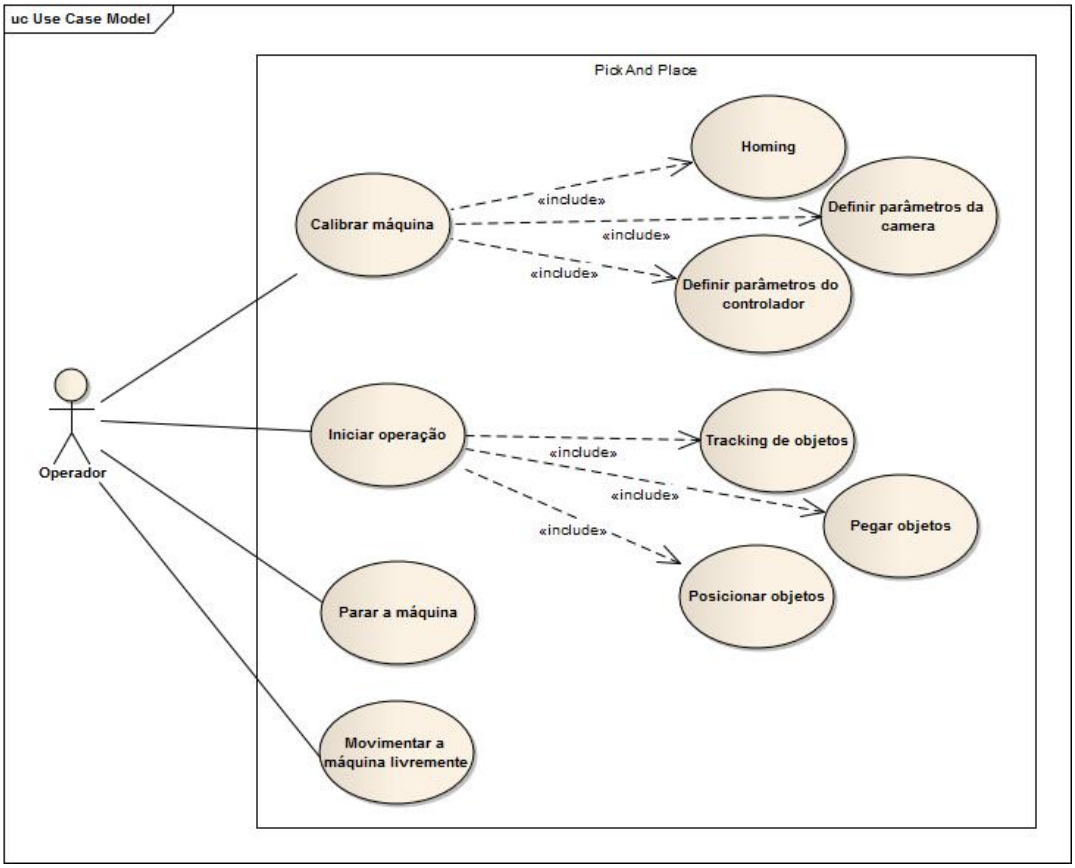

Figura 2.2: Diagrama de casos de uso do projeto.

2.2 *User stories*

O método usado no *SCRUM* (método para desenvolvimento ágil de *software* descrito no próximo capítulo) para definir e organizar os requisitos do sistema são as *User Stories* que proporcionam descrições simples, curtas e claras de uma funcionalidade do projeto descrita segundo o ponto de vista do usuário.

O projeto do aplicativo conta com as *User Stories* descritas na tabela 2.1.

Estas funcionalidades são utilizadas para guiar o desenvolvimento do sistema de tal forma que cada nova versão do aplicativo incluiu mais *User Stories* que a versão anterior.

Como um	**eu quero...**	**para...**
operador	conectar o controlador	iniciar a comunicação com o controlador
operador	desconectar o controlador	encerrar o uso do controlador
operador	conectar a câmera	iniciar a comunicação com a câmera
operador	desconectar a câmera	encerrar o uso da câmera
operador	carregar as configurações do aplicativo	não precisar definí-las novamente
operador	salvar as configurações do aplicativo	usá-las na próxima sessão
operador	fazer operação de *homing*	definir a posição de origem da máquina
operador	alterar um parâmetro da câmera	melhorar as condições de aquisição e processamento de imagens
operador	alterar um parâmetro no controlador	melhorar a execução de movimentos da máquina
operador	movimentar a máquina	testar sua funcionalidade
operador	interromper a qualquer momento a máquina	evitar problemas durante seu funcionamento
operador	testar a precisão da máquina	conhecer
operador	iniciar a operação da máquina	começar a rotina da máquina
operador	encerrar a operação da máquina	terminar a rotina da máquina

Tabela 2.1: Tabela de *User Stories* do sistema.

Capítulo 3

Embasamento Teórico

O presente trabalho possui grande variedade de disciplinas, abrangendo diversas áreas da engenharia e de computação. Assim, são apresentados os conteúdos teóricos que formaram o embasamento para o desenvolvimento do trabalho. Como alguns dos conceitos possuem um conteúdo bastante extenso e requerer um conhecimento mais aprofundado por parte do leitor, é possível entender um pouco mais lendo-se os anexos existentes no fim da obra para aumentar a organização, compreensão e fluidez da leitura.

3.1 Engenharia de Software

A Engenharia de software é uma área do conhecimento da informática voltada para a especificação, desenvolvimento e manutenção de sistemas de software aplicando tecnologias e práticas de ciência da computação, gerência de projetos e outras disciplinas, objetivando organização, produtividade e qualidade. Para se atingir todos os objetivos da engenharia de software utilizam-se os modelos abstratos e precisos em que se pode projetar e implementar sistemas de software garantindo a qualidade do programa gerado [2].

3.1.1 Desenvolvimento ágil de software com Scrum

O Scrum é um método de desenvolvimento que proporciona uma estrutura de gerenciamento de projetos. Este método tem sido usado com muito sucesso em diversos projetos de desenvolvimento de software para diversas finalidades e em organizações de todos os tipos e tamanhos. Embora mais aplicado em softwares, pode ser usado para resolver outros tipos de problemas. O processo Scrum é adequado para projetos com mudanças rápidas ou exigências emergenciais. O desenvolvimento de software progride em uma série de iterações chamadas

sprints, que duram de uma a quatro semanas e geram uma nova versão mais funcional do sistema. Cada *sprint* começa com uma breve reunião de planejamento e é concluída com uma revisão do trabalho. As funcionalidades são dividas em *User Stories*, definidas como sendo sentenças que explicam as atividades que um usuário realiza ou precisa realizar ao utilizar o sistema. Desta forma, a implementação do software é divida em pequenas funções que facilitam a análise de requisitos do sistema. Estes são os princípios de gerenciamento de projetos com Scrum.

3.1.2 Programação Orientada a Objetos

Com o surgimento da Engenharia de *Software* várias técnicas de programação foram desenvolvidas visando aumentar a qualidade e produtividade dos *softwares*. Desta maneira, verificou-se que a reutilização de códigos era o segredo para promover a melhoria buscada pela Engenharia de *Software*. Desta maneira, foi verificado que as técnicas de programação estruturada não eram suficientes para alcançar satisfatoriamente o nível de melhoria buscado, assim sendo, na década de 60, surgiu a teoria de orientação a objetos [3].

De uma forma abstrata, o intuito básico da POO (Programação Orientada a Objetos) é o de aproximar o mundo real do virtual por meio do uso de Objetos. Assim sendo, o programador deverá moldar tais objetos e definir como deverá ser feita a interação entre os mesmos, de forma que atinjam funcionalidades requeridas para um correto funcionamento do *software*.

Em suma, a ideia principal de programação orientada a objetos consiste na possibilidade de aglomerar, em uma única estrutura de dados (classe), seus atributos (características) e seus métodos (funções). Desta forma, uma variável do tipo desta classe é chamada objeto, e este objeto conterá seus próprios dados e funções. As funções que um objeto pode executar são chamadas de métodos. Tais métodos, de modo geral, são o único meio de acesso aos seus campos de dados, chamados de variáveis de instância.

No anexo I, extraído de [4], serão explicadas as definições específicas da POO, bem como as mesmas serão caracterizadas de maneira mais profunda e clara.

3.1.3 UML

A UML, *Unified Modeling Language*, é um padrão internacional, que permite a especificação, construção e documentação de *softwares* que utilizem programação orientada a Objetos, desta maneira, a UML independe da linguagem de programação utilizada.

No anexo II deste trabalho, extraído de [4], serão descritos e exibidos os conceitos básí-

cos, sintaxes e significados de alguns diagramas para que, futuramente, os mesmos possam ser utilizados para facilitar o entendimento e documentação, principalmente das classes, do *software* final.

