TRABALHO DE GRADUAÇÃO

MÉTODO BASEADO EM VISÃO COMPUTACIONAL PARA RECONHECIMENTO DE DÍGITOS VISANDO A LEITURA DE CONSUMO EM HIDRÔMETROS COM INDICAÇÃO ANALÓGICA E DIGITAL

GAUBERT VINÍCIUS SANTIAGO

Brasília, Dezembro de 2017

ENGENHARIA MECATRÔNICA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

MÉTODO BASEADO EM VISÃO COMPUTACIONAL PARA RECONHECIMENTO DE DÍGITOS VISANDO A LEITURA DE CONSUMO EM HIDRÔMETROS COM INDICAÇÃO ANALÓGICA E DIGITAL

GAUBERT VINÍCIUS SANTIAGO

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Alberto José Álvares, ENM/UnB
Orientador

Prof. Bruno Luiggi M. Espinoza, CIC/UnB
Examinador interno

Prof. José Maurício S. T. Motta, ENM/UnB
Examinador interno

Brasília, Dezembro de 2017

FICHA CATALOGRÁFICA

GAUBERT VINÍCIUS SANTIAGO
Método Baseado em Visão Computacional para Reconhecimento de Dígitos Visando a Leitura de Consumo em Hidrômetros com Indicação Analógica e Digital,

[Distrito Federal] 2017.

29, 127p., 210 x 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2015). Trabalho de Graduação – Universidade de Brasília, Faculdade de Tecnologia.

1. Visão Computacional
2. Aprendizado de Máquina
3. Automação de Hidrômetro
4. Medição de Vazão

I. Mecatrônica/FT/UnB
II. Graduação em Engenharia de Controle e Automação

REFERÊNCIA BIBLIOGRÁFICA

SANTIAGO, G. V., (2017). Método Baseado em Visão Computacional para Reconhecimento de Dígitos Visando a Leitura de Consumo em Hidrômetros com Indicação Analógica e Digital. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$29, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 127p.

CESSÃO DE DIREITOS

AUTOR: Gaubert Vinícius Santiago

TÍTULO DO TRABALHO DE GRADUAÇÃO: Método Baseado em Visão Computacional para Reconhecimento de Dígitos Visando a Leitura de Consumo em Hidrômetros com Indicação Analógica e Digital.

GRAU: Engenheiro ANO: 2017

Universidade de Brasília

UnB - Brasília, Distrito Federal, 70910-900.

Dedicatória

Dedico este Trabalho de Graduação aos meus pais Carlos e Edna, meus avós Jesus e Zilda, minha noiva Hannah e minha irmã Giselle.

GAUBERT VINÍCIUS SANTIAGO

Agradecimentos

Agradeço a Deus pela concretização do sonho de me tornar Engenheiro de Controle e Automação. Agradeço também a minha família, amigos, professores e entidades envolvidas.

GAUBERT VINÍCIUS SANTIAGO

RESUMO

Este trabalho se concentrou no campo de visão computacional aplicada a instrumentos de medição. Seu objetivo foi fazer a extração de dígitos de um hidrômetro analógico para obter o consumo instantâneo por meio de *Optical Character Recognition* (OCR) e estimar a vazão de água proveniente. Inicialmente, realizou-se a detecção dos dígitos em duas etapas. Na primeira, alinhou-se a imagem horizontalmente, utilizando a transformada de Hough e encontrou-se o visor do hidrômetro baseando-se em seu formato geométrico. Na segunda, fez-se a detecção de bordas por Canny e filtraram-se os dígitos com base nas dimensões dos contornos e algumas regras específicas de proporção e distâncias. A extração, por sua vez, foi realizada assim que as coordenadas de recorte foram adquiridas para cada um dos sete dígitos da sequência. Os dígitos recortados foram agrupados em um conjunto de 5000 imagens para conduzir o aprendizado de máquina, o qual foi alcançado mediante os classificadores *Support Vector Machine* (SVM), *k-Nearest Neighbor* (kNN) e *Multilayer Perceptron* (MLP). Estes, com as melhores condições de contorno testadas, produziram taxas de erro que cresciam, respectivamente. Por causa de sua melhor performance, o SVM foi escolhido para executar o OCR no sistema de visão computacional. Em testes feitos no hidrômetro analógico de uma bancada de teste, também construída para o projeto, foi constatado que o sistema de visão funcionava relativamente bem sob determinadas condições. Com iluminação adequada dentro do laboratório e um material plástico semitransparente ao redor da *webcam*, usado para evitar reflexo de luz no visor do hidrômetro, a detecção de dígitos era invariante da posição da *webcam* para determinadas distâncias. Em um teste rodado por um período maior que vinte e seis horas, a máquina foi forçada a reconhecer os dígitos a cada minuto com uma permissão de cinco tentativas por minuto. Os dados resultantes revelaram as chances de sucesso do sistema era de 50% para fazer o reconhecimento de dígitos. Além disso, resultados mostraram que a medição de vazão possuía um erro de cerca de 9% em relação ao valor de vazão inspecionado na bancada.

Palavras Chave: Automação de Hidrômetros, Aprendizado de Máquina, Visão Computacional, Medição de Vazão

ABSTRACT

The field of study of this project is computational vision applied to measurement instruments. The project's main purpose was to recognize the digits of an analog hydrometer to obtain an instantaneous water consumption data through Optical Character Recognition (OCR), in addition to estimating the water flow from the hydrometer. Primarily, the detection of the digits was performed in two steps. First, the image was aligned horizontally, using Hough transform, and the hydrometer display was found based on its geometric shape. Second, edges were detected by a Canny operator and the desired digits were filtered based on the contours and specific rules of proportion and distances, yielding the digits coordinates for snipping. Then, each of the digits of the seven-digit chain were snipped and extracted. The snipped digits were grouped into a set of 5000 images to drive the machine learning, using the classifiers *Support Vector Machine* (SVM), *k-Nearest Neighbor* (kNN), and Multilayer Perceptron (MLP). After testing these three classifiers, output data indicated that SVM produced the lowest error rate, kNN was the second lowest, and MLP yielded the highest rate of error. Because of its decreased error rate, the SVM was chosen to perform the OCR in the computer vision system. Tests were run on a hydrometer built onto a test bench. After reviewing the tests' results, it was found that the artificial vision functioned relatively well under certain conditions. With adequate light inside the laboratory and a semi-transparent plastic material around the webcam, used to avoid light reflection on the hydrometer *display*, the detection of the digits was found to be invariant from the camera position for certain distances. In a test over a twenty-six-hour period, the machine was forced to recognize digits every minute with an allowance of five attempts per minute. The data showed a 50% success rate for the recognition of the digits. In addition, the results showed that the flow measurement had an error of around 9% in respect with the flow value inspected on the test bench.

Keywords: Hydrometer Automation, Machine Learning, Computer Vision, Flow Measuring

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

L	Quantidade de níveis de cinza; Limiar
b	Quantidade de bits; *offset* do hiperplano
M	Número de linhas; *Kernel* de Sobel
N	Número de colunas; número de camadas ocultas;
r	Tom de cinza
p	Função probabilidade
n	Quantidade de um determinado nível de cinza na imagem; não dígito
I	Imagem
K	*Kernel*
g	Gaussiana aplicada a diferença de intensidade
f	Gaussiana de suavização no espaço
P	Dados da imagem de entrada
J	Dados da imagem de saída
k	Função normalização; quantidade de vizinhos
G	Gradiente
$\mathcal{A}$	Adjacência
C	Classe de componentes de uma imagem; contorno; penalidade para erro; consumo
r	Raio, rho
E	Função de perda
t	Resultado esperado na função de perda
y	Resultado encontrado na função de perda
w	Normal ao hiperplano; Largura
x	Pontos distribuídos pelo espaço do *kernel*
c	Classe de dígitos
h	Altura
l	Profundidade
d	Distância
L	Lista
a	Taxa de aprendizado inicial
e	Estado randômico
V	Volume de água
Q	Vazão
t	Tempo

Grupos Adimensionais

i, j, k	Contador
u, v	Posições de píxeis em uma imagem
x, y	Coordenada de um píxel em um *kernel*
p, q	Píxeis

Símbolos Gregos

σ	Desvio padrão
Ω	Janela de interesse
θ	Limiar; Ângulo de inclinação
μ	Média aritmética
λ	Razão entre largura e altura dos dígitos
ϕ	Ângulo do gradiente de um
γ	Coeficiente do *kernel*
δ	Tolerância
α	Penalidade para normalização
Δ	Variação

Subscritos

r	Tom de cinza
avr	Média
x	Componente horizontal
y	Componente vertical
$lapl$	Laplaciano
ob	Objeto
pf	Pano de fundo
c	Centro da circunferência; camadas
h	Horizontal
v	Vertical
esp	Especificação
p	Projeto
d	Dígito
min	Mínimo
max	Máximo
f	Contornos filtrados
o	Inicial
opt	Ótima
cor	Corrente
ant	Anterior
a	Média
n	Nominal

Sobrescritos

’	Matriz ou valor modificado; primeira derivada
”	Segunda derivada
$\rightarrow$	Vetor

Siglas

ARM	Advance Risc Machine
Caesb	Companhia de Saneamento Ambiental
CCD	Charged-Couple Device
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
DF	Distrito Federal
DVI	Digital Visual Interface
GPRS	Global Position Radio System
GPS	Global Position System
Graco	Grupo de Automação e Controle
HOG	Histogram of Oriented Gradients
FoS	Factor of Safety
HD	High Definition
HDMI	High-Definition Multimedia Interface
ICDAR	International Conference on Document Analisys and Recognition
INPI	Instituto Nacional de Propriedade Industrial
JMEDS	Java Multi Edition Devices Profile for Web Services Stack
kNN	k-Nearest Neighbor
LED	Light Emitting Diode
LTS	Long Term Support
MLP	Multilayer Perceptron
MSER	Maximally Stable Extremal Regions
OCR	Optical Character Recognition
PDF	Portable Document Format
PGOMG	Coordenadoria de Gestão de Hidrômetros
PVC	Polyvinyl Chloride
RF	Rádio Frequência
RGB	Red Green Blue
ROI	Region of Interest
RRD	Round Robin Databases
SD	Secure Digital
SMS	Short Message Service
SSOCR	Seven Segment Optical Character Recognition
SVM	Support Vector Machine
UHS	Ultra High Speed
UnB	Universidade de Brasília
USB	Universal Seral Bus
UTC	Coordinated Universal Time
VGA	Video Graphics Array
VDC	Volts Direct Current
VAC	Volts Alternating Current
Wi-Fi	Wireless Fidelity
XML	Extensive Markup Language
YAML	Yet Another Markup Language

Capítulo 1

Introdução

Atualmente, existem três formas de fazer a leitura de medição de hidrômetros. A primeira consiste na tradicional leitura a domicílio, em que um leiturista passa de casa em casa nas ruas para coletar o consumo acumulado de água nas residências. A segunda maneira de o fazer é através de um processo chamada *walk by*, no qual também há um leiturista que percorre as ruas com um módulo receptor de dados conectado a um *data logger*. Os dados são recebidos de um módulo acoplado a um hidrômetro o qual faz a contagem de pulso para determinar a leitura do consumo de água. A terceira forma é realizada por transmissão Global Position Radio System (GPRS), em que se usa o mesmo módulo emissor mencionado anteriormente. Seus dados são enviados por Rádio Frequência para uma unidade concentradora. Tanto esta quanto o *data logger* alimentam um banco de dados com as leituras de consumo, e a caixa concentradora pode executar a leitura diariamente, permitindo um melhor supervisionamento do consumo dos clientes. Por exemplo, uma leitura periódica diária pode detectar vazamento de água. Tal detecção pode colaborar para com a medição do índice de perdas de água, segundo a Companhia de Saneamento Ambiental do Distrito Federal (Caesb). Nesse sentido, uma máquina de visão computacional de baixo custo poderia surtir o mesmo efeito, quando apontada para o hidrômetro. Este capítulo procura contextualizar o leitor sobre a escolha deste Trabalho de Graduação, apresentar o que motiva a execução deste, definir o problema no qual ele está envolvido, e apontar os objetivos do projeto, entre outros.

1.1 Contextualização

Em 2016, a Companhia de Saneamento Ambiental do Distrito Federal (Caesb-DF) entrou em contato com o Prof. Dr. Alberto Álvares, professor associado da Universidade de Brasília (UnB), para tratar sobre uma demanda interna que tinha acabado de surgir a respeito da manutenção de seus ativos. A demanda consistia da necessidade de uma solução que diminuísse os custos com a coleta de leituras em residências do DF, que estavam elevados, bem como o prejuízo estimado devido a vazamentos ou mal funcionamento de hidrômetros.

Na conversa com a empresa de saneamento foi dito que, de acordo com a Coordenadoria de Gestão de Hidrômetros (PGOMG) da Caesb, existem basicamente dois tipos de clientes com os

quais se tem gastos ao fazer as leituras de hidrômetros. O primeiro é o cliente comum, cuja residência recebe a visita de um leiturista que faz a coleta do consumo e gera a conta de água. Para tal serviço, há custos com o próprio leiturista, passagens de ônibus, o minimonitor, a impressora térmica portátil, a folha térmica, a transmissão de dados via *Global Position Radio System* (GPRS), o monitoramento via *Global Position System* (GPS) e o coletor/transmissor de dados. Somando todas essas despesas, a leitura com emissão simultânea de conta para a categoria cliente comum é de R$ 0,60 por unidade (Tabela I.1). O segundo tipo de cliente é chamado cliente especial — em que também se incluem situações específicas. Nesse grupo, os gastos são similares aos supracitados, salvo que, em vez de se usar um ônibus para se deslocar até o local de medição, uma motocicleta é utilizada. O preço unitário da leitura sai a R$0,59 (Tabela I.2).

Com base no levantamento de dados feito pelo PGOMG, contudo, o serviço de leituras de hidrômetros e emissão de contas não é tão barato como se imagina. No DF, as regiões administrativas foram subdivididas em quatro grupos pela Caesb, que são denominados lotes — Brasília, junto com o Lago Sul e o Lago Norte, por exemplo, faz parte do lote 01. Os lotes 01, 02, 03 e 04 possuem, respectivamente, 161.789, 147.163, 169.969, 201.337 ligações de água, e são concorridos através de licitações por várias empresas de medição terceirizadas. Ao todo, então, são mais de 680 mil ligações de água por todo o DF. E o custo gerado a Caesb passa de 20 milhões de reais (Tabela I.3).

De posse dessas informações, dentre outras, o Prof. Alberto Álvares estudou o caso e propôs à Caesb uma solução para o problema. Tal solução foi basicamente um método criado para supervisionamento remoto de medidores de vazão e energia, tais como hidrômetros e relógios de luz, respectivamente, via internet e *Short Message Service* (SMS). Sua vantagem era que se travava de uma abordagem não invasiva, excluindo a necessidade de se ter um leiturista para realizar a leitura, e não exigia troca dos hidrômetros analógicos ou digitais já instalados nas residências, uma vez que a captura de dados seria feita por uma câmera acoplada aos medidores de vazão. A solução foi registrada como patente na base de dados no Instituto Nacional de Propriedade Industrial (INPI) sob os números de registro BR1020160139970 de 2016, em sua primeira versão, e BR1020170208125 de 2017, em sua segunda.

Em um determinado momento, surgiu a oportunidade de colocar em prática boa parte dos conceitos abordados na patente do Prof. Alberto Álvares em forma de um Trabalho de Graduação. A Caesb foi comunicada nessa ocasião e se dispôs a ajudar com grande parte dos materiais que fossem necessários, inclusive fornecendo os hidrômetros, para construir uma bancada de testes.

1.2 Motivação

Do ponto de vista acadêmico-científico, existe ao menos um fato que motiva o desenvolvimento deste trabalho. Tal fato consiste no ineditismo associado de estado da arte. Isso porque, através de pesquisas realizadas sobre a temática, notou-se que as literaturas existentes abordam técnicas de visão computacional para automação de leitura de medidores de gás e energia elétrica apenas.

Analisando a solução proposta de uma perspectiva mercadológica, no entanto, há várias razões

pelas quais o desenvolvimento do produto em questão se torna viável. A primeira delas está relacionada ao barateamento de produtos de cunho tecnológico e seus constantes aprimoramentos, à medida que a tecnologia avança. Microprocessadores e microcontroladores com memórias suficientemente grandes e *clocks* velozes já atendem muito bem aplicações de sistemas embarcados; sem falar das enormes capacidades de armazenamento de dados de minúsculos cartões de memória que chegam a ter taxas de transmissão impressionantes para escrita e leitura. E o mesmo acontece para as câmeras, que possuem resoluções cada vez maiores e dimensões cada vez menores. Tudo isso, felizmente, pode ser encontrado por um preço acessível, a gosto do consumidor.

A segunda razão está no fato de que a internet está gradativamente ficando mais presente na vida do brasileiro. É o que diz um estudo realizado pelo Comitê Gestor de Internet no Brasil, pelo Centro Regional de Desenvolvimento da Sociedade de Informação e pelo Núcleo de Informação e Coordenação do Ponto BR que entrevistou 23.465 domicílios em todos os estados brasileiros, entre novembro de 2015 e junho de 2016. Segundo esse estudo, 58% da população brasileira navega *on-line* na rede, o que equivale a 102 milhões de internautas, sendo que o acesso por computadores de domicílios permanece praticamente constante (50%), quando comparado ao ano de 2014, e o acesso à internet por celulares é feito pela maioria do grupo estudado (89%). No caso dos usuários de *smartphones*, 87% deles acessam por *Wireless Fidelity* (Wi-Fi), e 72%, por 3G ou 4G [25]. Além do mais, um levantamento de preços de planos de internet residencial (Tabela I.4) e para celular (Tabela I.5) no DF mostra que também há preços razoáveis, tais como R$ 79,90 mensal — com 40 reais de desconto mensal após o 6º mês — e R$ 79,90, respectivamente.

A terceira razão consiste na estimação de que o produto a ser criado venha a ter melhor custo-benefício do que a solução por rádio frequência, atualmente usada por algumas companhias de saneamento para conduzir as leituras nas residências. A Caesb, por exemplo, tem um projeto de rádio frequência (RF) em desenvolvimento visando ao atendimento do Lago Norte todo, onde há 20 mil hidrômetros digitais da marca Hydrus com saída pulsada. Segundo Roberto Kitahara, engenheiro eletrônico e analista de sistemas de saneamento da Caesb, o projeto que prevê a instalação de um módulo transmissor de dados em cada hidrômetro digital de saída pulsada, bem como torres de concentradores de dados, cujo alcance geralmente é de 1 Km, podendo chegar a 5 Km, dependendo da empresa fabricante, conseguindo atender cerca de mil pontos cada, será lançado em edital em breve. Ainda segundo ele, apesar de ser uma solução bastante onerosa, seria de grande relevância por dois motivos: em um futuro próximo, evitaria o tempo necessário de 1 mês para completar a leitura dos milhares de hidrômetros lá existentes e diminuiria o índice de perdas de água de 24% para 12%, alcançando padrões europeus de medição.

No caso do dispositivo a ser desenvolvido neste Trabalho de Graduação, as vantagens são muitas e o trabalho com a leitura é simples e automático. Isto é, não será necessária nenhuma antena receptora/concentradora de dados, tampouco um funcionário para passar pelas ruas coletando dados referentes a leituras de hidrômetros de casa em casa, o qual posteriormente precisaria fazer sincronização com a base de dados através de um computador [22]. Na verdade, o dispositivo será *plug-and-play*; terá uma confiabilidade altíssima, sendo possível se checar o valor medido por meio da imagem usada para extração da leitura, tanto em tempo real quanto por histórico de consumo; e funcionará periodicamente, podendo ter seu período de coleta ajustado de acordo com

a necessidade do cliente ou da companhia gestora dos dados.

A quarta razão se dá pela possibilidade de integrar ao sistema agricultores de regiões distantes e acesso difícil ou inviável por leituristas. Para os donos de plantações, a leitura seria executada remotamente por uma companhia de saneamento, e o agricultor poderia estar antenado no consumo de seu próprio cultivo, assim como saber o preço da conta de água do mês a pagar. Dessa maneira, quando houvesse um período de seca, as entidades responsáveis pelo uso adequado da água poderiam acompanhar o consumo de água e multar agricultores que ultrapassassem a cota imposta, uma vez que o ser humano tem preferência, em detrimento da agricultura, segundo a legislação brasileira [26]. Para realizar a integração, bastaria que aquele determinado local do plantio no campo tivesse acesso a uma conexão GPRS razoável.

Por fim, os serviços passíveis de serem entregues com a solução idealizada são os fatores que mais motivam a execução deste Trabalho de Graduação. Isso porque, com eles, o acesso a dados tais como leitura, vazão e afins seria possível de qualquer local, a qualquer hora, quantas vezes fosse preciso, com apenas um clique, pois seria dado por meio de acesso à internet. Tanto o cliente como a companhia de água poderiam estipular metas, e o cliente poderia então receber avisos por SMS ou e-mail quando a predição referente a tal meta fosse de ultrapassar o consumo estipulado. Um alerta a respeito de um possível consumo fora do padrão, significando vazamento, poderia ser emitido para ambas as partes interessadas. Bonificações ou punições poderiam ser geradas e aplicadas de acordo com gastos residenciais ou empresariais.

1.3 Definição do Problema

Necessita-se de uma tecnologia que seja capaz de executar a medição de consumo de água para diferentes e variados relógios de hidrômetros, analógicos ou digitais. Tal tecnologia deve ser robusta para trabalhar sozinha em dias ensolarados, nublados, chuvosos, bem como em ambientes escuros, e acertar o valor real das leituras com confiabilidade maior ou igual à de um leiturista contratado por companhias de saneamento de água. Além disso, a solução tecnológica deve trabalhar 24 horas por dia, 7 dias por semana, por todo mês, todo ano, de maneira periódica, sendo o intervalo de atualização um parâmetro ajustável, cuja alteração seja concernente ao usuário ou aplicação. Também deve ser capaz de conduzir uma reinicialização automática por si própria, caso haja a falta ou pico de energia, e então continuar de onde parou, com acesso a métricas da última leitura tirada antes da interrupção repentina, as quais têm de ser salvas imediatamente depois de cada medição feita em um arquivo, a fim de se gerar um *log report* para usos futuros. De preferência, sua conexão com a internet tem que ser *wireless*, e seus dados gerados para transmissão não devem ser pesados nem sobrecarregar a rede em uso, independentemente se esta for Wi-Fi, *Ethernet*, ou GPRS. Por fim, o dispositivo tem que ser *plug-and-play*, de modo que o usuário final não tenha nenhum trabalho com sua instalação de seu *software*.

1.4 Objetivos do Projeto

Este projeto tem como objetivo principal fazer a medição de vazão em tempo real de um hidrômetro analógico via internet, utilizando um método de visão computacional, e validá-lo em uma bancada de testes. Secundariamente, outros objetivos são implementar um algoritmo de localização de região de interesse e detecção de dígitos invariante de ângulo e aplicar OCR com classificadores tais como SVM, kNN e MLP, analisando a performance de cada um.

1.5 Resultados Obtidos

Tendo em vista que se usou o Crontab para invocar o algoritmo a cada um minuto e que podiam ser feitas cinco tentativas por invocação no máximo, verificou-se que a coleta de leitura era feita 50% das vezes dentro de um período de aproximadamente 26 horas corridas. Nesse caso, empregou-se o classificador SVM, o qual teve a maior taxa de acertos no aprendizado de máquina. A média da vazão estimada pelo sistema de visão foi de cerca de 130 L/h. Tal valor ficou razoavelmente abaixo da vazão real do sistema, que era de 144 L/h. A média da variação do volume detectada foi de 4,5 L, o que sugere que, para medir uma diferença média de 1 L de uma leitura para outra, um período de execução de 15 segundos teria de ser usado, em vez de 60 segundos. Por outro lado, a média de variação do tempo determinada pelos testes foi de aproximadamente 120 segundos, indicando que o sistema poderia atrasar o dobro do valor de tempo qualquer estipulado para atualizar a leitura do hidrômetro *on-line.* Finalmente, observou-se que o método de localização e região de interesse e detecção de dígitos é invariante de ângulo para até cera de 19 cm do visor do hidrômetro. Além disso, constatou-se a melhor forma de publicar a vazão do hidrômetro é por média móvel, uma vez que há um desvio padrão de cerca de 25,84 L/h em torno da vazão estimada.

1.6 Apresentação do Manuscrito

Este Trabalho se concentra especificamente no âmbito de processamento digital de imagens, aprendizado de máquina e visão computacional. Nesse sentido, a Revisão de Literatura visa fornecer conceitos básicos acerca dessas áreas, tais como filtragem no domínio espacial, separando filtros lineares dos não lineares, os dois tipos de transformada de Hough, aprendizado de máquina, evidenciando HOG, OCR e classificadores de padrões como o SVM, kNN e MLP. Além disso, define o que é visão computacional e apresenta vários exemplos de aplicação dela. Trabalhos Correlatos, compostos de dois artigos, dois *softwares* abertos e uma solução de telemetria por RF, aborda métodos bem eficientes para solucionar o desafio de automatizar instrumentos de medição. Já o capítulo de Desenvolvimento visa detalhar todo o procedimento tomado para construir a bancada de testes bem como todo o desenvolvimento do *software* criado, incluindo as partes de localização, detecção e extração de dígitos, treinamento, tomada de decisões e divulgação de métricas de medição via internet. Na sequência, Resultados traz os dados obtidos em um teste feito com o sistema de visão computacional durante um período de aproximadamente 26 horas consecutivas,

frisando parâmetros como variação de volume, de tempo e vazão estimada. Por fim, Conclusões faz um desfecho do trabalho em função dos resultados alcançados, avaliando a performance do dispositivo de visão artificial desenvolvido.

Capítulo 2

Fundamentação Teórica

Este trabalho se trata de uma aplicação prática de visão computacional, em que se fará o uso constante e recorrente de uma câmera embarcada em um computador em plataforma *Raspberry Pi* de arquitetura *Advanced RISC Machine* (ARM). Tal tipo de sensor será, provavelmente, o instrumento de maior relevância, pois todo o artifício de alinhamento horizontal de imagem, filtragem de ruídos, aprendizado de máquina, interpretação de dados e tomada de decisões serão aplicados e derivados de *frames* advindos dele. Por essa razão, é de extrema importância a aquisição de uma base sólida sobre técnicas consagradas de visão computacional. Com o mesmo teor de importância, é necessário que estas sejam utilizadas na ordem adequada e de maneira moderada. Tudo isso para que se possa ter um desenvolvimento de *software* eficiente e eficaz. Dessa maneira, faz-se necessário uma certa revisão teórica de determinados conteúdos que se julga serem chaves para resolução do problema. Nesse sentido, serão abordados vários tópicos acerca de Processamento Digital de Imagem, Aprendizado de Máquina e Visão Computacional, nas Seções 2.1, 2.2 e 2.3, respectivamente.

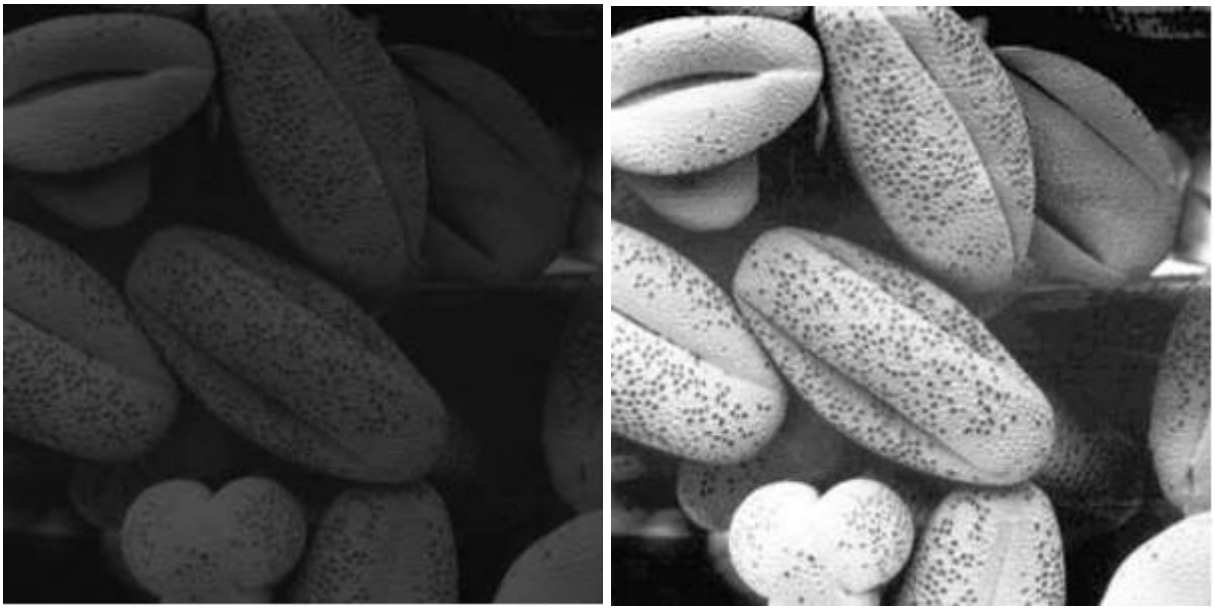
2.1 Conceitos de Processamento Digital de Imagens

Imagens, em geral, podem carregar muitos ruídos. Seja por causa do ambiente impróprio onde a foto é tirada, do objeto em foco que impossibilita uma captura razoável, pela câmera não ser robusta suficiente para produzir uma boa foto, ou pela falta de habilidade da pessoa que está fotografando, a foto perfeita às vezes foge do alcance. Em virtude disso, eventualmente, pode-se acabar tendo de usar uma foto indesejável.

Em situações como a descrita acima, é importante ter um conhecimento básico sobre processamento digital de imagens. Seu uso, sobretudo de filtros clássicos, pode deixar a foto adequada aos padrões necessários para uma determinada finalidade. Nesse intuito, serão apresentadas algumas técnicas de processamento digital de imagens, as quais serão as mais relevantes para este trabalho, em tese.

2.1.1 Equalização de Histograma

A equalização de histograma é um processo não-linear que visa ressaltar o brilho de maneira apropriada para análise visual humana [3]. Seu objetivo é mudar a imagem de tal forma que o histograma dela se torne o mais plano possível, fazendo com que os níveis de intensidade que o compõem fiquem equiprováveis. Assim sendo, o processo de equalização de histograma proporciona um maior contraste à imagem em que está sendo aplicado. Isso é exemplificado na Figura 2.1.



(a) Imagem original.

(b) Resultado da equalização de histograma.

Figura 2.1: Efeito da equalização de histograma [1].

Na Subfigura 2.1(a) não se consegue ver com clareza as fronteiras dos corpos ali ocupantes, nem sequer o fundo do sistema. Já na Subfigura 2.1(b), depois que a equalização de histograma é aplicada, as nuances se tornam mais claras. Diante disso, a equalização de histograma pode ser bem útil em aplicações em que se necessita detectar contornos característicos, fronteiras, arestas, [27].

Para entender como a equalização de histograma se dá, considera-se uma imagem com L tons de cinza, em que a faixa de valores varia no intervalo de 0 a $L-1$. L é dado por 2^b, em que o b é quantidade de bits por píxel que compõem a imagem, a qual tem uma dimensão $M \times N$, sendo M o número de linhas e N o número de colunas. Considera-se também a existência de um parâmetro r, que é tido como um tom de cor qualquer dentro do intervalo supracitado, tal que $r = 0$ representa a tonalidade preta e $r = L-1$, a tonalidade branca. Calcula-se a equalização de histograma por meio de uma transformação T, mais conhecida como mapeamento de intensidades [1], que é dada por:

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) = \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j \tag{2.1}$$

em que $k = 0, 1, 2, 3, \ldots, L-1$, p_r é a probabilidade da tonalidade r estar presente na imagem, e n, a quantidade de um determinado nível de cinza na imagem.

2.1.1.1 Exemplo de Aplicação

Considere uma imagem monocromática de dimensões 64×64, cujo píxel é composto por 3 bits, e que possua a distribuição de intensidades de tom conforme a Tabela 2.1, na qual também estão contidos passos para executar o mapeamento de intensidades.

Tabela 2.1: Equalização de histograma de uma imagem 64×64, composta por oito tons de cinza [1].

r_k	n_k	$p_r = \frac{n_k}{MN}$	$s_k = \frac{(L-1)}{MN} \sum_{j=0}^{k} n_j$	s_k
$r_0 = 0$	790	0,19	1,33	1
$r_1 = 1$	1023	0,25	3,08	3
$r_2 = 2$	850	0,21	4,55	5
$r_3 = 3$	656	0,16	5,67	6
$r_4 = 4$	329	0,08	6,23	6
$r_5 = 5$	245	0,06	6.65	7
$r_6 = 6$	122	0,03	6,86	7
$r_7 = 7$	81	0,02	7,00	7

A primeira coluna da Tabela 2.1 mostra índices referentes às intensidades de tons de cinza da imagem. A segunda coluna apresenta quantos píxeis existem para cada nível de cor. Na terceira, a quantidade de píxeis de cada cor é divida pela quantidade total de píxeis existentes na imagem, criando assim uma probabilidade de distribuição de cores de píxeis. A quarta, traz o resultado do mapeamento de intensidades, que é então arredondado para um valor inteiro mais próximo, de modo que tal resultado fique contido dentro do conjunto amostral de tons possíveis.

Ao fim, percebe-se que a nova imagem gerada pela transformação possui apenas cinco cores em seu espectro resultante e que as quantidades destas tendem a formar uma distribuição de níveis equiprováveis. Além do mais, nota-se que alguns níveis desapareceram, gerando lacunas e, assim, contribuindo com o aumento do contraste da imagem.

2.1.2 Filtragem no Domínio Espacial

Filtros espaciais são um conjunto de ferramentas usadas principalmente com o propósito de melhorar a qualidade de imagens, seja para enfatizar ou remover contornos característicos delas,

e possuem uma larga variedade de aplicações. Suas operações basicamente agem de maneira a borrar, realçar imagens, ou realçar bordas [28].

Os filtros no domínio espacial funcionam de maneira equivalente aos filtros passivos e ativos na eletrônica, de modo a rejeitar ou aceitar bandas de frequências. Daí vem a escolha de usar o termo "filtro" no campo de processamento digital de imagens.

Um exemplo para ilustrar a equivalência mencionada é o filtro passa-baixas. Na eletrônica, ele só aceita que as baixas frequências passem em uma rede elétrica. Já em uma imagem qualquer, o referido filtro espacial — também conhecido como máscara espacial, *kernel*, *template*, ou janela — produz um borramento por toda a matriz de píxeis [1].

A filtragem é baseada em uma operação feita na vizinhança, ou adjacência, do píxel do qual se quer mudar o valor. É feita por toda a imagem, alterando todos os píxeis da matriz.

2.1.2.1 Convolução

A filtragem linear de uma imagem pode ser realizada através de uma operação de convolução, em que existem a matriz imagem e a matriz *kernel*. Essa última, na prática, tem que ser sempre menor do que a primeira, e deve ser quadrada de tamanho ímpar, para que possua um píxel central.

Na operação de convolução, o píxel central da matriz *kernel* é posto sobre o píxel da primeira linha e primeira coluna da matriz imagem, e é feita a soma da multiplicação individual dos píxeis pareados — os píxeis da borda da matriz imagem, por sua vez, são convoluídos mediante a inserção de um *padding* adequado de píxeis pretos ao redor das bordas (vide Figura 2.5), para evitar um fenômeno denominado *aliasing*, que é uma distorção na imagem, em linhas gerais. Na sequência, o resultado dessa soma vai para o píxel da primeira linha e primeira coluna da matriz resultante.

De modo geral, por definição, a convolução de uma imagem I por um *kernel* K é expressa por:

$$I^{'}(u,v) = \sum_{i}^{n} \left[\sum_{j}^{n} I(u-i,\, v-j) \cdot K(i,\, j) \right] \tag{2.2}$$

em que u e v são os índices de posição de píxeis da imagem, e i e j são os índices do *kernel* — cada par significa linha e coluna, respectivamente. n é o tamanho da matriz *kernel*, que é sempre quadrada.

Outra maneira de representar a Equação 2.2 é:

$$I' = I * K \tag{2.3}$$

Para ilustrar a operação de convolução, como explicada acima, considere que a matriz imagem I e a matriz *kernel* K são as seguintes:

$$I = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

$$K = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$$

Agora, assuma que a operação esteja sendo feita no píxel da linha 2 e coluna 4 da matriz imagem. Sendo assim, o píxel central da matriz *kernel* deve coincidir com o píxel mencionado da matriz imagem. Salientando que a matriz K é rotacionada 180^o em torno de seu píxel central, pelo fato de se tratar de uma operação de convolução, a sobreposição do *kernel* na imagem é a matriz subsequente, em que se percebe números menores sendo o *kernel* e os maiores sendo a imagem [28]:

$$I * K = \begin{bmatrix} 17 & 24 & 1^2 & 8^9 & 15^4 \\ 23 & 5 & 7^7 & 14^5 & 16^3 \\ 4 & 6 & 13^6 & 20^1 & 22^9 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

Então, é feita a soma da multiplicação dos elementos pareados, representados de forma meramente ilustrativa por bases e expoentes, da seguinte maneira:

$$1 \cdot 2 + 8 \cdot 9 + 7 \cdot 7 + 14 \cdot 5 + 16 \cdot 4 + 13 \cdot 6 + 20 \cdot 1 + 20 \cdot 1 + 22 \cdot 9 = 575$$

Por fim, o resultado encontrado é inserido na posição linha 2 e coluna 4 da matriz imagem resultante I', que adquire o seguinte aspecto:

$$I' = \begin{bmatrix} i'_{11} & i'_{12} & i'_{13} & i'_{14} & i'_{15} \\ i'_{21} & i'_{22} & i'_{23} & 575 & i'_{25} \\ i'_{31} & i'_{32} & i'_{33} & i'_{34} & i'_{35} \\ i'_{41} & i'_{42} & i'_{43} & i'_{44} & i'_{45} \\ i'_{51} & i'_{52} & i'_{53} & i'_{54} & i'_{55} \end{bmatrix}$$

Na prática, no entanto, há vários tipos de *kernel*, sendo que cada um deles tem um propósito específico — e seus valores, ou pesos, não são aleatórios como aquele *kernel* do exemplo anterior, obviamente.

2.1.3 Filtros Passa-Baixas

O objetivo de um filtro passa-baixa é permitir que sinais de baixa frequência sejam filtrados em detrimento dos sinais de alta frequência. O efeito causado por tal filtragem é a diminuição

ou desaparecimento de contornos característicos em uma determinada imagem, como acontece no filtro média e no gaussiano vistos a seguir.

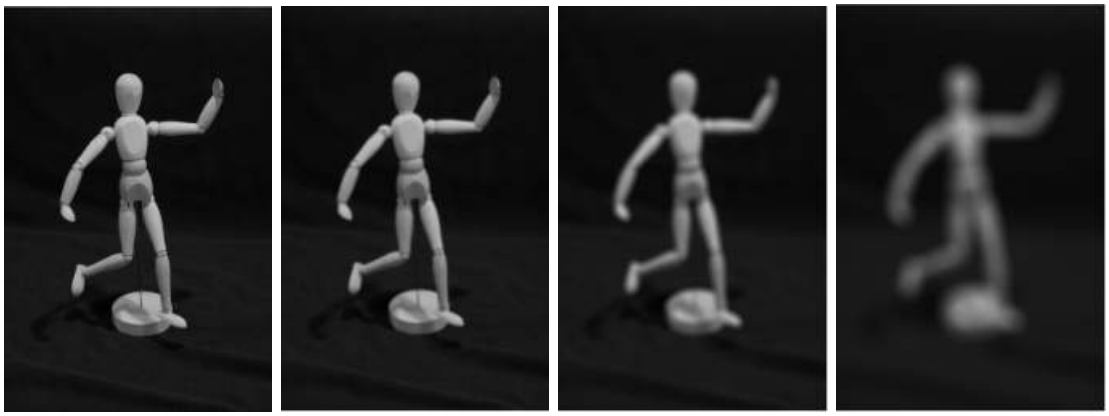
2.1.3.1 Filtro Média

O filtro média possui um *kernel* em que os pesos de seus elementos são iguais, de modo que a soma deles gere um valor unitário. A função desse arranjo é fazer com que a operação realizada em uma determinada adjacência nunca produza valores acima do branco, representado por um valor específico de tonalização, dependendo da quantidade de níveis. Um exemplo de operador média 3×3 é o *kernel* K_{avr}:

$$K_{avr} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

O objetivo do filtro média é reduzir ruídos. A desvantagem em seu uso, no entanto, é que a média é realizada com base nas vizinhanças produz um borramento na imagem.

Como foi comentado na Subseção 2.1.2, trata-se de um filtro passa-baixas. Por esse motivo, o operador média móvel rejeita componentes de frequências altas, que às vezes nada mais são do que ruídos, e pecam por diminuir o nível de detalhes [3]. Nesse contexto, é importante ressaltar que quanto maior for a tamanho do filtro média, maior será a rejeição de altas frequências. A Figura 2.2 faz uma ilustração desse comportamento do operador média em uma imagem.



(a) Imagem original. (b) *Kernel* 7×7. (c) *Kernel* 15×15. (d) *Kernel* 41×41.

Figura 2.2: Influência da dimensão do filtro média na imagem resultante [2].

Como pode-se perceber na sequência de imagens da Figura 2.2, ao passo que o tamanho do *kernel* do operador média aumenta, o borramento também fica maior e as componentes de alta frequências, caracterizadas pelos tons de cinza claros, vão se desaparecendo.

2.1.3.2 Filtro Gaussiano

O filtro gaussiano tem sido considerado um filtro muito bom para fazer suavização de imagens [3]. Seu *kernel* é concebido com base em valores de saída da Equação 2.4, cujas entradas são o desvio padrão σ, e as coordenadas x e y de cada píxel em um determinado *kernel*.

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{2.4}$$

Para um operador gaussiano 5×5 e desvio padrão igual 1, geram-se os seguintes resultados apresentados na Figura 2.3.

0.002	0.013	0.022	0.013	0.002
0.013	0.060	0.098	0.060	0.013
0.022	0.098	0.162	0.098	0.022
0.013	0.060	0.098	0.060	0.013
0.002	0.013	0.022	0.013	0.002

Figura 2.3: *Kernel* gaussiano 5×5, com $\sigma = 1$ [3].

Como se nota na Figura 2.3, a distribuição de valores no *kernel* é simétrica com relação ao píxel central. É importante perceber também que, como esperado, os valores do operador decaem à medida que se aproxima das bordas da matriz, o que remete a um aspecto de curva gaussiana 3-D (Figura 2.4).

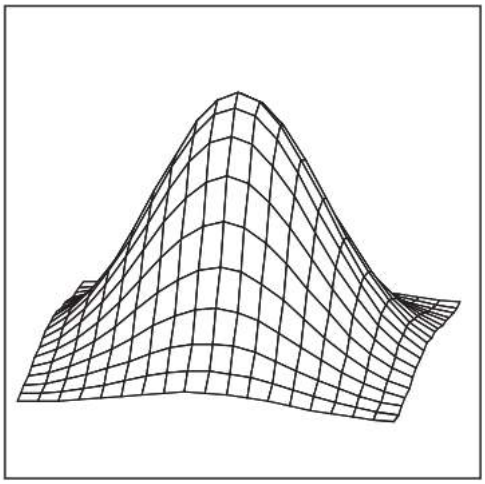


Figura 2.4: Aspecto de curva gaussiana 3-D aludido por um *kernel* gaussiano [3].

O operador gaussiano consegue remover ruídos em uma imagem que estiverem em um raio de 3σ, considerando como centro o píxel central do *kernel* [3]. Desse modo, a variância está diretamente relacionada com o tamanho matricial do filtro. Quanto maior este o for, maior será sua variância, e menor será a conservação de *features*, como mostra a Figura 2.5.

(a) *Kernel* 3×3. (b) *Kernel* 5×5. (c) *Kernel* 7×7.

Figura 2.5: Deterioração de *features* em função do tamanho do filtro gaussiano [3].

2.1.3.3 Filtro Bilateral

O filtro bilateral é derivado da gaussiana de suavização, contudo, previne o borramento de bordas de acontecer ao diminuir os pesos da filtragem quando a diferença de intensidade é muito expressiva [3]. Portanto, tal filtro reduz os ruídos na imagem, preservando os contornos das *features* praticamente intactos, geralmente. Sua fórmula é regida pela Equação 2.5:

$$\boldsymbol{J_s} = \frac{1}{k(\boldsymbol{s})} \sum_{c \in \Omega} f(\boldsymbol{c} - \boldsymbol{s}) g(\boldsymbol{P_c} - \boldsymbol{P_s}) \boldsymbol{P_c} \tag{2.5}$$

em que $\boldsymbol{J_s}$ se refere ao resultado da filtragem para o píxel s da nova imagem J, $k(\boldsymbol{s})$ é usado para fazer uma normalização, Ω é a janela de interesse, $\boldsymbol{P}$ são dados da imagem de entrada, f é a gaussiana de suavização no espaço e g é a gaussiana aplicada na diferença de intensidade [3]. A Figura 2.5 exibe o efeito de remoção de textura e conservação de contornos.