Capítulo 4

Materiais e Métodos

O foco do trabalho é um sistema de comando que permite controlar um máquina para indexar objetos espalhados em uma área a partir da captura de imagens digitais. Este capítulo divide o trabalho começando pelo protótipo capaz de testar o sistema e, dessa forma, implementar de fato o projeto para poder executá-lo em uma aplicação com sucesso. Após apresentar a parte física do projeto apresenta-se o software de comando, detalhando seus aspectos mais importantes. A seguir serão descritos as partes do sistema que foram desenvolvidas.

4.1 Protótipo

No início do projeto foi desenvolvido um protótipo de manipulador *pick and place*, com o auxílio do autor deste trabalho, principalmente na parte de eletrônica e controle. O projeto mecânico foi desenvolvido por terceiros, e o desenho da máquina pode ser visto na Figura 4.1. A máquina conta com quatro eixos de liberdade para realizar movimentação dentro de um determinado espaço, sendo os três eixos espaciais X, Y e Z, além de um quarto eixo (U) com movimentos rotatórios que permite uma pipeta capturar objetos, girar e posicioná-los com um ângulo específico.

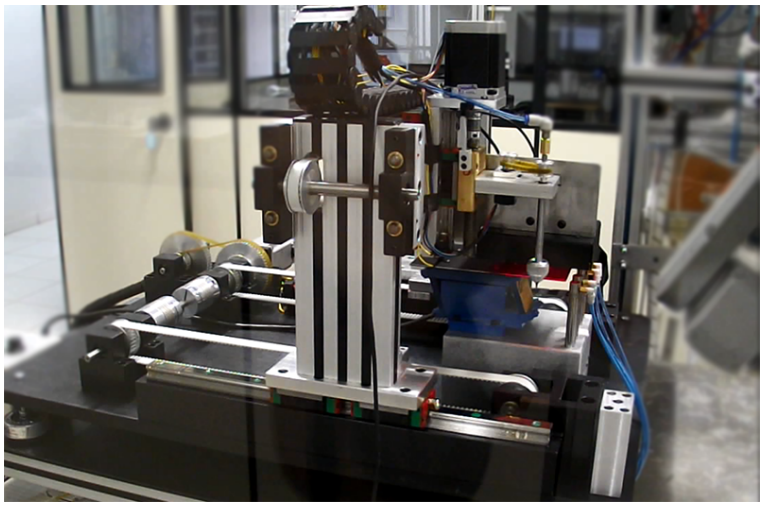

Figura 4.1: Protótipo da *pick and place*.

Os motores foram dimensionados para suportar as cargas necessárias (o eixo X movimenta os outros eixos), e os eixos X e Y contam com um sistema de redução por polias para diminuir a força necessária para movimentação. Os motores dos eixos Z e U são motores de passo , dimensionados com a potência, torque, precisão e repetitividade adequadas para o funcionamento do processo. Na Figura 4.2 pode-se ver um desenho simples de um motor de passo como utilizado no projeto.

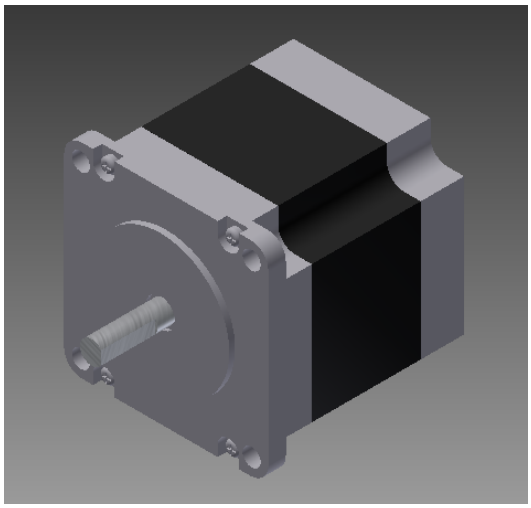

Figura 4.2: Desenho de um motor de passo.

Os motores dos eixos X e Y são do tipo servo com *encoder* (ilustrado na Figura 4.3). Os *encoders* são equipamentos eletromecânicos, utilizados para conversão de movimentos rotativos ou deslocamentos lineares em impulsos elétricos de onda quadrada, que geram uma quantidade exata de impulsos por volta em uma distribuição perfeita dos pulsos ao longo dos

360 graus do giro do eixo. Com eles é possível encontrar a posição angular dos eixos dos motores, necessário para o controle dos mesmos.

Os eixos X e Y ainda contam com um sistema de redução para suportar as cargas necessárias. Assim como os motores descritos anteriormente, estes foram dimensionados também para atender as necessidades de potência, torque, precisão e repetitividade necessários ao produto final.

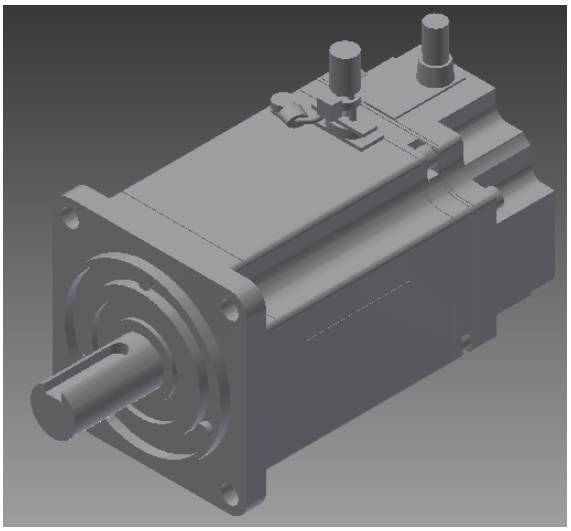

Figura 4.3: Desenho de um servo motor com *encoder*.

A máquina possui também sensores de Fim de Curso nos eixos X, Y e Z para realizar a operação de *homing*.

Quando a *pick and place* é ligada, os eixos podem estar em quaisquer posições, pois elas são desconhecidas pelo software de controle. O processo que associa a posição angular dos motores com o software é chamada de *homing*. Quando os eixos estão nas posições detectadas pelos sensores de Fim de Curso, o sistema de controle sabe que a máquina está na sua posição de origem.

O protótipo ainda possui um sistema de segurança que utiliza mais sensores de fim de curso nos eixos X, Y e Z (dois por eixo, um no final de cada guia) para desligar a máquina com uma contatora em casos de emergência, por exemplo, na falha de alguma operação o sistema pode perder a posição sua posição e forçar os motores no fim das guias, e por consequência, danificar e inutilizar os mesmos.

Para que seja possível efetuar a operação de transportar objetos previamente identificados é preciso utilizar um método para segurar tais objetos. A opção escolhida para este trabalho foi um sistema de geração de vácuo na pipeta e nos locais de entrega das pequenas peças

também para garantir que eles não saiam de seu destino por acaso.

Por fim, para que estes objetos sejam utilizados em um próximo passo num processo automático, a máquina conta com uma interface capaz de instruir qual o próximo local com a presença de um objeto.

4.2 Software de comando

O planejamento e o desenvolvimento do sistema seguiram as bases do Scrum para se alcançar um bom desempenho nesta fase do trabalho. O programa em sua versão final atende aos requisitos necessários listados nas *User Stories* descritas no capítulo 2. As *sprints* utilizadas para a implementação do aplicativo não serão focadas e sim sua versão final.

4.2.1 Arquitetura do software

A arquitetura do software pode ser vista na Figura 4.4. Este tipo de arquitetura é, em geral, mais utilizada para desenvolvimento web do que em aplicativos *desktop* (como é o caso deste sistema), mas foi adaptado para obter as vantagens deste tipo de arquitetura: design claro, modularidade eficiente e a facilidade de manutenção e expansão do software.

A figura 4.5 ilustra as principais classes utilizadas pela parte dos controladores responsáveis por manipular os dados e atualizar as *Views* para o usuário.

A figura 4.6 representa as classes mais importantes que modelam objetos que guardam dados sobre o estado do programa.

A figura 4.7 representa as *Views*, componentes responsáveis por renderizar a UI (Interface com o usuário). Em aplicações MVC (*Model-View-Controller*) comuns, as *Views* apenas exibem informações e o controle manipula e responde pelas interações com o usuário. No caso desta aplicação, as *Views* possuem uma lógica básica de manipulação das interações que entrega para os controladores apenas a parte de processamento de dados.