(a) Imagem original. (b) Depois da aplicação do filtro.

Figura 2.6: Potencial do filtro bilateral para remoção de textura e conservação de contornos [4].

2.1.4 Filtros Passa-Altas

Filtros passa-altas, como o próprio nome já indica, deixam passar apenas os sinais de alta frequência, produzindo assim um realce nas bordas da imagem. A seguir, são apresentados dois filtros (Sobel e laplaciano) que o executa, assim como um método de detecção de bordas, com base em Sobel, chamado Canny.

2.1.4.1 Filtro de Sobel

O filtro de Sobel é usado para detectar bordas. Antigamente, quando ainda não existiam técnicas de detecção de bordas fundadas com uma base teórica, foi o mais popular detector de bordas, por se sobressair entre seus contemporâneos [3]. Sua diferença está no fato de possuir duas componentes *kernels*, sendo uma para o eixo x e outra para o eixo y, que são normalmente matrizes quadradas de ordem 3 — embora haja como construir maiores através de algoritmos. A Figura 2.7 mostra como os operadores x e y do filtro de Sobel.

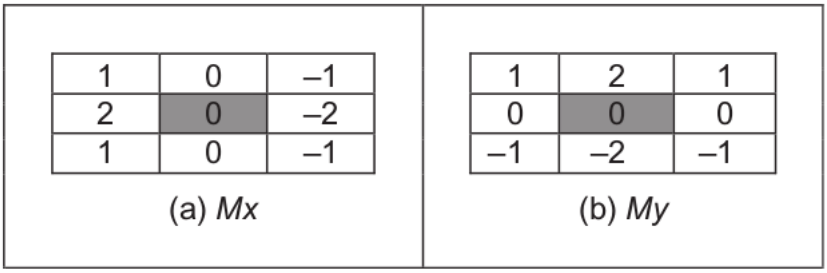


Figura 2.7: Componentes x e y do filtro de Sobel [3].

Observando o arranjo de elementos das matrizes M_x e M_y, constata-se que a segunda é simplesmente a transposta da primeira.

Considerando-se que existem uma matriz imagem I a ser processada e que S_x e S_y são resultados da convolução de M_x e M_y I, respectivamente, têm-se as Equações 2.6 e 2.7:

$$S_x = I * M_x \tag{2.6}$$

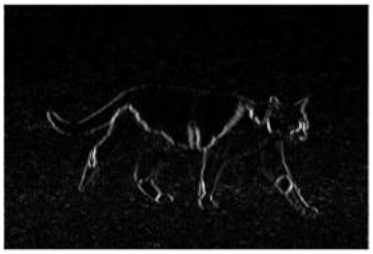
$$S_y = I * M_y \tag{2.7}$$

Daí, juntam-se essas duas componentes para formar a Equação 2.8, cujo resultado da combinação produz um vetor gradiente no píxel p. Tal vetor aponta para o maior crescimento de brilho da imagem I em volta do píxel p [5].

$$\overrightarrow{S}(p) = S_x(p)\overrightarrow{i} + S_y(p)\overrightarrow{j} = I * M \tag{2.8}$$

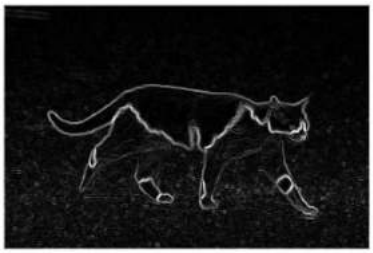
A Figura 2.8 detalha o efeito de cada uma das componentes M_x e M_y, bem como o efeito delas em conjunto.

(a) Imagem original.



(b) Sobel em x.

(c) Sobel em y.



(d) Sobel.

Figura 2.8: Realce de contornos por meio do operador de Sobel [5].

É importante salientar que M_x realça as bordas na direção x, afetando assim as bordas verticais, como é possível visualizar na Subfigura 2.8(b). Da mesma forma, M_y torna as bordas horizontais mais visíveis, como na Subfigura 2.8(c).

2.1.4.2 Filtro Laplaciano

O filtro Laplaciano é usado para implementar uma derivada de segunda ordem em imagens. Como se deve saber, uma derivada de segunda ordem avaliada em um píxel x pode ser expressa da forma que se encontra na Equação 2.9:

$$f''(x) \cong f'(x) - f'(x+1) \tag{2.9}$$

Por Séries de Taylor, todavia, sabe-se que a derivada de primeira ordem em torno do píxel x é dada pela Equação 2.10:

$$f'(x) \cong \frac{f(x+\Delta x) - f(x)}{\Delta x} - O(\Delta x) \tag{2.10}$$

em que $O(\Delta x)$ é o erro associado a operação sendo realizada no sistema, que é significativo somente para distâncias relativamente grandes [3], e Δx é a distância do píxel que está sendo avaliado até outro mais próximo em unidade de píxel, que no caso é 1, por se tratar de vizinhos imediatos. Assim sendo, desconsidera-se a existência de $O(\Delta x)$, por causa do erro associado ser ínfimo. Dessa maneira, a Equação 2.10 fica sendo equivalente a Equação 2.11:

$$f'(x) = f(x+1) - f(x) \tag{2.11}$$

Substituindo, então, a Equação 2.11 na Equação 2.9, chega-se à Equação 2.12:

$$f''(x) \cong f(x+1) - f(x) - f(x+2) + f(x+1) = -1 \cdot f(x) + 2 \cdot f(x+1) - 1 \cdot f(x+2) \quad (2.12)$$

Com isso, uma parte do *kernel* é obtida, a qual é a derivada de segunda ordem horizontal. Para encontrar o restante do *kernel*, combinam-se tal parte com a derivada de segunda ordem vertical, unindo e somando seus respectivos píxeis centrais. Finalmente, encontra-se o *kernel* do filtro laplaciano como é apontado pela Figura 2.9:

0	−1	0
−1	4	−1
0	−1	0

Figura 2.9: *Kernel* laplaciano gerado por meio da composição das derivadas de segunda ordem em x e y [3].

Outros *kernels* de filtro laplaciano são apresentados pelas matrizes K_{lapl_1} e K_{lapl_2} [5]:

$$K_{lapl_1} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$K_{lapl_2} = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

No caso do kernel da Figura 2.9, percebe-se que é feita a diferença entre um píxel e a média aritmética de seus quatro vizinhos imediatos. Já em K_{lapl_1}, é realizada a diferença entre um píxel e a média de seus oito vizinhos imediatos.

Para qualquer um dos *kernels* de operador laplaciano vistos até aqui, ressalta-se a inversão de sinal de seus elementos não afeta em nada na funcionalidade do filtro. O que importa, na verdade, é que a soma dos coeficientes seja igual a zero, para que bordas não sejam detectadas em regiões de brilho uniforme [3].

Uma aplicação da filtragem com o operador laplaciano é aludida pela Figura 2.10. A detecção de bordas é realizada a matriz imagem da Subfigura 2.10(a) e se revela na Subfigura 2.10(b) como sendo os zeros que apareceram nas bordas da matriz L.

2.1.4.3 Canny

Canny é um método de detecção de bordas bastante popular. A fim de entender como ele funciona, por ser um algoritmo multifase, é interessante seguir alguns passos determinados pelo seu criador, John Canny, os quais são enumerados abaixo [6]:

$$p = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 1 & 2 & 1 \\ 2 & 2 & 3 & 0 & 1 & 2 & 2 & 1 \\ 3 & 0 & 38 & 39 & 37 & 36 & 3 & 0 \\ 4 & 1 & 40 & 44 & 41 & 42 & 2 & 1 \\ 1 & 2 & 43 & 44 & 40 & 39 & 1 & 3 \\ 2 & 0 & 39 & 41 & 42 & 40 & 2 & 0 \\ 1 & 2 & 0 & 2 & 2 & 3 & 1 & 1 \\ 0 & 2 & 1 & 3 & 1 & 0 & 4 & 2 \end{bmatrix} \qquad L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -31 & -47 & -36 & -32 & 0 & 0 \\ 0 & -44 & 70 & 37 & 31 & 60 & -28 & 0 \\ 0 & -42 & 34 & 12 & 1 & 50 & -39 & 0 \\ 0 & -37 & 47 & 8 & -6 & 33 & -42 & 0 \\ 0 & -45 & 72 & 37 & 45 & 74 & -34 & 0 \\ 0 & 5 & -44 & -38 & -40 & -31 & -6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) Matriz imagem original. (b) Depois da aplicação do operador laplaciano.

Figura 2.10: Efeito da filtragem com o operador laplaciano [3].

1^o passo: Reduzir ruídos

Uma vez que ruídos podem atrapalhar a detecção de bordas, um operador gaussiano 5×5 é utilizado para removê-los.

2^o passo: Encontrar gradientes de intensidade da imagem

Implementa-se o operador Sobel tanto na direção horizontal quanto na vertical para obter os gradientes respectivos G_x e G_y, gerando duas imagens filtradas. Destas duas, encontra-se a magnitude e o ângulo do gradiente de cada píxel por meio das Equações 2.13 e 2.14:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{2.13}$$

$$\theta = \tan^{-1}\left(\frac{G_y^2}{G_x^2}\right) \tag{2.14}$$

Na Equação 2.14, o gradiente, que é sempre perpendicular às bordas, tem seu ângulo arredondado para um dos quatro ângulos que representam a vertical, a horizontal, e as duas diagonais.

3^o passo: Suprimir pontos de não-máximos

É feita uma varredura por ao longo das bordas para checar se não há píxeis enganosos, que eventualmente não constituem bordas. Para tal, o máximo local é analisado para cada píxel, tendo em vista sua adjacência na direção do gradiente do conjunto. A Figura 2.11 exemplifica isso. Nela, compara-se o píxel A, que está em uma borda, com os píxeis C e B, pelo fato de todos estarem na direção horizontal do gradiente. Como A é um máximo local na adjacência que compõe uma borda, o valor de sua magnitude permanece o mesmo. Caso contrário, a magnitude zero seria atribuída ao seu valor de intensidade no momento da varredura.

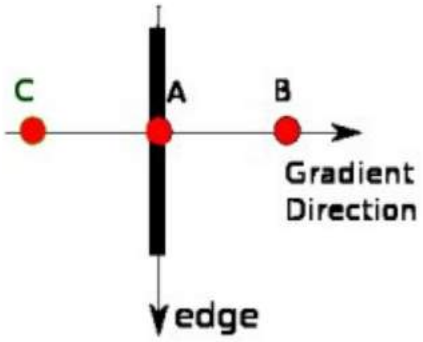


Figura 2.11: Análise de píxeis para descobrir o máximo local [6].

4^o passo: Limiarizar a histerese

Define-se aqui quais píxeis são bordas de verdade e quais não são. Para isso, escolhem valores para dois limiares: um máximo e outro mínimo. Todo píxel cuja intensidade está acima do limiar máximo é considerado uma borda-verdadeira. Em contrapartida, todo aquele cuja intensidade se encontra abaixo do limiar mínimo é definitivamente não-borda. Mas aqueles que estão entre os dois limiares são classificados de acordo com a sua conectividade mútua. Nesse caso, se um deles está ligado a um píxel borda-verdadeira, este passa a ser considerado parte da borda. Caso contrário, ele é descartado. Isso fica claro na Figura 2.12:

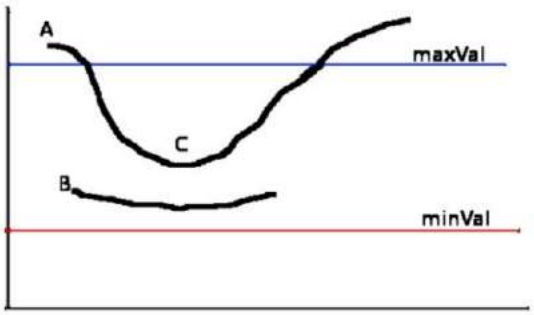


Figura 2.12: Limiares de máximo e mínimo que definem o que é borda de verdade [6].

Na Figura 2.12, A é um píxel considerado borda-verdadeira, por estar acima do limiar máximo. Com isso, C torna-se um píxel componente da borda, já que está conectado a *feature* da qual A faz parte. No entanto, o píxel B não é considerado integrante de nenhuma borda. Sua magnitude, então, é alterada para zero.

Tendo em vista os quatro passos apresentados, a aplicação do filtro de Canny produz a detecção de bordas mostrada na Figura 2.13:

2.1.5 Operações Morfológicas

As operações morfológicas são não-lineares, e servem como ferramenta para extrair componentes de uma imagem (estrutura e forma) que são úteis para descrição e representação, podendo

(a) Imagem original. (b) Depois da aplicação do filtro.

Figura 2.13: Efeito da implementação do filtro de Canny [3].

ser utilizada a priori ou a posteriori em um processamento de imagens *[29]*. Na sequência, serão vistas as operações de dilatação, erosão, abertura e fechamento.

2.1.5.1 Dilatação

A operação de dilatação consiste basicamente de um *kernel* K, circular ou quadrado na maioria das vezes, que desliza sobre uma imagem I. Tal *kernel*, denominado elemento estruturante na ocasião, sobrepõe a imagem I. Os píxeis que ficam empareados nessa sobreposição são então computados e o máximo valor entre eles é posto no lugar do píxel da imagem I que faz par com o píxel central. Matematicamente, a fórmula da dilatação é expressa pela Equação 2.15:

$$I' = I \oplus K \tag{2.15}$$

em que I' é a imagem resultante.

Outra maneira de expressar a operação dilatação em uma imagem I é indicada na Equação 2.16 [8]:

$$I' = \underset{\forall q \in \mathcal{A}(p)}{max} \{I(p) + K(q-p)\} \tag{2.16}$$

em que q e p representam os píxeis do elemento estruturante K e da imagem I — o primeiro tem que estritamente fazer parte do conjunto de píxeis adjacentes $\mathcal{A}$ do segundo.

É comum que os valores de $K(q-p)$ sejam nulos. Nesse caso, elemento estruturante é denominado planar.

Para exemplificar o efeito da dilatação em uma imagem, recorre-se a Figura 2.14. Nela, ao convoluir um elemento estruturante de tamanho 3×3 na imagem, observa-se que basicamente os píxeis de alta intensidade, representados pela cor branca, dilatam, expandem. Ou seja, a letra da imagem "engorda". Mas deve-se atentar ao fato de que, se as cores das Subfiguras 2.14(a) e 2.14(b) estivessem invertidas, a letra j emagreceria. Por isso, é importante salientar que somente os valores de intensidade claros que se expandem na operação de dilatação.

2.1.5.2 Erosão

A operação de erosão é bem semelhante à operação de dilatação. A única diferença é que na sobreposição do *kernel*, ao se analisar os píxeis adjacentes, escolhe-se aquele de menor intensidade para substituir o píxel da imagem resultante, o qual fica rente ao píxel central do elemento estruturante. Em virtude disso, ambas as fórmulas que regem a operação de erosão sofrem uma leve alteração, como é visto nas Equações 2.17 e 2.18 [8]:

$$I' = I \ominus K \tag{2.17}$$

$$I' = \min_{\forall q \in \mathcal{A}(p)} \{I(p) - K(q - p)\} \tag{2.18}$$

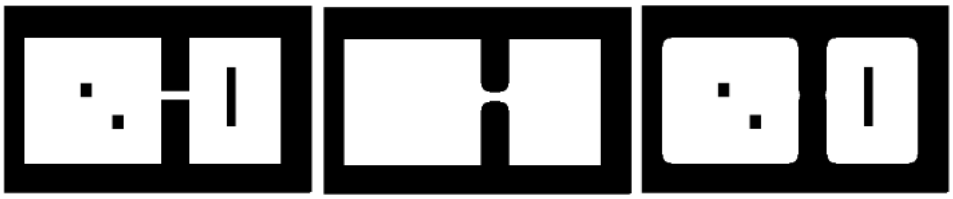
Como se sabe que agora o valor mínimo de intensidade que é computado, o operador erosão produz um efeito em uma imagem reverso ao operador dilatação, o que é exibido pela Figura 2.14. Nela, confere-se que as regiões claras de uma imagem se afinam depois da convolução feita com um elemento estruturante de tamanho 3×3.

(a) Imagem original. (b) Imagem dilatada. (c) Imagem erodida.

Figura 2.14: Efeito dos filtros de dilatação e erosão com elemento estruturante 3×3 planar em uma imagem binária [7].

2.1.5.3 Fechamento e Abertura

Os filtros de fechamento e abertura tem o objetivo de reduzir a degradação de imagens causada pelos operadores de dilatação e de erosão [5]. O fechamento consiste na operação de dilatação seguida de uma erosão, como indicado pela Equação 2.19. É voltado para a remoção de buracos, os quais são considerados como regiões escuras em uma imagem, como na Subfigura 2.15(b). Já a abertura é dada por uma convolução de erosão seguida de uma dilatação, o que é mostrado na Equação 2.20. Com esse filtro, visa-se remover pequenos objetos (regiões) claros em um fundo preto, ilustrado pela Subfigura 2.15(c). Ressalta-se que o elemento estruturante utilizado para demonstrar o efeito de cada filtro foi um disco de raio 10, planar, e que os conceitos sobre as operações morfológicas vistos valem tanto para imagens binárias quanto para aquelas em tons de cinza.



(a) Imagem original. (b) Imagem fechada. (c) Imagem aberta.

Figura 2.15: Ação de um disco (elemento planar) de raio=10 para fechamento e abertura [8].

$$I' = (I \oplus K) \ominus K \tag{2.19}$$

$$I' = (I \ominus K) \oplus K \tag{2.20}$$

2.1.6 Limiarização

A limiarização é geralmente usada para separar objetos de seus panos de fundo em uma imagem. É feita através de um limiar de tonalidade que, quando aplicado, faz com que tons mais escuros que sua cor sejam preenchidos com zero (preto) e tons mais claros, com um valor da outra extremidade do espectro que represente o branco.

Há dois tipos de limiarização bastante usuais, conhecidos como limiarização simples e limiarização adaptativa. O primeiro, como o próprio nome já indica, é realizado por meio aplicação direta do limiar de interesse, denominado limiar ótimo, na imagem. Sua ação pode ser expressa pela Equação 2.21:

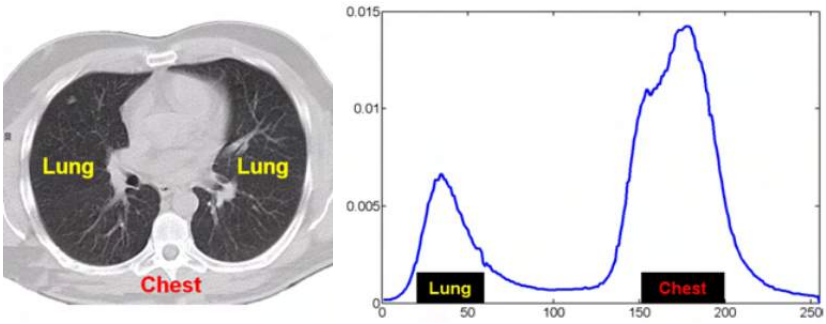
$$\begin{cases} I(x,y) = 0, \ I(x,y) \leq \theta \\ I(x,y) = L-1, \ I(x,y) > \theta \end{cases} \tag{2.21}$$

Para encontrar o limiar ótimo, é importante saber que objetos e panos de fundo têm picos, ou moda 1-D de sinal, diferentes entre si [9], como é visto na Figura 2.16:

Por inspeção, pode-se deduzir que, no caso da imagem da caixa torácica apresentada na Subfigura 2.16(b), o limiar ótimo θ na Subfigura 2.15(c) pode ser 100. Assim sendo, separam-se os pulmões (objeto) do peito (pano de fundo).

No caso da limiarização adaptativa, no entanto, a escolha da posição do limiar segue uma metodologia distinta. Passa a ser iterativa e procura lidar melhor com erros de separação, evitando atribuir os píxeis de objeto aos de pano de fundo e vice-versa.

Para entender como a limiarização adaptativa funciona, considere que a cada iteração j, o filtro executa os seguintes passos [9]:



(a) Caixa torácica em tons de cinza.

(b) Histograma da caixa torácica.

Figura 2.16: Picos característicos de objeto e pano de fundo [9].

1^o passo: Classificar cada tom de cinza

Faz-se uma espécie de limiarização simples, de tal forma a separar objeto do pano de fundo em duas classes. Píxeis com valores abaixo de um determinado limiar vão para classe pano de fundo e os acima, para a classe objeto, como mostra a Equação 2.22:

$$I(x,y) \in \begin{cases} C_{pf:[j]} \; se \; I(x,y) \leq \theta_j \\ C_{ob:[j]} \; se \; I(x,y) > \theta_j \end{cases} \tag{2.22}$$

em que $I(x,y)$ diz respeito a píxel da linha x e coluna y pertencente a imagem I, $C_{pf:[j]}$ e $C_{ob:[j]}$ são classes pano de fundo e objeto na iteração j e θ_j é o limiar θ na iteração j.

2^o passo: Computar as médias de tons de cinza

Computa-se a média de tons de cinza para cada uma das classes, conforme as Equações 2.23 e 2.24:

$$\mu_{pf:[j]} = \frac{1}{|C_{pf:[j]}|} \sum_{(x,y) \in C_{pf:[j]}} I(x,y) \tag{2.23}$$

$$\mu_{ob:[j]} = \frac{1}{|C_{ob:[j]}|} \sum_{(x,y) \in C_{ob:[j]}} I(x,y) \tag{2.24}$$

em que $|C|$ é a quantidade de píxeis na região C.

3^o passo: Calcular o novo limiar

Calcula-se, então, o novo limiar, que é dado pela Equação 2.25:

$$\theta_{j+1} = \frac{1}{2}\left(\mu_{pf:[j]} + \mu_{ob:[j]}\right) \tag{2.25}$$

A partir da metodologia usada na aplicação da limiarização adaptativa, percebe-se que também se visa a procurar o limiar ótimo, o qual tende a ficar no centro geométrico entre o pico do objeto e do pano de fundo.

2.1.7 Transformada de Hough

A transformada de Hough foi originalmente criada para encontrar retas em imagens binárias, de forma relativamente rápida. Atualmente, seu algoritmo foi ampliado para encontrar outros padrões geométricos tais como círculos e outros formatos simples [12]. É sugerido, contudo, que se aplique um filtro de detecção de bordas primeiramente [10] — o que significa que a imagem não precisa estar necessariamente binarizada.

2.1.7.1 Transformada de Hough para Retas

A implementação da transformada de Hough para detectar retas em uma imagem é de fácil entendimento. Inicialmente, deve-se considerar um ponto em um plano cartesiano, o qual possui um coeficiente angular a e um coeficiente linear b, localizado bem na interseção da semirreta azul com a reta vermelha na Figura 2.17.s

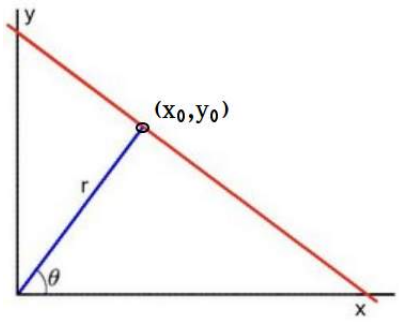


Figura 2.17: Localização do par ordenado (x_0, y_0) [10].

Fazendo uma rápida análise trigonométrica, tem-se as Equações 2.26, 2.27 e 2.28:

$$x = \frac{r}{\cos\theta} \tag{2.26}$$

$$y = b = \frac{r}{\sin\theta} \tag{2.27}$$

$$a = -\tan\theta = -\frac{x}{y} = -\frac{\frac{r}{cos\theta}}{\frac{r}{sin\theta}} = -\frac{\sin\theta}{\cos\theta} \tag{2.28}$$

Com isso, substituem-se a e b na equação geral da reta, obtendo a Equação 2.29:

$$y = \left(-\frac{\sin\theta}{\cos\theta}\right)x + \left(\frac{r}{\sin\theta}\right) \tag{2.29}$$

Rearranjando a Equação 2.29, tem-se a Equação 2.30, que representa a posição de um ponto (x, y) qualquer em termos de coordenadas polares (r, θ), na qual o ponto genérico (x, y) foi substituído pelo ponto específico (x_0, y_0):

$$r(\theta) = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta \tag{2.30}$$

Fixando os parâmetros $x_0 = 8$ e $y_0 = 6$, e variando o parâmetro ângulo θ, encontram-se vários valores de raio r, formando assim vários pares ordenados (r, θ). Tais pares permitem fazer a plotagem do gráfico na Figura 2.18, considerando apenas valores que seguem as seguintes restrições: $r > 0$ e $0 < \theta < 2\pi$ [10].

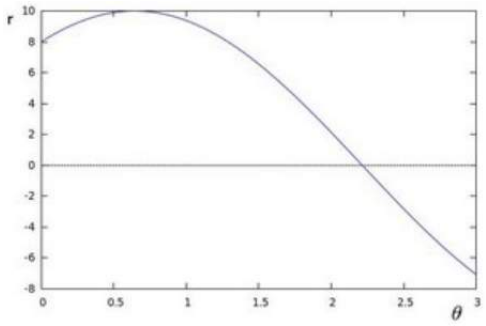


Figura 2.18: Curva formada ao fixar o ponto $(8, 6)$ e variar θ [10].

Se se fixam outros valores para x e y, tais como os pontos $(4, 9)$ e $(12, 3)$, descobrem-se outros pares ordenados (r, θ), os quais levam à plotagem feita na Figura 2.19.

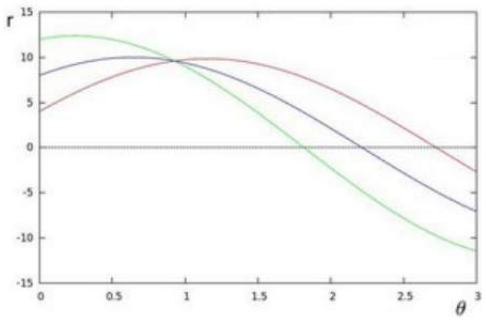


Figura 2.19: Curva formada ao fixar os pontos $(4, 9)$ e $(12, 3)$, além do $(8, 6)$, e variar θ [10].

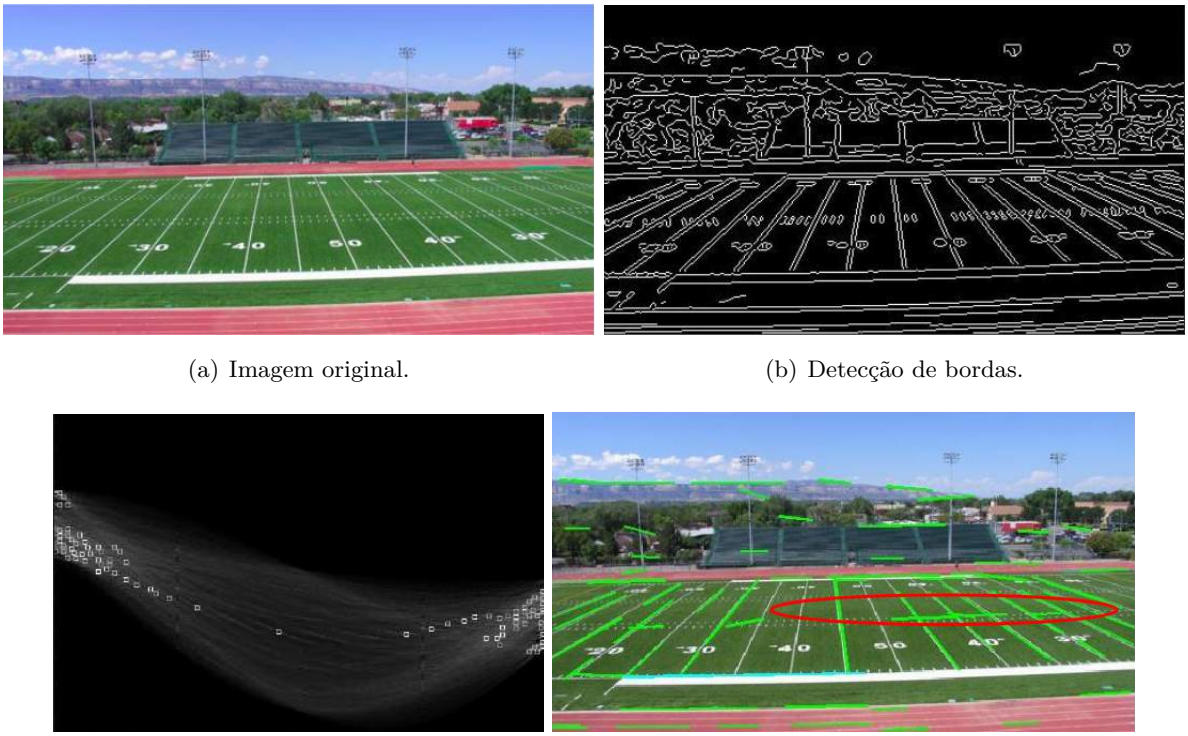
Daí, nota-se que as curvas plotadas possuem um ponto de interseção, que é definido pela coordenada polar $(0,925; 9, 6)$. Nesse contexto, quanto maior for a quantidade de curvas que cruzarem o ponto mencionado, maior será a chance de aquela sequência de pontos em coordenadas cartesianas formar ou ser uma reta.

Olhando de uma perspectiva de processamento de imagens, o plano cartesiano é a imagem e o os pontos de coordenadas x e y são os píxeis. Analogamente, quanto maior for a quantidade de curvas senoidais, características de cada píxel da imagem em análise, cruzando-se em um ponto em comum, maior a probabilidade dos píxeis correspondentes àquela interseção serem partes de uma mesma reta.

Para definir de vez quantos pontos são necessários para considerar aquela interseção como sendo correspondente a uma reta, há de se estabelecer um limiar — conhecido também por quantidade de votos. Isto é, ao estabelecer a quantidade mínima de votos necessários, píxeis que produzem curvas senoidais que se cruzam entre si recebem votos. Então, o conjunto de píxeis daquela determinada interseção tem que alcançar ou ultrapassar o número mínimo de votos preestabelecido para que seus píxeis sejam considerados uma reta como um todo.

Portanto, a aplicação da transformada de Hough para linhas retorna pares ordenados do tipo (r, θ), os quais são as interseções de curvas com quantidades suficientes de votos.

Como exemplo de aplicação, tem-se a Figura 2.20. Nela, a imagem inicial colorida é filtrada (Subfigura 2.20(a)), passando inclusive por um operador de detecção de borda (Subfigura 2.20(b)). Depois a transformada de Hough é aplicada, detectando-se as interseções das senoides (Subfigura 2.20(c)). E, por fim, as retas reveladas por meio de tal detecção são plotadas na imagem original (Subfigura 2.20(d)).



(a) Imagem original.

(b) Detecção de bordas.

(c) Transformada de Hough.

(d) Detecção de retas.

Figura 2.20: Aplicação prática da transformada de Hough para retas [11].

2.1.7.2 Transformada de Hough para Circunferências

A transformada de Hough para Círculos, como o próprio nome sugere, é utilizada para encontrar círculos em imagens. Seu princípio de funcionamento é levemente diferente daquele da transformada de Hough para retas explanado anteriormente. Isso porque, há mais uma dimensão a se considerar, que é o raio r do círculo, tendo em vista a equação da circunferência disposta na

Equação 2.31:

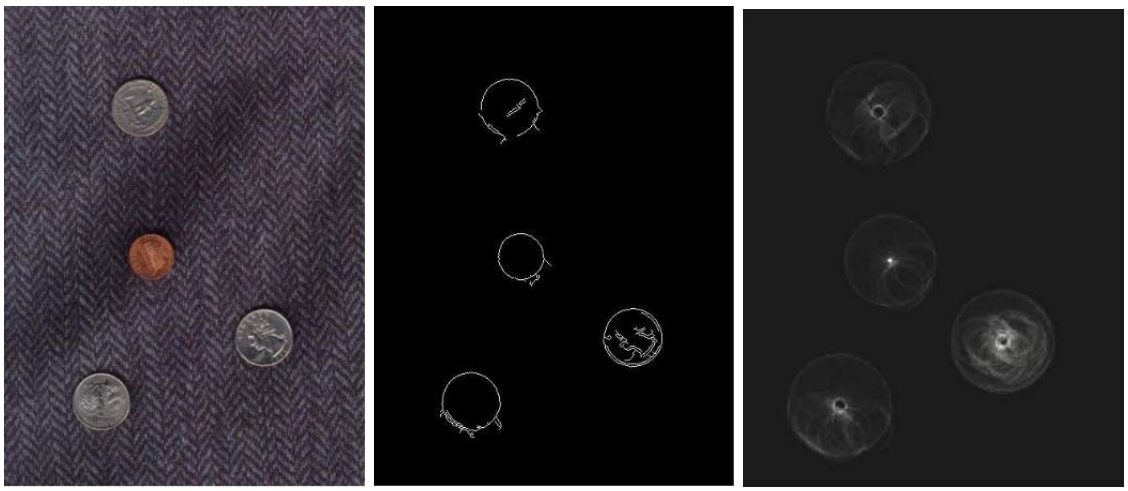
$$r^2 = x_c^2 + y_c^2 \tag{2.31}$$

em que x_c e y_c representação as coordenadas cartesianas da posição do píxel central da circunferência de raio r.

O fato de haver uma dimensão a mais a ser computada implica a necessidade de uma expansão de dimensão para detectar as interseções, passando o acumulador de votos de uma área para um volume. Isso aumentaria a ordem de complexidade do algoritmo.

Para solucionar tal gargalo, usa-se o Método do Gradiente de Hough. Esse método consiste em filtrar a imagem com Canny, fazendo a detecção de bordas, e depois com Sobel, para determinar o vetor gradiente de cada píxel não-zero. Ao longo da reta indicada pela direção e sentido do gradiente de cada píxel, faz-se uma varredura e incrementa-se um acumulador de votos à medida que se computam os píxeis candidatos a fazer parte daquela determinada semirreta — pois distâncias mínimas e máximas são especificadas para tal. Ao passo que isso acontece, a posição de cada píxel é armazenada. Na sequência, os píxeis candidatos a centro são selecionados com base em uma quantidade de votos mínima estipulada e na quantidade de votos para centro de píxeis em sua adjacência. Tais píxeis candidatos a centro são ordenados em ordem decrescente, de modo que aqueles mais votados apareçam primeiro. Depois disso, pega-se o píxel candidato com maior número de votos e analisa-se todos os píxeis não-zeros antes computados, da menor distância até a maior especificadas, procurando um padrão geométrico circular que vá de acordo tanto com as condições de contorno do problema quanto com a Equação 2.31. O píxel é dito centro de uma circunferência se ele tiver votos suficientes e se estiver dentro da distância mínima dos outros centros selecionados [12].

Um exemplo de como se dá a transformada de Hough para Círculos é apresentado na Figura 2.21. Por meio dela, consegue-se ter uma ideia de como a votação para centros ocorre.



(a) Imagem original. (b) Detecção de bordas. (c) Votação para centros.

Figura 2.21: Efeito do operador erosão em uma imagem [11].

2.2 Aprendizado de Máquina

O objetivo do aprendizado de máquina é transformar dados em informação [12]. Com o auxílio de uma base de dados enorme o suficiente, normalmente na casa dos milhares, consegue-se conduzir a aprendizagem, que por sua vez foca na descrição — por intensidades, gradientes, distâncias, entre outros — e separação de features dos diferentes padrões existentes entre os dados em análise. De acordo com os parâmetros descritivos, dados previamente organizados em classes passam a ser expressos em forma de vetores unidimensionais que abrigam numericamente as características estudadas das *features*. São criados um pequeno conjunto de testes e um largo conjunto de treinamento, contendo cerca de 10% e 90% dos dados (ou 20% e 80%), respectivamente — dependendo do classificador, pode haver também dados separados para a validação, como é o caso do Multilayer Perceptron. Ambos são passados para um algoritmo de classificação, o qual, de forma cega, tenta predizer os dados de teste com base nos de treinamento. Como, secretamente, as correspondências são conhecidas, confrontam-se os resultados do classificador com o que se sabe, e probabilidades de acerto são criadas. Se é obtido êxito na aprendizagem de máquina, encontrando altíssimas chances de o ser, abre-se uma nova via de comunicação. Nela, o computador é potencialmente capaz de, por si próprio, responder a determinadas perguntas a respeito de dados semelhantes aos quais ele foi treinado.

Esta Seção é destinada a reproduzir alguns conceitos intrinsecamente atrelados ao âmbito do aprendizado de máquina. Aqui será visto um método vastamente empregado na caracterização de formatos e aparências de objetos, bem como métodos supervisionados (pois as amostras de entrada são conhecidas) para classificação de padrões que, a propósito, usam-no. Além disso, será apresentada uma aplicação específica de aprendizado de máquina.

2.2.1 Histograma de Gradientes Orientados (HOG)

O histograma de gradientes orientados é utilizado para caracterizar aparência e formato de objetos. Seu objetivo é capturar as bordas, isto é, a estrutura de gradientes, que caracterizam formatos locais, de tal forma a tornar os padrões de cunho próprio, criados para o aprendizado de máquinas, relativamente imunes a pequenas variações de aparência ou formatos [3].

Para implementá-lo, inicialmente faz-se a detecção de bordas em uma imagem utilizando um filtro de primeira ordem aprimorado, por exemplo, como mostrado na Figura 2.22. De posse das bordas, aplica-se uma divisão quadricular com o propósito de obter vários quadrados ao longo da imagem, como se houvesse uma grade a sobrepondo 2.22(c). Os quadrados gerados são chamados células, as quais são 8×8 no exemplo. Dentro de cada uma destas, após se ter feito uma conversão de magnitude, direção e sentido dos gradientes para coordenadas polares, os píxeis são separados por caixas, em que cada uma é responsável por armazenar píxeis com ângulos entre uma determinada faixa de valores. Dependendo da tipagem numérica dos gradientes, pode-se ter ângulos de 0^o a 180^o se sem sinal, e de 0^o a 360^o se com sinal. E a quantidade de caixas é basicamente a quantidade de intervalos que se deseja, até completar 180^o ou 360^o — no exemplo são nove caixas com 20^o de intervalo cada. Assim, os píxeis são armazenados de acordo com seu

ângulo na direção horizontal do histograma. Suas magnitudes, por sua vez, os acompanham, sendo convertidas proporcionalmente em uma quantidade de votos em cada caixa e contribuindo para com o crescimento vertical de suas respectivas barras no histograma. Na sequência, com intuito de deixar o algoritmo imune a leves variações de magnitudes de gradientes devido a mudanças na iluminação e contraste com pano de fundo, aplica-se uma normalização local. Esta é normalmente aplicada em blocos de células, os quais são feitos de 3x3 (9 células) 2.22(d).

(a) Imagem original.

(b) Magnitude de gradiente.

(c) Grade com células 8x8.

(d) Agrupamento para normalização de contraste.

Figura 2.22: Aplicação do histograma de gradientes orientados [3].

2.2.2 Métodos para Classificação de Padrões

O objetivo final dos métodos de classificação de padrões é obter uma etiqueta de classe correta para um dado objeto, tendo como base um conhecimento prévio sobre o conjunto de características de features do grupo de objetos possíveis [30]. No âmbito do aprendizado de máquinas, existem dezenas de métodos de classificação de padrões. Tais métodos podem executar o reconhecimento com base em diferentes técnicas, assim como podem se basear em diferentes atributos, propriedades e peculiaridades de objetos quaisquer. A seguir, no entanto, serão apresentados de forma concisa os métodos com mais potencialidade de uso, por haver bibliotecas implementadas em OpenCV, bem como no Scikit-Learn.

2.2.2.1 k-Nearest Neighbor (kNN)

O kNN está entre as mais simples técnicas de classificação. Nela, todos os dados que compõem o conjunto usado para treinamento são armazenados. Novos pontos recebem um rótulo de classificação que é correspondente a quão próximos estão daquela classe cujo rótulo leva o nome. Isto é, quando kNN é chamado para classificar um novo ponto, ele procura em sua base de dados

pelos k pontos vizinhos mais próximos daquele novo ponto. Feito isso, rotula-o de acordo com a classe do conjunto de treinamento que contém a maioria dos vizinhos mais próximos computados. A técnica pode ser bem eficaz, mas também pode demandar um custo computacional grande e, às vezes, executar devagar, tendo em vista o fato de ter de armazenar todo o conjunto de dados de treinamento [12].

2.2.2.2 MultiLayer Perceptron (MLP)

O MLP é a rede neural que possui um dos melhores índices de performance entres classificadores de padrões. Quando usado, é bastante rápido, pois sua implementação é realizada mediante uma série de produtos internos seguidos de uma simples função não-linear. Em contrapartida, seu treinamento pode ser significativamente longo, por empregar gradientes descendentes através das camadas ocultas que possuem neurônios densamente conectados entre si, para minimizar o erro advindo da propagação.

Em uma rede neural, como já se deve estar ciente, neurônios artificiais, ou simplesmente neurônios, são nódulos interconectados ao longo de toda a rede. Quando as conexões são tais que saem de cada neurônio da camada i e alimentam todos os neurônios da camada $i + 1$, e assim sucessivamente, do início ao fim da rede, tem-se a chamada rede multicamada completamente conectada.

A fim de se entender melhor o funcionamento do MLP, considera-se a seguinte rede multicamada completamente conectada da Figura 2.23.

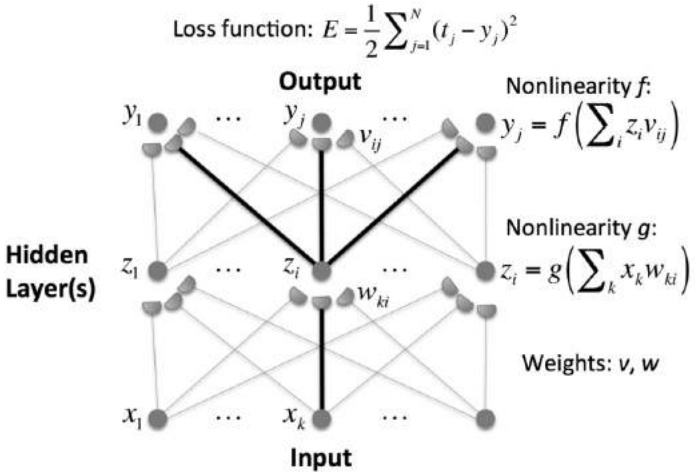


Figura 2.23: *Layout* do MLP [12].

Inicia-se com dados de entrada sendo passados para os neurônios x na camada de entrada. Desta, são enviados a camada de nódulos z, que por sua vez consiste em uma ou mais camadas ocultas da rede. Nos caminhos, ou conexões, a serem percorridos de x para z, são multiplicados por pesos escalares — um peso por conexão. Ao chegar nos neurônios z, os resultados dos produtos são filtrados com uma função não-linear, sendo este um sigmoide ou uma função de ativação retificadora. Os novos valores presentes nos nódulos z continuam se propagando por outras ca-

madas ocultas da rede, seguindo o mesmo princípio, até chegarem a camada de saída, atingindo os neurônios y. Por toda a extensão da rede, usa-se uma função, denominada função de perda, para medir o desempenho da propagação em cada camada. Tal função é expressa pela seguinte Equação 2.32:

$$E = \frac{1}{2}\sum_{k=1}^{N}(t_k - y_k)^2 \tag{2.32}$$

em que N é a quantidade de nódulos em uma camada qualquer k, t é o resultado esperado e y, o encontrado.

Com o objetivo de se obter a melhor classificação possível, alcançando índices de acertos altos, um algoritmo chamado *backpropagation* esforça-se em minimizar a função de perda, o que se dá pela aplicação da regra da cadeia nas funções de ativação — derivando com relação ao peso atribuído às conexões de suas respectivas camadas. Para tal, encontra-se o erro na saída da rede e então o propaga de trás para frente, camada oculta por camada oculta. Nesse processo, cada um dos nódulos de uma camada recebe as informações de erros dos N nódulos de sua camada anterior, combina-as e computa as derivadas necessárias. As informações são passadas assim até que se chegue a camada de entrada [12].

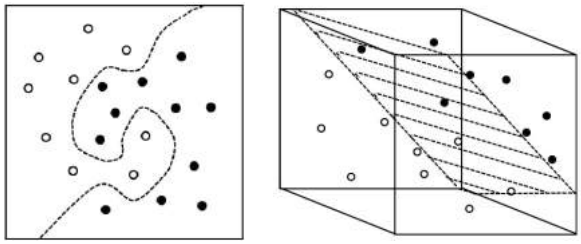
2.2.2.3 Support Vector Machine (SVM)

O SVM é um método de classificação que, em sua forma básica, é utilizado para agrupar duas classes em meio a um conjunto de dados exemplares — embora haja como fazer a separação em multicamadas através de uma extensão do conceito. No método, os pontos são distribuídos em um lugar dimensional chamado *kernel* de número de dimensão igual a dois, por exemplo, e então são mapeados para um lugar dimensional maior — no caso, tridimensional — chamado espaço do *kernel*. Em tal espaço, como é evidenciado na Figura 2.24, é possível determinar um plano classificador linear, conhecido como hiperplano, que separe os pontos em duas classes (Subfigura 2.24(a)), enquanto que no espaço bidimensional necessita-se de um classificador não-linear (Subfigura 2.24(b)). Além do hiperplano, o SVM também gera duas margens paralelas ao classificador, sendo uma de cada lado e tendo a máxima distância possível, como é ilustrado pela Figura 2.25, de modo a esbarrar nos pontos das duas classes mais próximos. Esses pontos de ambas as classes que ficam encostados nas margens são chamados de vetores suportes [12].

Formalmente, um SVM linear é descrito pela função decisão, com base em seu hiperplano, expressa na Equação 2.33:

$$\overrightarrow{w} \cdot \overrightarrow{x} + b = 0 \tag{2.33}$$

em que $\overrightarrow{x}$ representa todos os pontos distribuídos pelo espaço do *kernel*, $\overrightarrow{w}$ define a normal ao hiperplano e b o indica o *offeset* do hiperplano.



(a) *Kernel* 2-D inicial. (b) Espaço de *kernel* 3-D.

Figura 2.24: Expansão de dimensão característica do SVM [12].

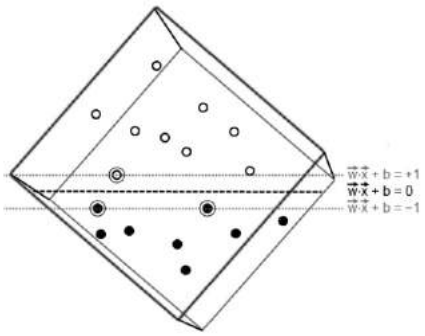


Figura 2.25: Localização de margens e vetores suportes [12].

2.2.3 Reconhecimento Óptico de Caracteres (OCR)

OCR é o processo de classificação óptica de caracteres alfanuméricos, ou de outros tipos de símbolos, contidos em uma imagem digital [13]. Sua técnica é baseada em três passos cruciais que são a segmentação, a extração de *feature* e a classificação. Com esse método que vem ganhando atenção tanto na indústria quanto em pesquisas, portanto, é possível converter vários tipos de documentos tais como documentos escaneados, PDFs, ou fotografias digitais em dados editáveis e passíveis de pesquisa por palavras. Infelizmente, no entanto, suas aplicações ainda não substituem por completo a capacidade humana de leitura por não alcançar desejados índices de acurácia, às vezes.

No âmbito do reconhecimento de padrões, o OCR tem sido empregado amplamente por meio de dois princípios. O primeiro consiste na execução de um reconhecimento óptico online, em que o computador reconhece o caractere ao passo que este está sendo desenhado. Já o segundo é o reconhecimento offline que se dá depois da escrita ou impressão. Para ambas as formas de reconhecimento, a qualidade da imagem a ser processada é um fator bastante pertinente. Quanto maior o for, mais chances se tem de obter resultados de classificação satisfatórios. Outro fator que pode colaborar com o sucesso do reconhecimento de padrões é a restrição de formato dos caracteres. No caso da grafia a mão, por exemplo, pode-se ter resultados questionáveis, às vezes, em virtude de cada pessoa ter seu próprio estilo de letra. A Figura 2.26 faz um resumo esquemático das diferentes áreas de reconhecimento de caracteres.

Este Trabalho de Graduação ficará concentrado na área do canto esquerdo da Figura 2.26, que

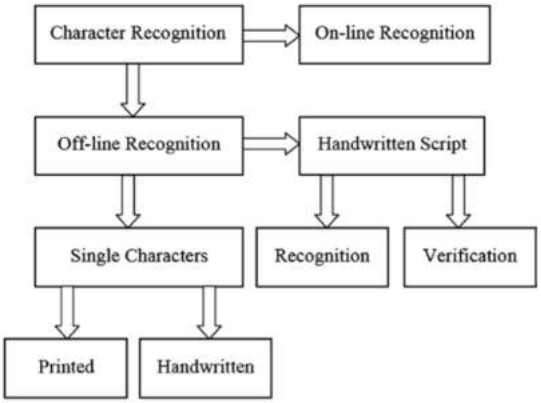


Figura 2.26: Quadro esquemático sobre as diferentes áreas de reconhecimento de padrão [13].

se refere à execução de reconhecimento de caractere de imprensa individual de forma offline. É importante ressaltar ainda que o conjunto de classes será composto por números somente.

2.3 Visão Computacional

A visão computacional é a análise automática de imagens e vídeos por um computador com o propósito de adquirir algum entendimento sobre a informação visual do ambiente em sua volta [17, 31]. Ela se difere do campo de processamento digital de imagens por uma simples razão. Para uma dada imagem 2-D, enquanto o processamento de imagens gera uma imagem 2-D, filtrada e alterada para uma certa finalidade, a resposta na saída de um algoritmo de visão computacional é expressa em dados geométricos 3-D [14]. Estes geralmente são manipulados e analisados pelo sistema de visão de forma a compor um raciocínio artificial lógico para tomada de decisões.

Um algoritmo de visão computacional possui basicamente três etapas [14]. A primeira é a visão em baixo nível, que lida com o pré-processamento de imagem, por meio da aplicação de filtros, realce e restauração. A segunda consiste na visão em nível intermediário, em que é feita a extração de características usando segmentação e descrição de imagem. A última etapa é dada pela visão em alto nível, a qual faz inferências mediante um conhecimento prévio do modelo em questão, o que se traduz em análise da imagem e inteligência artificial. As três etapas descritas são parte da estrutura de visão artificial esquematizada pela Figura 2.27. É importante notar que cada passo, desde a aquisição da imagem até o reconhecimento e interpretação dela, é interligado a uma espécie de módulo interpretador por uma via dupla de comunicação, em que se é possível receber informações e atuar tanto no sistema como um todo quanto individualmente no passo sendo executado.

No que diz respeito ao *hardware* de uma máquina de visão computacional, segue-se um determinado padrão, quase que invariante, para construí-lo. Normalmente, o sistema com visão consiste de [15]:

- 01 aparelho de iluminação que ilumine a amostra sob teste;

- 01 câmera matricial *charge-couple device* (CCD) de estado sólido ou *complementary metal-oxide semiconductor* (CMOS), usada para fazer a aquisição da imagem;
- 01 gravador de quadros, para fazer a conversão analógica digital de linhas escaneadas em píxeis, em uma foto de N linhas e M colunas;
- 01 computador pessoal ou sistema microprocessador, para que se possa gravar imagens em uma memória e para que haja capacidade computacional para manipulá-la por meio de aplicações de *software* específicas; e
- 01 monitor de alta resolução, para ajudar a visualizar imagens e efeitos advindos de técnicas de processamento digital de imagens.

Figura 2.27: Estrutura de um sistema de visão artificial [14].

A configuração supracitada é ilustrada pela Figura 2.28, e pode ser encontrada, por exemplo, em diversos laboratórios em que há pesquisas sendo desenvolvidas com alimentos. Tanto nesse caso como em vários outros, o principal objetivo é livrar inspetores humanos de um trabalho tedioso, repetitivo, que toma bastante tempo, e deixá-los disponíveis para atividades que demandem habilidades humanas mais profissionais. Assim sendo, a visão computacional, além de fornecer alta repetibilidade, flexibilidade e custo baixo, também permite uma taxa de transferência alta em linhas de produção sem prejudicar a acurácia necessária para um sistema [15].

A Figura 2.29 mostra, à borda do círculo maior, vários benefícios que a visão computacional traz — algum deles mencionados anteriormente —, tais como confiabilidade, economia, robustez, velocidade, rentabilidade. Dentro do círculo, na região mais clara, estão algoritmos, conceitos e técnicas nos quais a visão computacional se respalda ou usa como subsídio na resolução dos problemas. Nesse contexto, vale a pena evidenciar que, para desenvolver as soluções, faz-se o uso de transformada de Hough, redes neurais, processamento de imagens, filtragem, detecção de *features*, morfologia e reconhecimento de padrões, os quais foram explanados no decorrer deste capítulo.

Visão computacional é aplicada largamente em indústrias, permitindo a inspeção automática

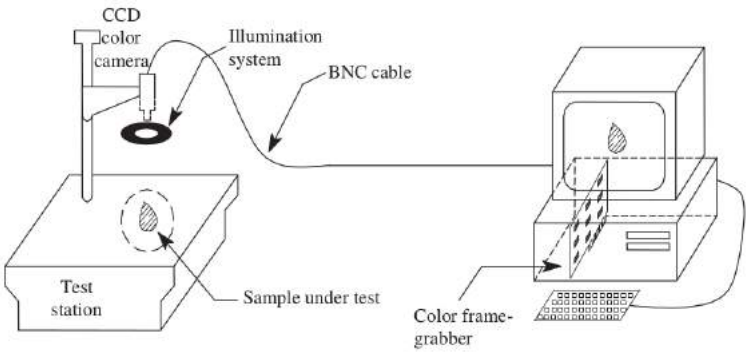


Figura 2.28: Elementos essenciais de um sistema de visão computacional típico [15].

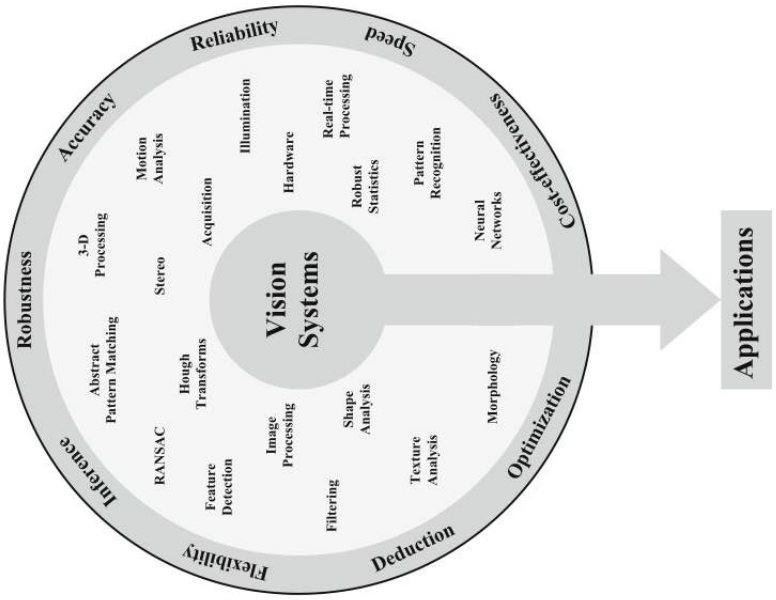


Figura 2.29: Conceitos e benefícios de aplicações de um sistema de visão computacional [16].

de bens de manufatura em qualquer estágio em linhas de produção. Exemplos de aplicações com visão computacional são [17]:

- inspeção de placas de circuito impresso para garantir que componentes e trilhos foram colocados corretamente, na Figura 2.30;
- inspeção da qualidade de rótulos, na Figura 2.31;
- inspeção de garrafas para assegurar que estão apropriadamente cheias, na Figura 2.31;
- inspeção de maçãs para determinar se possuem algum amasso;
- localização de chocolates em linhas de produção para que um braço robótico possa pô-lo de volta nos corretos locais das caixas.

Além disso, visão computacional é utilizada em vários outros segmentos do lado de fora das

fábricas, em que normalmente não há controle de luminosidade, tornando a solução do problema mais complexa e desafiante. Exemplos são [17]:

- leitura automática de placas de veículos ao passarem por portagens em rodovias principais;
- determinação de distâncias para faltas em jogos televisionados, bem como de distância que cada jogador percorre durante as partidas;
- checagem de biométricas para segurança em aeroportos usando imagens de faces e impressões digitais, na Figura 2.32;
- assistência de motoristas ao lhes avisar quando estão à deriva, saído fora da faixa de condução correta;
- detecção de minas terrestres por imagens infravermelhas, na Figura 2.32.

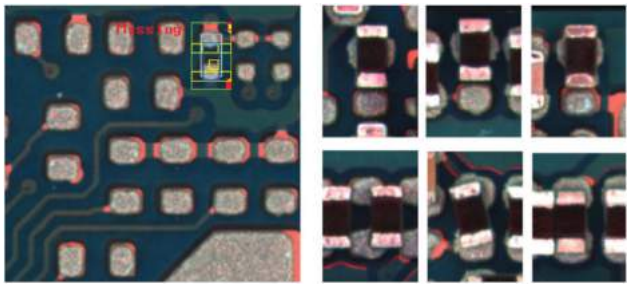


Figura 2.30: Inspeção dos terminais de placas de circuito impresso (esquerda), e imagens de alguns defeitos detectados na superfície de montagem de componentes (direita) [17].

Figura 2.31: Checagem da qualidade de impressão de datas de validade (esquerda), e monitoramento de nível até o qual as garrafas devem ser enchidas (direita) [17].

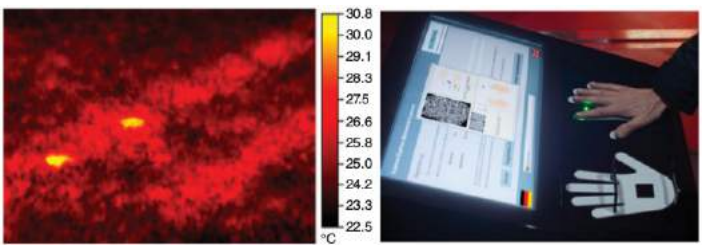


Figura 2.32: Detecção de minas terrestres enterradas (esquerda), e sistema de reconhecimento de impressão de mão (direita) [17].

Capítulo 3

Trabalhos Correlatos

Antes de iniciar a implementação da proposta, foi necessário realizar uma pesquisa sobre soluções disponíveis no mercado e, principalmente, sobre artigos escritos acerca do âmbito de supervisionamento remoto de instrumentos de medição. Encontraram-se vários resultados pertinentes, que de fato proporcionavam tanto técnicas para atacar o problema com maestria como ideias interessantes que poderiam se acoplar ao escopo da resolução. Serão abordados aqui, contudo, trabalhos que parecem ser mais aplicáveis e que apresentam artifícios exequíveis no contexto da proposta. Dessa maneira, serão brevemente demonstradas aplicações em medição de gás, energia elétrica — e teoricamente até água —, instrumentos digitais de medição em geral, bem como uma solução existente de medição por rádio frequência.

3.1 Leitura de Medidor de Gás Robusta Invariante de Ângulo

Na investigação realizada por Gallo *et al.* [18], procurou-se resolver um problema encontrado por companhias de medição de gás. Tal problema consiste no fato de que muitas vezes o leiturista ter de se deslocar a residências ou indústrias para realizar a leitura. Além de coletá-la, ele tira uma foto do visor do medidor a priori, a qual é checada a posteriori a fim reduzir chances de erro humano de leitura. Essa conferência pode demandar muito tempo, ser maçante e até custosa.

Para solucionar o problema, implementa-se um algoritmo de leitura automática por meio de fotos tiradas de medidores de gás. Da fotografia até a sequência numérica digital que expressa o valor de consumo, são realizados basicamente três passos: a localização do medidor, a detecção dos dígitos contidos nele e a leitura do consumo que aquele conjunto de dígitos representa. Isso é aplicado em imagens de fotos tiradas de diferentes ângulos e que possuem cores, borramentos, iluminações e medidores variados e diversos.

No primeiro passo, que é o de localização do medidor em meio a um pano de fundo qualquer, usa-se o MLP resiliente com 27 neurônios na entrada, 27 em uma única camada oculta e um único neurônio na saída de sua rede. Os dados de entrada são derivados de um *kernel* 3×3 que é deslizado em cada uma das imagens em RGB de um conjunto denominado “área vermelha”, tendo em vista que 99% dos visores de medidor de gás são vermelhos entorno de dígitos, de acordo com

Gallo *et al.*, e outro "pano de fundo" no treinamento. O neurônio de saída aponta a probabilidade de uma dada região da imagem ter a área vermelha. Uma vez que o medidor é encontrado no pano de fundo, é feita uma dilatação com elemento estruturante 5×5 e uma filtragem com operador gaussiano 5×5. Assim, torna-se possível localizar a região mais provável em que os dígitos estão e fazer uma rotação da imagem, de modo que os dígitos fiquem de pé em um alinhamento horizontal. A Figura 3.1 se remete ao procedimento, da esquerda para direita, para localizar o medidor e a região dos dígitos.

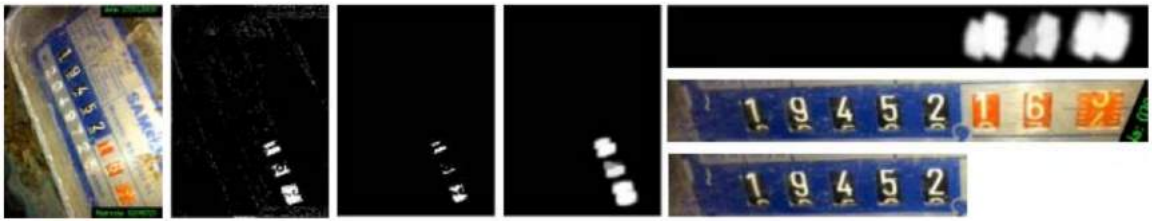


Figura 3.1: Localização e alinhamento do contador de consumo [18].

No segundo passo, utiliza-se um método chamado *Maximally Stable Extremal Regions* (MSER) para localizar os possíveis lugares em que os dígitos estariam contidos no medidor. Com isso, todas as regiões que teoricamente circundam dígitos são quadriculadas e aplica-se HOG nelas, a fim de que fiquem como a Figura 3.2 mostra. Em seguida, emprega-se SVM em cada um dos quadrados para classificar as regiões de acordo com as classes $c_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, n\}$, em que n significa não-dígito.

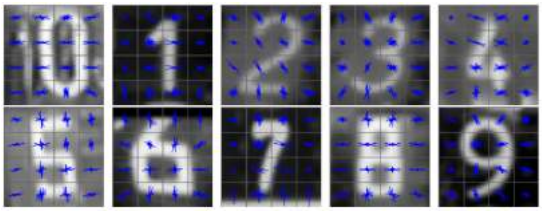


Figura 3.2: Aplicação de HOG nos recortes de dígitos [18].

O terceiro passo, faz-se a união de todos os recortes feitos pela localização através de MSER na região de possíveis dígitos, de forma que um sobreponha o outro. Depois disso, cada grupo formado pela união é ampliado por um fator escalar negativo, tornando-se menor. Então, esse grupo reduzido é transformado em um pequeno retângulo que abrange os limites de seus contornos, o qual estaria sobre um dígito se não houvesse falsos positivos. Esses retângulos minúsculos são organizados em ordem decrescente de tamanho e, em seguida, é feita uma espécie de regressão linear em relação ao centroide de cada um para encontrar uma reta que os corte pela região mais central possível. É computado um valor máximo para a distância de cada centroide a reta, e retângulos cujos centroides estão a uma distância maior do que a máxima, os falsos positivos, são excluídos do vetor que os armazena em forma decrescente. Daí todos os restantes são redimensionados em função das distâncias características de um dígito a outro, seus centroides e o tamanho dos dígitos. Se um deles é não-dígito, este é novamente enviado para reclassificação pelo SVM. Caso contrário, todos os rótulos atribuídos são mantidos. Finalmente, o valor de consumo correspondente àquela

sequência decrescente de recortes retangulares de dígitos é transformado em uma sequência de caracteres. As principais partes do processamento desse terceiro passo são apresentadas na Figura 3.3.

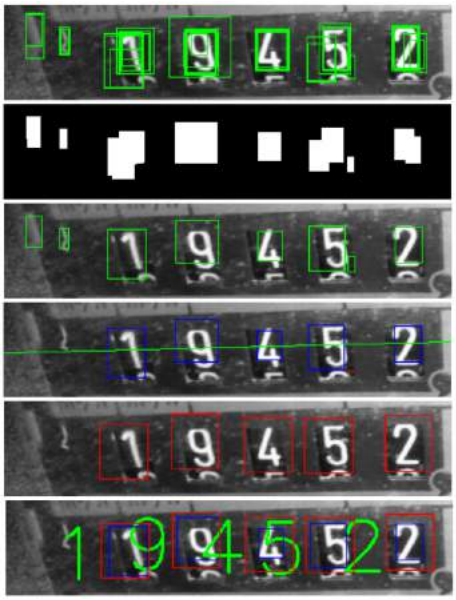


Figura 3.3: Alusão às fases do terceiro passo, de cima para baixo, que é o reconhecimento de dígitos [18].

Na prática, por meio dos testes realizados, nota-se que o algoritmo desenvolvido pelo grupo é relativamente robusto o suficiente para a demanda levantada. Com os melhores parâmetros de configuração, a acurácia de leitura para uma base de dados de 168.958 imagens de tamanho 640 × 480 chegou a 67,5%. Isso se explica pelo fato de boa parte das imagens terem sido problemáticas, tendo luminosidade, borramento, e distorção de perspectivas, como evidencia a Figura 3.4, o que fez com que o desempenho de MSER caísse drasticamente. Por outro lado, há de se considerar que a proposta trabalha com vários tipos de medidores, o que aumenta o grau de complexidade do problema. O tempo médio gasto para fazer o processamento com OCR de cada imagem foi de 1,6 segundos.

Figura 3.4: Base de dados com distorção de perspectivas, luminosidade e borramento [18].

3.2 Prática com OpenCV: OCR para Medidor de Energia Elétrica

No trabalho desenvolvido por Kompf [19], decidiu-se implementar um OCR para medidores de eletricidade por ter percebido que seria bem útil para monitoramento remoto, já que não há medidores do tipo que fornecem um acesso direto e padronizado a computadores. Para tal, criou-se um programa cujo algoritmo foi baseado no fluxograma da Figura 3.5.

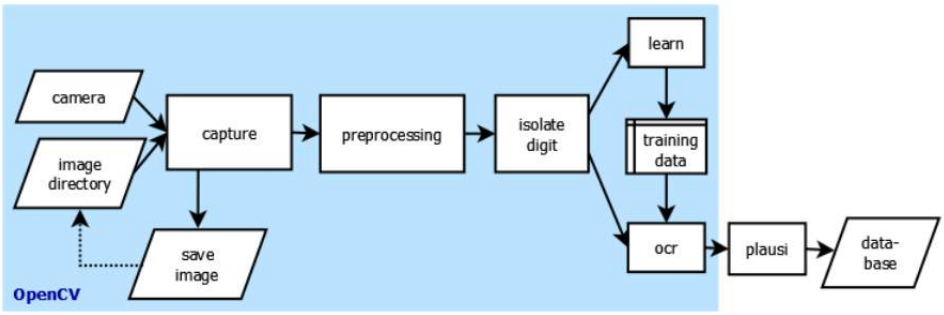


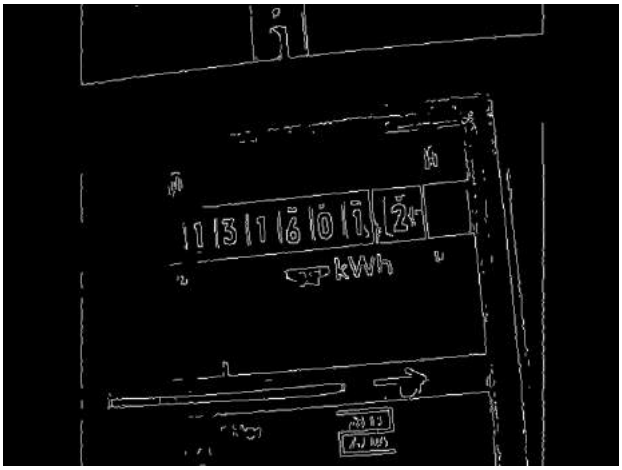
Figura 3.5: Fluxograma que define o algoritmo do programa [19].

Como aludido, o programa começa pela captura de imagem em RGB de tamanho de 640 × 480, feita por uma *webcam* USB. A fim de garantir uma boa imagem, usa-se um LED de baixo consumo com um mecanismo de difusão de iluminação feito com um material plástico branco semitransparente. Com isso, a imagem do medidor capturada adquire a forma exibida na Figura 3.6.

Figura 3.6: Foto original que é tirada do medidor [19].

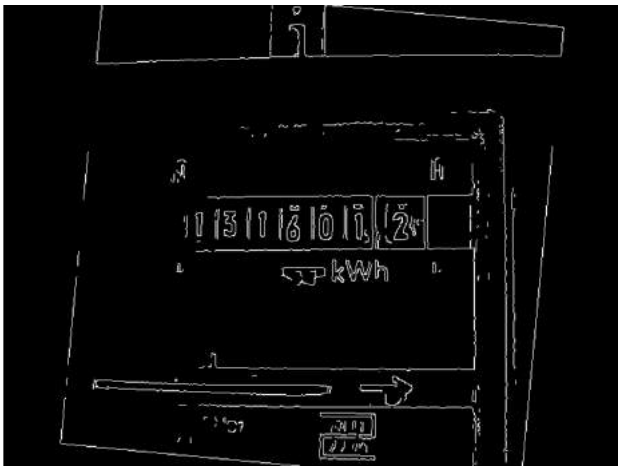
Em seguida, a foto tirada, além de ser salva para poder ser objeto de testes posteriormente, é processada de diversas maneiras. Primeiro, com o intuito de alinhar a foto, faz-se uma rotação com um ângulo pré-definido, o qual procura compensar o ângulo de construção do relógio medidor de energia dentro do padrão de luz, passando os dígitos da posição deitada para a de pé, como pode ser visto na Figura 3.7. Após isso, como a imagem não está totalmente na horizontal ainda, por causa da falta de ajuste na junta esférica da *webcam*, melhora-se o alinhamento. Com tal

propósito, o filtro Canny é convoluído nela para detectar as bordas ali existentes, especialmente as que compõem semirretas (Subfigura 3.7(a)). O ângulo residual de desvio com relação a horizontal é então encontrado executando a transformada de Hough. Para isso, faz-se uma média aritmética de todos os ângulos obtidos para uma determinada quantidade mínima de votos para possíveis retas (Subfigura 3.7(b)). Essa média serve então de entrada angular para rotacionar a imagem novamente e alinhá-la com a horizontal de vez (Subfigura 3.7(c)).



(a) Filtragem com Canny.

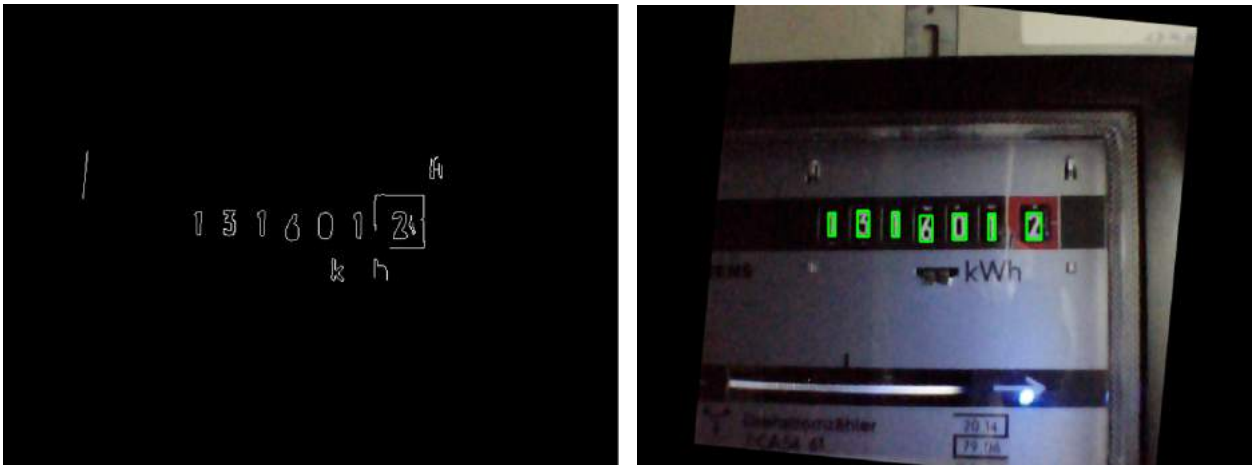
(b) Transformada de Hough para retas.



(c) Imagem alinhada horizontalmente.

Figura 3.7: Processo de alinhamento horizontal da imagem [19].

Na sequência, são feitos a detecção e o isolamento de dígitos. Para isso, filtram-se os contornos externos ao fazer uma restrição de comprimentos que os contornos necessitam para serem considerados dígitos, sendo a altura preestabelecida e sabendo-se que esta tem que ser maior do que a largura. Dessa maneira, afunila-se a quantidade de contornos, como é mostrado na Subfigura 3.8(a). Daí, avaliam-se as posições no eixo y e as alturas dos contornos com um algoritmo cuja função é detectar contornos similares em uma linha horizontal, considerando todas as combinações possíveis. Com isso, obtém-se a sequência retângulos delimitadores de dígitos ordenada da esquerda para direita. Esses retângulos estão desenhados no entorno dos dígitos na Subfigura 3.8(b).



(a) Filtragem de contornos. (b) Detecção dos dígitos.

Figura 3.8: Filtragem de contornos e detecção de dígitos [19].

O próximo passo é fazer o aprendizado de máquina. Escolhe-se aqui o kNN como método classificador de padrões. Para poder utilizá-lo, transforma-se todo o conjunto de dados, que até então é bidimensional, em unidimensional, além de torná-lo por completo expresso em ponto flutuante. Feito o ajuste nas amostras, seus respectivos rótulos são criados. Depois disso, ambos são passados para o modelo kNN para serem treinados. Após o treino, os dados gerados são armazenados em um arquivo no formato YAML, no qual o classificador vai se basear sempre para fazer a estimativa e rotular as imagens de dígitos.

Por fim, procura-se ter certeza de que os dados interpretados pelo kNN são verídicos utilizando testes de plausibilidade. Nesse sentido, são implementadas 3 regras básicas. A primeira estabelece que tem de haver sete dígitos reconhecidos no relógio de luz. A segunda regra exige que o valor corrente não seja menor do que o anterior. E a terceira restringe que, em um determinado intervalo de tempo, não haja um valor de potência consumida maior do que a potência gerada pela maior corrente admissível pelo fusível multiplicada pela tensão de alimentação do padrão. Os dados, quando checados e corretos, são armazenados em uma Round Robin Database (RRD) para contagem de análise, a qual permite acesso por internet a sua base de dados.

3.3 Abordagem Universal Econômica para Leitura Remota de Medidor Usando Serviços de Rede e Visão Computacional

Na investigação conduzida por Puttnies *et al.* [20], tenta-se resolver um problema contemporâneo que é a inserção de dispositivos tradicionais no contexto da Internet das Coisas. Em específico, trabalha-se acerca do desafio que é tornar um relógio de luz analógico em um relógio automático e inteligente, com a capacidade de se conectar à internet e usar serviços de rede para proporcionar uma leitura remota. A forma de atacar o problema propõe um método de visão computacional para identificar e extrair a leitura não somente de medidores de energia elétrica, mas também de gás e água. Os passos para tal compreendem desde de a captura da imagem até a emissão de

dados de leitura pela internet, como apresenta a Figura 3.9.

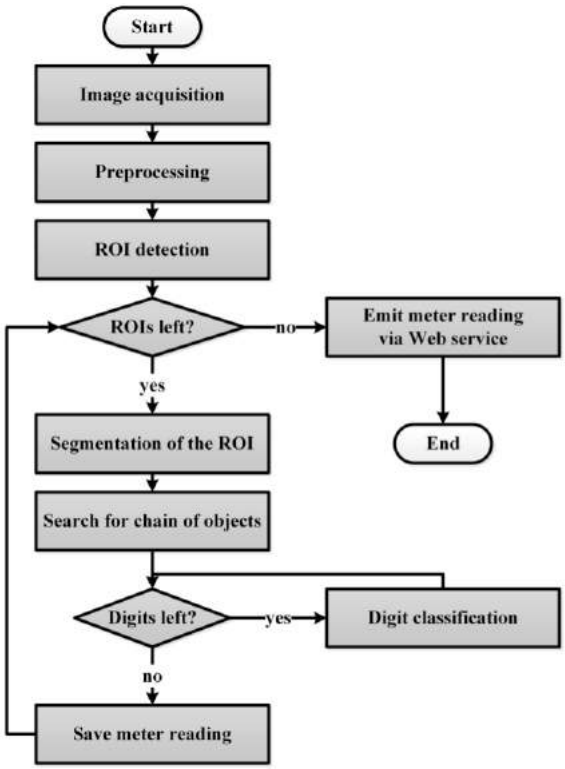


Figura 3.9: Fluxograma proposto para fazer a automação do relógio de luz [20].

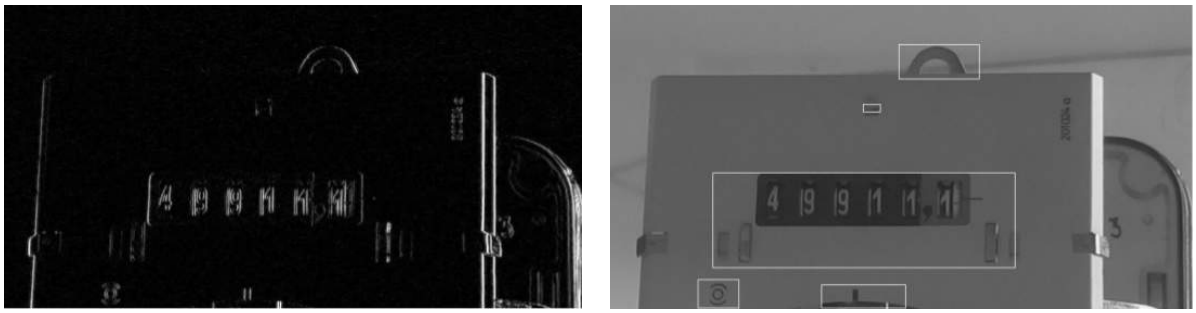
Como visto no fluxograma do algoritmo do sistema, inicia a execução pela fase de aquisição de imagem. Nela, uma *webcam* captura uma imagem colorida em alta resolução. Esta imagem é convertida em tons de cinza, reduzindo assim a complexidade computacional.

Na parte de processamento da imagem, ela é cortada em torno de sua região central, passando a ter 900×600 de tamanho, o que reduz custos computacionais ainda mais.

Segue-se então para a etapa de detecção de região de interesse (ROI). Esta, no artigo, consiste na região que abriga os dígitos do medidor. Em vez de usar redes neurais como no artigo da Seção 3.1, ROIs aqui são detectadas com base em retas verticais. Para isso, convolui-se a imagem em tons de cinza com um operador Sobel em x (Subfigura 2.7), a fim de deixar as bordas verticais mais visíveis. Depois, um *kernel* quadrado de filtro média é convoluído para suavizar e realçar as regiões de borda vertical. E finalmente, um limiar é aplicado para separar a frente da imagem de seu fundo, o que é evidenciado pela Figura 3.10. Daí as ROIs são detectadas partindo da premissa de que devem estar onde há grandes concentrações de bordas verticais (Subfigura 3.10(b)).

Após a detecção da ROI, é feita segmentação. Como na região extraída ainda existe parte do pano de fundo, a região dos dígitos é separada deste por meio de um filtro de limiar adaptativo.

Feito isso, vem a seção de procura por uma cadeia de objetos (dígitos). Sua finalidade é distinguir os dígitos do relógio dos objetos não desejados que circundam a imagem. O sistema reconhece automaticamente identifica o número de dígitos válidos, e é generalista no sentido de poder o fazer para qualquer modelo de medidor em que seu relógio tenha uma cadeia de no mínimo



(a) Bordas verticais. (b) ROIs.

Figura 3.10: Detecção de ROIs por meio de bordas verticais [20].

5 dígitos. Para alcançar tal robustez, usa-se um conjunto de três regras. A primeira demanda que haja uma distância horizontal d_h equidistante entre dígitos. A segunda estabelece que haja uma pequena distância vertical d_v entre os dígitos. E a terceira determina que todos eles devam ter a mesma altura h e a mesma largura w, praticamente. As métricas impostas pelas três regras são ilustradas na Figura 3.11.

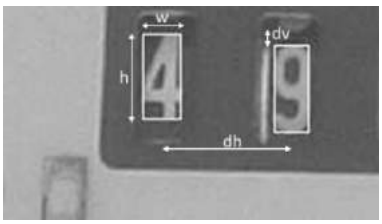


Figura 3.11: Parâmetros de regras criados para detectar a cadeia de dígitos [20].

Dando sequência, a parte de classificação usa redes neurais. Comenta-se que, apesar dos melhores resultados terem sido alcançados por redes neurais convolucionais, MLP consegue atingir resultados tão altos quanto, além de ser encontrada em diversas bibliotecas de *software*. Por essa razão, MPL foi escolhida como classificador de padrões. Como dados de treinamento, usaram-se uma pequena parte do conjunto robusto de dados para leitura do *International Conference on Document Analisys and Recognition* (ICDAR) 2003, várias imagens reais de dígitos de medidores de água, luz e gás, e imagens artificiais de dígitos, as quais foram distorcidas com a inserção de um ruído gaussiano branco. Ao todo, somaram-se 907 imagens de treino. Exemplo delas se encontram na Figura 3.12.

(a) Imagem real de um medidor de luz. (b) Imagem artificial com inserção de ruído branco. (c) Imagem pega da ICDAR.

Figura 3.12: Composição dos dados amostrais para OCR.[20].

O último passo consiste na emissão da leitura. Foi feito um artifício de serviço de rede implementado em Java, recorrendo a biblioteca *Java Multi Edition Devices Profile for Web Services Stack* (JMEDS) que é adequada para dispositivos embarcados. Na implementação, consegue-se ter acesso à leitura quando preferir ou se inscrever em uma lista para receber uma mensagem assim que houver uma alteração de leitura.

Como resultado de performance do sistema como um todo, teve-se a taxa de acertos do OCR é de 98%. O tempo de execução foi de aproximadamente 1,45 segundos.

3.4 Reconhecimento Óptico de Caracteres de Display de Sete Segmentos (SSOCR)

No trabalho produzido por Auerswald [21], o OCR criado é destinado a reconhecer dígitos de instrumentos digitais de medição que possuam *display* de sete segmentos. Uma imagem que tenha uma linha de segmentos entra no algoritmo do sistema, e caracteres numéricos em formato padrão são retornados como resposta, incluindo pontos flutuantes e números negativos, ou uma combinação dos dois. Como forma de exemplo, considera-se a Figura 3.13.

Figura 3.13: Medição vinda de um *display* de sete segmentos [21].

A implementação do reconhecimento de dígito começa pela transformação da imagem colorida em monocromática. Subsequentemente, emprega-se uma filtragem por limiarização, fazendo assim a segmentação da imagem. Uma vez que a imagem está binarizada, identifica-se a posição de cada dígito. Para isso, varre-se horizontalmente a imagem da esquerda para direita a procura de colunas de píxeis pretos na imagem. Por meio dessas colunas, encontram-se as distâncias de afastamento, bem como os dígitos, os quais são enquadrados, um por um, por um retângulo delimitador cinza.

Tendo os dígitos, segue-se para parte de reconhecimento deles. Nessa etapa, outra varredura é feita, porém na vertical, no centro do topo de cada retângulo delimitador. Os píxeis encontrados na parte do topo, meio e base são considerados como segmentos das respectivas partes do retângulo. Em seguida, os segmentos verticais são descobertos ao atravessar uma linha de escaneamento a um quarto do topo e outra a um quarto da base. Píxeis da direita e esquerda representam segmentos, respectivamente, da direita e da esquerda. Assim sendo, todos os dígitos, exceto o um, são obtidos recorrendo a uma tabela de correspondência implementada por um *switch*. No caso do numeral um, o dígito o é quando sua largura é menor que um quarto de sua altura.

Já no caso de símbolos tais como o ponto decimal e o hífen, que representa o sinal negativo, usa-se outro raciocínio. Para o primeiro, verifica-se o tamanho de cada dígito restante — que não foi reconhecido como um ainda. Caso altura e largura sejam significativamente menor que seus respectivos tamanhos máximos, o dígito é considerado um ponto decimal. O sinal de negativo, por sua vez, é encontrado quando a altura do dígito é menor do que um terço de sua largura.

A Figura 3.13 apresenta a saída do OCR para a Figura 3.14. Repara-se que às bordas esquerda e direita estão as colunas vermelha e verde, respectivamente, e linha verde entre essas colunas evidencia que a conexão das duas forma um dígito. Além disso, as pequenas retas verticais vermelha, verde e azul inferem que seus respectivos seguimentos foram encontrados na varredura vertical. Já a varredura horizontal acha seguimentos verticais indicados pelas cores vermelha e verde para esquerda e direita, respectivamente.

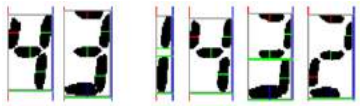


Figura 3.14: Resultado de reconhecimento de dígitos por SSOCR do *display* de sete segmentos [21].

O trabalho termina apresentando um exemplo de aplicação. Nele, relata-se que o SSOCR foi empregado na leitura de *tokens* emitidos em um *display* de sete segmentos de um dispositivo de segurança, visto na Figura 3.15. Explica-se que a razão pela escolha se deu pelo fato de que OCRs genéricos não são capazes de reconhecer dígitos em um *display* do tipo, principalmente por causa das desconexões dos segmentos.

Figura 3.15: Aplicação do SSOCR em um dispositivo de emissão de *tokens* [21].

3.5 Sistema de Telemetria para Hidrômetros e Medidores: Aquisição Móvel e Fixa de Dados por Rádio Frequência

Na solução comercial Ciasey [22], há duas alternativas para medição remota de dados de hidrômetros, assim como para medidores de gás ou energia elétrica.

A primeira alternativa consiste no *walk by* — indicado para um número de pontos a medir razoavelmente grande. Nela, um funcionário é responsável por passar nas ruas com um dispositivo de aquisição de dados, composto de um *data logger* e uma antena receptora, o que permite o acesso ao consumo dos moradores sem sequer entrar em suas residências. Feito o trajeto na vizinhança de onde se desejava coletar as leituras, o coletor é sincronizado com um computador via *bluetooth* para que as informações de medição possam ser inseridas em um banco de dados.

A segunda alternativa já é baseada em comunicação GPRS, o que garante uma medição autônoma, e é indicada para atender menos pontos, quando comparada com a solução *walk by*. Nesse caso, um concentrador é posto estrategicamente em uma região de medição, de modo a colher os dados de pontos distando até quinhentos metros dele, embora outros concentradores possam ser inseridos para aumentar a área de cobertura do sinal, atendendo mais pontos. Periodicamente, os hidrômetros daquela região enviam informações de consumo para tal concentrador. Este, por sua vez, recebe-as e as armazena em um arquivo Extensible Markup Language (XML) e, por GPRS de novo, repassa as informações a um servidor para que sejam guardadas em um banco de dados.

Com os dois tipos de solução supracitados, a Ciasey proporciona uma melhor gerência de consumo de água para seus clientes. Abnormalidades nos hidrômetros podem ser detectadas por meio de características previamente parametrizadas para a aplicação. Quando os parâmetros estão fora de um determinado padrão, portanto, um alarme é soado. Isso ajuda a verificar problemas emergentes instantaneamente, bem como fraudes e condições dos medidores. A Figura 3.16 ilustra como o monitoramento de consumo remoto é feito em uma interface gráfica de um programa criado pela empresa.

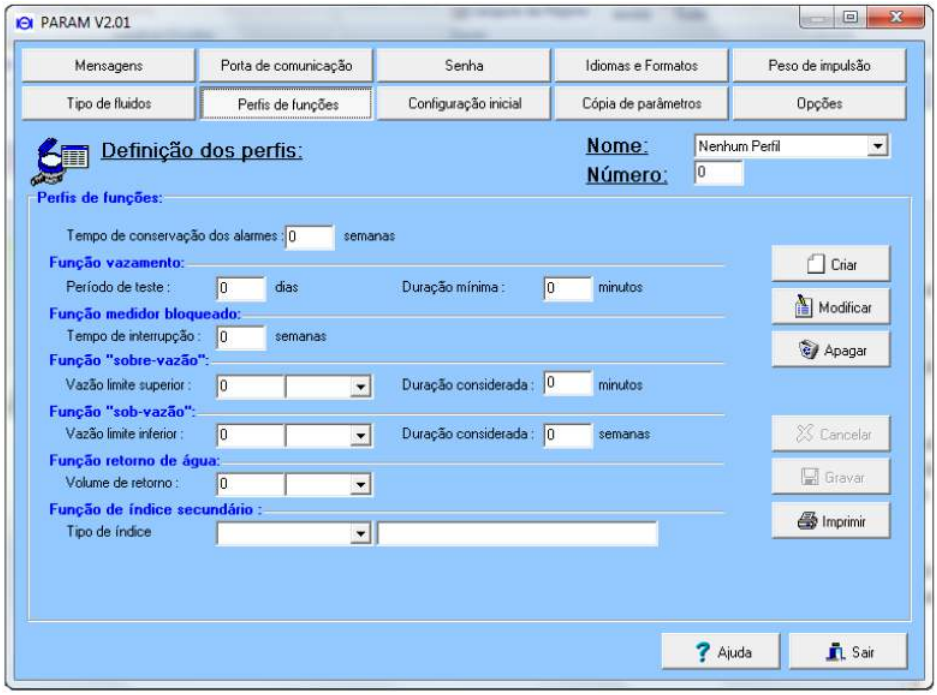


Figura 3.16: Interface gráfica desenvolvida para servir de monitoramento remoto de hidrômetro [22].