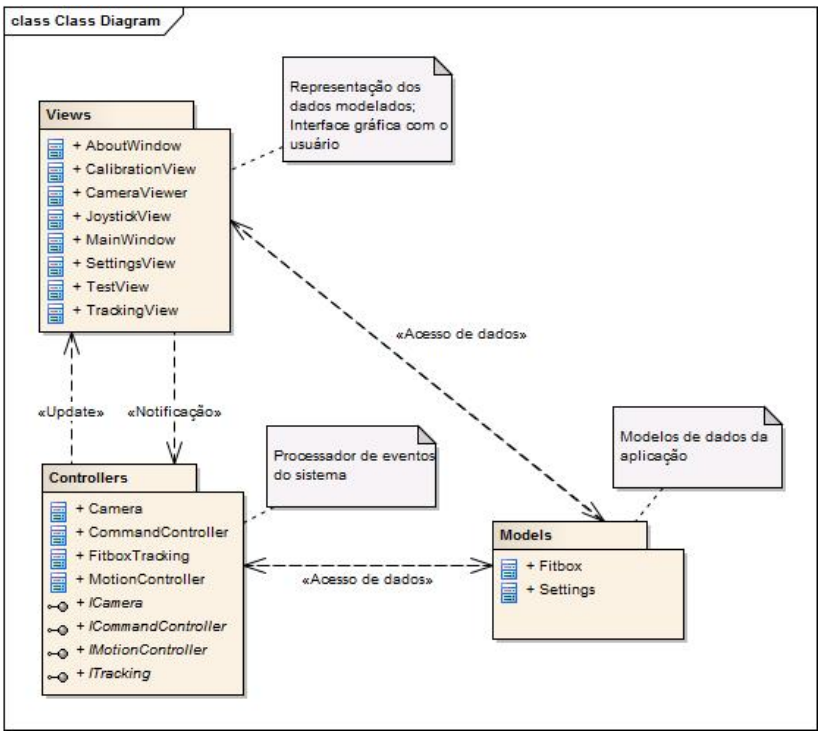

Figura 4.4: Modelo MVC do aplicativo.

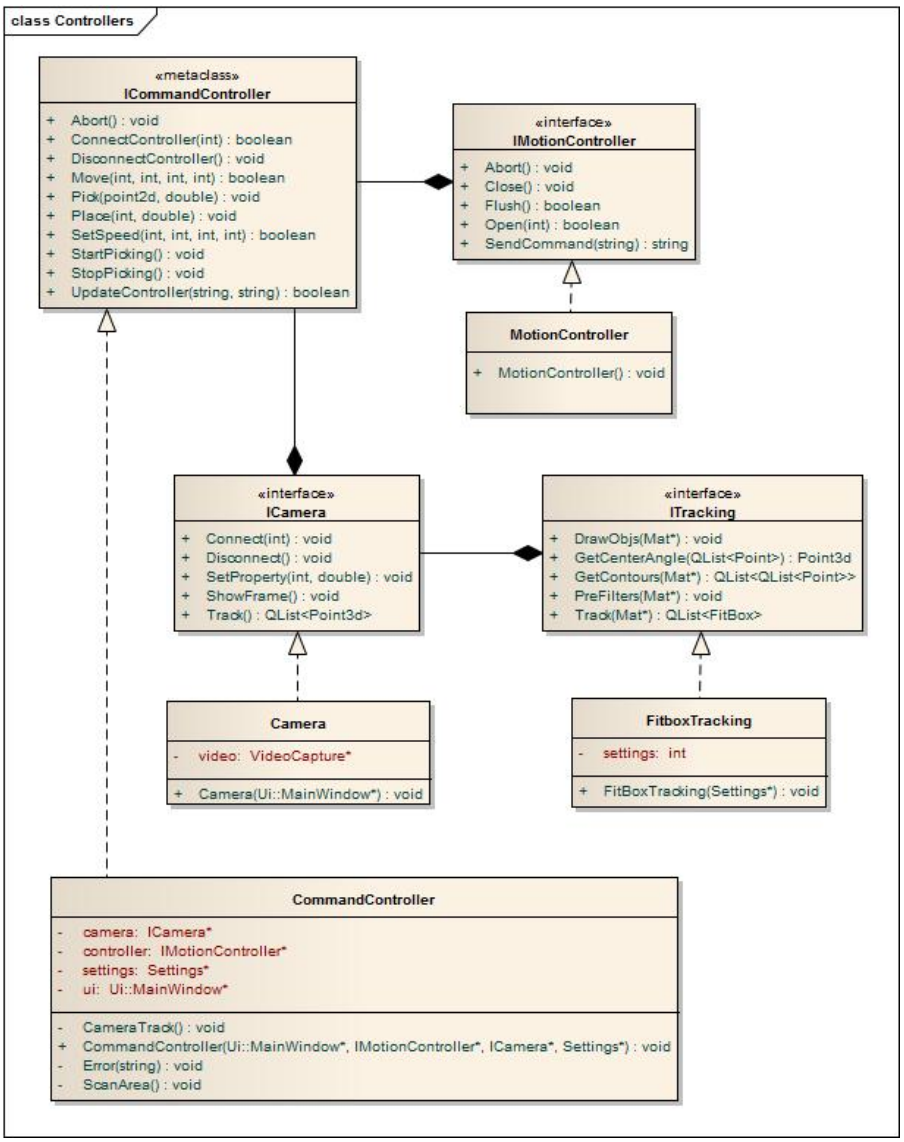

Figura 4.5: Diagrama de classes dos controladores.

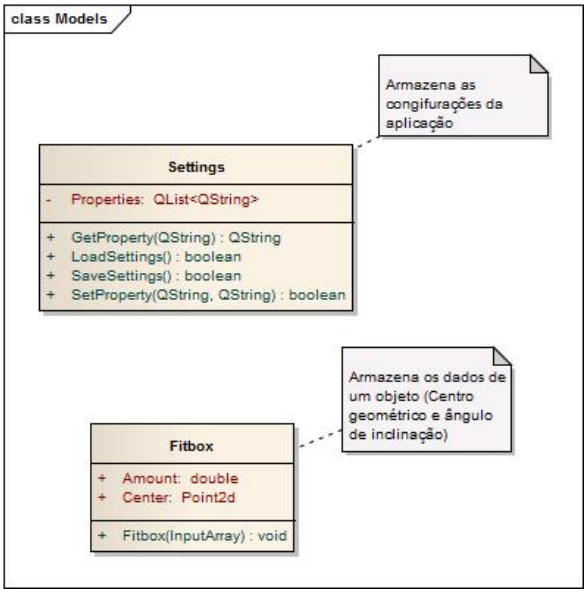

Figura 4.6: Diagrama de classes dos modelos.

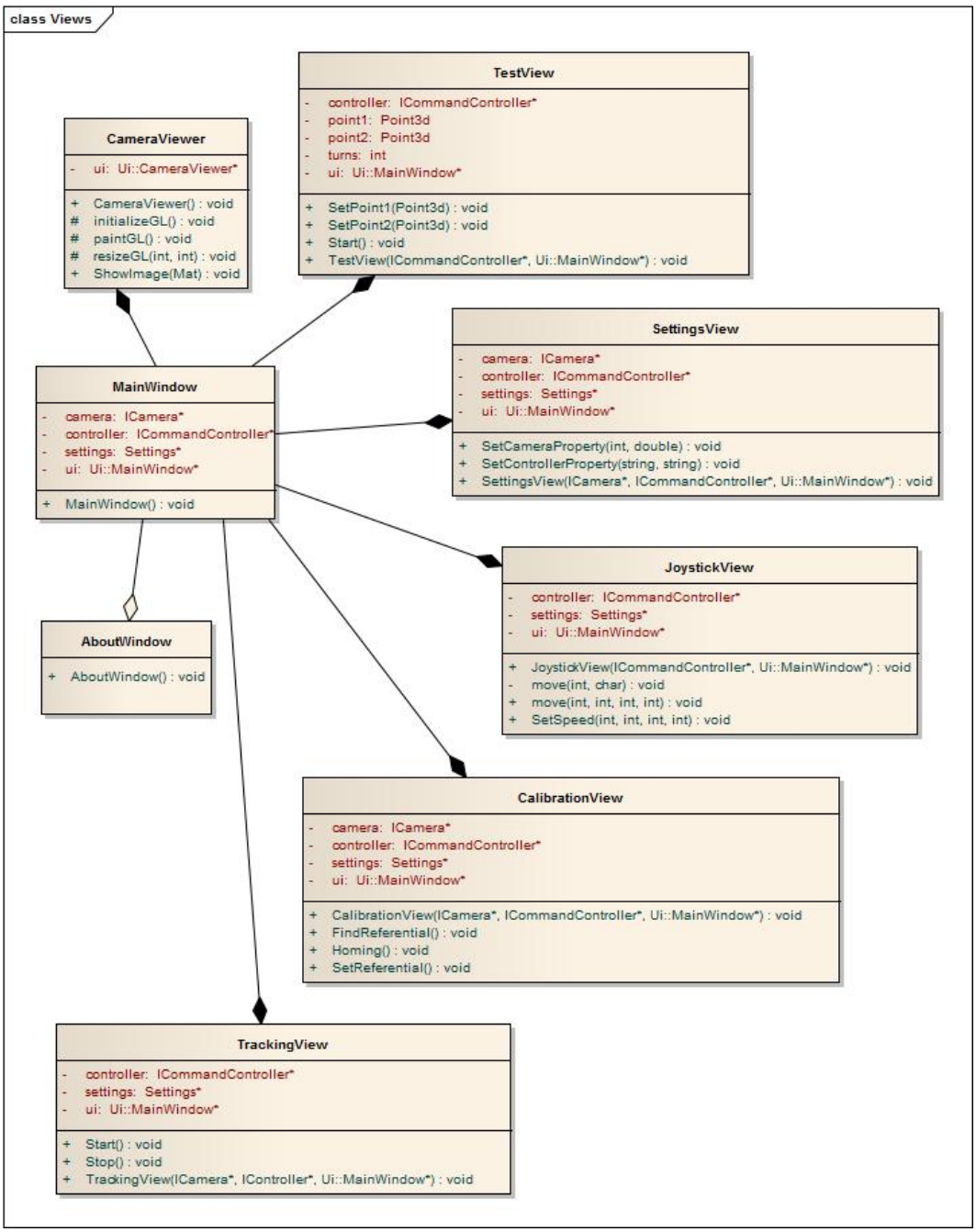

Figura 4.7: Diagrama de classes dos *Views*.

4.2.2 Calibração da câmera

Usando o modelo matemático da câmera *pinhole*, a câmera física é tratada como tendo um ponto no espaço onde os raios de luz passam do espaço tridimensional e são projetados num plano de imagem. Cada câmera tem um conjunto de parâmetros intrínsecos que descrevem a posição deste ponto virtual relativamente à coordenada de origem da imagem. O processo de calibração é usado para encontrar esses parâmetros que podem ser usados para reduzir sua

distorção e até mesmo em reconstrução 3D.

O processo de calibração requer a captura de um número sucessivo de imagens pela câmera usando um padrão similar a um tabuleiro de xadrez com dimensões conhecidas dos quadrados e do padrão inteiro. A rotina de calibração exige um número suficiente de imagens capturadas da referência em diferentes orientações e localizações do campo de vista da câmera. Neste trabalho foram usadas um total de 16 imagens para o processo de calibração.

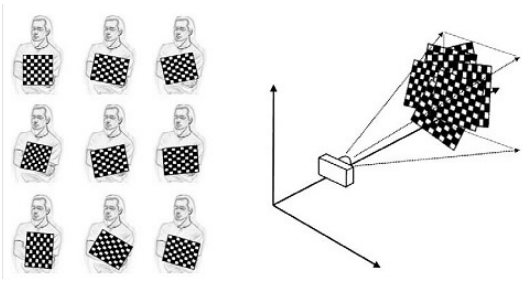

Figura 4.8: Exemplo do método de calibração com um tabuleiro de xadrez. (Fonte da imagem: Livro *Learning OpenCV*, 2008)

O resultado da calibração da câmera com seus parâmetros intrínsecos podem ser vistos na tabela 4.1 organizados de forma matricial. Os parâmetros adicionais resultantes do processo de calibração são os coeficientes de distorção que caracterizam distorção radial e tangencial das lentes da câmera. As imagens produzidas pela câmera estudada apresentaram distorção baixíssima e, portanto, os parâmetros de distorção não foram usados e o aplicativo não trata destes parâmetros.

Tabela 4.1: Os parâmetros intrínsecos da câmera usada no estudo.

Notação de parâmetro			Câmera		
fx	0	cx	920.5	0.0	650.1
0	fy	cy	0	924.5	366.7
0	0	1	0	0	1

Após a calibração da câmera foi possível fazer um processamento de imagens mais confiável, pois sem a distorção das lentes os algoritmos que convertem as posições de *pixel* em unidades reais são simplificados, pois podem ser convertidos utilizando um modelo linear.

4.2.3 Seleção de objetos

Para a implementação do *software* foi utilizado a biblioteca *OpenCV*, desenvolvida por terceiros e aberta para o uso gratuitamente. Como o foco do trabalho não é o desenvolvimento de algoritmos de visão computacional, a utilização desta biblioteca teve grande importância para o rápido desenvolvimento do projeto, uma vez que ela conta com mais de 2500 funções de processamento de imagem prontas para o uso. O algoritmo de seleção de objetos atua basicamente da seguinte forma:

- **Aquisição de imagem** - *frames* obtidos da câmera.

- **Filtro de ruídos** - Filtro passa-baixa gaussiano com máscara 3x3 para eliminar eventuais ruídos que atrapalhem a identificação dos objetos.

- **Binarização da imagem** - Separação das peças do fundo da imagem utilizando um *threshold* com intervalo de 200 aé 255 (pois os objetos possuem cor bem clara).

- **Detecção de contornos** - Utilização do algoritmo de *Canny* pra detecção de bordas dos objetos a partir da imagem binária gerada anteriormente.

- **Cálculo do centro de massa de contornos** - O centro de massa é calculado baseado na área dos contornos identificados anteriormente, considerando-se a densidade constante destas áreas.

- **Cálculo do ângulo dos contornos** - O ângulo das peças é determinado a partir da diferença entre os seus lados (pois trata-se de um objeto no formato retangular).

- **Conversão das coordenadas** - A conversão das coordenadas de *pixel* para unidades reais é feita através da multiplicação por uma constante determinada experimentalmente (pois a câmera foi calibrada e não possui distorções consideráveis).

A Figura 4.9 mostra a última versão do *software* de comando desenvolvido no trabalho.

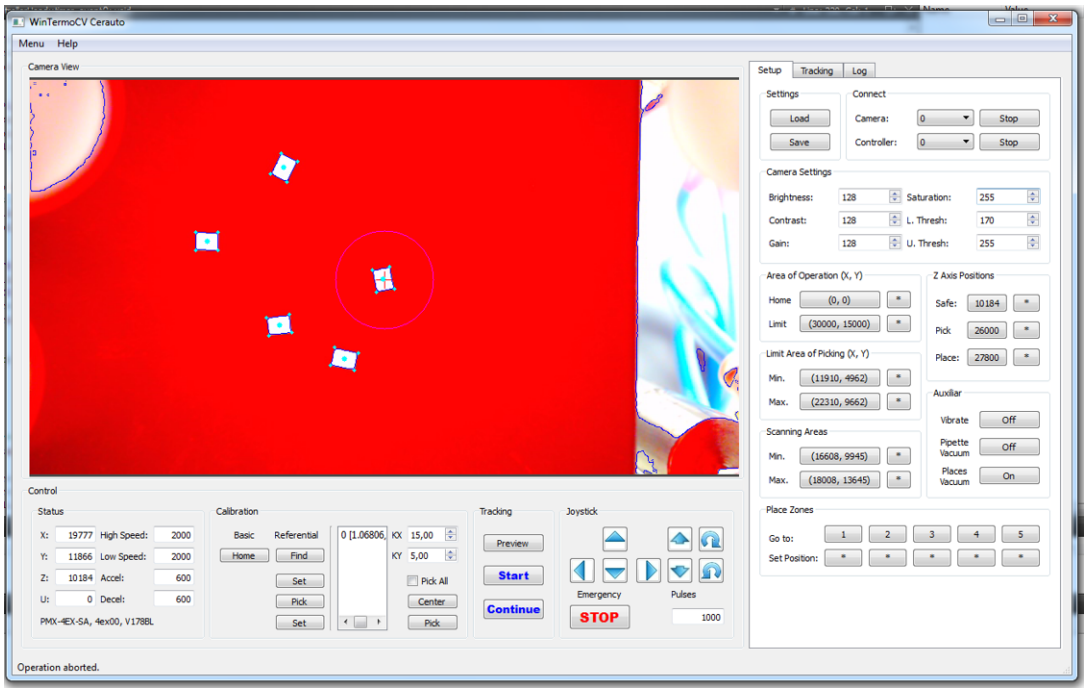

Figura 4.9: *Software* de comando desenvolvido.

Capítulo 5

Resultados

5.1 Resolução de problemas

Para que a implementação da aplicação, assim como os testes fossem bem sucedidos foi preciso lidar com situações muitas vezes imprevistas ou indesejáveis. Obstáculos sempre surgem em qualquer projeto e a maneira como os problemas são resolvidos pode ser um fator determinante no sucesso do mesmo. Portanto, esta seção trata de alguns aspectos importantes levados em consideração no desenvolvimento do trabalho.

Comunicação com o controlador de movimentos

A comunicação com a *pick and place* é um fator muito importante no projeto, pois se ela falhar podem acontecer problemas que comprometem o funcionamento da mesma com movimentos indesejados causando grandes prejuízos como colisões. Para garantir que os comandos não falhem, é preciso que a interface seja bem escolhida e bem programada: o controlador precisa responder a todos os comandos que receber para que o computador saiba o estado da máquina. A parte de comunicação entre o controlador e o computador apresentou alguns problemas a princípio e para contornar estes problemas foram levados em consideração os seguintes itens:

- Configuração do *timeout* da troca de mensagens levando em conta o tempo de processamento dessas mensagens por parte do controlador, que pode ser bastante grande (mais de 3 milisegundos).
- Reenvio de mensagens sem resposta para garantir que o comando seja executado, mas após um número limite (entre três e cinco por exemplo) o software exibe mensagens de erro sobre a comunicação

- O *driver* de comunicação utilizado no computador para enviar as mensagens para o controlador. Este foi um problema enfrentado neste trabalho e a solução foi atualizar o Sistema Operacional. Este tipo de problema é mais complicado pois o programador não tem acesso à esse nível para desenvolvimento.