Como pode ser visto, o algoritmo que é alimentado pela coleta periódica de leituras tem várias funcionalidades: consegue identificar algum vazamento da rede de forma automática; percebe quando o medidor pode estar avariado; aponta se o medidor foi subdimensionado, tendo em vista a vazão corrente, alarmando inclusive quando há constantemente uma alta vazão que pode trazer fadiga e danos ao equipamento; detecta fluxo reverso, que pode ser sinal de fraude ou problemas na rede; e consegue identificar fraude mecânica ou magnética, acionando alarme.

A Ciasey também possui uma plataforma online chamada Web Service Mizar. Tal plataforma pode ser utilizada pelo concentrador GPRS, armazenando informações sobre leituras em seu banco de dados. Para acessá-la, o usuário navega até seu site e insere *login* e senha. Quando dentro do sistema, é possível visualizar os dados de consumo, alarmes e informações cadastrais do ponto de inscrição, conforme a Figura 3.17. Além disso, relatórios diários e mensais personalizados podem ser emitidos e exportados para planilhas, como mostra a Figura 3.18, e há como visualizar o consumo em forma de gráfico de barras.

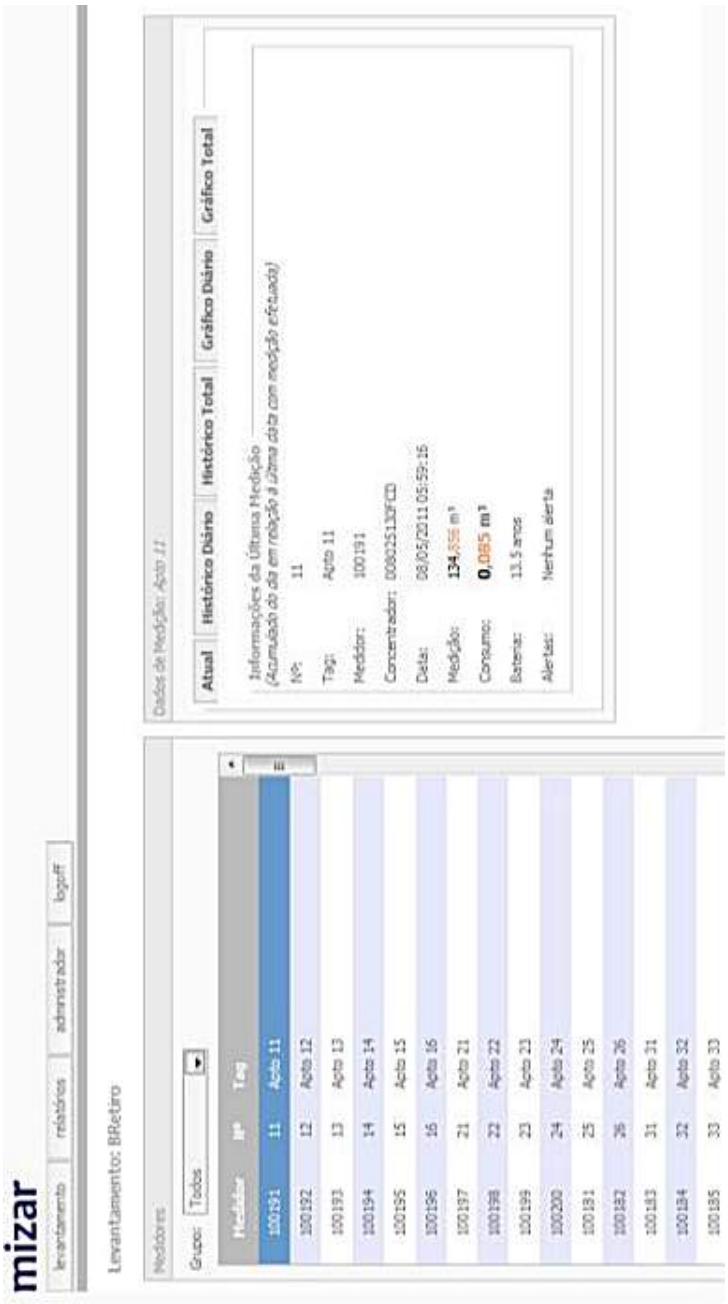
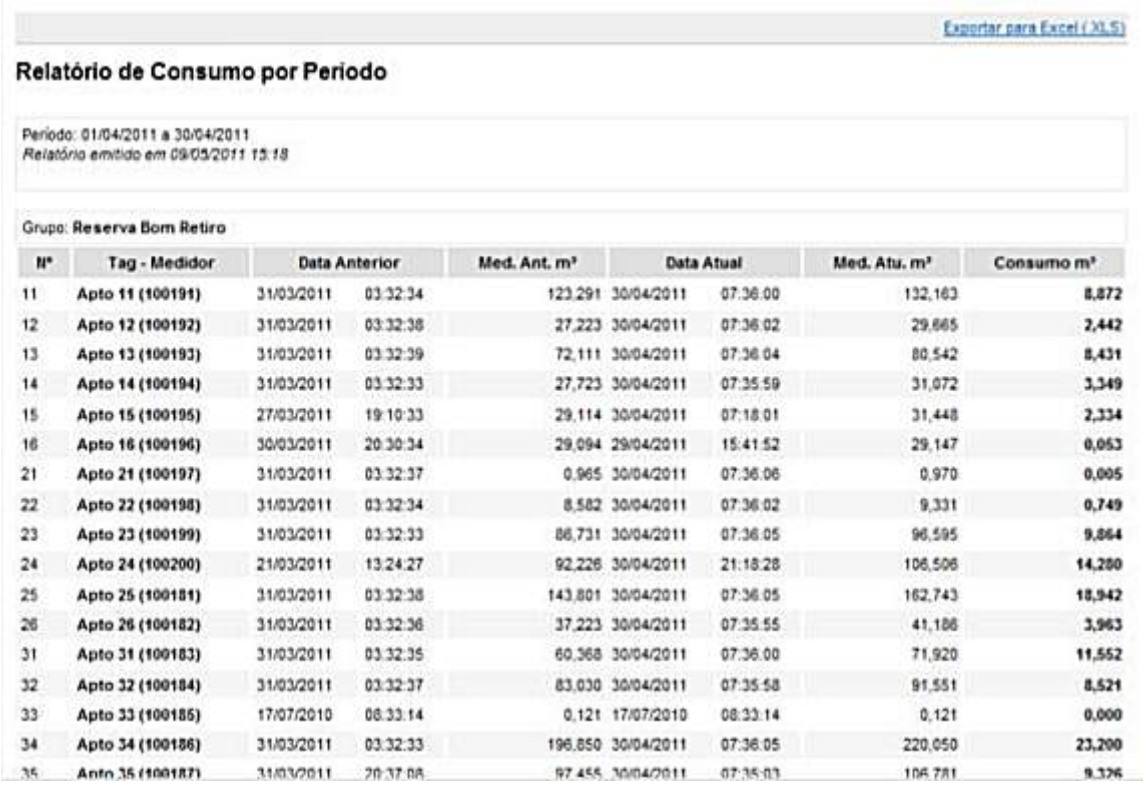


Figura 3.17: Página *web service* com informações sobre última medição de um hidrômetro [22].



Exportar para Excel (.XLS)

Relatório de Consumo por Periodo

Período: 01/04/2011 a 30/04/2011
Relatório emitido em 09/05/2011 15:18

Grupo: **Reserva Bom Retiro**

Nº	Tag - Medidor	Data Anterior		Med. Ant. m³	Data Atual		Med. Atu. m³	Consumo m³
11	Apto 11 (100191)	31/03/2011	03:32:34	123,291	30/04/2011	07:36:00	132,163	8,872
12	Apto 12 (100192)	31/03/2011	03:32:38	27,223	30/04/2011	07:36:02	29,665	2,442
13	Apto 13 (100193)	31/03/2011	03:32:39	72,111	30/04/2011	07:36:04	80,542	8,431
14	Apto 14 (100194)	31/03/2011	03:32:33	27,723	30/04/2011	07:35:59	31,072	3,349
15	Apto 15 (100195)	27/03/2011	19:10:33	29,114	30/04/2011	07:18:01	31,448	2,334
16	Apto 16 (100196)	30/03/2011	20:30:34	29,094	29/04/2011	15:41:52	29,147	0,053
21	Apto 21 (100197)	31/03/2011	03:32:37	0,965	30/04/2011	07:36:06	0,970	0,005
22	Apto 22 (100198)	31/03/2011	03:32:34	8,582	30/04/2011	07:36:02	9,331	0,749
23	Apto 23 (100199)	31/03/2011	03:32:33	86,731	30/04/2011	07:36:05	96,595	9,864
24	Apto 24 (100200)	21/03/2011	13:24:27	92,226	30/04/2011	21:18:28	106,506	14,280
25	Apto 25 (100181)	31/03/2011	03:32:38	143,801	30/04/2011	07:36:05	162,743	18,942
26	Apto 26 (100182)	31/03/2011	03:32:36	37,223	30/04/2011	07:35:55	41,186	3,963
31	Apto 31 (100183)	31/03/2011	03:32:35	60,368	30/04/2011	07:36:00	71,920	11,552
32	Apto 32 (100184)	31/03/2011	03:32:37	83,030	30/04/2011	07:35:58	91,551	8,521
33	Apto 33 (100185)	17/07/2010	08:33:14	0,121	17/07/2010	08:33:14	0,121	0,000
34	Apto 34 (100186)	31/03/2011	03:32:33	196,850	30/04/2011	07:36:05	220,050	23,200
35	Apto 35 (100187)	31/03/2011	20:37:08	97,455	30/04/2011	07:35:03	106,781	9,326

Figura 3.18: Relatório de consumo emitido [22].

Capítulo 4

Desenvolvimento

Este trabalho foi divido em duas partes principais. A primeira consistiu na concepção e construção de uma bancada de teste compacta e robusta, para realização de testes em um hidrômetro analógico e outro digital, os quais foram objetos de estudo essenciais para o desenvolvimento do protótipo de medidor autômato — afinal de contas, a proposta era fazer o supervisionamento remoto de medição de vazão via internet. Essa parte de produção da bancada de teste, bem como da instalação das câmeras e configuração do sistema de visão computacional (*webcams*, *Raspberry Pi* e monitor), foi desenvolvida ao longo do Trabalho de Graduação I. A segunda parte compreendeu todo o desenvolvimento do software, desde a captura de imagens até a publicação de métricas de medição on-line pela internet, sendo realizado no escopo do Trabalho de Graduação II. Ao fim, foram criados um arquivo de *log* com dia e horário de medições e afins e uma pasta específica para armazenar todas as imagens de leituras coletadas do hidrômetro. Essa pasta seria utilizada caso houvesse o interesse de conferir e certificar-se de que o sistema de visão computacional era transparente e confiável — dependendo da performance do reconhecimento de dígitos.

4.1 Bancada de Teste

A bancada de teste foi o primeiro elemento constituinte deste trabalho a ser projetado. A Caesb colaborou bastante ao fornecer boa parte do material necessário, sobretudo os hidrômetros analógico e digital, os quais eram fundamentais e tinham custo elevado no mercado. Se não fosse pela Caesb, o projeto poderia ter se tornado inviável por comprometer o orçamento disponível. Da mesma maneira, o Laboratório de Desenvolvimento de Produto: Prototipagem Rápida e Engenharia Reversa foi outro grande parceiro deste projeto. O laboratório emprestou duas câmeras CCDs da Microsoft, uma de qualidade HD e outra VGA, para serem usadas na bancada de teste. Caso fosse necessário adquiri-las, o projeto ficaria bastante dispendioso.

4.1.1 Concepção

Antes mesmo de realizar testes com o hidrômetro, foi necessário arquitetar uma forma de simular o consumo de água por intermédio de seu contador, como se ele estivesse sendo usado em uma residência ou indústria — uma vez que a robustez do sistema medidor não poderia ser alcançada com uma única sequência de dígitos presente no visor do instrumento de medição. Nesse sentido, vislumbrou-se que o sistema, na verdade, precisava ser um circuito hidráulico fechado. Tal circuito demandaria um pequeno reservatório, substituindo os grandes reservatórios de companhias de saneamento que armazenam a aguá a ser distribuída. Outro requisito do sistema seria uma bomba de drenagem de água, para funcionar como as potentes bombas que escalonam água até os pontos de consumo. No caso, a bomba de drenagem trabalharia de forma a proporcionar uma vazão constante no circuito hidráulico fechado.

O sistema hidráulico teria de ser capaz de funcionar por tempo indeterminado, sem que houvesse vazamento algum, para não acarretar perda de água do circuito e nem inundar o laboratório de medição, onde haveria aparelhos eletroeletrônicos. Em decorrência dessa exigência, o motor da bomba requereria um mecanismo de dissipação de calor e resfriamento, para evitar curto-circuito ou qualquer outro dano em caso de superaquecimento. Para tal, um elemento constituinte de aletas e uma ventoinha de tamanho mediano poderia ser eficiente. O vazamento, por sua vez, seria impedido por meio do uso de adaptadores e conectores adequados, bem como de uma cola especial para conexões de tubulações de *polyvinyl chloride* (PVC). Assim, formar-se-ia a bancada de teste, a qual teria ainda uma válvula esfera para cada hidrômetro, o que permitiria a fácil atuação na variação da vazão de cada um deles quando necessária.

A bancada de teste, ou planta, estaria restrita a determinadas dimensões. Como estaria em uma sala de laboratório, não poderia ser mais larga do que a porta de acesso dele, e teria de caber na mesa em que ficaria. Além disso, a altura do reservatório deveria proporcionar uma certa pressão de água razoável no nível em que os hidrômetros se encontrassem. No entanto, não poderia ser tão alta a ponto de ficar desproporcional a base e aumentar a propensão a instabilidade. Na verdade, seria muito importante que a estrutura da planta fosse bastante rígida e estável. Sua base deveria ser suspensa, distando poucos centímetros da mesa, caso houvesse a necessidade de colocar algo embaixo dela ou de passar fios por ali. E nas laterais, no rumo dos hidrômetros, seria interessante levantar duas hastes para que se pudesse fixar as webcams para fazer a medição dos instrumentos.

4.1.2 Construção

Primeiramente, procurou-se adquirir todos os materiais necessários para construção da bancada de teste. Nesse sentido, contatou-se a Caesb para comunicar sobre a montagem do protótipo a qual se estava dando início na forma de Trabalho de Graduação e solicitar ajuda na aquisição de componentes para o circuito hidráulico. Queria-se criar uma parceria, já que o trabalho era de interesse deles também. Como resposta, a companhia de saneamento forneceu os seguintes materiais:

- 01 hidrômetro digital Hydrus, com interface para saída pulsada;
- 01 hidrômetro analógico Elster M170-XI;
- 04 adaptadores específicos pretos, para possibilitar a conexão entre hidrômetros e tubos PVC;
- 01 adesivo plástico para PVC Tigre (75 g);
- 01 rolo de fita veda rosca Polyfita de dimensões $18\,mm \times 5\,m$;
- 01 tubo PVC com 3/4" de diâmetro e 3 m de comprimento;
- 04 luvas soldáveis com rosca de 3/4" de diâmetro;
- 03 luvas soldáveis para tubos com 3/4" de diâmetro; e
- 04 joelhos soldáveis para tubos com 3/4" de diâmetro.

Outras entidades envolvidas no trabalho foram o Laboratório de Desenvolvimento de Produto: Prototipagem Rápida e Engenharia Reversa, a marcenaria da UnB, o almoxarifado do SG-11, o almoxarifado do SG-09 e o Laboratório do Grupo de Automação e Controle (Graco). Com a colaboração dessas partes, adquiriram-se os seguintes materiais:

- 01 webcam LifeCam VX-1000;
- 01 webcam LifeCam Cinema;
- 07 abraçadeiras de canduite 25 mm;
- 14 parafusos de fenda simples e cabeça redonda com 5 mm de diâmetro 15 mm de comprimento;
- 06 chapas de compensado $15 \times 45 \times 45$ mm;
- 07 parafusos de fenda simples e cabeça chata com 5 mm de diâmetro e 50 mm de comprimento;
- 01 chapa de compensado $20 \times 705 \times 705$ mm;
- 01 chapa de compensado $20 \times 575 \times 705$ mm;
- 08 parafusos de fenda simples e cabeça chata com 5 mm de diâmetro e 30 mm de comprimento, com arruelas e porcas;
- 01 perfil quadrado de metalon 20×20 mm com 5 m de comprimento;
- 02 perfis retangulares de metalon 20×40 mm com 230 mm de comprimento;
- 03 folhas de lixa fina para metal 225×275 mm;
- 02 tubos de metal com 20 mm de diâmetro e 225 mm de comprimento;

- 02 tubos de metal com 26 mm de diâmetro e 50 mm de comprimento;
- 02 chapas de aço $2 \times 20 \times 45$ mm;
- 02 cantoneiras $20 \times 20 \times 95$ mm;
- 02 chapas de aço $3 \times 50 \times 180$ mm;
- 01 chapa de aço $3 \times 170 \times 180$ mm;
- 01 chapa de aço $3 \times 95 \times 170$ mm;
- 01 chapa de aço $2 \times 20 \times 170$ mm;
- 02 parafusos de fenda simples e cabeça chata com 6 mm de diâmetro e 25 mm de comprimento, com arruelas e porcas;
- 02 chapas de aço triangulares com catetos 90×120 mm e 5 mm de espessura;
- 08 chapas de aço triangulares com catetos 100×100 mm e 3 mm de espessura;
- 01 lata de tinta preta;
- 01 rolo de pintura com 50 mm de largura;
- 01 ventoinha *Boxer Fan* WS2107FL-7;
- 01 dissipador de calor de aletas finas ligeiramente envergadas com 90 mm de diâmetro e 30 mm de espessura;
- 01 dissipador de calor com dimensões $18 \times 45 \times 55$ mm e aletas de espessura 2 mm;
- 01 pasta térmica;
- 10 abraçadeiras de *nylon*;
- 02 pares de bornes; e
- 02 tomadas machos com fios de 1 m de extensão.

Para terminar a lista, foi necessário comprar os seguintes materiais:

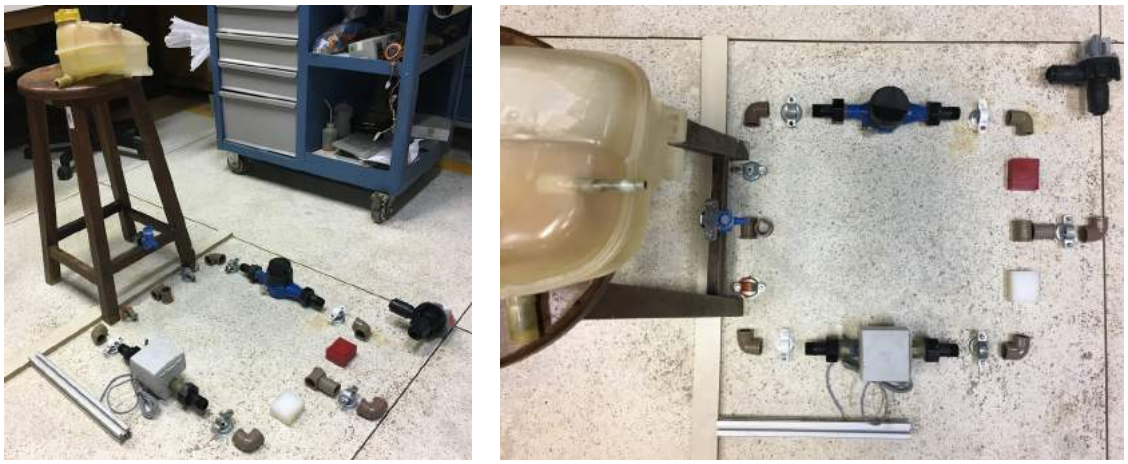
- 01 mangueira preta com 1 m de comprimento e $^3/_8$" de diâmetro;
- 01 mangueira transparente com 600 mm de comprimento e $^3/_4$" de diâmetro;
- 03 luvas soldáveis para tubos com $^3/_4$" de diâmetro;
- 02 joelhos soldáveis para tubos com $^3/_4$" de diâmetro;
- 03 luvas soldáveis com rosca de 25 mm;
- 02 adaptadores soldáveis e roscáveis de 25 mm;

- 01 luva com rosca;
- 03 abraçadeiras $^3/_8$" - $^1/_2$";
- 04 abraçadeiras 19 mm - 25 mm;
- 02 abraçadeiras 32 mm - 51 mm;
- 03 conexões T;
- 01 bico de metal para torneira $^3/_4$" × $^1/_2$";
- 03 adaptadores de rosca $^3/_4$" para mangueira $^3/_4$";
- 01 reservatório de água do radiador de um Astra (2 L de capacidade);
- 01 bujão $^3/_4$" de plástico;
- 02 válvulas esferas $^3/_4$" de plástico;
- 01 bomba de drenagem de água BAV1121-02UC 220 VAC; e
- 02 adaptadores que estabelecem uma ligação física entre a bomba e mangueiras de máquina de lavar.

Depois de reunir todo o material necessário, seguiu-se para a etapa de preparação para construção da bancada. Por meio de medições realizadas na porta do laboratório do Grupo de Automação e Controle (Graco) da Universidade de Brasília e na mesa em que a bancada de teste ia ficar, verificou-se que a base da planta deveria ter uma largura w_{esp} menor que 1000 mm, uma h_{esp} altura menor que 2100 e uma profundidade l_{esp} menor que 800 mm da porta do laboratório. Com base nessas especificações, definiu-se que a base teria largura $w_p = 750\,mm$ e profundidade $l_p = 750\,mm$, para garantir uma certa flexibilidade no que diz respeito à posição de montagem. Agora, para escolher a altura h_p, atinou-se que uma pessoa deveria ser capaz de passar pela porta do laboratório carregando a planta sem necessidade de agachar. Em virtude disso, estabeleceu-se $h_p = 650\,mm$, sendo que a superfície da mesa distaria 50 mm da superfície da base, e desta até o tubo de descarga de água do reservatório haveria 600 mm.

Diante das dimensões escolhidas para planta, organizaram-se os componentes do sistema para checar se seria possível dispô-los naquela área útil formada por $l_p \times w_p$. A Figura 4.1 mostra como os componentes ficaram distribuídos no espaço delimitado. Repare que o tamborete em que o reservatório está representa bem a altura definida para este e que a régua de madeira e o pedaço de perfil de alumínio, associados ao rejunto do piso, reproduzem a área disponível. Assim sendo, observou-se que a área era suficiente para abrigar os elementos o circuito hidráulico na superfície da base. Naquela configuração, os hidrômetros estavam espaçados por uma distância d_h de 500 mm. De modo igual, as duas luvas deitadas, paralelas uma a outra na base, distavam d_h uma da outra.

Sabendo-se de que a estrutura do circuito hidráulico tomaria um aspecto quadrado no nível dos hidrômetros, cortaram-se pedaços no tudo de PVC de forma que, quando montados com os joelhos,



(a) Vista lateral. (b) Vista superior.

Figura 4.1: Disposição dos elementos do circuito hidráulico.

conexões T, válvulas esferas e luvas soldáveis e rosqueáveis, tornassem-se simétricos com relação aos eixos vertical e horizontal imaginários do centro do quadrado. A Figura 4.2, em especial, a Subfigura 4.2(b), é um reflexo da simetria alcançada. Nessa fase, é importante salientar que as conexões não foram feitas com cola, mas sim só por pressão, por causa das possíveis adaptações futuras.

(a) Vista lateral. (b) Vista superior.

Figura 4.2: Montagem parcial do circuito hidráulico da base.

Na sequência, partiu-se para a etapa de montagem da estrutura metálica. Fizeram-se o esboço do desenho técnico a mão e o preenchimento do formulário de ordem de serviço do bloco SG-09, em que foi especificado o material para construir a bancada de teste, assim como os tipos de serviços a serem realizados. No documento, colheram-se as assinaturas do orientador do trabalho de graduação e do professor supervisor no SG-09. Feito isso, pegou-se o material no almoxarifado do bloco da mecânica e, com a ajuda de um técnico da oficina, fizeram-se os cortes exatos no perfil de metalon e chapas de aço a fim de adquirir as peças constituintes da planta, de acordo

com o esboço previamente produzido. Imediatamente depois, conduziu-se o processo de soldagem, unindo as peças de perfil e chapas cortadas anteriormente. Por fim, quando a estrutura metálica já estava montada, esmerilharam-se os pontos de solda muito irregulares e os cordões espessos de maneira a deixá-los lisos e finos. Além disso, limaram-se todos os cantos vivos e vazaram-se furos nos lugares em que parafusos atravessariam.

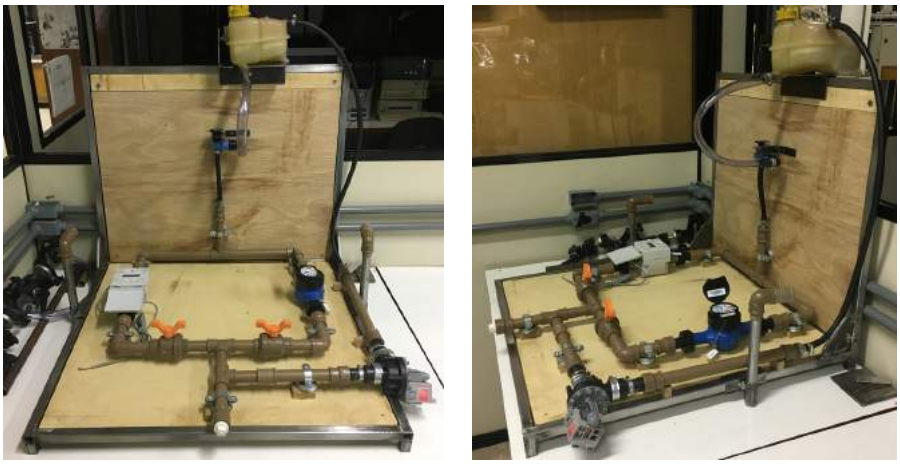
Ressalta-se aqui que além da estrutura metálica, foram fabricadas outras três peças na oficina utilizando recursos tais como chapas, tubos e cantoneiras de aço. Todos os três materiais foram arranjados e soldados para produzir um par de suportes. Cada um dos dois contava com uma interface quadrada para se prender no metalon por pressão e outra circular e perpendicular para poder abrigar uma haste circular de diâmetro ligeiramente menor, perpendicular ao plano dos hidrômetros. A terceira peça, por sua vez, foi produzida por meio de chapas apenas, que foram envolvidas em um processo de soldagem e furação. Dessa maneira, criou-se o suporte do reservatório, que entrava por pressão sobre o perfil quadrado do metalon e cujo conjunto aceitava um par de parafusos vazantes para os manterem fixos um no outro. As três peças também passaram por um processo de acabamento idêntico ao da estrutura metálica.

Subsequentemente, foi-se a marcenaria da UnB, onde duas chapas de compensado foram devidamente cortadas e furadas. Ali mesmo, também se colaram as seis chapas de compensado quadradas menores na chapa quadrada maior (base) com cola madeira, levando em consideração os furos de ambas as chapas, de forma que ficassem concêntricos.

De volta ao laboratório Graco, fez-se a limpeza dos componentes, bem como a lixamento das peças de aço e de madeira. Em seguida, montou-se a bancada por completo. Desta vez, acrescentaram-se outras partes ao circuito hidráulico, tais como a bomba de drenagem, o reservatório e seu suporte, uma válvula de acionamento por solenoide e as hastes verticais de aço e seus respectivos suportes. Para associá-los com o circuito, usaram-se abraçadeiras, elementos hidráulicos de conexão, fita veda rosca, mangueiras e tubos de PVC. Todas as juntas soldáveis foram coladas com o adesivo plástico para PVC, e elementos tais como a madeira da parede da bancada, o reservatório e seu suporte, assim como a válvula de acionamento solenoide, foram parafusados provisoriamente. A Figura 4.3 ilustra a bancada de teste depois de terminada a montagem provisória.

Visto que o tudo na bancada se encaixava perfeitamente nos devidos lugares, avançou-se para a fase de pintura. A bancada foi desmontada de maneira que as duas chapas de compensado, a estrutura metálica, as hastes circulares verticais e peças de aço soldadas (suportes) ficassem avulsas. A seguir, pintou-se uma por uma, utilizando uma tinta preta e um rolo de pintura. Ao fim, cada uma delas ficou com três camadas de tinta.

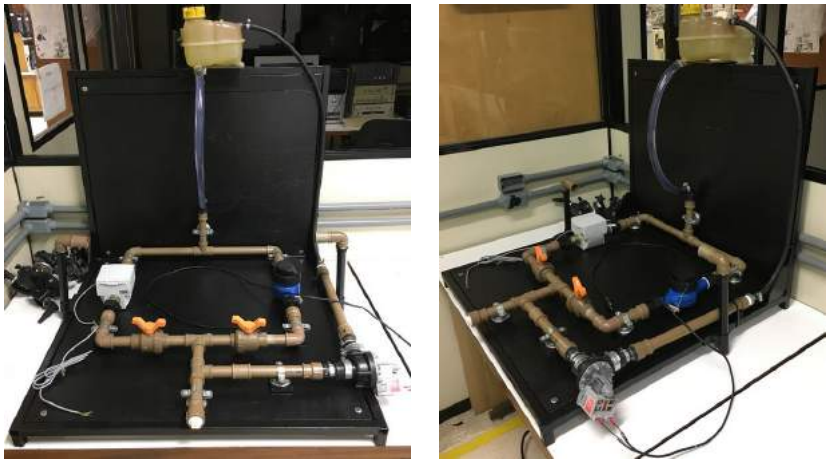
Depois de secas, montou-se a bancada novamente, parafusando todas as partes necessárias, inclusive os canduites na base. Quando finalizada a montagem, inundou-se o circuito com água através do reservatório e então ligou-se a bomba de drenagem. Observou-se que, apesar da tê-la ligado, não havia fluxo de água pelo circuito, pois os contadores dos hidrômetros não o registravam. Certificou-se de que os hidrômetros estavam postos no sentido de fluxo correto indicado em cada um deles, que era do reservatório para bomba, e na condução de uma inspeção minuciosa, ao abrir



(a) Vista frontal. (b) Vista lateral.

Figura 4.3: Montagem provisória da bancada de teste por completo.

o circuito à altura da válvula de acionamento solenoide, percebeu-se que a vazão de saída dela era ínfima, descobrindo ali a origem do problema de circulação da água. Diante disso, tirou-se tal válvula do circuito. Ela iria executar o desligamento do circuito, simulando uma intervenção remota da companhia de saneamento quando necessária. A Figura 4.4 apresenta a nova forma adquirida pela bancada. É importante ressaltar que a mangueira preta e o tubo PVC suspenso por um calço de madeira preto à direita foram presos à estrutura metálica por abraçadeiras de *nylon*.



(a) Vista frontal. (b) Vista lateral.

Figura 4.4: Conclusão da montagem da bancada de teste.

Por fim, inundou-se o circuito com água e ligou-se a bomba de drenagem a fim de verificar se haveria algum vazamento. Ligado por mais de trinta minutos, não apresentou nenhum vazamento sequer. No entanto, reparou-se que o núcleo do motor da bomba de drenagem estava bastante quente após aquele período. Com isso, surgiu-se então a necessidade de anexar um módulo de dissipação de calor no núcleo. Nesse sentido, dois dissipadores de calor foram justapostos no

material ferromagnético: um maior abrangendo as duas laterais da parte de cima e outro menor cobrindo toda a superfície do material na parte de baixo. Além do mais, para aumentar a taxa de dissipação de calor do sistema para o ambiente, pôs-se uma ventoinha de 115 VAC a cerca 120 mm do núcleo. Esta foi ajustada para uma posição qual que produzisse vento tanto nas aletas dos dissipadores quanto dentro do próprio núcleo do motor da bomba. Seu suporte foi fabricado dentro do Graco com chapas de aço, um metalon de perfil quadrado e outro de perfil retangular. Esse módulo que foi anexado a bancada, bem como os dissipadores de calor, pode ser visto na Figura 4.6.

4.1.3 Instalação do Kit Raspberry Pi

Além das *webcams* supracitadas na Subseção 4.1.2, o Laboratório de Desenvolvimento de Produto: Prototipagem Rápida e Engenharia Reversa disponibilizou os seguintes materiais:

- 01 estabilizador de tensão;
- 01 *Raspberry Pi* 2;
- 01 cartão de memória Samsung EVO micro SD *UHS-I Speed Class 1 (U1), Class 10* de 32 GB (taxa de transmissão até 48 MB/s);
- 01 fonte conversora abaixadora retificadora de duas ondas com saída 5,4 VDC e 2.0 A;
- 01 cabo USB para micro USB;
- 01 cabo de rede Ethernet;
- 01 cabo mini HDMI para DVI;
- 01 teclado;
- 01 mouse; e
- 01 monitor.

De posse dos materiais necessários, partiu-se para a instalação do kit *Raspberry Pi*. Iniciou-se esta a plugar a fonte no estabilizador de tensão, no qual se plugou o cabo USB cujo terminal micro USB foi plugado na *Raspberry*. Depois, conectou-se o minicomputador à rede pelo cabo ethernet, ao monitor pelo cabo mini HDMI para DVI e ao mouse, teclado e câmeras por intermédio de seus respectivos cabos USBs. Logo após isso, gravou-se a imagem da distribuição *4.9.50-1-ARCH Linux* no cartão de memória micro SD e o inseriu em seu *slot* na *Raspberry*. Finalmente, instalou-se um servidor Apache 2.4.27 (Unix) e a biblioteca OpenCV 3.3.0 no sistema operacional da placa microprocessada. A foto da tela do terminal, na Figura 4.5, foi tirada para evidenciar os detalhes concernentes às versões da distribuição e do Apache. A bancada, com o kit *Raspberry Pi* instalado e configurado, é vista na Figura 4.6. Nela, nota-se que há duas configurações, uma levemente diferente da outra. É que o conector no qual a mangueira transparente desaguava tinha

quebrado, por motivo desconhecido (Subfigura 4.6(a)). Com isso, teve-se de substitui-lo por outro que era um contector linear (Subfigura 4.6(b)), em vez de ser do mesmo formato do anterior (90º).

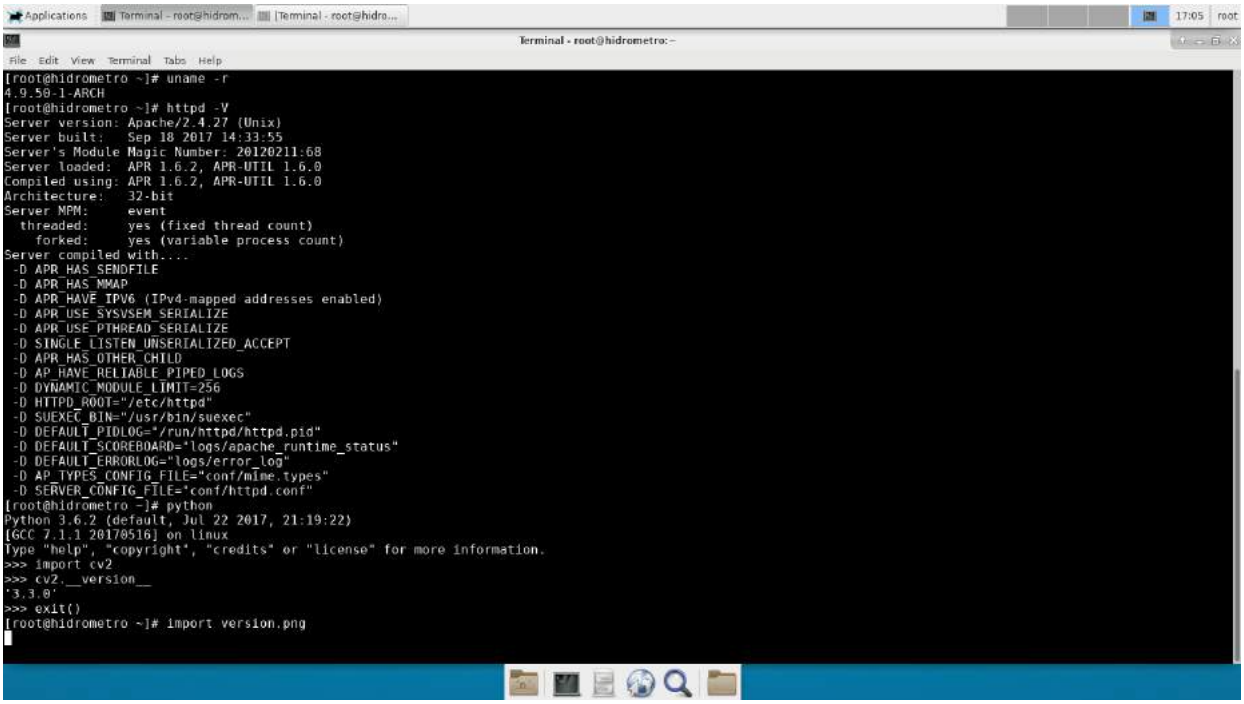


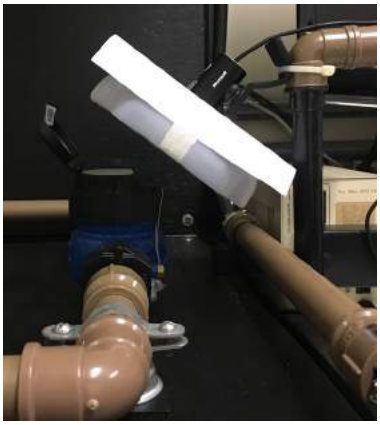
Figura 4.5: Versões do sistema operacional Arch e do servidor Apache.

(a) Bancada antes da avaria.

(b) Bancada depois da avaria.

(c) Câmera esquerda.

(d) *Raspberry* montada.



(e) Câmera direita.

Figura 4.6: Finalização de montagem do conjunto bancada-computador.

4.2 Desenvolvimento do Software para o Hidrômetro Analógico

O desenvolvimento do *software* para o hidrômetro analógico foi feito visando atender às definições do problema especificadas na Seção 1.3. Ao longo dele, usou-se uma distribuição Linux, cujo processador era Intel i7-4500U com quatro núcleos trabalhando a 1,8 GHz, o que tornava os testes mais rápidos do que se fosse o fazer em uma *Raspberry Pi 2*.

Para que fosse possível portar o *software* de uma máquina Linux para outra — no caso, do computador pessoal para a *Raspberry* —, usaram-se conceitos de distribuição de *software*, fazendo com que seus arquivos ficassem instalados nos devidos diretórios. Além do instalador, criou-se também um desinstalador para facilitar a retirada do aplicativo.

O programa criado entrega certa flexibilidade ao usuário em termos de sistema operacional. Isso porque foi desenvolvido em um *kernel* padrão para *Ubuntu 16.04 Long Term Support* (LTS) para rodar em um *kernel* de *Arch 4.9.50-1*. Assim, testes foram realizados em três diferentes versões de Python, uma vez que a biblioteca OpenCV tinha sido instalada com link simbólico

para as versões 2.7 e 3.5 na máquina Intel e 3.6 para a Arm. Por essa razão, a solução tornou-se executável para qualquer máquina Linux que tivesse uma das três versões mencionadas.

4.2.1 Considerações Iniciais

Para melhor entender o funcionamento do *software*, é importante estar à parte de alguns conceitos preliminares usados em seu desenvolvimento. A seguir, serão introduzidos os principais módulos, os diretórios e arquivos recorrentes, os argumentos de execução e quatro parâmetros chaves para o bom funcionamento do programa.

4.2.1.1 Módulos

O código fonte do *software* foi baseado em uma metodologia básica de processamento necessário para obter o valor do consumo de água em uma foto tirada do hidrômetro. Essa metodologia se traduzia em três passos:

1. a localização da região de interesse que era a região retangular ao redor dos dígitos (*roiLocator.py)*;
2. a detecção dos dígitos (*digitsDetector.py*); e
3. o reconhecimento deles (*digitsRecognizer.py*).

Assim, o código foi modularizado de modo que as funções usadas fossem organizadas e agrupadas por afinidade com os âmbitos existentes. Cada um destes ficou sendo um módulo.

4.2.1.2 Diretórios

Para padronizar o aplicativo e viabilizar o acesso a dados de manipulação do sistema, especificaram-se diretórios nos quais tais dados seriam salvos e lidos constantemente pelo programa. Ao todo, criaram-se quatro diretórios específicos. Estes e suas finalidades são descritos a seguir:

1. *ROI_PATH*: armazena imagem colorida da região de interesse, ou ROI;
2. *LOGFILE_PATH*: a título de verificação da coleta de leitura periódica, contém um arquivo *log* com métricas de medição tais como horário UTC local, *timestamp,* leitura, consumo instantâneo, variação de volume e tempo da leitura anterior para atual, vazão estimada, vazão média e quantidade de tentativas permitidas para fazer a leitura;
3. *METRICS_PATH*: armazena um arquivo de leitura e escrita que serve como *buffer* para o programa se situar sobre as métricas de medição da iteração anterior imediata; e
4. *IMS_UPDATE_PATH*: armazena foto da implementação de localização da ROI e o recorte ROI segmentado com a detecção de dígitos para exibição em página *html*.

4.2.1.3 Arquivos

Foi idealizado um conjunto de três arquivos que julgou serem indispensáveis para cumprir a demanda das funcionalidades entregues pelo *software.*

1. *hydrometerSetup.dat*: visa guardar dados do dia da configuração inicial (Subseção 4.2.10) tais como o horário e a data local, a leitura inicial verificada e inserida pelo usuário configurador e o nome (*timestamp)* da imagem recorte da região de interesse;
2. *prevHydroMetrics.var*: abriga todas as métricas computadas sempre referente a leitura anterior; e
3. *hydrometer.log*: coleciona todas as métricas das leituras feitas pelo sistema de visão computacional.

4.2.1.4 Argumentos

O programa foi desenvolvido para funcionar em função dos três argumentos de entrada listados a abaixo:

1. o número identificador do dispositivo *webcam* (*deviceID*);
2. o modo de execução (*debug*), podendo optar em fazer testes (*-d*) ou executar o programa pelo *Crontab* (*-n*); e
3. o modelo de aprendizado de máquina para classificação de padrões (*ML_Classifier*):
   (a) Support Vector Machine (*1*);
   (b) Multilayer Perceptron (*2*); e
   (c) k-Nearest Neighbor (*3*).

Criou-se o código fonte de forma que não fosse necessária a entrada de tais argumentos, sendo que, por padrão, o código já fosse inicializado referenciando à primeira e única câmera plugada na *Raspberry,* executando em modo *Crontab* por intermédio do SVM.

4.2.1.5 Parâmetros Fixados

Considere que foram fixados quatro parâmetros para o funcionamento do programa:

1. vazão nominal (*Qn=750*);
2. fator de segurança (*FoS=1*);
3. máximo número de tentativas (*Max_NoAttempts=5*); e
4. variação mínima de volume para registrar a leitura (*MIN_dV=1*).

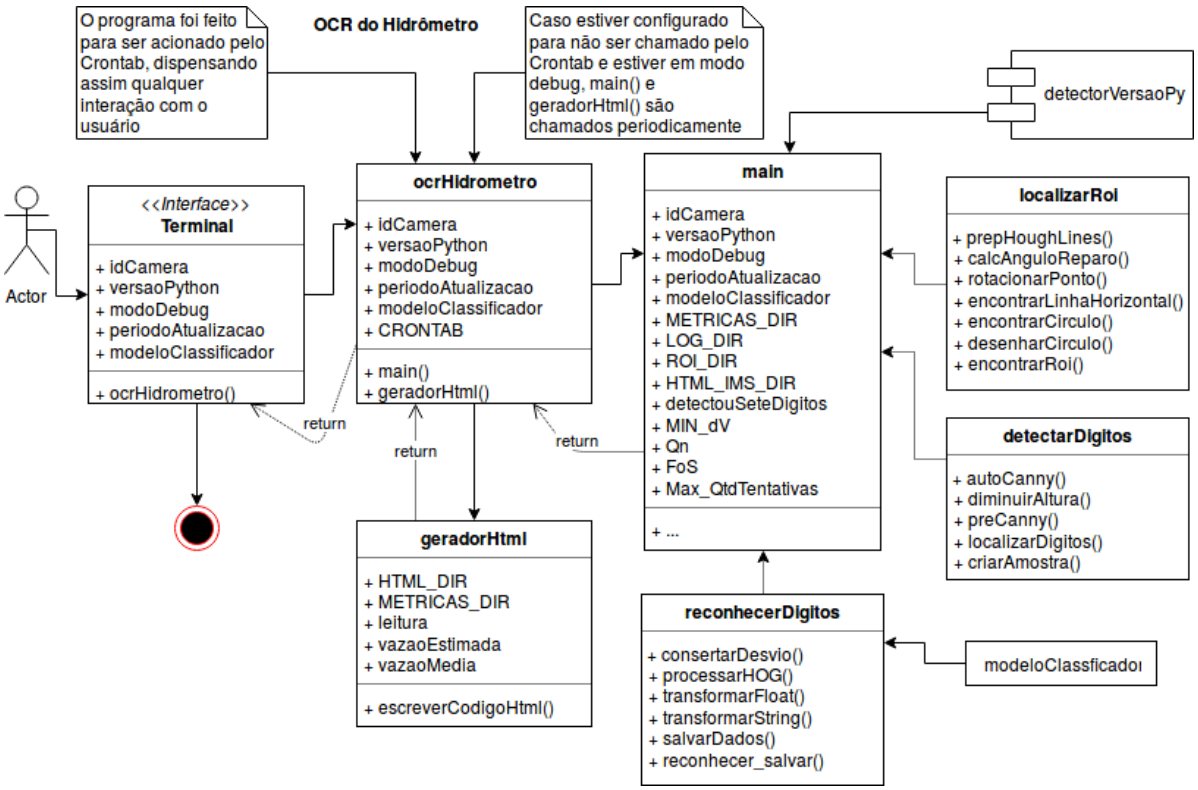


Figura 4.7: Diagrama de classes para o OCR do hidrômetro.