Visão Computacional

Para um ótimo funcionamento de um sistema artificial com percepção visual (extração de informações de imagens) é preciso ter um ambiente controlado, pois este sistema só deve funcionar para uma determinada variação do ambiente esperado, de tal forma que se a iluminação mudar ou se a distância da câmera para a calha onde ficam as peças mudar (por falta de calibração ou alguma pequena alteração que possa passar despercebido aos olhos humanos) o sistema não trará os resultados esperados. O aspecto mais sujeito a variação neste caso é o da luz ambiente para a aquisição de imagens. A cada mudança de lugar onde a *pick and place* estiver será preciso reajustar os parâmetros da câmera e, em algumas situações, não será possível por falta de iluminação. Ainda há casos em que sua própria geometria pode prejudicá-la com sua sombra. As lâmpadas fluorescentes normalmente utilizadas em residências não são aconselháveis para a Visão Computacional pois elas brilham em uma frequência baixa (60 Hz). A máquina utiliza uma proteção física no formato de uma cúpula para bloquear a luz ambiente e ainda conta com um módulo iluminador com *leds* vermelhos do tipo difuso. A iluminação contínua dos *leds* não atrapalha a câmera e a luz vermelha destaca os objetos, pois estes apresenta grande reflexão do espectro de cor incidente, enquanto que o fundo da calha (da cor preto fosco) e os ruídos ficam menos evidentes. A Figura 5.1 ilustra um dispositivo de iluminação que pode ser usado na prática.

Processamento de imagens

O processamento de imagens é considerado a parte mais importante do projeto. Os principais tratamentos de ruídos por software são obtidos por filtros passa-baixa e o ajuste contraste das imagens aquisitadas. Para que os resultados fossem confiáveis foram utilizadas médias de vários *frames* aquisitados pela câmera (com bons resultados variando de 3 até 10, sendo que no projeto foram usados 5) ao invés de utilizar apenas um *frame*. A definição da região de interesse é feita para o centro das imagens para evitar que outros objetos não sejam selecionados por engano (como pode acontecer fora da área de busca de peças). Um último aspecto importante de se comentar é o *delay*

Figura 5.1: Exemplo de iluminação difusa com *leds*.

introduzido antes da captura das imagens durante a operação da máquina. Este *delay* é introduzido para que as imagens capturadas se estabilizem antes do processamento e não apareçam borradas, prejudicando os resultados. A Figura 5.2 mostra como são identificados as peças no sistema de visão computacional do projeto.

Redundância de informações

Confiabilidade e disponibilidade são características indispensáveis na automação de complexos sistemas de supervisão e controle de máquinas e processos. Maior tolerância a falhas corresponde à maior disponibilidade da planta e maior segurança operacional e ambos os aspectos são importantes. Durante a implementação do projeto procurou-se tomar decisões a partir de mais de um resultado, utilizando-se médias no processamento e a possibilidade de executar um comando em mais de uma tentativa em casos de falha.

5.2 Falhas do sistema de comando

Ao fim deste trabalho foi lançada a versão final do aplicativo de comandos do qual foram extraídos os resultados apresentados neste capítulo. A última versão até o momento se mostrou robusta para a execução de sua rotina de busca e entrega de peças, atingindo assim seu objetivo de automação do processo desejado. No entanto, o software ainda apresentava algumas falhas durante sua execução (listadas nesta seção), que podem ser tratados no futuro para um aperfeiçoamento de sua rotina. Assim como qualquer equipamento, a *pick and place* neces-

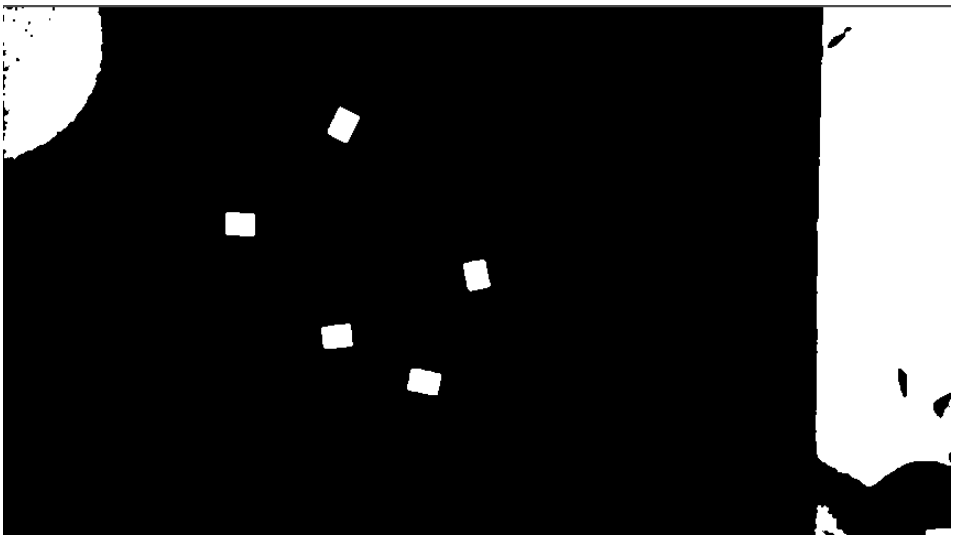

Figura 5.2: Detecção de peças.

sita de uma pessoa para começar seu funcionamento, dar continuidade e tratar problemas que possam surgir enquanto opera e para encerrar sua atividade. A máquina é capaz de notificar o usuário por meio de mensagens ao usuário no software de comando. As principais falhas que ela apresentou para sua avaliação neste projeto foram:

Dificuldade em pegar o objeto mais próximo

O algoritmo de processamento de imagens é preparado para tratar ruídos nas imagens aquisitadas e identificar as peças de forma otimizada. Porém, nem sempre os resultados são semelhantes: em alguns casos, a máquina identifica uma peça como sendo mais próxima do centro da câmera e em seu processo iterativo de alcançar aquela peça ela pode acabar identificando outra em seu lugar, pois ao se movimentar a posição das peças muda em relação à câmera.

Operação próxima aos limites da área de busca

A operação da *pick and place* deve ser segura, a ponto de que ela nunca ultrapasse limites definidos (por meio da calibração inicial dela) para evitar colisões e, assim, danos em seu funcionamento. Uma vez que foram definidos os limites da área de busca de objetos o algoritmo deve identificar e pegar objetos apenas dentro dessa área. Em alguns casos, quando o objeto detectado está muito próximo a essas bordas e o centro da câmera não está próximo ao mesmo o sistema pode calcular como se ele estivesse dentro da área de busca e ao se aproximar recalcula como estando fora da área de busca, por estar mais perto e ter mais precisão na posição.

5.3 Análise de desempenho da *pick and place*

Após o desenvolvimento do projeto ele foi submetido à uma série de testes a fim de fornecer informações sobre sua qualidade em relação ao contexto em que ele deve operar. A norma ISO 9216 fornece um modelo de propósito geral que define seis características de qualidade de software: Funcionalidade, Confiabilidade, Usabilidade, Eficiência, Manutenibilidade e Portabilidade [5]. Apesar de nem todos os atributos serem avaliados com profundidade no sistema de comando deste trabalho, eles servem de modelo para a análise do projeto como um todo não limitando-se ao aplicativo.

Primeiramente, o projeto é funcional, pois satisfaz as necessidades do usuário, assim como definidas nas *User Stories* com o método Scrum. O sistema é também confiável, pois possui robustez para tolerar possíveis falhas durante sua rotina e manter seu funcionamento (as falhas que não são recuperáveis notificam o usuário para dar continuidade no processo). O sistema de comando possui uma interface simples, fácil de usar, o que configura sua usabilidade. Como explicado no capítulo anterior, o desenvolvimento do projeto foi bem modularizado utilizando um padrão adaptado do MVC tornando sua manutenção mais rápida. Por fim, o sistema é portável porque foi planejado para ter o mínimo de dependência do ambiente físico em que se encontra e o aplicativo foi programado visando uma instalação simples em outros computadores por meio da geração de instaladores contendo todos as dependências do projeto (bibliotecas compartilhadas e recursos do programa).

Utilizando uma amostra de 60 objetos, foram cronometrados os tempos de entrega de peças pela *pick and place*. Os resultados se apresentam na Figura 5.3. O tempo mantém uma média de 9 segundos, apenas com duas variações mais evidentes, na peça de número 11 e na de número 50, em que o sistema encontrou maior dificuldade na operação.