Os três primeiros parâmetros foram necessários para a manutenção do bom funcionamento do sistema de medição por visão computacional. *Qn* tomou aquele valor pelo fato de a vazão nominal do hidrômetro analógico ser 750 L/h, e *Max_NoAttempts* foi escolhido com o intuito de diminuir o consumo de recursos causado pelo período que a câmera ficava habilitada. O último parâmetro, por sua vez, apenas reflete a vontade do usuário ou necessidade da aplicação com relação a variação mínima de volume a ser computada.

4.2.2 Diagrama de Classes

Para melhor saber como se dá o processo de reconhecimento de dígitos e publicação de dados pela internet, a linguagem unificada de modelagem do funcionamento do *software* foi produzida na Figura 4.7. Também foi criada uma linguagem unificada de modelagem para a fase de zeramento da máquina de visão computacional. Isso é exibido na Figura 4.8. Tanto a Figura 4.7 quanto a Figura 4.8 serão explicadas de forma mais detalhada nas subseções seguintes.

4.2.3 Execução do Programa

Um *script* em Bash para execução do programa foi modelado para possuir até cinco argumentos, os quais são descritos a seguir na ordem de inserção:

1. versão de Python com a qual se quer executar o programa em Python (deve ser ou 2, ou 3);

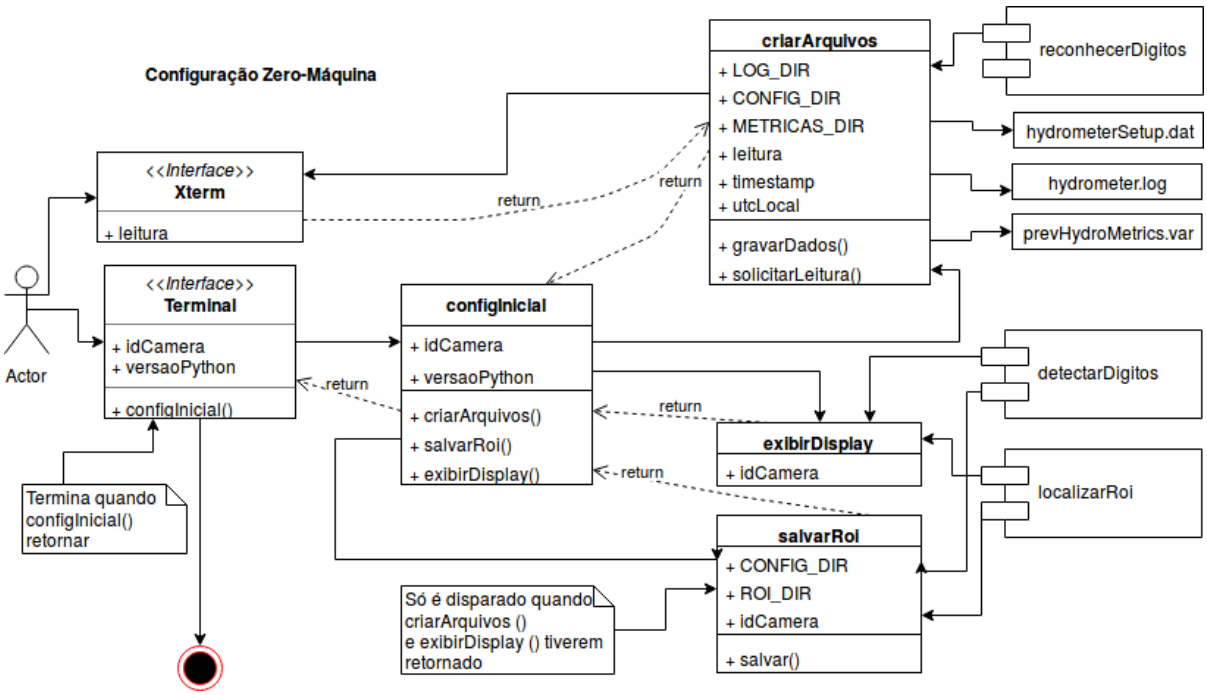


Figura 4.8: Diagrama de classes para o zeramento da máquina.

2. número de identificação da câmera;
3. modo de teste, ou *debug*;
4. período de atualização dado em segundos usado em modo *debug*; e
5. algoritmo classificador de padrão desejado para fazer o OCR.

Se nenhum deles é fornecido no ato de rodar o *script*, este roda em modo padrão, utilizando a versão Python 3. Mas se se deseja a inserção dos argumentos, estes têm de ser postos naquela mesma sequência. Por exemplo, para mudar o modelo de treinamento, todos os cinco argumentos devem ser inseridos. Se se quiser alterar o período de leitura, todos os quatro primeiros argumentos devem ser acrescentados.

Foram criados dois modos de execução, os quais eram manipulados através de uma variável booleana em função de uma determinada finalidade. Verdadeiro ligava o modo Crontab e falso, o modo *debug*. Ambos os módulos faziam as mesmas tarefas, mas tinham princípios diferentes. As tarefas eram executar o código fonte e depois atualizar um arquivo *html* que era gerado pelo módulo *htmlGen*. Este módulo acessava ao arquivo *prevHydroMetrics.var*, selecionava dados, tais como data, horário, consumo instantâneo, vazão estimada e vazão média, e os substituía nos devidos lugares do código em *html* que, por sua vez, era sobrescrito no arquivo que era lido pelo servidor. O tempo de atualização automática da página *html* era de 30 segundos.

4.2.4 Código Fonte

O código fonte foi projetado para executar vários passos no processo de leitura do valor de consumo através de uma foto tirada do hidrômetro pela *webcam*. A Figura 4.9 trata-se de um fluxograma construído que busca apontar os principais processos envolvidos, mostrar os dados mais relevantes gerados e utilizados pelos processos, bem como os critérios de decisão tomados, do início ao fim. É importante salientar que o algoritmo representado pelo fluxograma pode ser executado repetidas vezes, mesmo terminando a cada ciclo, pois as variáveis necessárias de uma iteração para outra são salvas em arquivos.

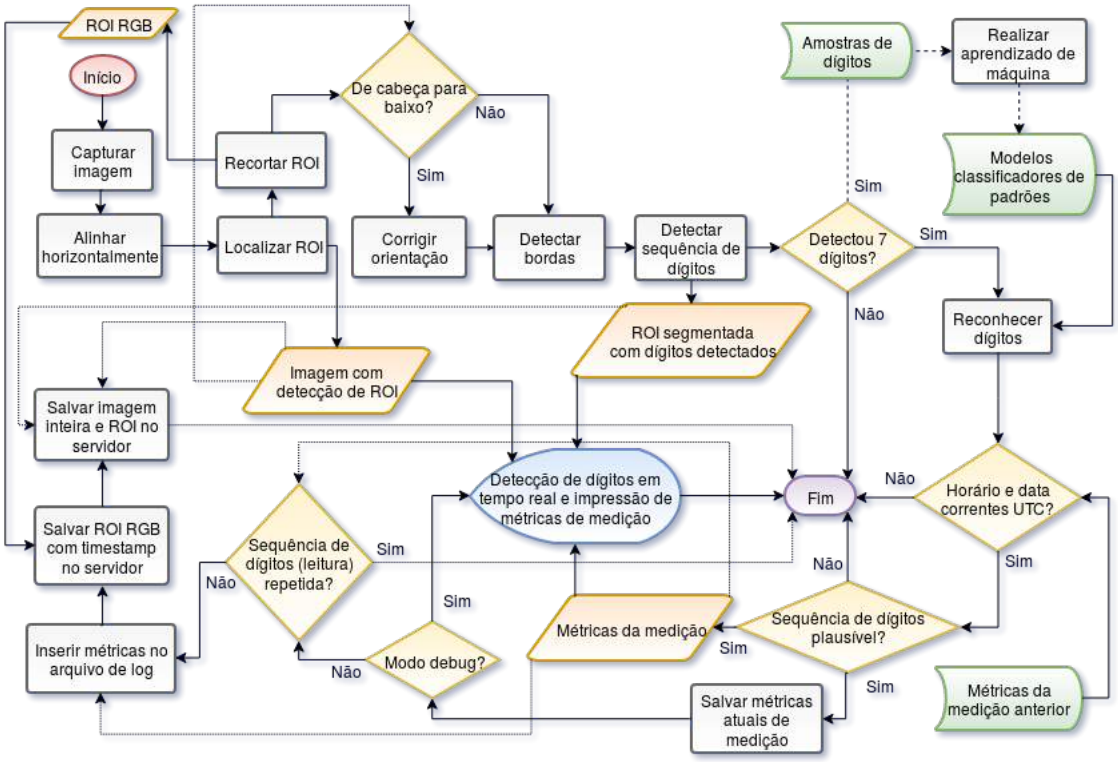


Figura 4.9: Fluxograma do código fonte.

Repare que o fluxograma está em forma de ações executadas a fim de alcançar o objetivo principal, que é proporcionar a medição de vazão em tempo real de um hidrômetro analógico via internet. Para cada passo, existe uma ou mais funções que, combinadas, têm por finalidade chegar a um propósito comum. O propósito é ditado de acordo com os módulos *headers* do código. Por exemplo, o módulo *roiLocator* compreende os processos de alinhamento horizontal e localização de ROI; *digitsDetection* subsidia a detecção de bordas e a detecção da sequência de dígitos; e *digitsRecognizer* reúne as funções que executam o reconhecimento de dígitos, o salvamento de dados da medição, da imagem inteira processada e do recorte de ROI, tanto o colorido como o segmentado. Além disso, tal módulo faz a inserção de dados no arquivo de *log*. Da Subseção 4.2.6 à 4.2.9, o procedimento em cada uma das partes funcionais é discorrido.

Ressalta-se aqui que, por intermédio de testes realizados com diferentes resoluções disponíveis da *webcam* HD, escolheu-se a 640×480 para a captura de fotos com a câmera por ela ter apresentado

os melhores resultados em termos de velocidade de processamento e qualidade da imagem. Além disso, frisa-se que em parte do código lidou-se com uma imagem colorida e outra equivalente em tons de cinza. Operações feitas nesta eram replicadas naquela.

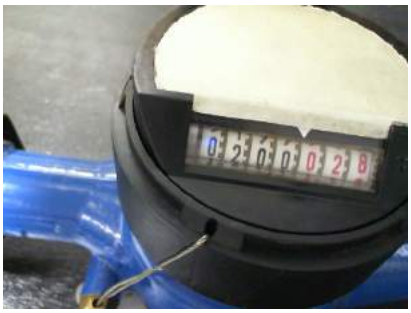
4.2.5 Identificação de Versão Python

O código fonte conta com um módulo extra chamado *versionHandler* responsável por detectar a versão de Python chamada no momento da inicialização da execução. De acordo com a versão detectada por ele, faz-se a decisão de chamar os três principais módulos do programa (Subseção 4.2.1.1) já compilados, economizando tempo de processamento do programa. Tudo isso é feito por uma única função chamada *headerPicker* que não recebe argumentos alguns, mas retorna os três módulos compilados. Assim, torna-se possível a portabilidade do programa para as versões 2.7, 3.5, e 3.6 de Python. O módulo seria uma forma de conservar o material desenvolvido privado, sem que alguém pudesse ter acesso a informações de escopo das funções de um dos três módulos. Isso porque, depois de compilados, eles se transformam em uma sequência de hexadecimais.

4.2.6 Localização de Região de Interesse

Antes de mais nada, é importante ressaltar que todos os parâmetros usados no processamento digital de imagens nesta Subseção e na Subseção 4.2.7 foram determinados empiricamente.

Para localizar a região de interesse, com o módulo *roiLocator*, uma vez obtida a foto colorida do hidrômetro pela captura da *webcam*, preparou-se a foto para a aplicação da transformada de Hough para retas. Nesse sentido, primeiramente, converteu-se a foto de RGB para tons de cinza, aplicou-se um filtro bilateral para reduzir ruídos sem degradar as bordas da imagem, as quais, na sequência foram realçadas mediante a convolução por um *kernel* realçador específico. Daí, usou-se o operador Canny para fazer a detecção de bordas. A sequência de operações pode ser vista na Figura 4.10.

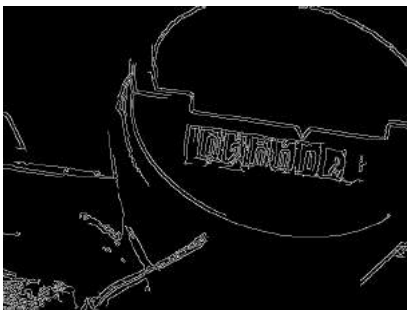


(a) Imagem 640 × 480 RBG.

(b) Imagem monocromática.

(c) Filtro bilateral ($d = 5$, $\sigma_{clr} = 1000$ e $\sigma_{spc} = 1000$).

(d) Filtro realçador de bordas (imagem subtraída de seu próprio laplaicano).



(e) Canny ($A_{sbl} = 5$, $L_{min} = 600$ e $L_{max} = 2500$).

Figura 4.10: Preparação da foto para aplicação da transformada de Hough.

Em seguida, a fim de alinhar a imagem horizontalmente, teve-se de detectar algum segmento de reta explícito pela detecção de borda (Subfigura 4.10(e)). Para isso, usou-se a transformada de Hough com um limiar de 100. Como resultado, surgiu-se uma reta paralela a *feature* retangular do *display* do hidrômetro, o que foi forçado a acontecer ao inserir uma fita adesiva branca em parte do visor superior em que não havia informações relevantes mas só dados considerados ruídos para o sistema em questão. Por meio do ângulo de inclinação dessa reta retornado pela transformada de Hough, calculou-se o ângulo de compensação θ_{comp} necessário para endireitar a imagem. Então, usou-se θ_{comp} para fazer o giro com relação ao centro desta, chegando ao alinhamento horizontal indicado pela Figura 4.11.

(a) Reta detectada pela transformada de Hough ($\theta_{resol} = 1^o$, $\rho_{resol} = 1$ e $L_{vts} = 100$).

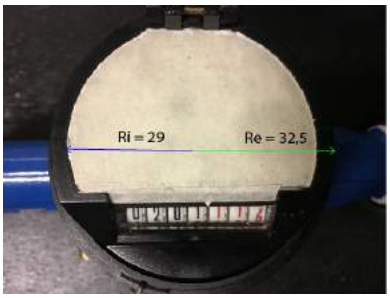
(b) Alinhamento horizontal.

Figura 4.11: Uso da transformada de Hough para fazer alinhamento horizontal.

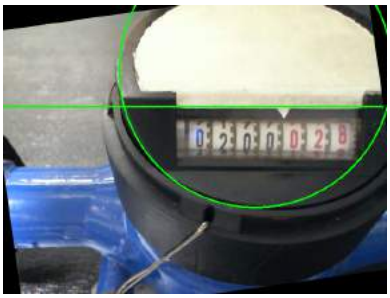
Logo após isso, explorou-se o aspecto geométrico circular do hidrômetro para afunilar o desafio que era de fazer a localização da região de interesse (*display*), já que esta estava dentro da circunferência em questão. No caso, pegou-se a imagem monocromática recém-alinhada e a submeteu a um filtro bilateral ($d = 9$, $\sigma_{clr} = 10$ e $\sigma_{spc} = 10$), novamente, para retirar ruídos, conservando as *features* do hidrômetro à medida do possível. Daí, empregou-se uma limiarização, usando um limiar alto para filtrar apenas os sinais de alta frequência — característicos da parte da fita adesiva branca —, e então procurou-se pelo círculo mínimo de fechamento da imagem. Acresceu-se seu raio com um valor relativo de cerca de 10%, que queria-se chegar ao raio externo da circunferência ($R_e = 32,5$ mm), tendo o raio interno ($R_i = 29$ mm). Tal circunferência externa foi desenhada na imagem RGB alinhada do hidrômetro. A sucessão de operações pode ser conferida na Figura 4.12.

(a) Imagem monocromática alinhada.

(b) Revelação do formato geométrico circular do *display* por limiarização ($L = 225$).



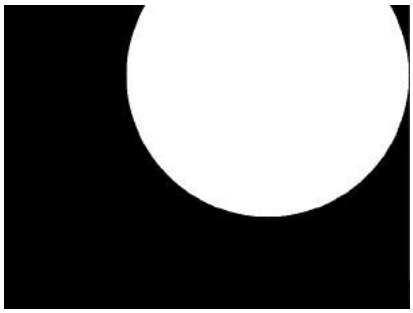
(c) Relação entre raio externo e raio interno do visor do hidrômetro.



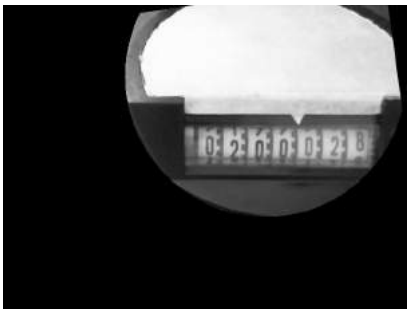
(d) Detecção do visor do hidrômetro.

Figura 4.12: Detecção da circunferência que abriga a região de interesse.

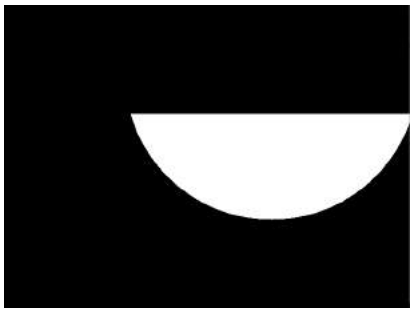
Na sequência, construiu-se uma máscara binária em que dentro da circunferência era nível alto e fora, nível baixo. Com essa máscara e a imagem monocromática alinhada fez-se uma operação lógica *and*. Sabendo da altura da reta gerada pela transformada de Hough, somaram-se todas as quantidades acumuladas de tonalidades de cinza a partir de nível 127 do histograma da imagem resultante da operação lógica, tanto na região acima da reta quanto na região abaixo da reta. Como a somatória da parte superior era maior que a da inferior, indicando uma maior densidade de tons claros no lado de cima, atribuiu-se zero a toda região superior à altura da reta horizontal. Então, efetuou-se um *and* entre essa nova máscara binária e a imagem monocromática alinhada, gerando assim uma meia-lua na imagem, em que a região exterior era toda nível baixo. O procedimento efetivado é exibido na Figura 4.13.



(a) Máscara binária criada com base na circunferência do *display*.



(b) Operação *and* entre a máscara e a imagem monocromática alinhada.



(c) Máscara com eliminação de região mais clara.

(d) Imediações da região de interesse.

Figura 4.13: Retirada de informações desnecessárias da imagem.

Tendo retirado as informações irrelevantes, o passo posterior foi detectar a região de alta frequência característica do *display*. Para tal, primeiro aplicou-se um filtro de borramento com um *kernel* quadrado de dimensão 30 para borrar a imagem semicircular. Depois, limiarizou-se a um nível relativamente baixo para ainda conseguir revelar a região mais clara que tinha sido distorcida. Por fim, o contorno da região branca da imagem binária foi computado e seus pontos mais externos foram levados em consideração para formar um retângulo que foi posto sobre a imagem RGB alinhada. Com isso, localizou-se a região de interesse, como é apresentada na Figura 4.14. É importante salientar que se o retângulo fosse maior do que o esperado, tendo dimensões tal que a razão entre a largura e a altura ultrapassasse 2,6, a meia-lua monocromática sofreria uma equalização de histograma, a fim de aumentar seu contraste, e passaria pelo mesmo procedimento usado para chegar a um novo retângulo que desta vez, possivelmente, retrataria a região de interesse correta.

(a) Operador média com *kernel* 30×30.

(b) Limiarização ($L = 100$).

(c) ROI retangular.

Figura 4.14: Localização da região de interesse.

4.2.7 Detecção de Dígitos

O detector de dígitos, executado pelo módulo *digitsDetector*, inicia sua tarefa a partir do recorte feito da região de interesse, evidenciado pela Figura 4.15. Antes de mais nada, o algoritmo verifica se a imagem na qual foi feita a localização da região de interesse está de cabeça para baixo, recorrendo ao mesmo princípio usado para zerar a parte acima da imagem a partir do rumo da reta verde horizontal (Subfigura 4.13(d)). Isto é, se a densidade de píxeis claros acima da reta é maior que abaixo, a imagem não sofre alteração alguma. Caso contrário, ela é rotacionada 180^o em relação ao seu centro, ficando assim na orientação certa. Tal processo de checagem de orientação e possível rotação é aplicado na imagem do recorte de dígitos também.

Figura 4.15: Recorte da região de interesse.

A seguir, aplicou-se Canny na imagem dos dígitos com o propósito de detectar as bordas na horizontal, em especial. O objetivo aqui era se livrar de ruídos ao redor dos dígitos. Para isso, empregou-se a transformada de Hough, novamente com um limiar 100, e computaram-se as dez primeiras retas mais votadas dentro do conjunto retornado por tal função. Essas retas foram separadas em dois grupos. O primeiro grupo era composto de retas do meio da imagem para cima, e o segundo, do meio para baixo. Depois, para fazer um novo recorte, tomaram-se como referência a coordenada y de cima e a de baixo mais próximas do centroide da imagem recortada, dada pelos parâmetros de coordenada cilíndrica (r, θ) de retorno da transformada de Hough. O resultado do recorte pode ser verificado na Figura 4.16.

(a) Detecção de bordas por Canny. (b) Novo recorte na ROI.

Figura 4.16: Redução vertical na imagem para retirar bordas indesejáveis.

Posteriormente, visou-se à eliminação de ruídos, mas mantendo ou até melhorando os contornos característicos dos dígitos. Nesse sentido, utilizou-se uma filtragem de borramento gaussiano, com *kernel* 5×5, três vezes consecutivas, seguido de um operador laplaciano. Como a função da biblioteca OpenCV que executa o laplaciano tem caráter *float* com sinal, fez-se uma limiarização com limiar zero, de modo que valores os de píxeis negativos fossem todos zerados e os positivos fossem todos para nível alto. Dessa maneira, a imagem tornou-se binária. Dando sequência, fizeram-se uma abertura e um fechamento na imagem com um elemento estruturante planar 2×2 na imagem. O primeiro foi realizado para tirar pontualmente os ruídos de alta frequência. Já o segundo foi empregado com o intuito de melhor as *features* dos números. Isso é verificado na Figura 4.17.

(a) Operação de borramento. (b) Filtro laplaciano. (c) Abertura seguida de fechamento.

Figura 4.17: Preparação da região de interesse para detecção de dígitos.

Por fim, aplicou-se Canny na imagem com o fechamento para ter acesso às bordas dela. Diante das bordas, usou-se a função do OpenCV para encontrar os contornos mais externos e acessar suas respectivas dimensões, já que ele faz uma varredura do canto superior esquerdo até o canto inferior direito e retorna o ponto inicial de cada contorno, bem como sua largura e altura. Dessa forma, por meio dos parâmetros de retorno de tal função, criou-se um algoritmo com o objetivo de filtrar os contornos dos dígitos, em detrimento dos outros contornos. Lembrando que os contornos capturados eram os mais externos, a estratégia de filtragem conduzida para obter os contornos dos dígitos consistiu de duas etapas.

Na primeira etapa, foi feita uma varredura por toda a lista de contornos verificando se a razão λ_d definida pela altura dividida pela largura era de 1,5 (para números diferentes de um) a 4,5 (no caso do número um), já que as alturas dos dígitos no hidrômetro era cerca de 4 mm e a largura, aproximadamente 2 mm, ou 1 mm para o dígito um. Além disso, para ser considerado dígito, o contorno tinha de ter uma altura entre 50% e 90% da altura da imagem. Se o contorno satisfizesse a essas duas restrições impostas, ele era inserido em uma outra lista, a qual era ordenada por ascendência posteriormente, tendo como referência a coordenada x de cada contorno. Ou seja, a sequência de contornos dentro da lista passava a ser dada da esquerda para direita.

Já na segunda etapa, se a lista não estivesse vazia, ela era varrida do início ao fim. Pegavam-se sempre o centro em termo da coordenada x do contorno atual C_i e do posterior C_{i+1} e checava-se, respectivamente, se eram maior que 8% e menor que 98% da largura total do recorte da região de

interesse — respectivos início e fim da zona provável para cogitar contornos. Depois, comparava-se se a distância do centro C_i até o de C_{i+1} era menor do que uma distância mínima d_{min} equivalente a 8% da imagem. Se não, o contorno C_i era posto na lista L_f, que era a lista de contornos filtrados. Se sim, retirava-se C_{i+1}, atualizava-se o tamanho da lista e computava-se novamente o centro em x do novo contorno C_{i+1}. Se a distância entre o centro deste e o de C_i fosse maior do que d_{min} e maior do que uma distância máxima d_{max} de 16% da imagem, o contorno C_i era anexado em L_f. Atualizava-se o índice i, e voltava-se ao início da segunda etapa. Caso detectado que faltava um contorno para chegar à sequência desejada de sete dígitos ao fim da filtragem dos contornos de dígitos, fazia-se uma outra comparação. Isso porque o último elemento anexado à lista L_f da cadeia de contornos de dígitos foi o C_i, deixando o C_{i+1} de lado por não haver sucessor deste na próxima iteração. Por essa razão, para o caso especial do último elemento, pegava-se o centro em x de C_{i+1} e subtraia do centro de C_i para encontrar a distância entre eles. Essa distância também tinha de ser maior que d_{min} e menor que d_{max} para que o último contorno fosse adicionado à lista L_f. Finalmente, se de L_f fosse igual a sete, seus contornos eram envolvidos por caixas retangulares verdes, fazendo então a detecção dos dígitos como apresenta a Figura 4.18.

(a) Filtro Canny. (b) Detecção de dígitos.

Figura 4.18: Resultado da detecção de dígitos.

4.2.8 Aprendizado de Máquina

O aprendizado de máquina foi possível ser desencadeado a partir do momento que se conseguiu detectar os dígitos. A seguir, será discorrido sobre a metodologia utilizada para adquirir todos os dígitos de forma variada e diversa, bem como sobre o que foi feito para organizá-los em classes para daí criar uma imagem enorme composta de 5 mil amostras, sendo 500 de cada classe. Ao fim, também será apresentado como se deu o processo de treinamento com a base de dados (imagem de milhares de dígitos) para modelos de classificação de padrão consagrados, tais como o *Support Vector Machine*, *Multilayer Perceptron* e *k-Nearest Neighbor*.

4.2.8.1 Aquisição de Dados Amostrais

Para reunir uma quantidade suficientemente grande de amostras, inicializou-se a aplicação e a deixou rodando por cerca de quatro horas, conseguindo, ao final, uma quantidade amostral de mais de 30 mil dígitos avulsos de tamanho 20×10, já que a havia a razão λ_d entre a largura e altura, respectivamente. Cada um desses dígitos recebeu uma etiqueta do tipo *xwyz_dn.png* como nome, em que *xwyz* era uma variável que se alterava da unidade z no sentido da unidade de milhar x em função da iteração que era iniciada a cada três minutos e *dn* era a posição do enésimo dígito da sequência da esquerda para a direita, podendo assumir valores de um a sete. Os dígitos rotulados iam para um único arquivo.

Terminada a parte de obtenção de dígitos, separaram-se os três dígitos menos significativos dos quatro mais significativos, pelo fato daqueles girarem mais rápido do que estes. A propósito, foi detectado que o dígito menos significativo do rolete do hidrômetro concluía um ciclo a cada vinte segundos, com a válvula do hidrômetro digital totalmente fechada e a do analógico totalmente 100% aberta, o que conferia a bomba uma vazão de 0,05 L/s (antes de uma avaria, que fez com que a nova vazão fosse para 0,04 L/s). Continuando, cada um dos três dígitos foi para seu respectivo diretório específico, em que foram renomeados de *0000.png* até *XWYZ.png*, sendo *XWYZ* a quantidade de dígitos existentes que dependia do diretório. A essa altura, é importante salientar que o processo de recorte era paralelo e que nem sempre se conseguia 7 recortes por vez, pois o algoritmo não ter estado tão bem otimizado antes.

Tendo os dígitos menos significativos devidamente separados em três arquivos diferentes, fez-se uma nova separação para dividi-los em dez diferentes classes, as quais representavam o valor do dígito, podendo ser de zero a nove, obviamente. Para tal criou-se um *script* Bash que chamava um programa em Python, iterativamente. Esse programa basicamente percorria o arquivo, imagem por imagem, mostrando na tela um dígito por vez. Ao reconhecer o dígito, o usuário tinha de digitar seu valor correspondente. Daí o *script* lia aquele valor de retorno e reencaminha o dígito-imagem para um arquivo específico concernente a classe daquele dígito.

4.2.8.2 Construção da Imagem de Dígitos

Com as imagens minúsculas de dígitos 20×10 foi construída uma imagem grande de 1800×1250, na qual havia 4 linhas com 125 dígitos cada, para cada classe, somando assim 500 dígitos de cada classe. Essa grande imagem pode ser visualizada na Figura 4.19.

4.2.8.3 Treinamento dos Modelos Classificadores de Padrões

Para iniciar o processo de treinamento, usou-se uma função para recortar os dígitos novamente da imagem grande e, de acordo com as conhecidas posições deles nela, fazer a rotulação, de modo que se obtivesse dois vetores de 5 mil elementos cada, sendo o primeiro com imagens de dígitos e o segundo com seus rótulos de classe correspondentes. Feito isso, empregaram-se três passos no processamento de dados para as amostras de treinamento com seus respectivos rótulos.

O primeiro passo foi permutar o vetor de imagens, juntamente com o de rótulos. Par isso, usou-se uma função para definir um estado randômico, que era passado para outra função de permutação para rearranjar os índices em função do grau de aleatoriedade em questão. Esses índices eram então levados em conta para reorganizar o vetor de 5 mil imagens, bem como o de rótulos, provocando um grau de desordem, mas mantendo a correlação.

O segundo passo consistia em verificar se os dígitos estavam com uma inclinação admissível. Nele, usou-se o cálculo do momento central dos dígitos. Se o dígito em análise tivesse um momento de segunda ordem em y com relação ao centro menor que 0,01, tal dígito permanecia daquele jeito.

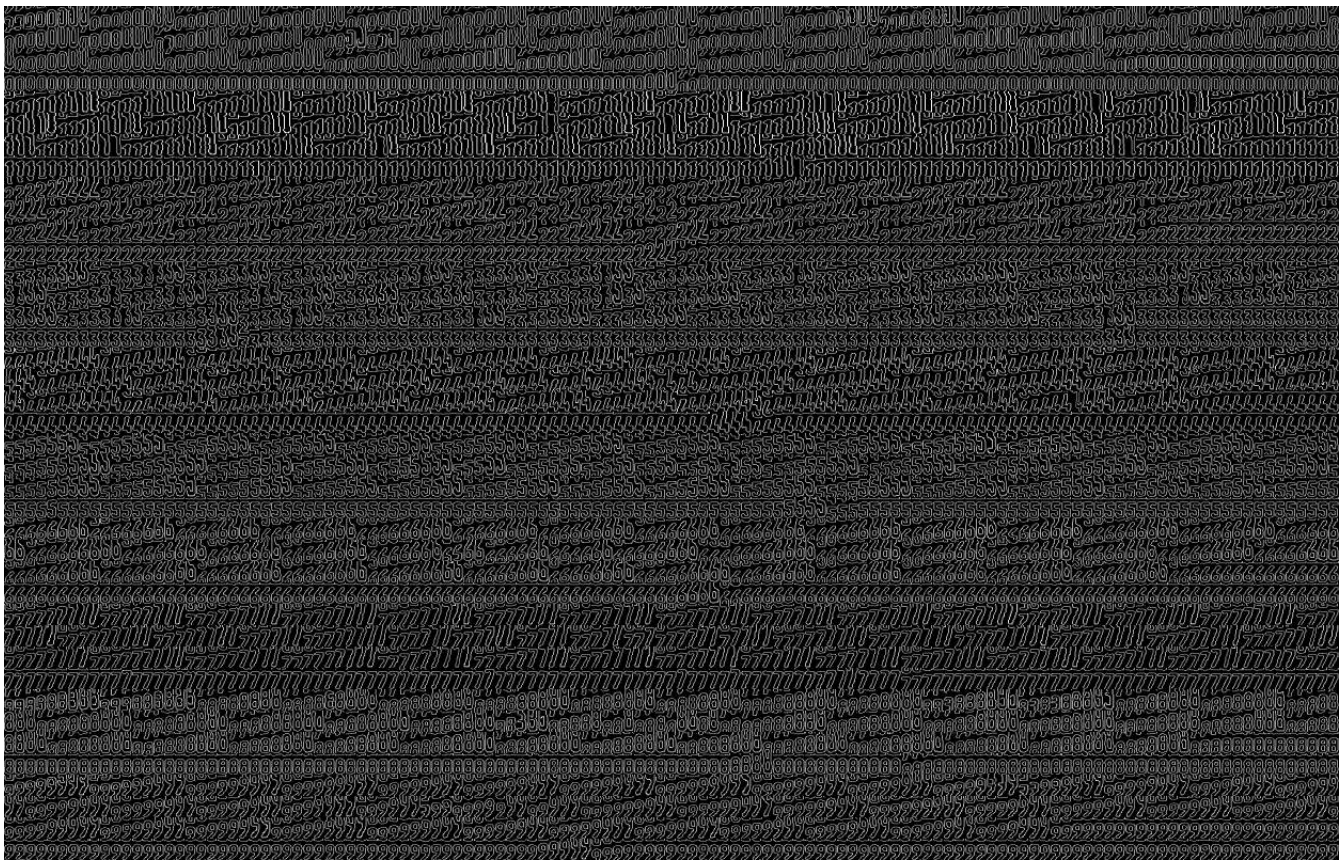


Figura 4.19: Imagem para treinamento de classificadores, composta de 5 mil dígitos.

Caso contrário, o seu ângulo de desvio com relação à vertical era calculado pela combinação de dois momentos de segunda ordem e o valor era posto numa matriz de rotação. Esta era passada como parâmetro para uma função para consertar a inclinação da imagem.

O último passo era dado pela execução do histograma de gradientes orientados, mais conhecido como HOG (Subseção 2.3), em cada um das imagens de dígitos. Utilizou-se o operador Sobel em x e em y, como de praxe, para que então pudesse obter tanto a magnitude quanto o ângulo do gradiente de cada píxel. Na sequência, dividiu-se a imagem 20×10 em quatro células de 10×5. Então, separou-se o vetor de ângulos de cada uma das quatro células em 16 caixas, de tal forma que ângulos no intervalo $0^o \leq \phi < 22,5^o$, em que ϕ é o ângulo do gradiente, ficassem na primeira caixa, aqueles no intervalo $22,5^o \leq \phi < 45^o$, na segunda caixa, e assim por diante, até o intervalo $337,5^o \leq \phi < 360^o$. Acompanhados dos valores de ângulo, os respectivos valores de magnitude foram separados também, formando então um histograma, no qual as barras verticais eram constituídas do somatório das magnitudes em cada caixa. Por fim, os quatro histogramas foram todos colocados em um vetor unidimensional, o qual foi normalizado e submetido ao cálculo da distância de Hellinger (para cada célula) por meio da norma de Frobenius (L^2). Para formar o conjunto amostral final, o terceiro passo foi repetido até que zerasse a quantidade de 5 mil imagens minúsculas.

Finalizado o processamento de amostras, repartiu-se o conjunto amostral final em dois, ficando com 90% de amostras de treinamento e 10% de amostras de teste. E seus respectivos rótulos

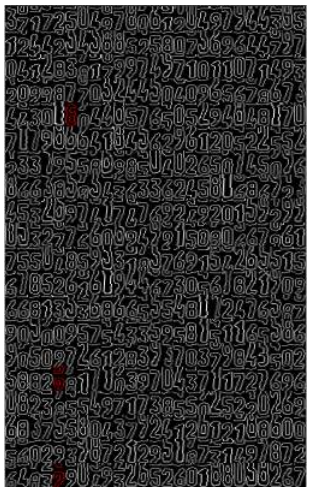
também foram repartidos, mantendo a correlação. Daí, seguiu-se para a etapa a parte de ajuste de parâmetros dos modelos de classificação de padrão, visando sempre ao maior índice de acertos. A seguir, os treinamentos para SVM, MLP e kNN são relatados. Ressalta-se que, para armazenar o resultado do treinamento (estrutura de objeto Python interpretável apenas por computador) nos três casos, foi utilizado o módulo Pickle, o que demandou um treinamento dos três classificadores para cada versão Python, posteriormente. Tal módulo permitia a escrita do resultado em um arquivo, assim como a leitura deste, economizando bastante tempo de processamento por não ser necessário repetir o treinamento a cada iteração no âmbito da aplicação. Para criar os três modelos de classificação, foram bibliotecas do *Scikit-Learn Developers* que proviam algoritmos para a implementação dos classificadores em *software*. No treinamento, o menor índice de erro foi encontrado para o SVM, como é aludido abaixo. Por isso, este foi o classificador de padrões escolhido para fazer o reconhecimento dos dígitos do hidrômetro.

i. Classificador Support Vector Machine (SVM)

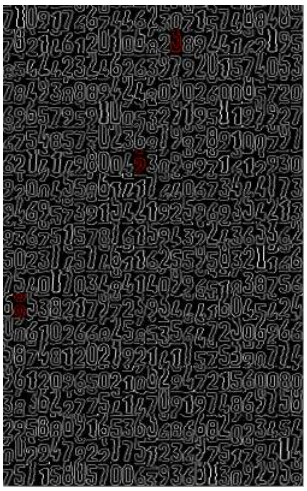
Foram feitas várias tentativas acerca do parâmetro de penalidade para erro C do coeficiente do *kernel* γ. Os melhores valores para esses parâmetros foram:

- C=*2,67* ; e
- γ=*5,383*.

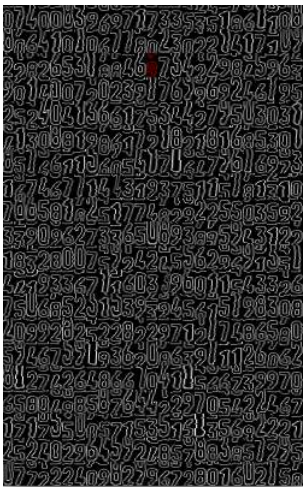
O restante dos parâmetros permaneceu com seus valores padrões. Como exemplo, três resultados para esses valores com três estados randômicos diferentes E_{rand} são exibidos na Figura 4.20. Observa-se que os dois primeiros resultados, da esquerda para direita, têm uma taxa de erro de 0.6%, já no último houve somente um erro, abaixando a taxa para 0.2% apenas de erro. Além disso, nota-se as matrizes de confusão das duas primeiras (Subfiguras 4.20(d) e 4.20(e)) apontam que o classificador confundiu uma vez o nove com três em dois tipos diferentes de permutação.



(a) Amostras de teste, $E_{rand} = 123$.



(b) Amostras de teste, $E_{rand} = 377$.



(c) Amostras de teste, $E_{rand} = 321$.

```
[[47  0  0  0  0  0  0  0  0  0]
 [ 0 45  0  0  0  0  0  0  0  0]
 [ 0  0 47  0  0  0  0  0  0  1]
 [ 0  0  0 43  0  0  0  0  0  0]
 [ 0  0  0  0 47  0  0  0  0  0]
 [ 0  0  0  0  0 58  0  0  0  0]
 [ 0  0  0  0  0  0 55  0  0  0]
 [ 0  0  0  0  0  0  0 51  0  0]
 [ 0  0  0  0  0  0  1  0 53  0]
 [ 0  0  0  1  0  0  0  0  0 51]]
```

(d) Matriz de confusão, $E_{rand} = 123$.

```
[[48  0  0  0  0  0  0  0  0  0]
 [ 0 50  0  0  0  0  0  0  0  0]
 [ 0  0 52  0  0  0  0  0  0  1]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 53  0  0  0  0  0]
 [ 0  0  0  1  0 51  0  0  0  0]
 [ 0  0  0  0  0  0 53  0  0  0]
 [ 0  0  0  0  0  0  0 55  0  0]
 [ 0  0  0  0  0  0  0  0 38  0]
 [ 0  0  0  1  0  0  0  0  0 60]]
```

(e) Matriz de confusão, $E_{rand} = 377$.

```
[[45  0  0  0  0  0  0  0  0  0]
 [ 0 57  0  0  0  0  0  0  0  0]
 [ 0  0 61  0  0  0  0  0  0  0]
 [ 0  0  0 44  0  0  0  0  0  0]
 [ 0  0  0  0 40  0  0  0  0  0]
 [ 0  0  0  0  0 51  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  0  0  0  0  0  0 55  0  0]
 [ 0  0  0  0  0  0  0  0 46  1]
 [ 0  0  0  0  0  0  0  0  0 48]]
```

(f) Matriz de confusão, $E_{rand} = 321$.

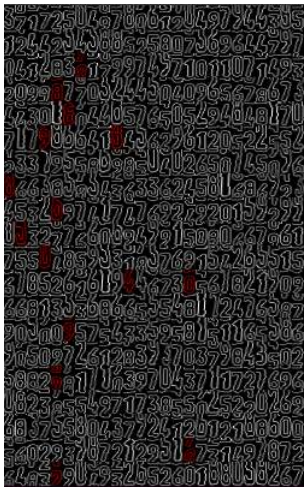
Figura 4.20: Alguns dos treinamentos executados para SVM.

ii. Classificador Multilayer Perceptron (MLP)

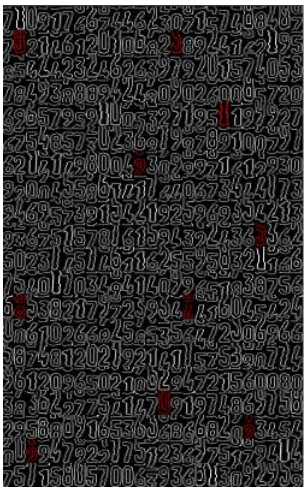
Vários testes foram realizados para o MLP, também, que por sinal tinha um *backpropagation* resiliente, o que significa que ele podia ir até o fim da rede, voltar para o início e depois para o fim várias vezes. Os parâmetros alterados dele foram o número de camadas ocultas N_c, a taxa de aprendizagem inicial a_o, o estado randômico e_{rand}, a tolerância δ_{opt} e termo de penalidade para normalização (L^2) α. Os valores da otimização foram os seguintes:

- $N_c = 380$;
- $a_o = 10^{-2}$;
- $e_{rand} = 1$;
- $\delta_{opt} = 10^{-5}$; e
- $\alpha = 10^{-5}$.

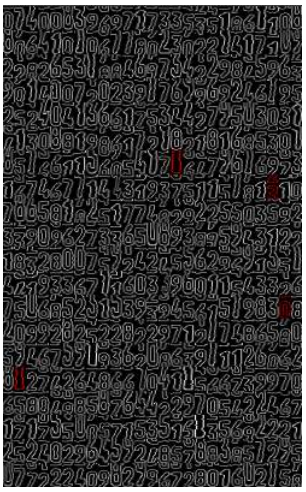
Variando o estado randômico E_{rand} das amostras para o teste, as taxas de erro registradas como resultados do treinamento foram de 3,0%, 2,0% e 0,8%, respectivamente, para os graus de desordem 123, 377 e 321, como mostra a Figura 4.21. Ao analisar as matrizes de confusão, verificou-se que o MLP confundiu três vezes o oito com o zero, o dois com o nove, e duas vezes o nove com o dois, o oito com o seis, o cinco com o três, o um com o zero.



(a) Amostras de teste, $E_{rand} = 123$.



(b) Amostras de teste, $E_{rand} = 377$.



(c) Amostras de teste, $E_{rand} = 321$.

```
[[45  0  0  0  1  0  0  0  1  0]
 [ 0 45  0  0  0  0  0  0  0  0]
 [ 0  0 47  0  0  0  0  0  0  1]
 [ 0  0  0 43  0  0  0  0  0  0]
 [ 0  0  0  0 46  0  1  0  0  0]
 [ 0  0  0  1  0 57  0  0  0  0]
 [ 0  0  0  0  0  1 54  0  0  0]
 [ 0  0  0  0  0  0  0 51  0  0]
 [ 3  0  0  0  0  0  2  0 48  1]
 [ 0  0  2  1  0  0  0  0  0 49]]
```

(d) Matriz de confusão, $E_{rand} = 123$.

```
[[47  0  0  0  0  0  0  0  1  0]
 [ 1 49  0  0  0  0  0  0  0  0]
 [ 0  0 50  0  0  0  0  0  0  3]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 52  0  0  1  0  0]
 [ 0  0  0  2  0 50  0  0  0  0]
 [ 0  0  0  0  0  0 53  0  0  0]
 [ 0  0  0  0  0  0  0 55  0  0]
 [ 0  0  0  0  0  0  0  0 37  1]
 [ 0  0  0  1  0  0  0  0  0 60]]
```

(e) Matriz de confusão, $E_{rand} = 377$.

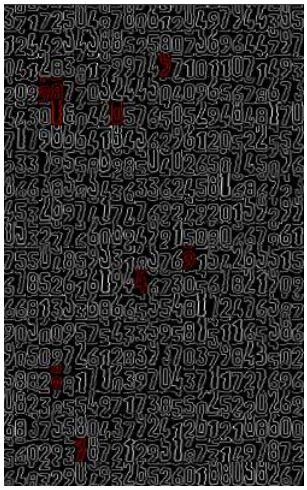
```
[[44  0  0  0  0  0  0  0  1  0]
 [ 2 55  0  0  0  0  0  0  0  0]
 [ 0  0 61  0  0  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0  0  1]
 [ 0  0  0  0 40  0  0  0  0  0]
 [ 0  0  0  0  0 51  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  0  0  0  0  0  0 55  0  0]
 [ 0  0  0  0  0  0  0  0 47  0]
 [ 0  0  0  0  0  0  0  0  0 48]]
```

(f) Matriz de confusão, $E_{rand} = 321$.

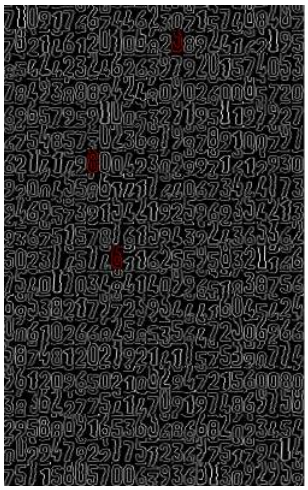
Figura 4.21: Alguns dos treinamentos executados para MLP.

iii. Classificador k-Nearest Neighbor (kNN)

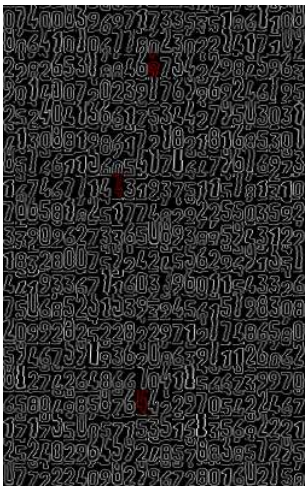
Com o classificador kNN, repetiram-se os mesmos graus de desordem usados para o SVM e o MLP. Naquela mesma sequência, foram obtidas taxas de erro de 1,8%, 0,6% e 0,6% para uma quantidade de vizinhos *k=3*. Isso é evidenciado pela Figura 4.22. Nesse caso, notou-se que foram feitas duas confusões entre o oito e o zero e outras duas entre o nove e o dois.



(a) Amostras de teste, $E_{rand} = 123$.



(b) Amostras de teste, $E_{rand} = 377$.



(c) Amostras de teste, $E_{rand} = 321$.

```
[[47  0  0  0  0  0  0  0  0  0]
 [ 1 44  0  0  0  0  0  0  0  0]
 [ 0  0 48  0  0  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0  0  0]
 [ 0  0  0  0 46  0  1  0  0  0]
 [ 0  0  0  0  0 58  0  0  0  0]
 [ 0  0  0  0  0  0 55  0  0  0]
 [ 0  0  0  0  0  0  0 51  0  0]
 [ 2  0  0  1  0  0  0  0 51  0]
 [ 0  0  2  1  0  0  0  1  0 48]]
```

(d) Matriz de confusão, $E_{rand} = 123$.

```
[[48  0  0  0  0  0  0  0  0  0]
 [ 0 50  0  0  0  0  0  0  0  0]
 [ 0  0 53  0  0  0  0  0  0  0]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 53  0  0  0  0  0]
 [ 0  0  0  1  0 51  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  1  0]
 [ 0  0  0  0  0  0  0 55  0  0]
 [ 1  0  0  0  0  0  0  0 37  0]
 [ 0  0  0  0  0  0  0  0  0 61]]
```

(e) Matriz de confusão, $E_{rand} = 377$.

```
[[45  0  0  0  0  0  0  0  0  0]
 [ 0 57  0  0  0  0  0  0  0  0]
 [ 0  0 61  0  0  0  0  0  0  0]
 [ 0  0  0 44  0  0  0  0  0  0]
 [ 0  0  1  0 38  0  1  0  0  0]
 [ 0  0  0  0  0 51  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  0  0  0  0  0  0 55  0  0]
 [ 0  0  0  1  0  0  0  0 46  0]
 [ 0  0  0  0  0  0  0  0  0 48]]
```

(f) Matriz de confusão, $E_{rand} = 321$.

Figura 4.22: Alguns dos treinamentos executados para kNN.

4.2.9 Reconhecimento de Dígitos e Publicação de Dados

Para concluir o código fonte, com o objetivo de fazer o reconhecimento dos dígitos capturados a cada período de leitura, o módulo *digitsRecognizer* foi desenvolvido. Seu acionamento se dava se, e somente se, *digitsDectector* tiver detectado os sete dígitos no visor do hidrômetro, o que foi a primeira regra de plausibilidade aplicada.

Quando invocado, o algoritmo procurava qual o método classificador se queria usar. Ao descobri-lo, seguia para um adaptador de versão de Python que alterava a codificação no momento da leitura, caso a versão de Python fosse posterior à 3.0, para interpretar a estrutura de objeto em Pickle referente ao modelo escolhido. Então, o modelo e os recortes de dígitos eram passados para a função de predição. Esta, iterativamente, lia o recorte de dígito 20×10, remodelava-o de tal forma que virasse um vetor unidimensional 1×200, checava por intermédio de momentos centrais se a inclinação do dígito era admissível, rotacionando-o caso a inclinação não o fosse, aplicava HOG nele e fazia a predição usando as distâncias de Hellinger confrontadas com o modelo classificador. Isso era feito para os sete dígitos detectados, iterativamente.

Em seguida, para saber a leitura atual, transformava-se aquela sequência de caracteres em uma *string*. Da mesma maneira, para saber a o consumo instantâneo corrente C_{cor}, tal sequência era transformada em um valor de ponto flutuante com três casas decimais. Nesse caso, o valor lido era dado em metros cúbicos. Depois, as métricas da medição anterior tais como tempo t_{ant}, consumo instantâneo C_{ant}, vazão estimada Q_{ant} e vazão média $Q_{a_{ant}}$ eram lidas do arquivo

prevHydroMetrics.var.

Dando sequência, checava-se se o horário atual t_{cor} era maior do que o interior através dos Unix *timestamp*, pois queria garantir que o sistema não registrasse coisa alguma no arquivo *hydrometer.log* se não estivesse em sincronia com o *Coordinated Universal Time* (UTC). Estando em sincronia, o algoritmo extraia o horário UTC, bem como o *timestamp*, calculava a variação de volume ΔV e de tempo Δt mediante $C_{cor} - C_{ant}$ e $t_{cor} - t_{ant}$, respectivamente. Daí, fazia $^{\Delta V}/_{\Delta t}$ para estimar a vazão corrente Q_{cor}, e a partir dela encontrava a vazão média $Q_{a_{cor}}$ empregando $^{(Q_{cor}+Q_{a_{ant}})}/_{2}$.

Depois disso, verificava-se se a leitura corrente era maior ou igual à leitura anterior, o que foi a segunda regra de plausibilidade imposta. Dentro da mesma estrutura de condição, também se conferia se a variação de volume ΔV era menor que a variação de vazão MIN_dV necessária para habilitar a leitura do hidrômetro, e analisava-se se a vazão corrente Q_{cor} era menor ou igual à vazão nominal Qn característica do hidrômetro multiplicada por um fator de segurança FoS (Subseção 4.2.1.5). Essa última condição foi a terceira e última aplicada como teste de plausibilidade.

Se a leitura atual fosse maior que a anterior e as outras duas condições fossem verdadeiras, o programa salvava o recorte da região de interesse segmentada com a detecção de dígitos (Subfigura 4.18(b)) e a imagem inteira com a localização da região de interesse (Subfigura 4.14(c)) na pasta do servidor *html* (Subseção 4.2.1.2). Além disso, armazenava-se o recorte da região de interesse colorida (Figura 4.15) em outra pasta específica, também acessível pelo servidor, colocando o *timestamp* registrado no momento do reconhecimento como nome. Todas as imagens mencionadas eram salvas em formato *jpeg* com fator de compressão igual a 75%, o que permitia os recortes terem cerca de 7 KB e a imagem inteira ter por volta de 43 KB. Ainda dentro do escopo, uma única função fazia a atualização dos arquivos *prevHydroMetrics.var* e *hydrometer.log*, sobrescrevendo e anexando métricas de medição (Subseção 4.2.1.2), respectivamente.

Por outro lado, se a leitura corrente fosse igual à atual e as outras condições verdadeiras, o único procedimento executado era atualizar as métricas de medição do arquivo *prevHydroMetrics.var*.

É importante salientar que era possível conduzir testes no programa através do modo *debug* criado em ambos os casos. Nesse modo, basicamente, exibiam-se a imagem inteira com localização de região de interesse (Subfigura 4.14(c)) e o recorte segmentando da região de interesse (Subfigura 4.18(b)), além dos dados de métricas sendo escritos no terminal periodicamente. A vantagem dele era que se podia especificar um tempo menor que 60 segundos como intervalo de atualização. Este, na verdade, era razoável quando maior ou igual a 25 segundos, pelo fato da maior vazão do hidrômetro ser de 0,04 L/s.

4.2.10 Zeramento da Máquina de Visão Computacional

Para que o algoritmo tivesse um zeramento de máquina, ou seja, um ponto de partida definido em termos do consumo instantâneo inicial e outras métricas de medição, surgiu-se a necessidade de criar um *script* em Bash que promovesse um trabalho nesse sentido. Tal *script* executa dois programas em paralelo e outro em seguida. Os dois primeiros são o criador e inicializador de

arquivos e o visualizador do visor. O último é o salvador de região de interesse.

4.2.10.1 Visualizador do Visor do Hidrômetro

O visualizador do visor do hidrômetro, apelidado *viewHydroDisplay*, foi criado a partir dos módulos *roiLocator* e *digitsDetection*. Seu objetivo era, obviamente, permitir ao usuário um acompanhamento em tempo real da visão computacional feita na região do visor do hidrômetro. Além de exibir os dígitos, através dos módulos supracitados, era possível perceber os melhores ângulos da *webcam* para que a detecção dos sete dígitos fosse suave e precisa, o que era efetivado sob uma boa localização da região de interesse.

4.2.10.2 Criador e Inicializador de Arquivos de Recorrência do Software

O criador e inicializador de arquivos, cujo módulo foi denominado *hydrometerSetup*, era executado por intermédio do console *Xterm*. Neste, inseria-se a leitura constatada pelo módulo *viewHydroDisplay* rodando em paralelo, como é retratado pela Figura 4.23. Quando o usuário desse *"enter"*, o programa disparava três processos diferentes. O primeiro consistia de criar o arquivo *hydrometerSetup.dat*, no qual ele o horário e data local em UTC, a leitura inicial e o nome da imagem da região de interesse colorida determinado pelo *timestamp*. O segundo processo era encarregado de criar o arquivo *hydrometer.log* se não existente (ou anexar se existente) e então anexar métricas tais como UTC local, *timestamp,* leitura e consumo instantâneo. O restante do conjunto de métricas era zerado. O último processo, por sua vez, era bem equivalente ao anterior, com a exceção do nome do arquivo. Desta vez, os mesmos dados eram sempre sobrescritos no arquivo recém-criado chamado *prevHydroMetrics.var*.

4.2.10.3 Salvador de Região de Interesse

Assim que todos os processos da execução paralela eram terminados, era importante que se salvasse o recorte da região de interesse para possíveis recorrências futuras. Dessa maneira, o código fonte era empregado até a etapa do recorte para o fazer, e o salvamento da imagem com o nome apropriado era dado pela abertura do arquivo *hydrometerSetup.dat* e a extração do *timestamp* referente a imagem. Como era o recorte colorido da região de interesse, este era salvo diretório *ROI_PATH*.

4.2.11 Instalador e Desinstalador

O instalador (*install.sh)* foi elaborado para colocar os arquivos nos diretórios corretos da distribuição do *software*. O desinstalador, obviamente, procura fazer o contrário. A seguir, encontram-se informações sobre módulos e *scripts* do *software*, bem como de seus respectivos diretórios.

Scripts em Bash:

1. hydrometerSetup.sh
2. hydrometerOCR.sh
3. install.sh
4. uninstall.sh

Códigos em Python

1. Módulos Principais
   (a) main.py
   (b) htmlGen.py
   (c) hydrometerSetup.py
   (d) viewHydroDisplay.py
   (e) saveRoi.py
2. Módulos Auxiliares
   (a) htmlGen.pyc || htmlGen.cpython-35.pyc || htmlGen.cpython-35.pyc
   (b) versionHandler.pyc || versionHandler.cpython-35.pyc || versionHandler.cpython-36.pyc
   (c) roiLocator.pyc || roiLocator.cpython-35.pyc || roiLocator.cpython-36.pyc
   (d) digitsDetector.pyc || digitsDetector.cpython-35.pyc || digitsDetector.cpython-36.pyc
   (e) digitsRecognizer.pyc || digitsRecognizer.cpython-35.pyc || digitsRecognizer.cpython-36.pyc

Ação do Instalador:

/usr/local/<BRAND>/

1. hydrometerSetup.sh
2. hydrometerOCR.sh
3. main:
   (a) main.py
   (b) mainCT.py
   (c) viewHydroDisplay.py
   (d) saveRoi.py
   (e) hydrometerSetup.py
   (f) versionHandler.py
   (g) versionHandler.pyc
4. ___pycache___:

(a) roiLocator.pyc
(b) digitsDetector.pyc
(c) digitsRecognizer.pyc
(d) versionHandler.cpython-35.pyc
(e) roiLocator.cpython-35.pyc
(f) digitsDetector.cpython-35.pyc
(g) digitsRecognizer.cpython-35.pyc
(h) versionHandler.cpython-36.pyc
(i) roiLocator.cpython-36.pyc
(j) digitsDetector.cpython-36.pyc
(k) digitsRecognizer.cpython-36.pyc

5. ml-models:
    (a) pickle:
        i. pickle2.7
            A. digits_knn.pkl
            B. digits_svm.pkl
            C. digits_mlp.pkl
        ii. pickle3.5
            A. digits_knn.pkl
            B. digits_svm.pkl
            C. digits_mlp.pkl
        iii. pickle3.6
            A. digits_knn.pkl
            B. digits_svm.pkl
            C. digits_mlp.pkl

/var/log/<BRAND>/

1. hydrometer.log

/usr/local/etc/<BRAND>/

1. hydrometerSetup.dat
2. prevHydroMetrics.var

/srv/http/<BRAND>/

1. index.html

2. ROIs:
    (a) <timestamps>.jpg
3. currentIms:
    (a) roi.jpg
    (b) digits.jpg

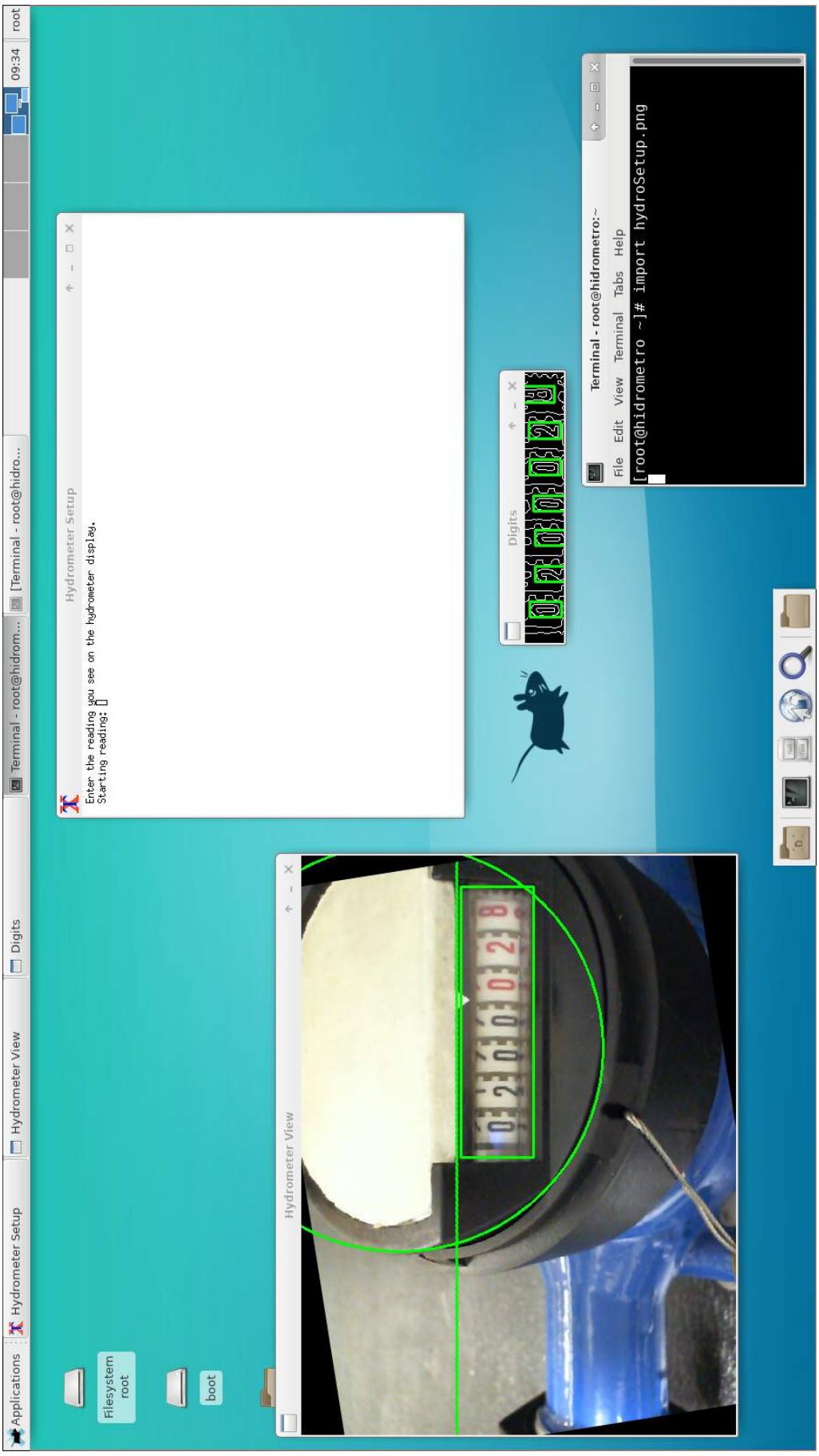


Figura 4.23: Execução do *script* para configuração inicial do *software* de medição.

Capítulo 5

Resultados

Os resultados adquiridos, exceto para a Seção 5.7, foram todos obtidos por meio de um teste conduzido com o código fonte subsidiado pelo classificador SVM, por ter apresentado as maiores taxas de acerto para o reconhecimento de padrões, como relata a Seção 5.1. Tal teste foi executado por mais de 26 horas seguidas, utilizando Crontab o com um período de execução recorrente de um minuto. O código fonte, por sua vez, tinha a permissão de cinco tentativas no máximo para fazer o reconhecimento de dígitos. Ao fim, conseguiu-se um conjunto amostral de 773 medições. O monitoramento pela internet não foi visualizado ao longo do extenso período de teste. Em contrapartida, todos as métricas de medição podiam ser lidas no arquivo de *log*, o qual permitiu postar importantes gráficos para avaliar o comportamento do sistema de visão computacional. Por intermédio desse mesmo arquivo, foi possível fazer uma constatação importante acerca da quantidade de tentativas por execução que a visão artificial precisava para detectar os sete dígitos fazer o OCR deles, ao usar um conjunto amostral de 4065 amostras (linhas de métricas no arquivo de log). O tempo em que a CPU ficava ocupada pelo *software* era de 9 segundos em média.

5.1 Treinamento do Aprendizado de Máquina

Depois de ter ajustado os parâmetros para cada modelo, como indicado na Subseção 4.2.8.3, escolheram-se dez valores de estado aleatório para permutação das amostras e rodou-se o treinamento para os três classificadores com cada um desses valores. Ao final, percebeu-se que a média de erro percentual do SVM era menor que a do kNN, que era menor que o MLP, conforme a Tabela 5.1. Tal resultado sugere que houve um *overfitting* no treinamento do MLP, uma vez que se esperava o menor erro percentual para este, por ter a capacidade de aprender muito melhor as informações sobre o conjunto amostral quando treinado apropriadamente.

5.2 Supervisionamento Remoto da Medição

O supervisionamento remoto via internet da medição de vazão foi exequível. Como resultado, o sistema de visão é responsável por atualizar *online* a foto da localização da região de interesse

Tabela 5.1: Treinamento dos classificadores de padrões com diferentes graus de desordem para permutação de amostras.

Estado aleatório	**Erro [%]**		
	SVM	**kNN**	**MLP**
50	1,0	2,2	2,4
100	1,0	1,0	1,8
150	0,4	0,8	2,4
200	0,4	1,0	1,2
250	0,6	0,6	3,4
300	0,6	1,2	3,2
350	0,6	1,8	1,8
400	0,4	1,8	1,8
450	0,4	1,4	2,6
500	1,0	1,6	3,0
Média	0,6	1,3	2,4

e do recorte segmentado da imagem desta com a detecção de dígitos. Além disso, o sistema publica ao usuário os dados mais interessantes tais como a data e o horário local, o valor da leitura, a vazão estimada e o valor resultante da média móvel desta e sua anterior imediata. Isso é explicitado na Figura 5.1 e pode ser verificado, enquanto a *Raspberry* estiver ligada, no endereço `http://medicaohidrometro.alvarestech.com`.

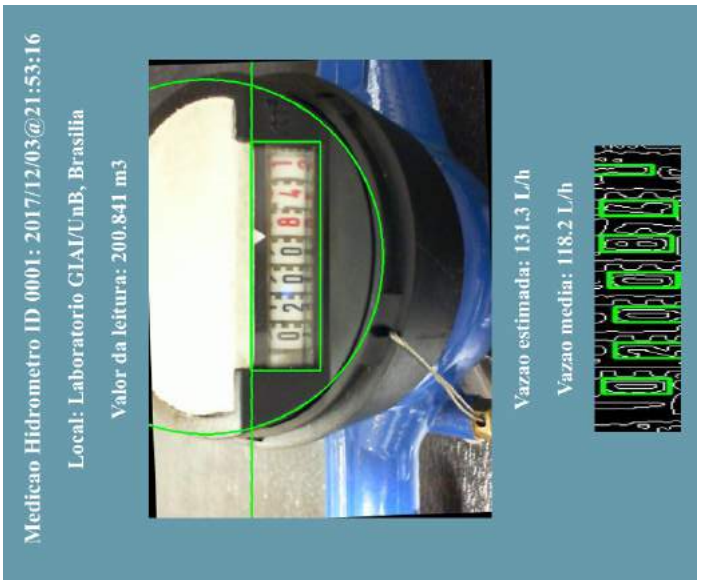


Figura 5.1: Dados divulgados pela página *html* do servidor.

Outra forma encontrada de fornecer um supervisionamento remoto para o usuário do hidrômetro foi publicando todas as métricas de medição feita pela visão artificial em um arquivo de *log*. Uma parte bem pequena do conjunto de dados desse arquivo é mostrada na Figura 5.2. Seu conteúdo pode ser acessado por completo pelo endereço `http://medicaohidrometro.alvarestech.`

com/log/log.txt.

UTC	Timestamp	Reading	Usage[m3]	dV[L]	dt[s]	Q[L/h]	Qma[L/h]	Try[x]
2017/11/22@16:09:08	1511374148	0195318	195.318	2.0	60	118.3	61.5	1
2017/11/22@16:13:10	1511374390	0195327	195.327	9.0	242	133.7	126.0	3
2017/11/22@16:14:08	1511374448	0195329	195.329	2.0	58	123.1	128.4	1
2017/11/22@16:15:08	1511374508	0195331	195.331	2.0	60	118.8	120.9	1
2017/11/22@16:17:09	1511374629	0195336	195.336	5.0	121	147.9	133.4	2
2017/11/22@16:18:10	1511374690	0195338	195.338	2.0	61	116.7	132.3	3
2017/11/22@16:20:11	1511374811	0195342	195.342	4.0	121	118.5	117.6	4
2017/11/22@16:21:08	1511374868	0195344	195.344	2.0	57	124.9	121.7	1
2017/11/22@16:25:10	1511375110	0195353	195.353	9.0	242	133.4	129.1	3
2017/11/22@16:26:10	1511375170	0195355	195.355	2.0	60	119.9	126.6	2
2017/11/22@16:28:11	1511375291	0195360	195.360	5.0	121	148.4	134.1	4
2017/11/22@16:29:08	1511375348	0195362	195.362	2.0	57	125.3	136.9	1
2017/11/22@16:30:12	1511375412	0195364	195.364	2.0	64	111.9	118.6	5
2017/11/22@16:31:08	1511375468	0195366	195.366	2.0	56	127.5	119.7	1
2017/11/22@16:32:08	1511375528	0195368	195.368	2.0	60	118.8	123.1	1
2017/11/22@16:33:08	1511375588	0195371	195.371	3.0	60	177.7	148.2	1
2017/11/22@16:34:08	1511375648	0195373	195.373	2.0	60	118.1	147.9	1
2017/11/22@16:36:09	1511375769	0195377	195.377	4.0	121	118.1	118.1	2
2017/11/22@16:37:09	1511375829	0195379	195.379	2.0	60	119.8	118.9	1
2017/11/22@16:39:10	1511375950	0195384	195.384	5.0	121	147.5	133.7	3
2017/11/22@16:40:10	1511376010	0195386	195.386	2.0	60	119.8	133.7	2
2017/11/22@16:42:09	1511376129	0195390	195.390	4.0	119	121.0	120.4	1
2017/11/22@16:44:09	1511376249	0195395	195.395	5.0	120	148.8	134.9	2
2017/11/22@16:45:11	1511376311	0195397	195.397	2.0	62	114.3	131.6	4
2017/11/22@16:46:10	1511376370	0195399	195.399	2.0	59	121.9	118.1	2
2017/11/22@16:47:11	1511376431	0195401	195.401	2.0	61	117.9	119.9	3
2017/11/22@16:48:10	1511376490	0195403	195.403	2.0	59	121.8	119.9	2
2017/11/22@16:50:10	1511376610	0195408	195.408	5.0	120	149.8	135.8	2
2017/11/22@16:51:12	1511376672	0195410	195.410	2.0	62	115.8	132.8	4

Figura 5.2: Pequena amostra de métricas de medição disponíveis pelo servidor.

5.3 Consumo Instantâneo

No teste consumo instantâneo foi plotado em função do tempo e obteve-se o gráfico da Figura 5.3. Verificou-se por inspeção, que o maior valor de consumo registrado era de 198,782 m^3 e o menor, de 195,316 m^3, resultando numa variação de volume de 3,466 m^3. Já que o intervalo completo de medição era 26 horas, encontrou-se uma vazão de 131,79 L/h, a qual tinha um erro 9,26% com relação a vazão constada por testes no hidrômetro analógico.

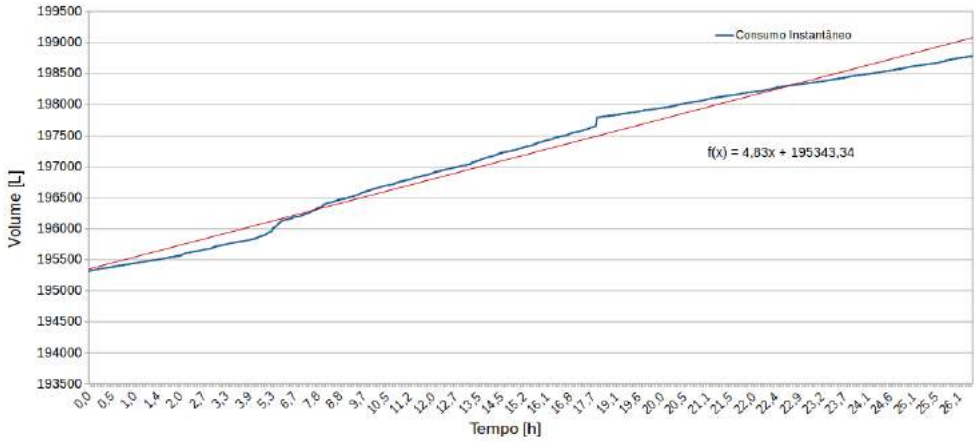


Figura 5.3: Consumo instantâneo e sua curva de ajuste linear no decorrer do tempo.

A regressão linear dos pontos da curva do consumo instantâneo, no entanto, reflete bem o

valor médio da detecção de variação de volume de uma leitura para outra expressa na Seção 5.4. Isso porque o valor do coeficiente angular da equação de regressão linear é 4,83, o que significa um incremento de volume em litros a cada iteração. Tal coeficiente é aproximadamente igual ao valor médio da variação do volume encontrado, que era de 4,49 L.

Ainda em relação ao gráfico, algumas irregularidades de medição podem ser notadas. Por exemplo, por volta da 18^a hora decorrida (09h53), o valor de volume foi alterado abruptamente. Para analisar o que aconteceu ali, recorreu-se ao arquivo de *log*. Descobriu-se que, como é exibido na Figura 5.4, por volta das dez horas, o sistema tinha parado de funcionar e perdurado assim por quase uma hora, o que indica que alguém deve ter desligado as luzes do laboratório, fazendo com que a o sistema de visão entrasse em hibernação por todo aquele período. Quando as luzes foram acesas novamente e a visão artificial voltou a trabalhar, já havia gasto cerca de 140 litros de água.

2017/11/23@09:43:12	1511437392	0197632	197.632	20.0	542	132.7	126.9	5
2017/11/23@09:44:09	1511437449	0197634	197.634	2.0	57	124.6	128.6	2
2017/11/23@09:45:10	1511437510	0197636	197.636	2.0	61	116.5	120.6	3
2017/11/23@09:48:10	1511437690	0197643	197.643	7.0	180	139.8	128.2	2
2017/11/23@09:50:12	1511437812	0197647	197.647	4.0	122	117.2	128.5	5
2017/11/23@09:53:12	1511437992	0197654	197.654	7.0	180	139.7	128.4	5
2017/11/23@10:57:08	1511441828	0197795	197.795	141.0	3836	132.3	136.0	1
2017/11/23@10:58:11	1511441891	0197797	197.797	2.0	63	113.7	123.0	4
2017/11/23@10:59:10	1511441950	0197799	197.799	2.0	59	121.6	117.6	3
2017/11/23@11:00:08	1511442008	0197801	197.801	2.0	58	123.2	122.4	1
2017/11/23@11:02:09	1511442129	0197806	197.806	5.0	121	148.6	135.9	1
2017/11/23@11:03:10	1511442190	0197808	197.808	2.0	61	117.7	133.1	2
2017/11/23@11:04:10	1511442250	0197810	197.810	2.0	60	119.6	118.7	3
2017/11/23@11:05:08	1511442308	0197812	197.812	2.0	58	123.6	121.6	1
2017/11/23@11:06:09	1511442369	0197814	197.814	2.0	61	117.4	120.5	2
2017/11/23@11:07:08	1511442428	0197817	197.817	3.0	59	181.5	149.4	1
2017/11/23@11:08:08	1511442488	0197819	197.819	2.0	60	118.6	150.1	1

Figura 5.4: Histórico do consumo de água com evidência de iluminação cortada (variação de volume de água igual a 141 litros).

5.4 Variação de Volume

A variação de volume de água de uma iteração para outra foi verificada de duas perspectivas. Na primeira, analisou-se a variação da vazão no decorrer do tempo, dada pela Figura 5.5. Nesta, além de ter ficado claro o período em que a variação alcançou a marca de 141 litros, percebeu-se que havia diversos outros pontos irregulares, os quais poderiam ser tanto erro quanto incapacidade de fazer a detecção dos dígitos com cinco tentativas. A Tabela 5.2 traz algumas grandezas estatísticas características do sistema. Nela, o fato de se ter um valor de variação mínima de volume igual a um litro sugere um erro de visão artificial. Isso porque a vazão era de um litro a cada 25 segundos e o sistema de visão era invocado a cada 60 segundos. Isto é, em 60 segundos, estritamente, deveria ser constatado uma gasto de 2 litros de água.

A segunda perspectiva usada para a análise baseou-se no histograma dos dados de variação de volume, como mostra a Figura 5.6. Com ela, sempre para a mesma variação de tempo (60 segundos), constatou-se que o algoritmo detecta até 3 litros de variação de uma leitura para outra

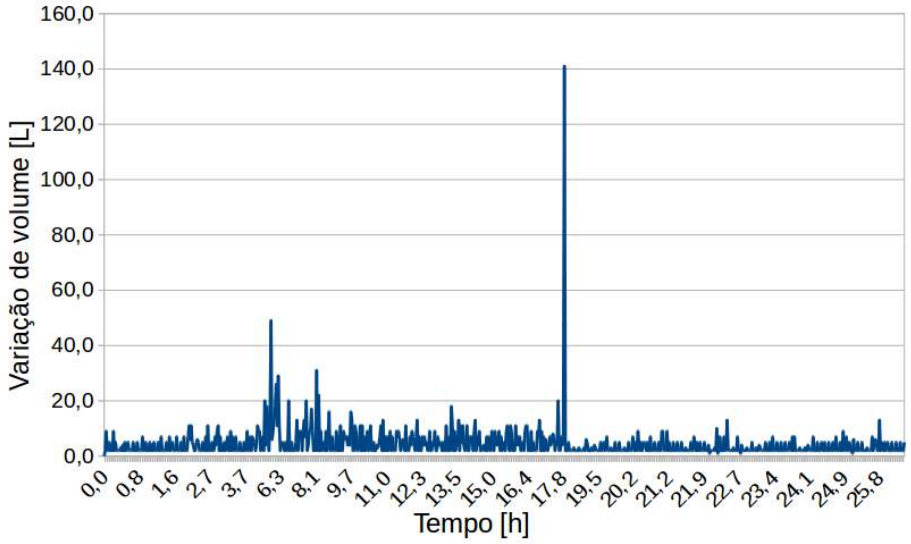


Figura 5.5: Variação de volume de água consumido no decorrer do tempo.

Tabela 5.2: Caracterização estatística da variação de volume consumido.

Variação de volume [L]	
média	4,49
mediana	2,00
moda	2,00
máximo	141,00
mínimo	1,00
desvio padrão	6,31

cerca de 58% das vezes (das 773 medições), e a variação de 4 a 6 litros com uma frequência relativa de 20%.

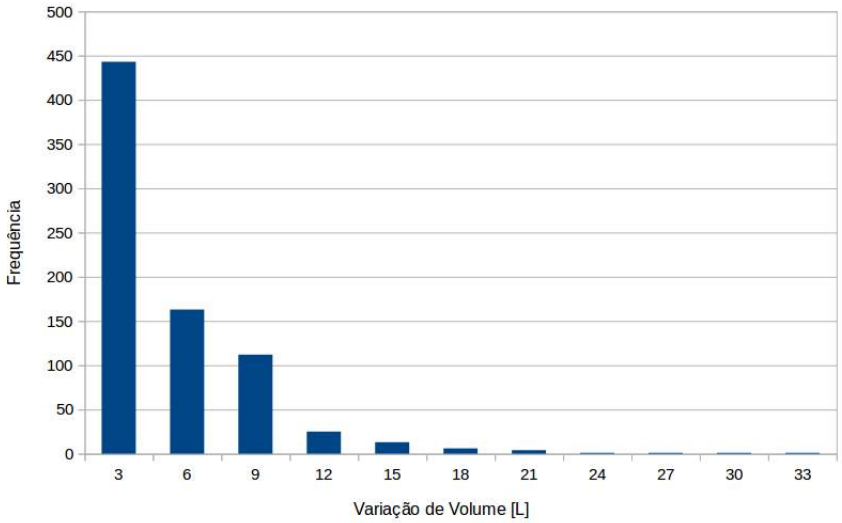


Figura 5.6: Distribuição de probabilidade da variação de volume.

5.5 Variação de tempo

Ao fazer o histograma da variação de tempo na Figura 5.7, verificou-se que se leva de 60 a 120 segundos para reconhecer os sete dígitos e fazer o processamento completo em aproximadamente 70% das vezes. Presume-se que no caso específico de se ter demorado cerca de 840 segundos duas vezes para fazer o reconhecimento seria um sinal de não robustez do programa. Nesse caso, nesse caso, OCR foi executado, mas não se conseguiu prever o valor exato para a medição do consumo. Acredita-se que o erro entre a leitura esperada e a leitura encontrada pelo OCR surtiu uma variação de volume de no mínimo 10 litros, uma vez que o sistema de visão ficou 14 min sem detectar nada, visto que a leitura corrente tinha de ser maior do que a anterior.

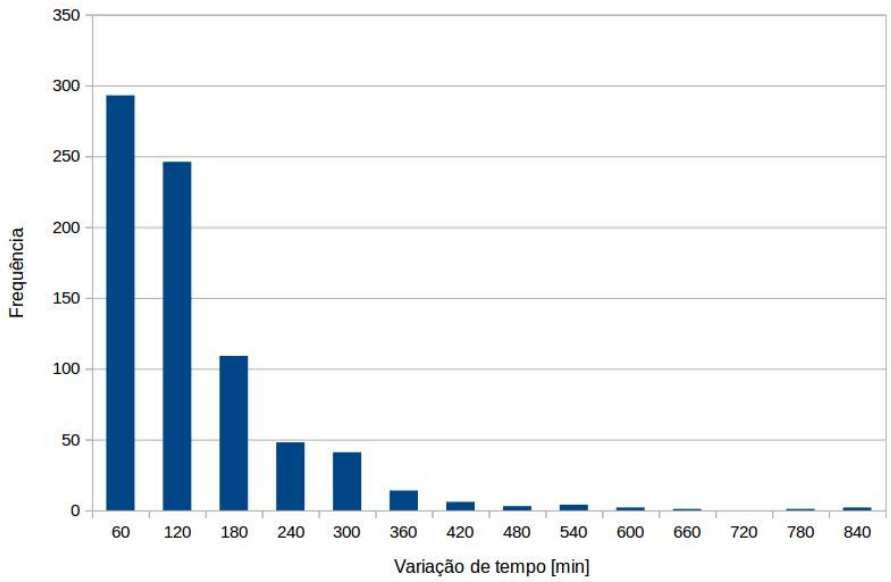


Figura 5.7: Distribuição de probabilidade da variação do tempo.

A Tabela 5.3 revela que a média de variação de tempo de invocação do programa pelo Crontab, com sucesso no reconhecimento dos dígitos, é de 117,37 segundos. Recapitula-se que o maior tempo (3836 segundos) se refere ao período pelo qual a sala ficou com as luzes apagadas.

Tabela 5.3: Caracterização estatística da variação de tempo.

Variação de Tempo [s]	
média	117,37
mediana	62,00
moda	60,00
máximo	3836,00
mínimo	56,00
desvio padrão	104,32

5.6 Vazões Estimada e Média

Nessa parte, plotou-se a vazão estimada, dada pela variação do volume de água divido pela variação do tempo de uma iteração para outra, e a ação do filtro média móvel sobre ela (vazão média), o que é visto na Figura 5.8. Como se lê na Tabela 5.4, a média da vazão estimada é de 128,65 L/h, o que é bem próximo do valor de vazão encontrado pela inclinação da curva de consumo instantâneo da Figura 5.3 (131,79 L/h). No entanto, ambos são razoavelmente menores do que a vazão esperada (144 L/h), detectada por inspeção no circuito hidráulico.

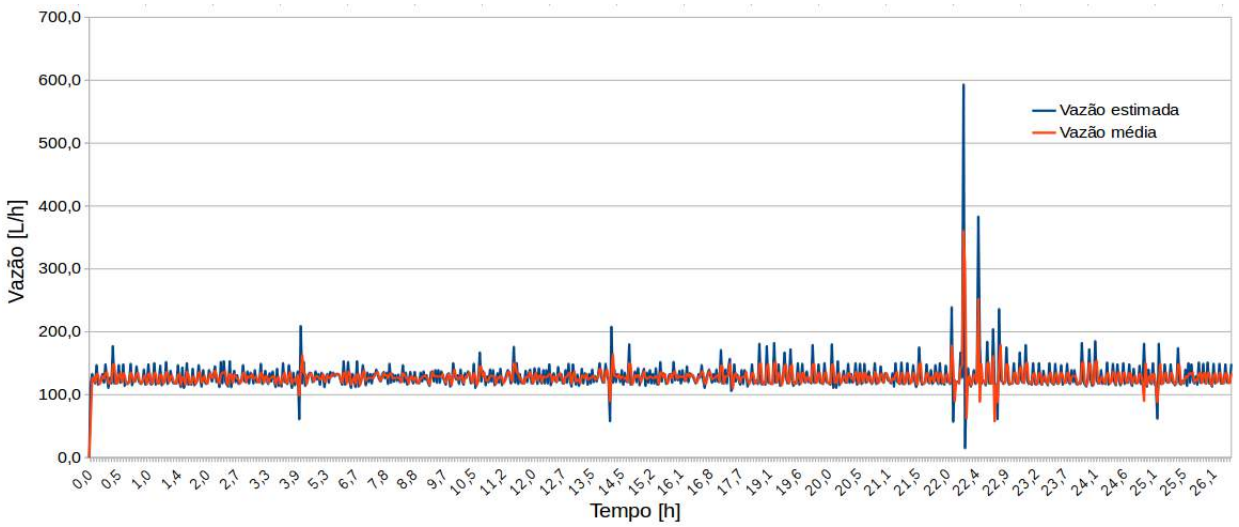


Figura 5.8: Vazões estimada e média no decorrer do tempo.

Tabela 5.4: Caracterização estatística da variação de vazão.

Vazão Estimada [L/h]	
média	128,65
mediana	121,00
moda	118,00
máxima	593,00
mínima	15,00
desvio padrão	25,84

As variações abruptas e grandes que ocorreram na plotagem do gráfico das vazões em função do tempo podem ser um sinal de inconsistência do SVM. Exemplo disso é a varição que se manifestou por volta da 22^a hora de medição (14h15), cujo pico chegou a aproximadamente 600 L/h, como evidencia a Figura 5.9. Ali fica claro que o classificador de padrões errou ao fazer o OCR, pois, para aquelas condições de operação do circuito hidráulico, não havia como se ter uma variação de volume de água de 10 litros dentro de um intervalo de tempo de 60 segundos.

2017/11/23@14:11:08	1511453468	0198221	198.221	2.0	59	121.0	120.1	1
2017/11/23@14:12:08	1511453528	0198223	198.223	2.0	60	118.6	119.8	1
2017/11/23@14:13:12	1511453592	0198226	198.226	3.0	64	167.9	143.3	5
2017/11/23@14:14:08	1511453648	0198228	198.228	2.0	56	127.4	147.6	1
2017/11/23@14:15:08	1511453708	0198238	198.238	10.0	60	593.5	360.4	1
2017/11/23@14:19:08	1511453948	0198239	198.239	1.0	240	15.0	304.2	1
2017/11/23@14:20:12	1511454012	0198241	198.241	2.0	64	112.4	63.7	5
2017/11/23@14:23:08	1511454188	0198248	198.248	7.0	176	142.7	127.6	1
2017/11/23@14:24:08	1511454248	0198250	198.250	2.0	60	118.5	130.6	1

Figura 5.9: Evidência de erro do OCR por causa da variação abrupta de vazão constatada.

5.7 Tentativas de Execução do Programa

Por meio do histograma feito com as quantidades de tentativas e suas frequências de aparecimento, plotou-se o gráfico da Figura 5.10. Com ela, reparou-se que o sistema tende a economizar, ao máximo, recursos do processador da *Raspberry Pi* em cerca de 40% das vezes que o sistema de visão conseguiu fazer o OCR do hidrômetro.