A Figura 5.4 ilustra a qualidade do sistema em função de sua velocidade de funcionamento (a velocidade dos motores estão em pulsos por segundo e variam de 1000 até sua maior velocidade, 10000). É possível ver que a máquina apresenta mais erros quando opera a velocidades muito altas - a identificação de objetos se torna mais difícil pois as imagens da câmera tendem a ficar mais borradas. Portanto, sua rotina opera em 6000 passos por minuto

Mais uma vez, a Figura 5.5 comprova a melhor velocidade de operação da *pick and place* em 6000 pulsos por segundo. Em velocidades baixas a visão funciona melhor, porém o tempo de movimentação não compensa o ganho. Em altas velocidade acontece o contrário, a parte de visão requer mais tempo para estabilizar as imagens e obter a posição das peças.

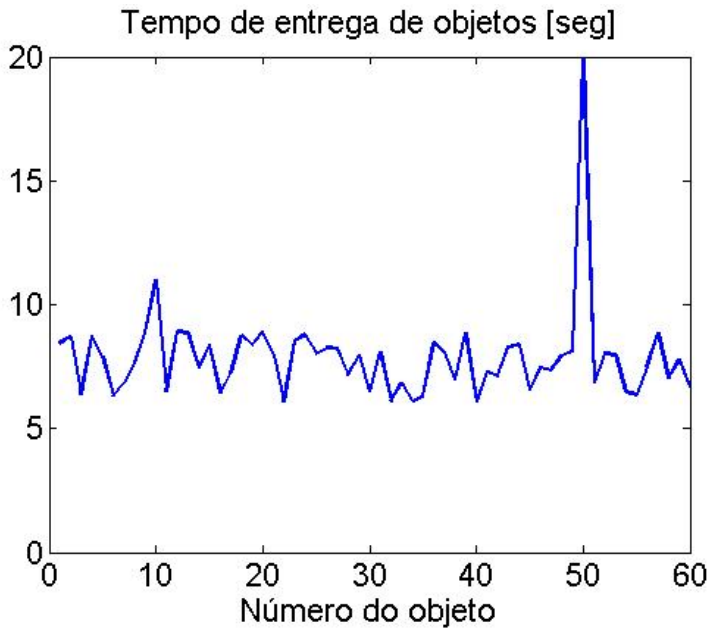

Figura 5.3: Desempenho da *pick and place* para uma amostra de 60 objetos.

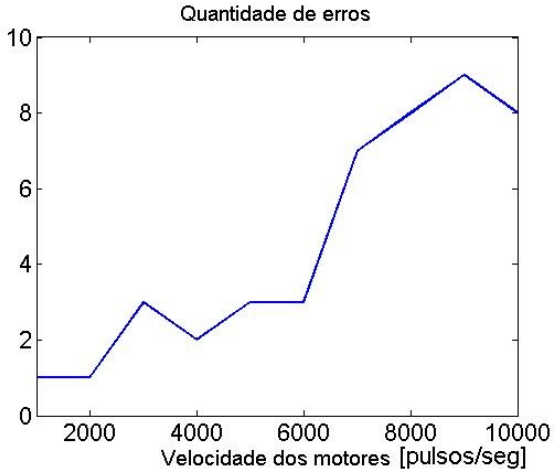

Figura 5.4: Taxa de erros obtidos em relação a velocidade dos motores.

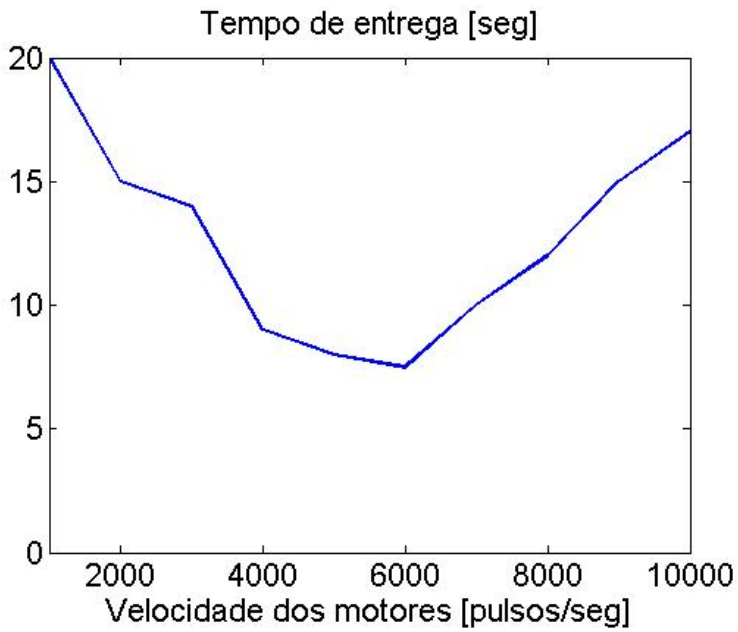

Figura 5.5: Média de tempo de entrega de peças em relação a velocidade dos motores.

A Figura 5.6 representa a produção da máquina durante três dias de teste da máquina utilizando a velocidade mais adequada. O tempo médio de execução dos dias é de, aproximadamente, 8 horas. para o primeiro dia, a média do tempo de entrega dos objetos foi de 9,9 segundos (total de 2900 peças) no primeiro dia, o segundo dia foi de 11,5 (2400 peças) e no terceiro dia foi de 10,66 segundos. Comparativamente, a Figura 5.7 demonstra que o processo manual é menos produtivo, chegando a uma média de apenas 1500 peças por dia, aproximadamente.

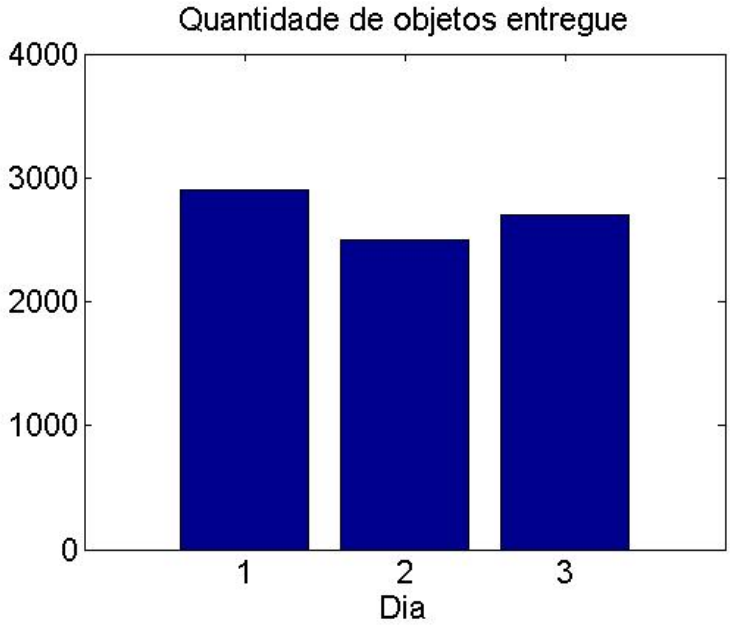

Figura 5.6: Produção da máquina em três dias de teste.

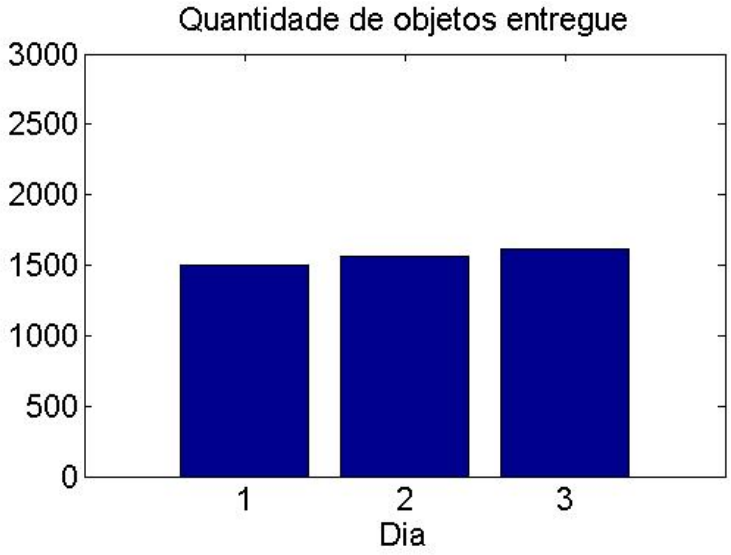

Figura 5.7: Comparativo do processo manual em três dias.

Capítulo 6

Conclusão

O sistema de comando desenvolvido apresentou a qualidade desejada em relação ao esperado em termos de funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

O trabalho expôs algumas falhas encontradas no software até a versão final usada para testes, presentes na divisão de visão computacional. Estas falhas evidenciam a dificuldade em converter medidas de distância de *pixels* em unidades do mundo real e vice-versa. Desta maneira, nota-se que o sistema pode ser mais bem trabalhado e personalizado de acordo com suas aplicações, melhorando sua funcionalidade e eficiência.

Finalmente, o projeto como um todo apresentou resultados positivos e propiciou o estudo de aplicações de diversas áreas envolvidas para o desenvolvimento de um projeto de engenharia, abordando áreas como controle, circuitos elétricos e eletrônicos (com o desenvolvimento do protótipo) e programação, visão computacional e processamento de imagens, e assim, mostrou-se como uma oportunidade ímpar para solidificação dos conhecimentos e vislumbre de novas áreas para estudo.

Referências Bibliográficas

[1] Leonardo Silveira and Weldson Q̇Lima. Um breve histórico conceitual da automação industrial e redes para automação industrial, 2003.

[2] Ian Sommerville. Software engineering, 2009.

[3] A. Sintes. *Sams Teach Yourself Object Oriented Programming in 21 Days*. Sams, 2001.

[4] Enzo Bertini Vieira e Lara Bertini Vieira. Sistema autônomo de vigilância baseado em dados biológicos com registro de dados na nuvem via smartphone, 2014.

[5] Luiz bianchi. Qualidade de produtos de software. `http://bianchi.pro.br/edutec/qualsoft.php`, Acesso em 10 de outubro de 2014.

[6] P. Coad e E. Yourdon. *Análise Baseada em Objetos*. Campus, 1992.

[7] G. Pollice e D. West. *Use a Cabeça! Análise e Projeto Orientado ao Objeto* B. McLaughlin. O'Reilly, 2007.

[8] M. C. Sampaio e E. L. Neto. *Material Sobre UML*. Universidade Federal de Campina Grande Centro de Engenharia Elétrica e Informática. `http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/`, Acesso em: 27 de Outubro de 2014.

[9] Scrum and agile training. `http://www.mountaingoatsoftware.com`, Acesso em 10 de outubro de 2014.

Anexo I

Programação Orientada a Objetos

Aqui apresenta-se uma ideia geral de programação Orientada a Objetos, baseado no trabalho da referência [4]. Como a aplicação deste conhecimento varia entre as linguagens de programação, nenhuma estrutura de códigos ou comandos será introduzida, assim sendo, não será focada em um tipo de linguagem. O intuito é saber a teoria por trás deste paradigma para ser capaz de aplicá-la em qualquer linguagem. Os estudos aqui demonstrados foram baseados nas referencias [6] [7].

I.1 Objetos

A base fundamental de Orientação a Objetos são os objetos, uma vez que esta entidade contém operações para manipular suas informações. Em linguagens de programação estruturada, variáveis são instâncias (exemplares) de uma estrutura. Já em POO, objetos são instâncias de classes. Nos objetos são definidos quais informações (atributos) ele conterá, bem como tais atributos poderão ser manipulados (métodos).

I.2 Atributos

Os atributos (ou variáveis membro) são os campos que armazenam as informações referentes a determinados objetos. O estado de um objeto é o conjunto de valores que seus atributos possuem em um determinado instante. Tomando-se um automóvel como sendo o objeto, como já explicado anteriormente, esse automóvel deverá ter seus atributos dados da seguinte maneira:

Classe: Automóvel

Atributos:

Número de rodas

Tipo de Combustível

Velocidade máxima

Cor

I.3 Métodos

Os métodos são os procedimentos que permitem ao objeto realizar suas ações. Tudo que se espera que um objetos deva fazer precisa ser especificado por um método.

Continuando com o exemplo anterior:

Classe: Automóvel

Atributos:

Número de rodas

Tipo de Combustível

Velocidade máxima

Cor

Métodos:

Acelerar

Frear

Virar para esquerda

Virar para direita

I.4 Classes

Uma classe é um conjunto de objetos com características e comportamentos comuns. Seguindo o exemplo:

Classe: Automóvel

Objeto: Caminhonete

Atributos:

Número de rodas: 4

Tipo de Combustível: Diesel

Velocidade máxima: 180 km/h

Cor: Verde

Métodos:

Acelerar

Frear

Virar para esquerda

Virar para direita

Objeto: Motocicleta

Atributos:

Número de rodas: 2

Tipo de Combustível: Gasolina

Velocidade máxima: 230 km/h

Cor: Azul

Métodos:

Acelerar

Frear

Virar para esquerda

Virar para direita

Note que tanto o objeto Caminhonete, quanto o Motocicleta possuem os mesmos métodos e os mesmos atributos, contudo os valores de seus atributos são diferentes.

I.5 Abstração

A abstração, a "grosso modo", significa que só deve ser representado o que realmente for ser utilizado, deixando de representar métodos ou variáveis membro desnecessárias. Pelo exemplo da classe Automóvel, se o projeto consiste num sistema para descrever suas características básicas, não é interessante conter atributos para "Número de tomadas de 12V"ou "Numero de antenas".

I.6 Encapsulamento

O encapsulamento consiste no princípio de projeto de que cada componente de um programa deve conter todas as informações necessárias para seu completo funcionamento. Na OO, os dados (variáveis membro), e os processos (métodos) estão encapsulados numa mesma

identidade, o objeto.

O encapsulamento é também o princípio pelo qual determinado componente, seja ele um método ou uma variável membro esteja pública ou privada. Se pública, determinado componente poderá ser acessado por qualquer outra classe, se privado, esse componente apenas poderá ser acessado dentro da própria classe.

Geralmente as variáveis membro são privadas, e implementam-se métodos para modificá-las. Desta maneira, sempre que necessário alterar seu valor, isso deverá ser feito através de algum método, impedindo que, acidentalmente, alguém acesse-a diretamente, corrompendo sua informação. Desta maneira, quase a totalidade dos programas numa linguagem OO (orientada a objetos) consiste na comunicação de objetos por meio de chamadas aos seus métodos, inclusive visando a modificação de suas variáveis membro.

A ideia básica do encapsulamento é a de disponibilizar um objeto com todas as suas funcionalidades sem necessariamente saber como ele funciona, ou seja, ser capaz de reutilizar classes sem necessariamente saber seu código, o importante é conhecer suas operações disponíveis e proteger suas características por meio de acessos públicos ou privados.

I.7 Ligações e Associações

As ligações e associações são os mecanismos pelos quais é possível estabelecer interações entre objetos e classes.

Uma associação descreve um grupo de ligações com estrutura e semântica comuns. A ligação é uma conexão entre duas instâncias de objetos, ou seja, a ligação é uma instância da associação.

I.8 Generalização e Herança

Generalização e herança são abstrações de compartilhamento de similaridades entre classes, preservando suas diferenças.

A generalização consiste no relacionamento entre uma classe e uma ou mais versões refinadas (especializações) da mesma. Já a herança representa a relação entre as classes por meio de hierarquias.

A classe sendo refinada é chamada de superclasse (ou classe base), já sua versão refinada é chamada de subclasse (ou classe derivada).

Dividem-se classes em subclasses, mantendo o princípio de que cada subclasse herda as características da classe da qual foi derivada e ainda cria algumas características particulares. Novamente, exemplificando com nossa classe Automóvel:

Superclasse: Automóvel

Atributos:

Número de rodas

Tipo de Combustível

Velocidade máxima

Cor

Métodos:

Acelerar

Frear

Virar para esquerda

Virar para direita

Subclasse: Caminhão

Atributos:

Número de carretas

Métodos:

Desacoplar carreta

A subclasse Caminhão possuirá os mesmos métodos e atributos que sua superclasse Automóvel, porém ela é mais especializada, uma vez que contém atributos e métodos próprios.

Ressalta-se ainda que generalização e herança podem ser recursivamente aplicadas a um número arbitrário de níveis, e ainda, um dos principais usos de classe derivadas é aumentar a eficiência da programação pela não necessidade da criação de códigos repetitivos, ou seja, ao tornar Caminhão subclasse de Automóvel, não há necessidade de fazer novamente os métodos e atribuições já existentes na superclasse.

I.9 Polimorfismo e Sobrecarga

O polimorfismo consiste no uso de um único nome para definir várias formas distintas, em outras palavras, refere-se a criação de uma família de funções que compartilham do mesmo nome, mas cada uma tem um código independente. A sobrecarga é um tipo particular de

polimorfismo. Por exemplo, seja o operador aritmético "+"(simbolo de soma). O computador executa operações completamente diferentes para somar números inteiros e números reais, porém a soma desses dois tipos de dados é representado por um único símbolo.

I.10 Construtores e Destrutores

Quando um objeto é criado, automaticamente é feita sua inicialização através de um método chamado Construtor, tal método possui o mesmo nome da classe e é executado automaticamente toda vez que um objeto for criado. Quando um objeto é destruído, há outro método especial que é chamado automaticamente. Esta função é chamada destrutor. Ressalta-se ainda que o destrutor não pode receber argumentos nem ser chamado explicitamente pelo programador.