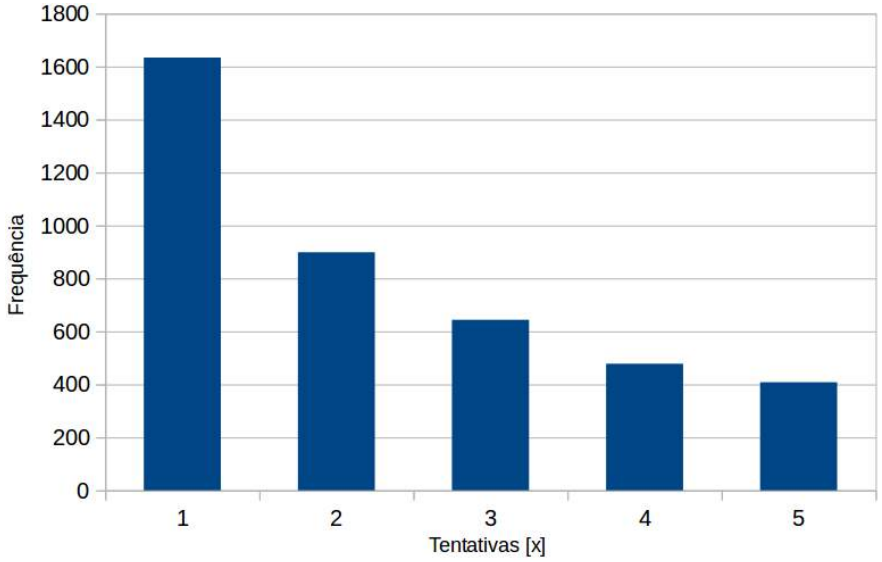


Figura 5.10: Distribuição de probabilidade para a quantidade de tentativas.

Capítulo 6

Conclusão

O trabalho desenvolvido no âmbito da automação de hidrômetros analógicos foi concluído com êxito. Pôde-se executar em tempo real a coleta de leituras e a publicação de métricas de medição de vazão na internet por um sistema de visão computacional feito por uma *webcam* plugada em uma *Raspberry Pi 2*.

O sistema se destacou de forma peculiar com o método de localização de região de interesse criado, ao o fazer dos mais variados ângulos. De forma bem satisfatória, também conseguiu detectar os dígitos no visor do hidrômetro, mesmo quando estes tinham metade de suas *features* cortadas. Ficou levemente lento, mas permaneceu com taxas de atualização de quadros suficientes para acompanhar o deslocar dos dígitos do rolete.

Os algoritmos de OCR também tiveram uma boa parcela na contribuição para o bom funcionamento. O SVM, em especial, tinha a maior taxa de acertos, seguido do kNN e então MLP. Não à toa, o SVM foi escolhido para ser o classificador de padrões a ser usado pelo programa. Acredita-se que, além de bons parâmetros de treinamento para o modelo, a quantidade amostral de imagens de dígitos, associada ao uso de distâncias de Hellinger ao fim do processamento de HOG, tenha sido o ponto chave para o bom reconhecimento de dígitos. No entanto, presume-se que os parâmetros de treinamento para o MLP estariam causando um *overfitting*, uma vez que tal topologia de rede neural é conhecida por apresentar taxas de acerto elevadíssimas.

Por outro lado, a visão artificial empregada possuía algumas limitações. Infelizmente, ela ficou fadada ao funcionamento unicamente dentro do laboratório e sob a constante iluminação de lâmpadas fluorescentes. Além disso, foi necessário colocar uma tampa semitransparente de plástico ao redor da *webcam* para que qualquer reflexo no visor fosse bloqueado.

O sistema de visão ainda tem muito o que melhorar. Como foi visto, no teste feito durante cerca de 26 horas, com uma tolerância de 5 tentativas para reconhecer os dígitos, ele colheu 773 leituras no período para uma colheita via Crontab a cada um minuto. Considerando que a quantidade esperada era de 1580 vezes, levando em consideração todo o tempo em que ficou ligado, o programa funcionou apenas 50% apenas das vezes. E o significado disso é claro: se uma certa aplicação demanda um intervalo de leitura de um minuto, o aconselhável é que se trabalhe com um período de medição que seja a metade disso ao menos, ou seja, 30 segundos.

Um reflexo de quanto o código ainda precisa ser otimizado está no histograma de tentativas (Figura 5.10). Apesar de se perceber uma distribuição normal ao redor da média um (uma vez), o desvio padrão ainda é alto. Isto é, a variação da quantidade de tentativas ainda varia bastante no intervalo de duas a cinco vezes.

Contudo, ainda é possível ter uma estimação da vazão do circuito hidráulico relativamente boa. Através de médias de experimentos, notou-se que ela gira em torno de um valor médio de 130 litros por hora, o que corresponde a uma diferença do valor esperado com uns 9,26% de erro. Tal imprecisão seria dada pelo simples fato de não ser possível fazer a captura dos sete dígitos ao mesmo tempo constantemente. Na verdade, à medida que o rolete de dígitos gira, pode-se ficar alguns segundos sem conseguir fazer a detecção da sequência dos sete dígitos. E esse período de impossível detecção pode coincidir com o momento de leitura da câmera.

Por causa das variações de picos e vales na estimação da vazão, podendo chegar a valores bem discrepantes, dependendo da confusão entre dígitos feita pelo algoritmo de SVM, o valor da média móvel acaba sendo o mais apropriado para descrever a vazão do sistema. Isso porque a média móvel é menos susceptível às variações abruptas.

Não foram feitos testes para outros tipos de hidrômetros analógicos com o sistema de visão. No entanto, acredita-se que o mesmo código desenvolvido não seria aplicável um hidrômetro diferente. Como visto, o algoritmo de visão artificial ainda nem é robusto o suficiente para reconhecer com uma alta frequência os dígitos do hidrômetro para o qual foi treinado. Há vários pontos da técnica de localização, detecção e reconhecimento de dígitos que precisam ser aperfeiçoados e otimizados para tornar o sistema robusto.

6.1 Perspectivas Futuras

Pode-se melhorar o desempenho do sistema em geral por meio de várias maneiras. A começar pela habilidade de executar medições no escuro, a troca da *webcam* por um módulo Picamera com LEDs em um sistema de iluminação difusa e um certo ajuste nos parâmetros da filtragem poderiam tornar o sistema capaz de trabalhar em ambientes escuros. E um suporte fixo, para tal, seria o mais interessante.

No que diz respeito ao código fonte, a implementação de um sistema de visão mais veloz e leve também seria de grande importância. Filtros poderiam ser retirados ou substituídos e uma outra linguagem de programação tal como C++ para OpenCV contribuiria para com um menor tempo de processamento. Talvez, a adoção de dimensões ainda menores do que 640×480 eventualmente poderia ser possível, reduzindo o tempo de execução.

No âmbito do OCR, sugere-se que se construa novamente a foto que contém as pequenas imagens de dígitos do hidrômetro, ainda com um espaço amostral de 5000 imagens, ou maior. Isso porque tal imagem gigante foi construída somente uma vez nas primeiras fases de implementação do algoritmo de detecção de dígitos. Posteriormente, no entanto, este foi aperfeiçoado. Com isso, poderia aumentar as taxas de acertos do OCR ainda mais. Além do mais, parâmetros do modelo

de SVM, kNN e MLP deveriam ser revistos e um novo treinamento, refeito.

Na parte de publicação de métricas de medição, uma interface mais amigável e interativa com o usuário poderia ser desenvolvida. O usuário poderia ser capaz de gerar gráficos de consumo através de um período determinado, inserido no *website*. Além disso, seria muito interessante se houvesse alguma forma de implementar buscas nos bancos de dados de *log* e de região de interesse (ROI), de modo que, assim que data e horário fossem inseridos no sistema para a pesquisa, a ROI colorida correspondente saísse, por exemplo.

Por fim, poderiam ser entregues alguns serviços de monitoramento, tais como alerta de meta de consumo sendo alcançada, aviso por e-mail sobre a detecção de um consumo fora do padrão, podendo ser resultante de vazamento, e disponibilização do valor a pagar bem como do histórico de consumo mensal dos últimos 12 meses.

REFERÊNCIAS

[1] GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing (black & white - text ok, images badly damaged)*. 3. ed. [S.l.]: Prentice Hall, 2007. (3rd Edition). ISBN 013168728X,9780131687288.

[2] SPATIAL Filtering. A. Disponível em: <http://www.coe.utah.edu/~cs4640/slides/Lecture5.pdf>.

[3] NIXON, M. S.; AGUADO, A. S. *Feature extraction & image processing for computer vision*. 3. ed. [S.l.]: Academic Press, 2012. ISBN 0123965497,9780123965493.

[4] TOMASI, C.; MANDUCHI, R. Bilateral filtering for gray and color images. In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. [S.l.]: Narosa Publishing House.

[5] FALCÃO, A. X. *Filtros Lineares*. Acesso em: 13 Nov. 2017. Disponível em: <http://www.ic.unicamp.br/~afalcao/mo443/slides-aula6.pdf>.

[6] Canny Edge Detection. Acessso em: 14 Nov. 2017. Disponível em: <https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html>.

[7] Eroding and Dilating. Acesso em: 14 Nov. 2017. Disponível em: <https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html>.

[8] FALCÃO, A. X. *Filtros Não-Lineares*. Acesso em: 14 Nov. 2017. Disponível em: <http://www.ic.unicamp.br/~afalcao/mo443/slides-aula9.pdf>.

[9] GIMELF'ARB, G. *Image Filtering and Segmentation*. Acesso em: 15 Nov. 2017. Disponível em: <https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/2013/CS373-IP-03.pdf>.

[10] Hough Lines Transform. Acesso em: 15 Nov. 2017. Disponível em: <https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html>.

[11] BEDROS, S. J. *Hough Transform and Thresholding*. Acesso em: 15 Nov. 2017. Disponível em: <http://me.umn.edu/courses/me5286/vision/Notes/2015/ME5286-Lecture9.pdf>.

[12] KAEHLER, A.; BRADSKI, G. *Learning OpenCV 3: Computer vision in C++ with the OpenCV library*. [S.l.]: O'Reilly, 2017. ISBN 1491937998.

[13] CHAUDHURI, A.; MANDAVIYA, K.; GHOSH, P. B. S. K. *Optical Character Recognition Systems for Different Languages with Soft Computing*. 1. ed. [S.l.]: Springer International Publishing, 2017. (Studies in Fuzziness and Soft Computing 352). ISBN 978-3-319-50251-9, 978-3-319-50252-6.

[14] VIEIRA, M. A. da C.; GONZAGA, A. *Introdução à Visão Computacional*. Acesso em 25 Nov. 2017. Disponível em: <http://iris.sel.eesc.usp.br/sel339/SEL0339%20-%20Aula%201%20-%20Fundamentos%20de%20Imagens%20Digitais.pdf>.

[15] SUN, D.-W. *Computer Vision Technology for Food Quality Evaluation*. [S.l.]: Academic Press, 2007. (Food Science and Technology). ISBN 0123736420,9780123736420.

[16] DAVIES, E. R. *Computer and machine vision : theory, algorithms, practicalities*. 4th ed. ed. [S.l.]: Elsevier, 2012. ISBN 9780123869081,0123869080,9780123869913,0123869919.

[17] DAWSON-HOWE, K. *A Practical Introduction to Computer Vision with OpenCV*. 1. ed. [S.l.]: Wiley, 2014. (Wiley-IS&T Series in Imaging Science and Technology). ISBN 1118848454,9781118848456.

[18] GALLO, I.; ZAMBERLETTI, A.; NOCE, L. Robust angle invariant GAS meter reading. In: *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. [S.l.]: IEEE, 2015.

[19] KOMPF, M. *OpenCV practice: OCR for the electricity meter*. Acesso em: 11 Nov. 2017. Disponível em: <https://www.mkompf.com/cplus/emeocv.html>.

[20] PUTTNIES, H. et al. Cost-efficient universal approach for remote meter reading using web services and computer vision. In: *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. [S.l.]: IEEE, 2015.

[21] AUERSWALD, E. *Seven Segment Optical Character Recognition*. Acesso em: 11 Nov. 2017. Disponível em: <https://www.unix-ag.uni-kl.de/~auerswal/ssocr/>.

[22] Sistema de Telemetria para Hidrômetros e Medidores Aquisição de Dados Móvel e Fixa por Rádio Frequência. Disponível em: <http://www.ciasey.com.br/materiais/hidrometros-telemetria/sistema-de-telemetria.pdf>.

[23] Internet banda larga em Brasília/DF. Acesso em: 04 Nov. 2017. Disponível em: <https://melhorescolha.com/internet/brasilia-df/>.

[24] Celular em Brasília/DF. Acesso em: 04 Nov. 2017. Disponível em: <https://melhorescolha.com/celular-pos/brasilia-df/>.

[25] BOCCHINI, B. *Pesquisa mostra que 58% da população brasileira usam a internet*. 2016. Acesso em: 04 Nov. 2017. Disponível em: <http://agenciabrasil.ebc.com.br/pesquisa-e-inovacao/noticia/2016-09/pesquisa-mostra-que-58-da-populacao-brasileira-usam-internet>.

[26] IDOETA, P. A. *A agricultura é vilã ou vítima na crise hídrica?* 2015. Acesso em: 05 Nov. 2017. Disponível em: <http://www.bbc.com/portuguese/noticias/2015/03/150302_agua_agricultura_pai>.

[27] PATEL, S.; GOSWAMI, M. Comparative analysis of histogram equalization techniques. In: *2014 International Conference on Contemporary Computing and Informatics (IC3I)*. [S.l.]: IEEE, 2014.

[28] What Is Image Filtering in the Spatial Domain? Acesso em: 12 Nov. 2017. Disponível em: <https://www.mathworks.com/help/images/what-is-image-filtering-in-the-spatial-domain.html>.

[29] OLIVEIRA, L. E. S. *Processamento de Imagens: Morfologia Matemática Binária*. Acesso em: 14 Dez. 2017. Disponível em: <www.inf.ufpr.br/lesoliveira/download/morfologia.pdf>.

[30] CHERIET, M. et al. *Character Recognition Systems: A Guide for Students and Practitioners*. [S.l.]: Wiley-Interscience, 2007. ISBN 9780471415701,0471415707.

[31] SHIRAI, Y. *Three-Dimensional Computer Vision*. 1. ed. [S.l.]: Springer-Verlag Berlin Heidelberg, 1987. (Symbolic Computation). ISBN 978-3-642-82431-9,978-3-642-82429-6.

ANEXOS

I. TABELAS

Tabela I.1: Detalhamento de custos com leitura de hidrômetro e impressão da conta de água para cliente comum.

Código	Descrição	uni.	Coef.	Custo unit.	Preço total
127000021001	Leiturista	h	0.025	15.18	0.38
127000021005	Passagem integral tipo Metropolitana 2	uni.	0.01	4	0.04
127000021008	Papel térmico (largura: 111 mm/ comprimento: 320 mm / densidade 75 g/cm²)	folha	1	0.08	0.08
127000021009	Transmissão de dados via GPRS, com monitoramento via GPS incluso	h	0.025	0.38	0.01
127000021007	Coletor/transmissor de dados via GPRS, com monitoramento via GPS incluso	h prod	0.025	0.86	0.02
127000021010	Impressora térmica portátil	h prod	0.025	0.68	0.02
127000021000	Monitor	h prod	0.0025	19.12	0.05
8201008010004	Leitura de hidrômetro com impressão simultânea da conta	uni.	-	0.60	-

Tabela I.2: Detalhamento de custos com leitura de hidrômetro e impressão da conta de água para cliente especial ou situações específicas.

Código	Descrição	uni.	Coef.	Custo unit.	Preço total
127000021001	Leiturista	h	0.025	15.18	0.38
2280019010001	MOTOCICLETA, gasolina, potência 12,8 HP (9,5 kW) cilindrada 125 cc - sem operador	h prod	0.005	6.95	0.03
2280019010002	MOTOCICLETA, gasolina, potência 12,8 HP (9,5 kW) cilindrada 125 cc - sem operador	h imp	0.035	0.44	0.02
127000021007	Coletor/transmissor de dados via GPRS, com monitoramento via GPS incluso	h prod	0.025	0.86	0.02
127000021009	Transmissão de dados via GPRS, com monitoramento via GPS incluso	h	0.025	0.38	0.01
127000021008	Papel térmico (largura: 111 mm/ comprimento: 320 mm / densidade 75 g/cm²)	folha	1	0.08	0.08
127000021000	Monitor	h prod	0.0025	19.12	0.05
8201008010004	Leitura de hidrômetro com impressão simultânea da conta	uni.	-	0.59	-

Tabela I.3: Gastos da Caesb com coleta de leituras e emissões simultâneas.

Lote 01				
Região Administrativa	Qtd Ligação Água	valor estimado pela Caesb para a execução dos serviços, com BDI segundo edital Edital de Concorrência CP 003/2016-Caesb		Valor Licitado:
		Mensal	Para 24 meses	Valor Licitado Para 24 meses
Brasília	37.139	R$ 198.519,02	R$ 4.764.456,52	R$ 4.636.422,86
Núcleo Bandeirante	6.262			
Guará	28.679			
Cruzeiro	7.847			
Lago Sul	11.726			
Riacho Fundo I	10.323			
Lago Norte	8.348			
Candangolândia	4.182			
Sudoeste	8.043			
Park Way	6.384			
St. Compl. Ind. E Abastecimento	574			
Setor de Ind. E Abastecimento	1.867			
Estrutural	8.385			
Octogonal	848			
Metropolitana	697			
Noroeste	130			
Saneamento Rural	533			
Vicente Pires	19.802			
	161.769			

Lote 02				
Região Administrativa	Qtd Ligação Água	valor estimado pela Caesb para a execução dos serviços, com BDI segundo edital Edital de Concorrência CP 003/2016-Caesb		Valor Licitado:
		Mensal	Para 24 meses	Para 24 meses
Sobradinho I	14.781	R$ 201.903,26	R$ 4.845.678,24	R$ 4.368.922,69
Planaltina	27.211			
Paranoa	9.619			
São Sebastião	19.067			
Varjão	2.072			
Sobradinho II	24.399			
Jardim Botânico	6.900			
Itapõa	14.785			
Fercal	2.192			
Arapoanga	13.113			
Mestre D'Armas	11.350			
Taquari/Lago Norte	820			
SMLIN/SMLN	823			
Jardins Mangueiral	31			
	147.163			

Lote 03				
Região Administrativa	Qtd Ligação Água	valor estimado pela Caesb para a execução dos serviços, com BDI segundo edital Edital de Concorrência CP 003/2016-Caesb		Valor Licitado:
		Mensal	Para 24 meses	Para 24 meses
Arniqueira	6.373	R$ 232.070,98	R$ 5.569.703,52	R$ 4.882.107,13
Gama	33.870			
Samabaia	56.334			
Santa Maria	30.872			
Recanto das Emas	31.064			
Riacho Fundo II	11.454			
Entorno	2			
	169.969			

Lote 04				
Região Administrativa	Qtd Ligação Água	valor estimado pela Caesb para a execução dos serviços, com BDI segundo edital Edital de Concorrência CP 003/2016-Caesb		Valor Licitado:
		Mensal	Para 24 meses	Para 24 meses
Taguatinga	57.247	R$ 258.086,01	R$ 6.194.064,22	R$ 5.587.026,36
Brazlândia	12.943			
Ceilândia	79.462			
Águas Claras	29.490			
Ceilândia II	22.195			
	201.337			
	680.238	R$ 890.579,27	R$ 21.373.902,50	R$ 19.474.479,04

Tabela I.4: Custos com internet residencial em função da operadora no DF. [23]

Operadora	Velocidade [Mb]	Wi-Fi Incluso	Franquia [GB]	Benefícios Adicionais	Preço [R$]
Claro	1	Sim	20	Velocidade 1 Mb em área 3G, 2 Mb em área 4G.	159,99 [a]
Net	5	Sim	20	Oferta exclusiva no site, Wi-Fi grátis.	79,90 [b]
	15	Sim	80	Adesão grátis, Wi-Fi, desconto e dobro da velocidade grátis no combo Multi com TV, fixo e celular.	115,90 [c]
	60	Sim	150	Adesão grátis, Wi-Fi, desconto e dobro da velocidade grátis no combo Multi com TV, fixo e celular.	149,90 [c]
	120	Sim	200	Adesão grátis e desconto no combo com TV e fixo.	179,90 [c]
Oi	2	Não	60	R$59,90 para clientes Oi Fixo.	118,07 [d]
	15	Sim	100	R$64,90 para clientes Oi Fixo.	123,07 [d]
	25	Sim	110	R$74,90 para clientes Oi Fixo.	133,07 [d]
	35	Sim	130	R$59,90 para clientes Oi Fixo.	143,07 [d]
Vivo	25	Sim	100	Sem informações adicionais.	119.99
	60	Sim	170	Sem informações adicionais.	144.99
	100	Sim	220	Sem informações adicionais.	169.99
	200	Sim	270	Sem informações adicionais.	194.99
	300	Sim	300	Sem informações adicionais.	244.99

[a] adesão: R$249,90
[b] R$39,90 após o 6^o mês
[c] adesão: R$180,00
[d] adesão: R$160,00

Tabela I.5: Custos com internet para celular em função de operadora no DF. [24]

Operadora	Minutos para fixo e celular	Franquia [GB]	Ilimitado para mesma operadora	Benefícios adicionais	Preço [R$]
Claro	Ilimitado local e DDD.	6	Sim	Roaming nacional gratis. Claro Vídeo e Claro Música grátis.	99,99
	Ilimitado local e DDD.	8	Sim	Roaming nacional gratis. Claro Vídeo e Claro Música grátis.	199,99
	Ilimitado local e DDD.	10	Sim	Roaming nacional gratis. Claro Vídeo e Claro Música grátis.	149,99
	Ilimitado local e DDD	15	Sim	Roaming nacional gratis. Claro Vídeo e Claro Música grátis.	199,99
	Ilimitado local e DDD.	25	Sim	Roaming nacional gratis. Claro Vídeo e Claro Música grátis.	299,99
Oi	500 minutos local e DDD.	2	Não	Roaming nacional grátis. Leve mais 2 GB de bônus até 31/07/2018.	79,90
	Ilimitado local e DDD.	3	Sim	Roaming nacional grátis. Leve mais 3 GB de bônus até 31/07/2018.	99,90
	Ilimitado local e DDD.	5	Sim	Roaming nacional grátis. Leve mais 5 GB de bônus até 31/07/2018.	149,90
	Ilimitado local e DDD.	10	Sim	Roaming nacional grátis. Leve mais 10 GB de bônus até 31/07/2018.	249,90
Tim	Ilimitado local e DDD.	4	Sim	Roaming nacional grátis. Grátis Tim Music by Deezer.	99,99
	Ilimitado local e DDD.	5	Sim	Roaming nacional grátis. Grátis Tim Music by Deezer. Bônus de 2 GB por 12 mês.	109,99
	Ilimitado local e DDD.	7	Sim	Roaming nacional grátis. Grátis Tim Music by Deezer. Bônus de 2 GB por 12 mês.	139,99
	Ilimitado local e DDD.	10	Sim	Roaming nacional grátis. Grátis Tim Music by Deezer. Bônus de 2 GB por 12 mês.	179,99
	Ilimitado local e DDD.	20	Sim	Roaming nacional grátis. Grátis Tim Music by Deezer. Bônus de 2 GB por 12 mês.	249,99
Vivo	Ilimitado local.	4	Sim	Roaming nacinal grátis.	99,99
	Ilimitado local.	6	Sim	Roaming nacinal grátis.	149,99
	Ilimitado local e DDD.	10	Sim	Roaming nacinal grátis. 1 linhas de dependentes grátis.	269,99
	Ilimitado local e DDD.	15	Sim	Roaming nacinal grátis. 2 linhas de dependentes grátis.	389,99
	Ilimitado local e DDD.	30	Sim	Roaming nacinal grátis. 53 linhas de dependentes grátis.	489,99
	Ilimitado local e DDD.	50	Sim	Roaming nacinal grátis. 4 linhas de dependentes grátis.	589,99
	Ilimitado local e DDD.	200	Sim	Roaming nacinal grátis. 5 linhas de dependentes grátis.	1299,99

II. CÓDIGOS

install.sh

```
#! /bin/bash

# ----> SET UP PATHS <----

CURRENT=$(pwd)"/"
# software's name
BRAND="ocr"
BRAND=$BRAND"/"

DIR_1="/usr/local/"
DIR_2="/var/log/"
DIR_3="/usr/local/etc/"
DIR_4="/srv/http/"


# ----> CREATE DIRECTORIES <----

declare -a DIR_LIST=("$DIR_1" "$DIR_2" "$DIR_3" "$DIR_4")

echo "Creating directories if they don't exist yet..."
i=1
for dir in ${DIR_LIST[@]}; do
        # check if the --> http <-- directory exists; if no, create it in /srv/.
        if [ $i -eq 4 ]; then
                if [ ! -d $dir ]; then
                        mkdir $dir
                        echo $i ":" ${dir:0:${#dir}-1}
                fi
        fi
        # check if there is the --> brand <-- directory in the path; if no, create it.
        if [ ! -d $dir$BRAND ]; then
                mkdir $dir$BRAND
                echo $i ":" $dir${BRAND:0:${#BRAND}-1}
        fi
        # check if there is --> main, main/*, __pycache__ & ml-models/* <-- directory in /usr/local/<
            BRAND>/;
        # if no, create them.
        if [ $i -eq 1 ]; then
                if [ ! -d $dir$BRAND"main" ]; then
                        mkdir $dir$BRAND"main"
                        echo $i ":" $dir$BRAND"main"
                fi
                if [ ! -d $dir$BRAND"main/__pycache__" ]; then
                        mkdir $dir$BRAND"main/__pycache__"
                        echo $i ":" $dir$BRAND"main/__pycache__"
                fi
                if [ ! -d $dir$BRAND"ml-models" ]; then
                        mkdir $dir$BRAND"ml-models"
                        echo $i ":" $dir$BRAND"ml-models"
                fi
                if [ ! -d $dir$BRAND"ml-models/pickle" ]; then
                        mkdir $dir$BRAND"ml-models/pickle"
                        echo $i ":" $dir$BRAND"ml-models/pickle"
                fi
                if [ ! -d $dir$BRAND"ml-models/pickle/pickle2.7" ]; then
                        mkdir $dir$BRAND"ml-models/pickle/pickle2.7"
                        echo $i ":" $dir$BRAND"ml-models/pickle/pickle2.7"
                fi
                if [ ! -d $dir$BRAND"ml-models/pickle/pickle3.5" ]; then
                        mkdir $dir$BRAND"ml-models/pickle/pickle3.5"
                        echo $i ":" $dir$BRAND"ml-models/pickle/pickle3.5"
                fi
                if [ ! -d $dir$BRAND"ml-models/pickle/pickle3.6" ]; then
                        mkdir $dir$BRAND"ml-models/pickle/pickle3.6"
                        echo $i ":" $dir$BRAND"ml-models/pickle/pickle3.6"
                fi
        fi
        # check if --> ROIs & currentIms <-- directory already exists; if doesn't, create them in /srv/
            http/
        if [ $i -eq 4 ]; then
                if [ ! -d $dir$BRAND"ROIs" ]; then
                        mkdir $dir$BRAND"ROIs"
```

```
                        echo $i ":" $dir$BRAND"ROIs"
                fi
                if [ ! -d $dir$BRAND"currentIms" ]; then
                        mkdir $dir$BRAND"currentIms"
                        echo $i ":" $dir$BRAND"currentIms"
                fi
        fi
        i=$[$i+1]
done
echo -en "\n"


# ----> COPY FILES <----

echo "Copying files to their respective directories..."


echo -en "\n"
# /usr/local/<BRAND>/
PREFIX_1="main/"
PREFIX_2="main/__pycache__/"
PREFIX_3="ml-models/pickle/"
PREFIX_4="__pycache__/"

declare -a SOFTWARE_BASH=("hydrometerSetup.sh" "hydrometerOCR.sh")
for file in ${SOFTWARE_BASH[@]}; do
        cp $file $DIR_1$BRAND
        echo $DIR_1$BRAND$file
done

declare -a SOFTWARE_MAIN=("main.py" "mainCT.py" "htmlGen.py" "viewHydroDisplay.py" "hydrometerSetup.py" "
    saveRoi.py" "versionHandler.py")
for file in ${SOFTWARE_MAIN[@]}; do
        cp $file $DIR_1$BRAND$PREFIX_1
        echo $DIR_1$BRAND$PREFIX_1$file
done

declare -a SOFTWARE_HEADERS=("roiLocator.pyc" "digitsDetector.pyc" "digitsRecognizer.pyc"\
 "roiLocator.cpython-35.pyc" "digitsDetector.cpython-35.pyc" "digitsRecognizer.cpython-35.pyc"\
 "roiLocator.cpython-36.pyc" "digitsDetector.cpython-36.pyc" "digitsRecognizer.cpython-36.pyc")
for file in ${SOFTWARE_HEADERS[@]}; do
        cp $PREFIX_4$file $DIR_1$BRAND$PREFIX_2
        echo $DIR_1$BRAND$PREFIX_2$file
done

declare -a SOFTWARE_MODELS=("digits_knn.pkl" "digits_svm.pkl" "digits_mlp.pkl")
for file in ${SOFTWARE_MODELS[@]}; do
        cp $PREFIX_3"pickle2.7/"$file $DIR_1$BRAND$PREFIX_3"pickle2.7/"
        echo $DIR_1$BRAND$PREFIX_3"pickle2.7/"$file
        cp $PREFIX_3"pickle3.5/"$file $DIR_1$BRAND$PREFIX_3"pickle3.5/"
        echo $DIR_1$BRAND$PREFIX_3"pickle3.5/"$file
        cp $PREFIX_3"pickle3.6/"$file $DIR_1$BRAND$PREFIX_3"pickle3.6/"
        echo $DIR_1$BRAND$PREFIX_3"pickle3.6/"$file
done

echo -en "\n"
```

uninstall.sh

```
#! /bin/bash

# ----> REMOVING PATHS <----

CURRENT=$(pwd)"/"
# software's name
BRAND="ocr"
BRAND=$BRAND"/"

DIR_1="/usr/local/"
DIR_2="/var/log/"
DIR_3="/usr/local/etc/"
DIR_4="/srv/http/"

declare -a DIR_LIST=("$DIR_1" "$DIR_2" "$DIR_3" "$DIR_4")

echo "Removing directories..."
echo -en "\n"
i=1
for dir in ${DIR_LIST[@]}; do
        rm -r $dir$BRAND
```

```
22          echo $i ":" $dir${BRAND:0:${#BRAND}-1}
23          i=$[$i+1]
24  done
25  echo -en "\n"
```

hydrometerSetup.sh

```
1   #!/bin/bash
2
3   : '
4   Args:
5           1st --> python version
6           2nd --> device ID
7   '
8
9   if [ $# -eq 0 ]; then
10          # use Python 3 if no arg is spcified
11          py_version="python3"
12  else
13          if [ $# -lt 1 ] || [ $# -gt 2 ]; then
14                  echo "You need to provide at least 1 argument, and up to 2 arguments, at maximum. Exiting
                        ..."
15                  exit
16          fi
17          if [ $1 != 2 ] && [ $1 != 3 ]; then # according to De Morgan's Law
18                  echo "Python major version is not allowed. This program handles only versions 2 and 3."
19                  exit
20          fi
21          py_version="python"$1
22  fi
23
24  PREFIX="/usr/local/ocr/main/"
25  # two programs running in parallel
26  $py_version $PREFIX"viewHydroDisplay.py" $2 & \
27  xterm -xrm 'XTerm.vt100.allowTitleOps: false' -T 'Hydrometer Setup' +hold -e $py_version $PREFIX"
        hydrometerSetup.py"
28  # kill the second one as soon as the first one is finished
29  pids=( $(pgrep -f viewHydroDisplay.py) )
30  for pid in ${pids[@]}; do
31    if [[ $pid != $$ ]]; then
32      kill $pid
33      wait $pid 2>/dev/null
34    fi
35  done
36
37  # save roi with as <timestamp>.jpg
38  $py_version $PREFIX"saveRoi.py" $2
```

hydrometerSetup.py

```
1   from __future__ import print_function
2   import datetime
3   import os
4   import time
5   import sys
6   import versionHandler as vh
7
8   if __name__ == '__main__':
9           _,_,direc = vh.headerPicker()
10
11          LOGFILE_PATH = "/var/log/ocr/"
12          SETUP_PATH = "/usr/local/etc/ocr/"
13          METRICS_PATH = "/usr/local/etc/ocr/"
14
15          # Open file that stores the inicial metrics of the hydrometer reading
16          f = open(SETUP_PATH+"hydrometerSetup.dat","w+")
17          while True:
18                  # Start dialog with user through xterm, verifying Python version first
19                  if sys.version_info >= (3,0):
20                          strReading = input("  Enter the reading you see on the hydrometer display.\n
                                Starting reading: ")
21                  else:
22                          strReading = raw_input("  Enter the reading you see on the hydrometer display.\n
                                Starting reading: ")
23                  if not strReading.isdigit():
24                          print ("  --> All chars must be digits. <--")
25                  else:
26                          break
27
```

```
28          # Write local UTC time and date, the starting reading, and the timestamp relative to the
29          # local UTC time and date as the filename of the region of interest image
30          f.write("UTC\t\t\tStarting Reading\tROI\n")
31          utc = datetime.datetime.utcnow().strftime("%Y/%m/%d@%H:%M:%S")
32          timestamp = int(time.time())
33          f.write(str(utc)+"\t"+str(strReading)+"\t\t\t"+str(timestamp)+".jpg"+"\n")
34          f.close()
35
36          # Create titles for each column of the file if the file has size zero KB
37          f = open(LOGFILE_PATH+"hydrometer.log", "a+")
38          if os.stat(LOGFILE_PATH+"hydrometer.log").st_size == 0:
39                  f.write("\tUTC\t       Timestamp     Reading    Usage[m3]    dV[L]    dt[s]    Q[L/h]    Qma[L/h
                        ]    Try[x]\n")
40          f.close()
41
42          # Feed both the metrics file (prevHydroMetrics.var) and log file (hydrometer.log)
43          direc.send2file(utc, timestamp, strReading, float(strReading)/1000.0, 0, 0, 0, 0, 0, LOGFILE_PATH
                , "hydrometer.log", "a+")
44          direc.send2file(utc, timestamp, strReading, float(strReading)/1000.0, 0, 0, 0, 0, 0, METRICS_PATH
                , "prevHydroMetrics.var", "w+")
```

viewHydroDisplay.py

```
1   from __future__ import print_function
2
3   import sys
4   import cv2
5   import numpy as np
6   # import versionHandler as vh # --> not necessary <--
7   import roiLocator as roid
8   import digitsDetector as did
9
10  if __name__ == '__main__':
11          #roid, did,_ = vh.headerPicker()
12
13          deviceID = 0 # Number of the device ID by default
14          if len(sys.argv) == 2:
15                  deviceID = sys.argv[1] # Can be overwritten if wanted
16
17          COLOR = (0,255,0) # Color green (BGR)
18
19          gotDigitsSequence = False # When True, it flags that all that seven digits were detected
20          debug=True # When True, it flags the program to show computer vision image processing
21
22          try:
23
24                  cap = cv2.VideoCapture(int(deviceID))
25
26                  while(True):
27
28                          # Capture frame-by-frame
29                          ret, imBGR = cap.read()
30
31                          # ------------------------- APPLY HOUGH LINES AND STRAIGHTEN IMAGE
                                ------------------------
32
33                          canny_1, imGray = roid.prep4HoughLines(imBGR)
34
35                          imHoughL = imBGR.copy()
36                          # imHoughL = cv2.cvtColor(imHoughL, cv2.COLOR_GRAY2BGR)
37                          if roid.findHorizontalLine(canny_1, imHoughL, COLOR) is None:
38                                  continue
39                          imHoughL, skewAngle, (y0,x0) = roid.findHorizontalLine(canny_1, imHoughL, COLOR)
40
41                          repairAngle = roid.calcRepairAngle(skewAngle)
42
43                          # Grab the dimensions of the image and calculate the center of the image
44                          (h, w) = imHoughL.shape[:2]
45                          center = (w / 2, h / 2) # w = width = y & h = hight = x
46
47                          # Calculate the new Hough point when the image is horizontally aligned
48                          new_houghPoint = roid.rotatePoint(y0,x0,float(h)/2.0,float(w)/2.0, repairAngle)
49
50                          # Rotate the image by 180 degrees around its center
51                          M = cv2.getRotationMatrix2D(center, repairAngle, 1.0)
52                          straightened = cv2.warpAffine(imHoughL, M, (w, h)) # it has a line drawn by
                                HoughLines()
53                          straight = cv2.warpAffine(imGray.copy(), M, (w, h))
54
55                          # ------------------------- FIND ROI -------------------------
```

```
56
57                      # Find a circle around the analog meter's display
58                      if roid.findCircle(straight, straightened, COLOR) is None:
59                              continue
60                      straightened, (center, radius) = roid.findCircle(straight, straightened, COLOR)
61
62                      # Leave just a semi-circular visible part, which is upon the ROI
63                      imMasked_2, (upper_whiteDensity, bottom_whiteDensity) = roid.formSemiCirleOnRoi(
                            straight, center, radius, new_houghPoint)
64
65                      # Find biggest contour in the picture
66                      if roid.findRoi(imMasked_2, straightened, COLOR) is None:
67                              continue
68                      straightened, (x,y,w,h) = roid.findRoi(imMasked_2, straightened, COLOR)
69
70                      if float(w)/float(h) <= 2.6:
71                              if roid.fixRatio(imMasked_2, straightened, COLOR):
72                                      continue
73                              straightened, (x,y,w,h) = roid.fixRatio(imMasked_2, straightened, COLOR)
74
75
76                      # ------------------------- MAKE SURE ROI IS RIGHT-SIDE-UP
                            -------------------------
77
78                      # Crop ROI from the whole image
79                      roi = straight[y:y+h,x:x+w]
80                      roiBGR = straightened[y:y+h,x:x+w]
81
82                      if bottom_whiteDensity > upper_whiteDensity:
83                              # ROI's center regarding the whole image
84                              (yRoi, xRoi) = (float(h/2.0),float(w/2.0))
85                              # Rotate roi by 180 degrees in respect with its center
86                              center = (int(xRoi), int(yRoi)) # w = width = y & h = hight = x
87                              M = cv2.getRotationMatrix2D(center, 180, 1.0)
88                              roi = cv2.warpAffine(roi, M, (roi.shape[1], roi.shape[0]))
89                              # Rotate rightSideUP by 180 degrees in respect with its center
90                              (h,w) = straightened.shape[:2]
91                              center = (w/2,h/2) # w = width = y & h = hight = x
92                              M = cv2.getRotationMatrix2D(center, 180, 1.0)
93                              rightSideUP = cv2.warpAffine(straightened, M, (straightened.shape[1],
                                    straightened.shape[0]))
94
95                      else:
96                              rightSideUP = straightened
97
98                      # Display the resulting frame
99                      if debug:
100                             cv2.imshow("Hydrometer View", rightSideUP)
101
102                     # ------------------------- FIND DIGITS -------------------------
103
104                     # Equalize its histogram
105                     equalized = cv2.equalizeHist(roi)
106
107                     # Darken the image's boundaries
108                     equalized = did.darkenImSides(equalized)
109
110                     # Crop out part of the top and the bottom
111                     if did.diminishHeight(equalized) is None:
112                             continue
113                     imDigits = did.diminishHeight(equalized)
114
115                     # Prepare the image to apply Canny next and then see all the digits contours
116                     closing = did.preCanny(imDigits)
117
118                     if did.locateDigits(closing, gotDigitsSequence, debug) is None:
119                             continue
120                     # edges --> segmented roi with green rectangles
121                     # digits --> just segmented roi
122                     # homeCoords --> digits' home coordenates
123                     digits, edges, homeCoords, gotDigitsSequence = did.locateDigits(closing,
                            gotDigitsSequence, debug)
124
125
126                     # Quit
127                     if cv2.waitKey(1) & 0xFF == ord('q'):
128                             break
129
130             cap.release()
131             cv2.destroyAllWindows()
132
```

```
133             except KeyboardInterrupt:
134                     pass
```

saveRoi.py

```
1   from __future__ import print_function
2
3   import sys
4   import cv2
5   import numpy as np
6   import time
7   import versionHandler as vh
8
9   if __name__ == '__main__' :
10          roid,_,_ = vh.headerPicker()
11
12          ROI_PATH = "/srv/http/ocr/ROIs/"
13          SETUP_PATH = "/usr/local/etc/ocr/"
14
15          deviceID = 0 # Number of the device ID by default
16          if len(sys.argv) == 2:
17                  deviceID = sys.argv[1] # Can be overwritten if wanted
18
19          COLOR = (0,255,0) # Color green (BGR)
20          #turn on video stream
21          cap = cv2.VideoCapture(int(deviceID))
22
23          while(True):
24
25                  # Capture frame-by-frame
26                  ret, imBGR = cap.read()
27
28                  # ------------------------- APPLY HOUGH LINES AND STRAIGHTEN IMAGE
                        -------------------------
29
30                  canny_1, imGray = roid.prep4HoughLines(imBGR)
31
32                  imHoughL = imBGR.copy()
33                  # imHoughL = cv2.cvtColor(imHoughL, cv2.COLOR_GRAY2BGR)
34                  if roid.findHorizontalLine(canny_1, imHoughL, COLOR) is None:
35                          continue
36                  imHoughL, skewAngle, (y0,x0) = roid.findHorizontalLine(canny_1, imHoughL, COLOR)
37
38                  repairAngle = roid.calcRepairAngle(skewAngle)
39
40                  # Grab the dimensions of the image and calculate the center of the image
41                  (h, w) = imHoughL.shape[:2]
42                  center = (w / 2, h / 2) # w = width = y & h = hight = x
43
44                  # Calculate the new Hough point when the image is horizontally aligned
45                  new_houghPoint = roid.rotatePoint(y0,x0,float(h)/2.0,float(w)/2.0, repairAngle)
46
47                  # Rotate the image by 180 degrees around its center
48                  M = cv2.getRotationMatrix2D(center, repairAngle, 1.0)
49                  straightened = cv2.warpAffine(imHoughL, M, (w, h)) # It has a line drawn by HoughLines()
50                  straight = cv2.warpAffine(imGray.copy(), M, (w, h))
51
52                  # ------------------------- FIND ROI -------------------------
53
54                  # Find a circle around the analog meter's display
55                  if roid.findCircle(straight, straightened, COLOR) is None:
56                          continue
57                  straightened, (center, radius) = roid.findCircle(straight, straightened, COLOR)
58
59                  # Leave just a semi-circular visible part, which is upon the ROI
60                  imMasked_2, (upper_whiteDensity, bottom_whiteDensity) = roid.formSemiCirleOnRoi(straight,
                        center, radius, new_houghPoint)
61
62                  # Find biggest contour in the picture
63                  if roid.findRoi(imMasked_2, straightened, COLOR) is None:
64                          continue
65                  straightened, (x,y,w,h) = roid.findRoi(imMasked_2, straightened, COLOR)
66
67                  if float(w)/float(h) <= 2.6:
68                          if roid.fixRatio(imMasked_2, straightened, COLOR):
69                                  continue
70                          straightened, (x,y,w,h) = roid.fixRatio(imMasked_2, straightened, COLOR)
71
72                  # ------------------------- MAKE SURE ROI IS RIGHT-SIDE-UP -------------------------
73
```

```
74                  # Crop ROI
75                  roi = straightened[y:y+h,x:x+w]
76
77                  # Invert it if upside down
78                  if bottom_whiteDensity > upper_whiteDensity:
79                          # ROI's center regarding the whole image
80                          (yRoi, xRoi) = (float(h/2.0),float(w/2.0))
81                          # Rotate ROI by 180 degrees in respect with its center
82                          center = (int(xRoi), int(yRoi)) # w = width = y & h = hight = x
83                          M = cv2.getRotationMatrix2D(center, 180, 1.0)
84                          roi = cv2.warpAffine(roi, M, (roi.shape[1], roi.shape[0]))
85
86
87                  f = open(SETUP_PATH+"hydrometerSetup.dat","r")
88                  lines = f.readlines()
89                  part = lines[1].split()
90                  timestamp = part[2][:-4]
91                  f.close()
92                  # cv2.imwrite(str(timestamp)+'.jpg',roi, [int(cv2.IMWRITE_JPEG_QUALITY), 90])
93                  cv2.imwrite(ROI_PATH+str(timestamp)+".jpg",roi, [cv2.IMWRITE_JPEG_QUALITY, 75])
94                  break
95
96          # turn off video
97          cap.release()
98          cv2.destroyAllWindows()
```