Anexo II

UML - Unified Modeling Language

A descrição, exibição e os conceitos básicos, sintaxes e significados de alguns diagramas de UML serão detalhados neste apêndice. Os estudos aqui demostrados foram baseados nas referências [7] [8], a partir do trabalho [4].

II.1 Diagramas de Classes

Um diagrama de classe mostra uma estrutura estática das classes que constituem o sistema, bem como o relacionamento entre diversas classes.

II.1.1 Representação de uma Classe

Veja a figura II.1 que representa a classe Automóvel (vide Apêndice A para maiores detalhes sobre esta classe usada para exemplos).

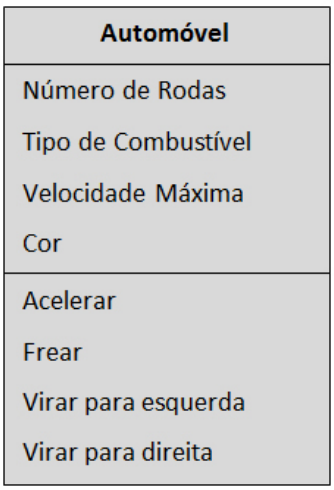

Figura II.1: Representação da classe Automóvel

A figura II.1 é composta por 3 retângulos. O primeiro deles contém o nome da classe e deve estar em negrito. Já o retângulo intermediário contém os atributos da classe. No último retângulo, deve haver os métodos que podem ser executados pela classe.

II.1.2 Associação Simples

A Associação Simples mostra o relacionamento ou dependência de uma classe com a outra. O símbolo que a representa pode ser verificado na figura II.2.

Figura II.2: Representação da Associação Simples

II.1.3 Generalização e Herança

A Herança ou Generalização indica que o comportamento da classe ou característica da classe muda. O símbolo para a Herança pode ser vista na figura II.3. Um exemplo de sua utilização se mostra na figura II.4, utilizando o exemplo da superclasse Automóvel e subclasse Caminhão, presentes no Apêndice A.

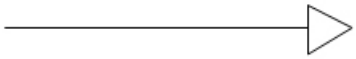

Figura II.3: Representação da Herança

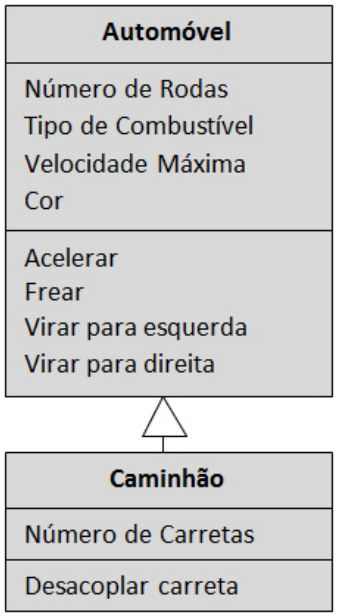

Figura II.4: Exemplo de Herança entre a superclasse Automóvel e a subclasse Caminhão

II.1.4 Agregação

Pode-se dizer que a agregação é uma associação em que um objeto é parte de outro, de tal forma que a parte pode existir sem o todo. Isso quer dizer que é uma relação do tipo "todo/parte"ou "possui um". É uma forma especializada de associação na qual um todo é relacionado com suas partes. Na figura II.5 pode-se verificar seu símbolo.

Figura II.5: Representação da Agregação

II.1.5 Composição

Tipo de associação de agregação, mas a diferença entre elas é que a composição faz parte do todo, e ainda, depende do todo. Em outras palavras, os objetos são inseparáveis, quando um objeto Pai (pertencente a uma Superclasse) é destruído o mesmo ocorre com o objeto Filho (pertencente a subclasse). Na figura II.6 pode-se verificar seu símbolo.

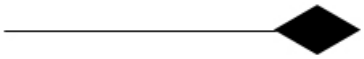

Figura II.6: Representação da Composição

II.2 Diagrama de Caso de Uso

Primeiramente, sabe-se que um Ator pode ser uma pessoa, um sistema, ou uma entidade externa. É ele quem realiza uma atividade e sempre atua sobre um caso de uso.

Um caso de uso é a descrição de uma funcionalidade (um uso específico) do sistema. Tal descrição dos usos é normalmente feita através de uma planilha contendo textos explicativos.

Em suma, os diagramas de caso de uso mostram um certo número de atores externos e suas conexões com os casos de uso que o sistema provê. Já a funcionalidade e o fluxo dos mesmos são apresentados através de diagramas de atividades

II.3 Diagrama de Atividade

Um diagrama de atividade mostra um fluxo sequencial de ações em um único processo, sendo que as ações são as unidades básicas de uma atividade. Também é verificado como uma

atividade depende da outra.

Esse diagrama pode especificar mensagens e objetos sendo enviados ou recebidos como parte das ações que estão sendo executadas. Condições e decisões, como também execução de ações paralelas, podem ser vistas neste diagrama.

II.4 Diagrama de Sequência

Consiste em um diagrama que tem o objetivo de representar como as mensagens entre os objetos são trocadas no decorrer do tempo para a realização de uma operação, ou seja, mostra a evolução de uma dada situação em determinado momento do *software*.

Anexo III

Scrum

Scrum é uma metodologia ágil para gestão e planejamento de projetos de software. Neste modelo, os projetos são divididos em ciclos (tipicamente mensais) chamados de *Sprints*. O *Sprint* representa um Time Box dentro do qual um conjunto de atividades deve ser executado. Metodologias ágeis de desenvolvimento de software são iterativas, ou seja, o trabalho é dividido em iterações.

As funcionalidades a serem implementadas em um projeto são mantidas em uma lista que é conhecida como *Product Backlog*. No início de cada *Sprint*, faz-se um *Sprint Planning Meeting*, ou seja, uma reunião de planejamento na qual o *Product Owner* prioriza os itens do *Product Backlog* e a equipe seleciona as atividades que ela será capaz de implementar durante o *Sprint* que se inicia. As tarefas alocadas em um *Sprint* são transferidas do *Product Backlog* para o *Sprint Backlog*.

A cada dia de uma *Sprint*, a equipe faz uma breve reunião (normalmente de manhã), chamada *Daily Scrum*. Ela tem como objetivo disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho a ser realizado no dia que se inicia.

Ao final de um *Sprint*, a equipe apresenta as funcionalidades implementadas em uma *Sprint Review Meeting*. Finalmente, faz-se uma *Sprint Retrospective* (identifica o que funcionou bem, o que pode ser melhorado e que ações serão tomadas para melhorar) e a equipe parte para o planejamento do próximo *Sprint*. Assim reinicia-se o ciclo.

O Resumo do processo de Scrum pode ser visto na Figura III.1, baseado na referência [9].

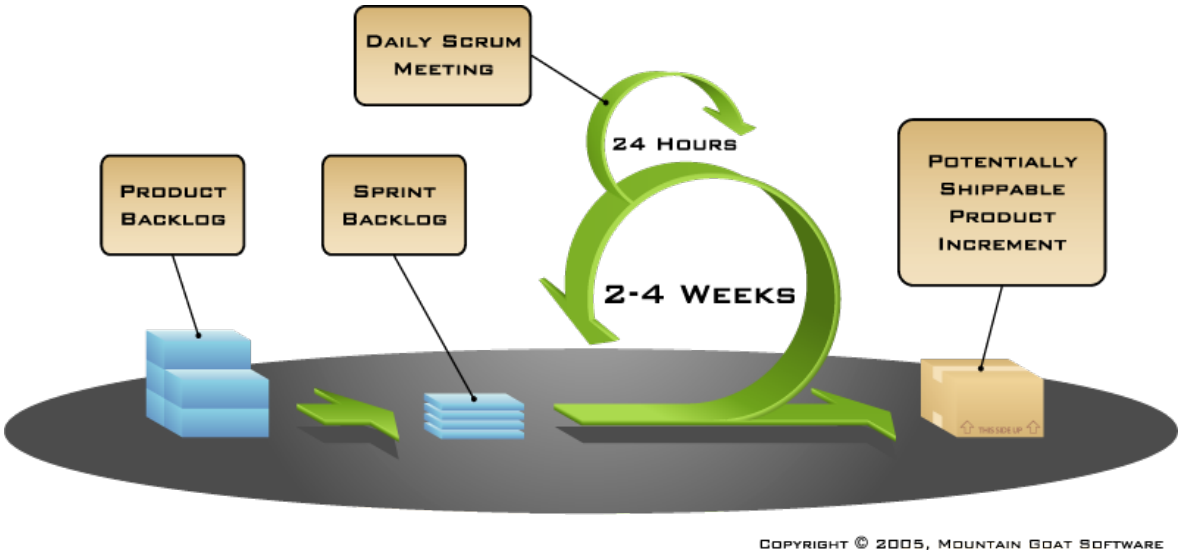

Figura III.1: Resumo do processo de Scrum. (Fonte da imagem: *website MountainGoat*)