hydrometerOCR.sh

```
1   #!/bin/bash
2
3   : '
4   Args:
5           1st --> python version
6           2nd --> device ID
7           3rd --> debug mode
8                   * -d : mode on
9                   * -n : mode off
10          4th --> update period
11          5th --> machine learning classifier model
12                  * 1 : Support Vector Machine
13                  * 2 : Multilayer Perceptron
14                  * 3 : k-Nearest Neighbor
15  '
16
17  if [ $# -eq 0 ]; then
18          # use Python 3 if no arg is spcified
19          py_version="python3"
20  else
21          if [ $# -lt 1 ] || [ $# -gt 5 ]; then
22                  echo "You need to provide at least 1 argument, and up to 5 arguments, at maximum. Exiting
                        ..."
23                  exit
24          fi
25          if [ $1 != 2 ] && [ $1 != 3 ]; then # according to De Morgan's Law
26                  echo "Python major version is not allowed. This program handles only versions 2 and 3."
27                  exit
28          fi
29          if [ $# -eq 5 ]; then
30                  if [ $5 -lt 1 ] || [ $5 -gt 3 ]; then
31                          echo "There only 3 machine learning classifiers. Please specify 1 for SVM, 2 for
                                MLP, or 3 for kNN. Exiting..."
32                          exit
33                  fi
34          fi
35          py_version="python"$1
36  fi
37
38  PREFIX="main/"
39  CRONTAB=false
40  if [ $CRONTAB = true ]; then
41          $py_version $PREFIX"main.py" $2 $3 $5
42          $py_version $PREFIX"htmlGen.py" & sleep $T
43  else
44          # execute main every <T> s
45          if [ $# -eq 4 ]; then
46                  T=$4
47          else
48                  T=20 # default (if no time period is specified)
49          fi
50          while [ true ]; do
```

```
51                  $py_version $PREFIX"main.py" $2 $3 $5
52                  $py_version $PREFIX"htmlGen.py" & sleep $T
53          done
54  fi
```

main.py

```
1   from __future__ import print_function
2   import sys
3   import cv2
4   import versionHandler as vh
5
6   if __name__ == '__main__' :
7           roid, did, direc = vh.headerPicker()
8
9           # Set up path constants
10          ROI_PATH = "/srv/http/ocr/ROIs/"
11          LOGFILE_PATH = "/var/log/ocr/"
12          METRICS_PATH = "/usr/local/etc/ocr/"
13          IMS_UPDATE_PATH = "/srv/http/ocr/currentIms/"
14
15          # Break down directories according to the Python version the program is written in
16          # Each directory has 3 models (kNN, SVM, MLP) created through machine learning
17          # Also, each one is conceived with a different version of Pickle format (file extension)
18          if sys.version_info >= (2,7) and sys.version_info < (3,0):
19                  LEARNING_PATH = "/usr/local/ocr/ml-models/pickle/pickle2.7/"
20          if sys.version_info >= (3,0) and sys.version_info < (3,6):
21                  LEARNING_PATH = "/usr/local/ocr/ml-models/pickle/pickle3.5/"
22          if sys.version_info >= (3,6) and sys.version_info < (4,0):
23                  LEARNING_PATH = "/usr/local/ocr/ml-models/pickle/pickle3.6/"
24
25          # ---> Default variables <---
26          # ID of the camera turned to the hydrometer display
27          deviceID = 0
28          # When True, default mode turns on and real-time image processing and metrics are shown
29          debug = False
30          # Machine Learning Classifier: . 1 --> SVM                . 2 --> MLP                . 3 --> kNN
31          ML_CLASSIFIER = 1
32
33          # ---> Override default values if arguments are passed with the module called <---
34          if len(sys.argv) == 2:
35                  deviceID = sys.argv[1]
36          elif len(sys.argv) == 3:
37                  deviceID = sys.argv[1]
38                  if sys.argv[2] == "-d":
39                          debug = True
40          elif len(sys.argv) == 4:
41                  deviceID = sys.argv[1]
42                  if sys.argv[2] == "-d":
43                          debug = True
44                  ML_CLASSIFIER = int(sys.argv[3])
45
46          COLOR = (0,255,0)
47          gotDigitsSequence = False
48
49          MIN_dV = 1.0 # Minimum delta necessary to update metrics [L]
50          Qn = 750.0 #  Nominal Flow [L/h]
51          FoS = 1.0 # Factor of safety
52
53          Max_NoAttempts = 5 # Maximum number of attemps the program has to get the OCR done
54          # in a while, periodically
55          counter = 0 # Initialization of a variable that counts the numbers of attempts
56
57          try:
58                  # Start video stream
59                  cap = cv2.VideoCapture(int(deviceID))
60
61                  while(True):
62
63                          # Limit the number of attempts for the OCR
64                          counter+=1
65                          if counter > Max_NoAttempts:
66                                  counter-=1
67                                  break
68
69                          # Capture one frame
70                          ret, imBGR = cap.read()
71
72                          # ---------------------------- APPLY HOUGH LINES AND STRAIGHTEN IMAGE
                                ----------------------------
```

```
73
74                          canny_1, imGray = roid.prep4HoughLines(imBGR)
75
76                          imHoughL = imBGR.copy()
77                          # imHoughL = cv2.cvtColor(imHoughL, cv2.COLOR_GRAY2BGR)
78                          if roid.findHorizontalLine(canny_1, imHoughL, COLOR) is None:
79                                  continue
80                          imHoughL, skewAngle, (y0,x0) = roid.findHorizontalLine(canny_1, imHoughL, COLOR)
81
82                          repairAngle = roid.calcRepairAngle(skewAngle)
83
84                          # Grab the dimensions of the image and calculate the center of the image
85                          (h, w) = imHoughL.shape[:2]
86                          center = (w / 2, h / 2) # w = width = y & h = hight = x
87
88                          # Calculate the new Hough point when the image is horizontally aligned
89                          new_houghPoint = roid.rotatePoint(y0,x0,float(h)/2.0,float(w)/2.0, repairAngle)
90
91                          # Rotate the image by 180 degrees around its center
92                          M = cv2.getRotationMatrix2D(center, repairAngle, 1.0)
93                          straightened = cv2.warpAffine(imHoughL, M, (w, h)) # It has a line drawn by
                                HoughLines()
94                          straight = cv2.warpAffine(imGray.copy(), M, (w, h))
95
96                          # ------------------------- FIND ROI -------------------------
97
98                          # Find a circle around the analog meter's display
99                          if roid.findCircle(straight, straightened, COLOR) is None:
100                                 continue
101                         straightened, (center, radius) = roid.findCircle(straight, straightened, COLOR)
102
103                         # Leave just a semi-circular visible part, which is upon the ROI
104                         imMasked_2, (upper_whiteDensity, bottom_whiteDensity) = roid.formSemiCirleOnRoi(
                                straight, center, radius, new_houghPoint)
105
106                         # Find biggest contour in the picture
107                         if roid.findRoi(imMasked_2, straightened, COLOR) is None:
108                                 continue
109                         straightened, (x,y,w,h) = roid.findRoi(imMasked_2, straightened, COLOR)
110
111                         if float(w)/float(h) <= 2.6:
112                                 if roid.fixRatio(imMasked_2, straightened, COLOR):
113                                         continue
114                                 straightened, (x,y,w,h) = roid.fixRatio(imMasked_2, straightened, COLOR)
115
116
117                         # ------------------------- MAKE SURE ROI IS RIGHT-SIDE-UP
                                -------------------------
118
119                         # Crop ROI
120                         roi = straight[y:y+h,x:x+w]
121                         roiBGR = straightened[y:y+h,x:x+w]
122
123                         if bottom_whiteDensity > upper_whiteDensity:
124                                 # ROI's center regarding the whole image
125                                 (yRoi, xRoi) = (float(h/2.0),float(w/2.0))
126                                 # Rotate roi by 180 degrees in respect with its center
127                                 center = (int(xRoi), int(yRoi)) # w = width = y & h = hight = x
128                                 M = cv2.getRotationMatrix2D(center, 180, 1.0)
129                                 roi = cv2.warpAffine(roi, M, (roi.shape[1], roi.shape[0]))
130                                 # Rotate rightSideUP by 180 degrees in respect with its center
131                                 (h,w) = straightened.shape[:2]
132                                 center = (w/2,h/2) # w = width = y & h = hight = x
133                                 M = cv2.getRotationMatrix2D(center, 180, 1.0)
134                                 rightSideUP = cv2.warpAffine(straightened, M, (straightened.shape[1],
                                        straightened.shape[0]))
135
136                         else:
137                                 rightSideUP = straightened
138
139                         if debug == True:
140                                 # Display the resulting frame
141                                 cv2.imshow("ROI",rightSideUP)
142
143                         # ------------------------- FIND DIGITS -------------------------
144
145                         # Equalize its histogram
146                         equalized = cv2.equalizeHist(roi)
147
148                         # Darken the image's boundaries
149                         equalized = did.darkenImSides(equalized)
```

```
150
151                         # Crop out part of the top and the bottom
152                         if did.diminishHeight(equalized) is None:
153                                 continue
154                         imDigits = did.diminishHeight(equalized)
155
156                         # Prepare the image to apply Canny next and then see all the digits contours
157                         closing = did.preCanny(imDigits)
158
159                         if did.locateDigits(closing, gotDigitsSequence, debug) is None:
160                                 continue
161                         # edges --> segmented ROI with green rectangles
162                         # digits --> just segmented ROI
163                         # homeCoords --> digits' home coordenates
164                         digits, edges, homeCoords, gotDigitsSequence = did.locateDigits(closing,
                                gotDigitsSequence, debug)
165
166                         if gotDigitsSequence:
167                                 rectDigits = did.createSample(digits, homeCoords)
168
169                                 # ------------------------- RECOGNIZE DIGITS -------------------------
170
171                                 direc.recogNsave(MIN_dV, Qn, FoS, rectDigits, roiBGR, edges, rightSideUP,
                                        ROI_PATH, IMS_UPDATE_PATH,\
172                                                                 LOGFILE_PATH, METRICS_PATH,
                                                                        LEARNING_PATH, ML_CLASSIFIER,
                                                                        counter, debug)
173                                 break
174
175                         if cv2.waitKey(1) & 0xFF == ord('q'):
176                                 break
177
178                 cap.release()
179
180                 if debug:
181                         cv2.waitKey(5000)
182                         cv2.destroyAllWindows()
183
184         except KeyboardInterrupt:
185                 pass
```

versionHandler.py

```
1   import sys
2
3   def headerPicker():
4           PATH="/usr/local/ocr/main/"
5           if sys.version_info >= (3,0):
6                   import importlib.util
7
8                   version = ""
9                   version += str(sys.version_info[0])
10                  version += str(sys.version_info[1])
11
12                  from pathlib import Path
13
14                  pyCache = Path(PATH+"__pycache__/roiLocator.cpython-"+version+".pyc")
15                  if pyCache.is_file():
16                          spec = importlib.util.spec_from_file_location("roiLocator", PATH+"__pycache__/
                                roiLocator.cpython-"+version+".pyc")
17                          roid = importlib.util.module_from_spec(spec)
18                          spec.loader.exec_module(roid)
19                  else:
20                          import roiLocator as roid
21
22                  pyCache = Path(PATH+"__pycache__/digitsDetector.cpython-"+version+".pyc")
23                  if pyCache.is_file():
24                          spec = importlib.util.spec_from_file_location("digitsDetector", PATH+"__pycache__
                                /digitsDetector.cpython-"+version+".pyc")
25                          did = importlib.util.module_from_spec(spec)
26                          spec.loader.exec_module(did)
27                  else:
28                          import digitsDetector as did
29
30                  pyCache = Path(PATH+"__pycache__/digitsRecognizer.cpython-"+version+".pyc")
31                  if pyCache.is_file():
32                          spec = importlib.util.spec_from_file_location("digitsRecognizer", PATH+"
                                __pycache__/digitsRecognizer.cpython-"+version+".pyc")
33                          direc = importlib.util.module_from_spec(spec)
34                          spec.loader.exec_module(direc)
```

```
35                  else:
36                          import digitsRecognizer as direc
37
38          else:
39
40                  import imp
41                  from pathlib2 import Path
42
43                  pyCache = Path(PATH+"__pycache__/roiLocator.pyc")
44                  if pyCache.is_file():
45                          roid = imp.load_compiled("roiLocator", PATH+"__pycache__/roiLocator.pyc")
46                  else:
47                          import roiLocator as roid
48
49                  pyCache = Path(PATH+"__pycache__/digitsDetector.pyc")
50                  if pyCache.is_file():
51                          did = imp.load_compiled("digitsDetector", PATH+"__pycache__/digitsDetector.pyc")
52                  else:
53                          import digitsDetector as did
54
55                  pyCache = Path(PATH+"__pycache__/digitsRecognizer.pyc")
56                  if pyCache.is_file():
57                          direc = imp.load_compiled("digitsRecognizer", PATH+"__pycache__/digitsRecognizer.
                                pyc")
58                  else:
59                          import digitsRecognizer as direc
60
61
62          return roid, did, direc
```

roiLocator.py

```
1   import cv2
2   import numpy as np
3
4   ######################################
5   ###### ROI FUNCTIONS
6   ######################################
7
8   #
        ____________________________________________________________________________________________________________________________
9
10  def prep4HoughLines(imBGR):
11          # Convert BGR to Gray
12          imGray = cv2.cvtColor(imBGR, cv2.COLOR_BGR2GRAY)
13          # Smooth the image without demaging the edges
14          bilateral_1 = cv2.bilateralFilter(imGray,5,1000,1000)
15          # Make edges sharper
16          kernel = np.matrix('1, 1, 1; 1, -7, 1; 1, 1, 1')
17          sharpened_1 = cv2.filter2D(bilateral_1,-1,kernel)
18          # Show pretty much only edges
19          canny_1 = cv2.Canny(sharpened_1,600,2500,apertureSize = 5)
20          return canny_1, imGray
21
22  #
        ____________________________________________________________________________________________________________________________
23
24  def calcRepairAngle(skewAngle):
25          # Define image's repair angle
26          if (skewAngle >= 0) and (skewAngle < 90):
27                  repairAngle = skewAngle - 90
28          if (skewAngle >= 90) and (skewAngle <= 180):
29                  repairAngle = skewAngle + 270
30          return repairAngle
31
32  #
        ____________________________________________________________________________________________________________________________
33
34  def rotatePoint (y,x,yC,xC,angle):
35          # Rotate the Hough point around the image center point
36          point = (float(y) , float(x))
37          center = (float(yC), float(xC))
38          alpha = angle/180.0*np.pi
39          dist = np.subtract(point,center)
40          new_dist_Y = dist[0]*np.cos(alpha) - dist[1]*np.sin(alpha)
41          new_dist_X = dist[0]*np.sin(alpha) + dist[1]*np.cos(alpha)
42          new_dist = (new_dist_Y, new_dist_X)
```

```
43          new_point = tuple(map(sum,zip(center,new_dist)))
44          return new_point
45
46  #       ------------------------------------------------------------------------------------------------------------------------------------------------------
47
48  def findHorizontalLine(canny_1, imHoughL, COLOR):
49          # Apply hough-line transform
50          lines = cv2.HoughLines(canny_1,1,1*np.pi/180,100)
51
52          if lines is None:
53                  return None
54
55          thetaAcc = 0
56          thetaQtd = 1
57          for i in range (0,thetaQtd):
58                  for rho,theta in lines[i,:]:
59                          thetaAcc += theta
60                          a = np.cos(theta)
61                          b = np.sin(theta)
62                          x0 = int(a*rho)
63                          y0 = int(b*rho)
64                          x1 = int(x0 + 1000*(-b))
65                          y1 = int(y0 + 1000*(a))
66                          x2 = int(x0 - 1000*(-b))
67                          y2 = int(y0 - 1000*(a))
68                          cv2.line(imHoughL,(x1,y1),(x2,y2),COLOR,2)
69          # Find skew angle --> regarding the vertival axis <--
70          thetaMean = float(thetaAcc)/float(thetaQtd)
71          skewAngle = thetaMean/np.pi*180.0
72
73          return imHoughL, skewAngle, (y0,x0)
74
75  #       ------------------------------------------------------------------------------------------------------------------------------------------------------
76
77  def findCircle(straight, straightened, COLOR):
78
79          # Smooth image while enhancing edges
80          bilateral_2 = cv2.bilateralFilter(straight,9,10,10)
81          # Threshold = 225
82          _, threshold_1 = cv2.threshold(bilateral_2, 225, 255, cv2.THRESH_BINARY)
83          _,contours,_ = cv2.findContours(threshold_1,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
84
85          if len(contours) is 0:
86                  return None
87
88          areas = [cv2.contourArea(c) for c in contours]
89          max_index = np.argmax(areas)
90          cnt=contours[max_index]
91          (x,y),radius = cv2.minEnclosingCircle(cnt)
92          center = (int(x),int(y))
93          radius = int(radius+int(0.108*radius))
94          cv2.circle(straightened, center,radius,COLOR,2)
95
96          return straightened, (center,radius)
97
98  #       ------------------------------------------------------------------------------------------------------------------------------------------------------
99
100 def formSemiCirleOnRoi(straight, center, radius, new_houghPoint):
101         # Create mask
102         mask = np.zeros(straight.shape, dtype = np.uint8) # All black at first
103         cv2.circle(mask,center,radius,(255,255,255),-1) # Add the circle found, filled with white
104         # And operatoin with the mask and the actual image
105         imMasked_1 = cv2.bitwise_and(straight,mask) # put the image into the mask (circular)
106         # -3 and +3 are taking the white Hugh line part off
107         upperPart = lowerPart = imMasked_1.copy()
108         upperPart = upperPart[0:int(new_houghPoint[0]-3)]
109         lowerPart = lowerPart[int(new_houghPoint[0])+3:-1]
110         # Find a quantitative value for the white density, both on the lower and upper part of the image
111         hist_upper = np.bincount(upperPart.ravel(),minlength=256)
112         hist_bottom = np.bincount(lowerPart.ravel(),minlength=256)
113         upper_whiteDensity = sum(hist_upper[127:-1])
114         bottom_whiteDensity = sum(hist_bottom[127:-1])
115         # Paint with black the part with higher white density
116         start = int(center[1] - radius)
117         end = int(center[1] + radius)
```

```
        if upper_whiteDensity > bottom_whiteDensity:
                if start < 0:
                        start = 0 # Make sure it is a valid y coordinate inside the image
                mask[start:int(new_houghPoint[0])] = 0
        else:
                if end > mask.shape[0]:
                        end = mask.shape[0] # make sure it is a valid y coordinate inside the image
                mask[int(new_houghPoint[0]):end] = 0
        # And operation with the mask and the actual image
        imMasked_2 = cv2.bitwise_and(imMasked_1,mask) # put the image into the mask (semi-circular)

        return imMasked_2, (upper_whiteDensity, bottom_whiteDensity)

#
```

```
def findRoi(imMasked_2, straightened, COLOR):

        # Smooth image
        blurred_1 = cv2.blur(imMasked_2,(30,30))
        # Threshold = 100
        _, threshold_2 = cv2.threshold(blurred_1, 100, 255, cv2.THRESH_BINARY)
        # Apply adaptative filter to show pritty much only edges in white more thickly
        edges = cv2.adaptiveThreshold(threshold_2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
                cv2.THRESH_BINARY_INV,11,5)
        _,contours,_ = cv2.findContours(edges,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

        if len(contours) is 0:
                return None

        areas = [cv2.contourArea(c) for c in contours]
        max_index = np.argmax(areas)
        cnt=contours[max_index]
        x,y,w,h = cv2.boundingRect(cnt)
        cv2.rectangle(straightened,(x,y),(x+w,y+h),COLOR,2)

        return straightened, cv2.boundingRect(cnt)

#
```

```
def fixRatio(imMasked_2, straightened, COLOR):
        # Equalize histogram
        equalized = cv2.equalizeHist(imMasked_2)
        # Smooth image while enhancing edges
        blurred_2 = cv2.blur(equalized,(5,5))
        # Threshold = 160
        retval, threshold_3 = cv2.threshold(blurred_2, 160, 255, cv2.THRESH_BINARY)
        # Find biggest contour in the pic
        _,contours,_ = cv2.findContours(threshold_3,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

        if len(contours) is 0:
                return None

        areas = [cv2.contourArea(c) for c in contours]
        max_index = np.argmax(areas)
        cnt=contours[max_index]
        x,y,w,h = cv2.boundingRect(cnt)
        cv2.rectangle(straightened,(x,y),(x+w,y+h),COLOR,2)

        return straightened, cv2.boundingRect(cnt)
```

digitsDetector.py

```
import cv2
import numpy as np

#####################################
###### DIGITS FUNCTIONS
#####################################

#
```

```
def auto_canny(image, sigma=0.33):
    # Compute the median of the single channel pixel intensities
    v = np.median(image)
```

```
13      # Apply automatic Canny edge detection using the computed median
14      lower = int(max(0, (1.0 - sigma) * v))
15      upper = int(min(255, (1.0 + sigma) * v))
16      edged = cv2.Canny(image, lower, upper)
17
18      return edged
19
20  #-------------------------------------------------------------------------------------------------------------------------------------------------
21
22  def diminishHeight(imRoi):
23      (h , w) = imRoi.shape[:2]
24      (hC, wC) = (2*h/3,2*w/3)
25      # Show only edges
26      autoCanny = auto_canny(imRoi)
27      # Find top and bottom points
28      bottomLines = list()
29      topLines = list()
30      LineQtd = 10
31      lines = cv2.HoughLines(autoCanny,1,1*np.pi/180,100)
32
33      if lines is None:
34          return None
35
36      if len(lines) < LineQtd:
37          LineQtd = len(lines)
38      for i in range (0,LineQtd):
39          for rho,theta in lines[i,:]:
40              a = np.cos(theta)
41              b = np.sin(theta)
42              x0 = int(a*rho)
43              y0 = int(b*rho)
44              if y0 < hC:
45                  topLines.append((y0, x0))
46              else:
47                  bottomLines.append((y0, x0))
48
49      if len(topLines) == 0 or len(bottomLines) == 0:
50          return None
51
52      # Find closest point to the image's center, both on the top and bottom
53      index = np.argmax(topLines, axis=0) # Regarding the height
54      topClosest = topLines[index[0]]
55      index = np.argmin(bottomLines, axis=0) # Regarding the height
56      bottomClosest = bottomLines[index[0]]
57      # Crop image, removing the black painted parts
58      imRoi = imRoi[topClosest[0]+7:bottomClosest[0]+7,:]
59      # imRoi = imRoi[int(imRoi.shape[0]*0.28):int(imRoi.shape[0]*0.72),:]
60
61      return imRoi
62
63  #-------------------------------------------------------------------------------------------------------------------------------------------------
64
65  # Darken the four sides, on the boundaries, removing the white stripes previously
66  # drawn to indicate where ROI was
67  def darkenImSides(im):
68      (h,w) = im.shape[:2]
69      im[0:3,:] = 0
70      im[:,0:3] = 0
71      im[h-2:h+1,:] = 0
72      im[:,w-2:w+1] = 0
73
74      return im
75
76  #-------------------------------------------------------------------------------------------------------------------------------------------------
77
78  def preCanny(imDigitsMasked):
79      # Smooth the image
80      # blur = cv2.blur(imDigitsMasked,(3,3))
81      blur = cv2.GaussianBlur(imDigitsMasked,(5,5),0)
82      blur = cv2.GaussianBlur(blur,(5,5),0)
83      blur = cv2.GaussianBlur(blur,(5,5),0)
84      # Smooth it a little more, not damaging the edges
85      bilateral = cv2.bilateralFilter(blur,5,100,100)
86      # Define contours of the digits
87      laplacian = cv2.Laplacian(bilateral,cv2.CV_64F)
```

```
    # Threshold = 0
    _, threshold = cv2.threshold(laplacian, 0, 255, cv2.THRESH_BINARY)
    # Apply opening and closing
    kernel = np.ones((2,2),np.uint8)
    opening = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, kernel)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)
    closing = np.uint8(closing)

    return closing

# ----------------------------------------------------------------------------------------------------------------

def locateDigits(closing, gotDigitsSequence, debug):

    # Apply Canny to show only edges
    edges = cv2.Canny(closing, 1000,1500, apertureSize = 3)
    digits = edges.copy()
    edges = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

    # Find contours and filter them to locate digits
    _,contours,_ = cv2.findContours(closing,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

    # test = edges.copy()
    # for i in range(0,len(contours)):
    #     cnt=contours[i]
    #     x,y,w,h = cv2.boundingRect(cnt)
    #     cv2.rectangle(test,(x-2,y-2),(x+w+2,y+h+2),(0,0,255),1)
    # cv2.imshow("test", test)

    if len(contours) is 0:
        return None

    rects = list()
    for i in range(0,len(contours)):
        cnt=contours[i]
        x,y,w,h = cv2.boundingRect(cnt)
        if float(h)/float(w) > 1.5 and float(h)/float(w) < 4.5\
        and h > 0.5*float(edges.shape[0]) and h < 0.9*float(edges.shape[0]):
            rects.append((cv2.boundingRect(cnt)))

    rects = sorted(rects, key=lambda x: x[0])

    if len(rects) > 0:
        distMIN = int(0.08*edges.shape[1])
        distMAX = int(0.16*edges.shape[1])
        START = int(0.08*edges.shape[1])
        END = int(0.98*edges.shape[1])

        filteredRects = list()
        loopEnd = len(rects)-1
        i=0
        while i < loopEnd:
            x1,y1,w1,h1 = rects[i]
            x2,y2,w2,h2 = rects[i+1]
            xC1 = x1+w1/2
            xC2 = x2+w2/2
            # Considering the position of the leftest and the rightest ractangle
            if xC1 > START and xC2 < END:
                if xC2-xC1 < distMIN and i is not len(rects)-2:
                    # Delete every next rectangle whose distance from the previous is below distMIN
                    del rects[i+1]
                    # Decrement loop end
                    loopEnd -= 1
                    # Analyze next rectangle
                    x3,y3,w3,h3 = rects[i+1]
                    xC3 = x3+w3/2
                    # If the distance is under the conditions, append this rectangle to the list
                    # Considering the position of the rightest ractangle
                    if xC3-xC1 > distMIN and xC3-xC1 < distMAX and xC3 < END:
                        filteredRects.append(rects[i])
                else:
                    filteredRects.append(rects[i])
            i += 1

        if len(filteredRects) == 6:
            x1,y1,w1,h1 = rects[i-1]
            x2,y2,w2,h2 = rects[i]
            xC1 = x1+w1/2
            xC2 = x2+w2/2
```

```
167                 if xC2-xC1 > distMIN and xC2-xC1 < distMAX and xC2 < END:
168                     filteredRects.append(rects[i])
169
170             # Make rectangles if, and only if, there are 7 digits
171             if len(filteredRects) == 7:
172                 for i in range(0,len(filteredRects)):
173                     x,y,w,h = filteredRects[i]
174                     cv2.rectangle(edges,(x-2,y-2),(x+w+2,y+h+2),(0,255,0),2)
175
176                 homeCoords = list(filteredRects)
177                 gotDigitsSequence = True
178
179                 if debug == True:
180                     cv2.imshow("Digits", edges)
181
182                 return digits, edges, homeCoords, gotDigitsSequence
183
184             else:
185
186                 return digits, edges, _ , gotDigitsSequence
187
188 #   ------------------------------------------------------------------------------------------------------------------------------------------------------
189
190 def createSample(imDigitsHomes, homeCoords):
191     rectDigits = list()
192     NoDIGITS = 7
193     maskDim = 40
194     digitDim = 30
195     for n in range(0,len(homeCoords)):
196
197         # Crop digit from image
198         PADDING = 1
199         x,y,w,h = homeCoords[n]
200         y_start = y-PADDING
201         x_start = x-PADDING
202         y_end = y+h+PADDING
203         x_end = x+w+PADDING
204         if y-PADDING < 0:
205             y_start = y
206         if x-PADDING < 0:
207             x_start = x
208         if y+h+PADDING > imDigitsHomes.shape[0]:
209             y_end = imDigitsHomes.shape[0]
210         if x+w+PADDING > imDigitsHomes.shape[1]:
211             x_end = imDigitsHomes.shape[1]
212         dn = imDigitsHomes[:y+h+PADDING,x-PADDING:x+w+PADDING]
213         label = "d"+str(n+1)
214         # Resize digit
215         h_digit = 20
216         w_digit = 10
217         rszd = cv2.resize(dn, (w_digit,h_digit), interpolation = cv2.INTER_AREA)
218
219         rectDigits.append(rszd)
220
221     return rectDigits
```

digitsRecognizer.py

```
1  from __future__ import print_function
2  from multiprocessing.pool import ThreadPool
3  import cv2
4  import numpy as np
5  from numpy.linalg import norm
6  from sklearn.neural_network import MLPClassifier
7  from sklearn.neighbors import KNeighborsClassifier
8  from sklearn import datasets, svm, metrics
9  import math
10 import pickle
11 import time
12 import datetime
13 import sys
14
15 '''
16     def deskew(img) and def preprocess_hog(digits) were found as samples
17     in the OpenCV library.
18 '''
19
20 w_SZ = 10
```

```
21  h_SZ = 20
22
23  def deskew(img):
24      # Align properly the little digit image if it is proned
25      m = cv2.moments(img)
26      if abs(m['mu02']) < 1e-2:
27          return img.copy()
28      skew = m['mu11']/m['mu02']
29      M = np.float32([[1, skew, -0.5*h_SZ*skew], [0, 1, 0]])
30      img = cv2.warpAffine(img, M, (w_SZ, h_SZ), flags=cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR)
31      return img
32
33  def preprocess_hog(digits):
34      # Perform hog in each little digit image among the seven ones
35      samples = []
36      for img in digits:
37          gx = cv2.Sobel(img, cv2.CV_32F, 1, 0)
38          gy = cv2.Sobel(img, cv2.CV_32F, 0, 1)
39          mag, ang = cv2.cartToPolar(gx, gy)
40          bin_n = 16
41          bin = np.int32(bin_n*ang/(2*np.pi))
42          bin_cells = bin[:10,:5], bin[10:,:5], bin[:10,5:], bin[10:,5:]
43          mag_cells = mag[:10,:5], mag[10:,:5], mag[:10,5:], mag[10:,5:]
44          hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in zip(bin_cells, mag_cells)]
45          hist = np.hstack(hists)
46
47          # Transform to Hellinger kernel
48          eps = 1e-7
49          hist /= hist.sum() + eps
50          hist = np.sqrt(hist)
51          hist /= norm(hist) + eps
52
53          samples.append(hist)
54      return np.float32(samples)
55
56  def predictDigits(model, rectDigits):
57      # Predict digits according to the specified machine learning model
58      intDigits = list()
59      for i in range(0,len(rectDigits)):
60          digits = np.array(rectDigits[i])
61          digits = digits.reshape(-1, h_SZ, w_SZ)
62          digits = list(map(deskew, digits))
63          digits = preprocess_hog(digits)
64          intDigits.append(int(model.predict(digits)[0]))
65      return intDigits
66
67  def set2float(intDigits):
68      # Transform set of digits into a float number
69      reading = 0
70      qtd = len(intDigits)
71      for i in range(0, qtd):
72          reading += intDigits[(qtd-1)-i]*math.pow(10,i)
73      return float(reading)/1000.0
74
75  def set2string(intDigits):
76      # Transform set of digits into a string
77      reading = ""
78      qtd = len(intDigits)
79      for i in range(0, qtd):
80          reading = str(reading) + str(intDigits[i])
81      return reading
82
83  def send2file(utc, timestamp, reading, usage, dV, dt, Q, Qma, counter, path, file, mode):
84      # Write data into hydrometer.log and prevHydroMetrics.var
85      f = open(path+file, mode)
86      line = str(utc)+"    "+str(timestamp)+"    "+str(reading)+"     "+str("%8.3f"%usage)\
87      +"  "+str("%6.1f"%dV)+"     "+str("%4d"%dt)+"    "+str("%6.1f"%Q)+"    "+str("%8.1f"%Qma)\
88      +"      "+str(counter)+"\n"
89      f.write(line)
90      f.close()
91      return None
92
93  def recogNsave(MIN_dV, Qn, FoS, rectDigits, roiBGR, edges, rightSideUP, ROI_PATH, IMS_UPDATE_PATH,\
94                 LOGFILE_PATH, METRICS_PATH, LEARNING_PATH, ML_CLASSIFIER, counter, debug):
95
96      # Recognize and save digits
97
98      if ML_CLASSIFIER == 1:
99          model="digits_svm.pkl"
100     if ML_CLASSIFIER == 2:
101         model="digits_mlp.pkl"
```

```
102     if ML_CLASSIFIER == 3:
103         model="digits_knn.pkl"
104 
105     if sys.version_info >= (3,0):
106         with open(LEARNING_PATH+model, "rb") as f:
107             svmModel = pickle.load(f, encoding="latin1")
108     else:
109         with open(LEARNING_PATH+model, "rb") as f:
110             svmModel = pickle.load(f)
111 
112     intDigits = predictDigits(svmModel, rectDigits)
113     reading = set2float(intDigits)
114     strReading = set2string(intDigits)
115 
116     f = open(METRICS_PATH+"prevHydroMetrics.var","r")
117     lines = f.readlines()
118     part = lines[0].split()
119     prevTime = float(part[1])
120     prevReading = float(part[3])
121     prevQ = float(part[6])
122     prevQa = float(part[7])
123     f.close()
124 
125     # Just proceed if the system time is bigger than the previous time the program executed
126     if float(time.time()) > prevTime:
127 
128         utc = datetime.datetime.utcnow().strftime("%Y/%m/%d@%H:%M:%S")
129         timestamp = int(time.time())
130         dV = (reading-prevReading)*1000
131         dt = float(time.time()-prevTime)
132         Q = (float(dV)/dt)*3600
133         Qma = float((Q + prevQ))/2.0
134 
135         '''Only proceed if...
136             1. the reading of the consumption predicted is bigger than the previous value
137             2. the variation of volume is bigger than the minimum variation accepted
138             3. if the flow detected is under the nominal or expected flow value
139         '''
140         if reading > prevReading and dV >= MIN_dV and Q <= Qn*FoS:
141             # Save processed whole image, the encountered digits, and the rectangular BGR ROI
142             cv2.imwrite(IMS_UPDATE_PATH+"roi.jpg", rightSideUP, [cv2.IMWRITE_JPEG_QUALITY, 75])
143             cv2.imwrite(IMS_UPDATE_PATH+"digits.jpg", edges, [cv2.IMWRITE_JPEG_QUALITY, 75])
144             cv2.imwrite(ROI_PATH+str(timestamp)+".jpg",roiBGR, [cv2.IMWRITE_JPEG_QUALITY, 75])
145 
146             if debug:
147                 print ("")
148                 print ("  UTC         = ", utc)
149                 print ("  Timestamp   = ", timestamp)
150                 print ("  READING     = ", strReading)
151                 print ("  Usage       = ", "%.3f" %reading, "m3")
152                 print ("  dV          = ", "%.1f" %dV, "L")
153                 print ("  dt          = ", "%d" %dt, "s")
154                 print ("  Q           = ", "%.1f" %Q, "L/h")
155                 print ("  Qma         = ", "%.1f" %Qma, "L/h")
156                 print ("  Try         = ", counter, "x")
157 
158             send2file(utc, timestamp, strReading, reading, dV, dt, Q, Qma, counter, METRICS_PATH, "
                    prevHydroMetrics.var", "w+")
159             send2file(utc, timestamp, strReading, reading, dV, dt, Q, Qma, counter, LOGFILE_PATH, "
                    hydrometer.log", "a+")
160 
161         elif reading == prevReading:
162 
163             if debug:
164                 print ("")
165                 print ("  UTC         = ", utc)
166                 print ("  Timestamp   = ", timestamp)
167                 print ("  READING     = ", strReading)
168                 print ("  Usage       = ", "%.3f" %reading, "m3")
169                 print ("  dV          = ", "%.1f" %dV, "L")
170                 print ("  dt          = ", "%d" %dt, "s")
171                 print ("  Q           = ", "%.1f" %Q, "L/h")
172                 print ("  Qma         = ", "%.1f" %Qma, "L/h")
173                 print ("  Try         = ", counter, "x")
174 
175             send2file(utc, timestamp, strReading, reading, dV, dt, Q, Qma, counter, METRICS_PATH, "
                    prevHydroMetrics.var", "w+")
```

htmlGen.py

```
from __future__ import print_function

if __name__ == '__main__':

        METRICS_PATH = "/usr/local/etc/ocr/"
        HTML_PATH = "/usr/http/"

        f = open(METRICS_PATH+"prevHydroMetrics.var","r")
        lines = f.readlines()
        part = lines[0].split()
        utc = str(part[0])
        usage = str(part[3])
        Q = str(part[6])
        Qa = str(part[7])
        f.close()


        f = open(HTML_PATH+"index.html","w+")
        f.write(\
        """
        <!doctype html>
        <html>
        <meta http-equiv="Refresh" content="30; url=./index.html">
        <body bgcolor="#679AAA" text="ffffff")
                <hr>
                <center>
                <h1> Laboratório GIAI/UnB, Brasília </h1>
                <h2>
                        <font size="6">
                                <p>
                                        HID: 0001
                                </p>
                        </font>

                        <p style="line-height:1.25;">
                                Reading
                                <br>
                                <img src="roi.jpg" alt="Region of Interest" title="Region of Interest"
                                     height="320" width="427">
                        </p>

                        <p>
                                Digits Detection
                                <br>
                                <img src="digits.jpg" alt="Detected Digits" title="Detected Digits"
                                     height=40 width="220">
                        </p>

                        <p>
                                Metrics
                                <br>
                        </p>
                        <p style="text-align:justify; margin-left:400px">
                                <font size="3">
                                        UTC: """+utc+"""
                                        <br>
                                        Instant Usage: """+usage+""" m3
                                        <br>
                                        Estimated Flow: """+Q+""" L/h
                                        <br>
                                        Average Flow: """+Qa+""" L/h
                                        <br>
                                </font>
                        </p>


                </h2>
        </body>
        </html>
        """
        )
        f.close()
```

common.py

```
#/usr/bin/env python

'''
This module contais some common routines used by other samples.
'''
```

```
6   from __future__ import print_function
7   import numpy as np
8   import cv2
9   import os
10  from contextlib import contextmanager
11  import itertools as it
12
13  image_extensions = ['.bmp', '.jpg', '.jpeg', '.png', '.tif', '.tiff', '.pbm', '.pgm', '.ppm']
14
15  class Bunch(object):
16      def __init__(self, **kw):
17          self.__dict__.update(kw)
18      def __str__(self):
19          return str(self.__dict__)
20
21  def splitfn(fn):
22      path, fn = os.path.split(fn)
23      name, ext = os.path.splitext(fn)
24      return path, name, ext
25
26  def anorm2(a):
27      return (a*a).sum(-1)
28  def anorm(a):
29      return np.sqrt( anorm2(a) )
30
31  def homotrans(H, x, y):
32      xs = H[0, 0]*x + H[0, 1]*y + H[0, 2]
33      ys = H[1, 0]*x + H[1, 1]*y + H[1, 2]
34      s  = H[2, 0]*x + H[2, 1]*y + H[2, 2]
35      return xs/s, ys/s
36
37  def to_rect(a):
38      a = np.ravel(a)
39      if len(a) == 2:
40          a = (0, 0, a[0], a[1])
41      return np.array(a, np.float64).reshape(2, 2)
42
43  def rect2rect_mtx(src, dst):
44      src, dst = to_rect(src), to_rect(dst)
45      cx, cy = (dst[1] - dst[0]) / (src[1] - src[0])
46      tx, ty = dst[0] - src[0] * (cx, cy)
47      M = np.float64([[ cx,  0, tx],
48                      [  0, cy, ty],
49                      [  0,  0,  1]])
50      return M
51
52
53  def lookat(eye, target, up = (0, 0, 1)):
54      fwd = np.asarray(target, np.float64) - eye
55      fwd /= anorm(fwd)
56      right = np.cross(fwd, up)
57      right /= anorm(right)
58      down = np.cross(fwd, right)
59      R = np.float64([right, down, fwd])
60      tvec = -np.dot(R, eye)
61      return R, tvec
62
63  def mtx2rvec(R):
64      w, u, vt = cv2.SVDecomp(R - np.eye(3))
65      p = vt[0] + u[:,0]*w[0]    # same as np.dot(R, vt[0])
66      c = np.dot(vt[0], p)
67      s = np.dot(vt[1], p)
68      axis = np.cross(vt[0], vt[1])
69      return axis * np.arctan2(s, c)
70
71  def draw_str(dst, x, y, s):
72      cv2.putText(dst, s, (x+1, y+1), cv2.FONT_HERSHEY_PLAIN, 1.0, (0, 0, 0), thickness = 2, lineType=cv2.
            CV_AA)
73      cv2.putText(dst, s, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.0, (255, 255, 255), lineType=cv2.CV_AA)
74
75  class Sketcher:
76      def __init__(self, windowname, dests, colors_func):
77          self.prev_pt = None
78          self.windowname = windowname
79          self.dests = dests
80          self.colors_func = colors_func
81          self.dirty = False
82          self.show()
83          cv2.setMouseCallback(self.windowname, self.on_mouse)
84
85      def show(self):
```

```
86          cv2.imshow(self.windowname, self.dests[0])
87
88      def on_mouse(self, event, x, y, flags, param):
89          pt = (x, y)
90          if event == cv2.EVENT_LBUTTONDOWN:
91              self.prev_pt = pt
92          if self.prev_pt and flags & cv2.EVENT_FLAG_LBUTTON:
93              for dst, color in zip(self.dests, self.colors_func()):
94                  cv2.line(dst, self.prev_pt, pt, color, 5)
95              self.dirty = True
96              self.prev_pt = pt
97              self.show()
98          else:
99              self.prev_pt = None
100
101
102 # palette data from matplotlib/_cm.py
103 _jet_data =   {'red':   ((0., 0, 0), (0.35, 0, 0), (0.66, 1, 1), (0.89,1, 1),
104                          (1, 0.5, 0.5)),
105                'green': ((0., 0, 0), (0.125,0, 0), (0.375,1, 1), (0.64,1, 1),
106                          (0.91,0,0), (1, 0, 0)),
107                'blue':  ((0., 0.5, 0.5), (0.11, 1, 1), (0.34, 1, 1), (0.65,0, 0),
108                          (1, 0, 0))}
109
110 cmap_data = { 'jet' : _jet_data }
111
112 def make_cmap(name, n=256):
113     data = cmap_data[name]
114     xs = np.linspace(0.0, 1.0, n)
115     channels = []
116     eps = 1e-6
117     for ch_name in ['blue', 'green', 'red']:
118         ch_data = data[ch_name]
119         xp, yp = [], []
120         for x, y1, y2 in ch_data:
121             xp += [x, x+eps]
122             yp += [y1, y2]
123         ch = np.interp(xs, xp, yp)
124         channels.append(ch)
125     return np.uint8(np.array(channels).T*255)
126
127 def nothing(*arg, **kw):
128     pass
129
130 def clock():
131     return cv2.getTickCount() / cv2.getTickFrequency()
132
133 @contextmanager
134 def Timer(msg):
135     print (msg, '...',
136     start = clock())
137     try:
138         yield
139     finally:
140         print ("%.2f ms" % ((clock()-start)*1000))
141
142 class StatValue:
143     def __init__(self, smooth_coef = 0.5):
144         self.value = None
145         self.smooth_coef = smooth_coef
146     def update(self, v):
147         if self.value is None:
148             self.value = v
149         else:
150             c = self.smooth_coef
151             self.value = c * self.value + (1.0-c) * v
152
153 class RectSelector:
154     def __init__(self, win, callback):
155         self.win = win
156         self.callback = callback
157         cv2.setMouseCallback(win, self.onmouse)
158         self.drag_start = None
159         self.drag_rect = None
160     def onmouse(self, event, x, y, flags, param):
161         x, y = np.int16([x, y]) # BUG
162         if event == cv2.EVENT_LBUTTONDOWN:
163             self.drag_start = (x, y)
164         if self.drag_start:
165             if flags & cv2.EVENT_FLAG_LBUTTON:
166                 xo, yo = self.drag_start
```

```
                x0, y0 = np.minimum([xo, yo], [x, y])
                x1, y1 = np.maximum([xo, yo], [x, y])
                self.drag_rect = None
                if x1-x0 > 0 and y1-y0 > 0:
                    self.drag_rect = (x0, y0, x1, y1)
            else:
                rect = self.drag_rect
                self.drag_start = None
                self.drag_rect = None
                if rect:
                    self.callback(rect)
    def draw(self, vis):
        if not self.drag_rect:
            return False
        x0, y0, x1, y1 = self.drag_rect
        cv2.rectangle(vis, (x0, y0), (x1, y1), (0, 255, 0), 2)
        return True
    @property
    def dragging(self):
        return self.drag_rect is not None


def grouper(n, iterable, fillvalue=None):
    '''grouper(3, 'ABCDEFG', 'x') --> ABC DEF Gxx'''
    args = [iter(iterable)] * n
    return it.zip_longest(fillvalue=fillvalue, *args)

def mosaic(w, imgs):
    '''Make a grid from images.

    w    -- number of grid columns
    imgs -- images (must have same size and format)
    '''
    imgs = iter(imgs)
    img0 = imgs.__next__()
    pad = np.zeros_like(img0)
    imgs = it.chain([img0], imgs)
    rows = grouper(w, imgs, pad)
    return np.vstack(map(np.hstack, rows))

def getsize(img):
    h, w = img.shape[:2]
    return w, h

def mdot(*args):
    return reduce(np.dot, args)

def draw_keypoints(vis, keypoints, color = (0, 255, 255)):
    for kp in keypoints:
            x, y = kp.pt
            cv2.circle(vis, (int(x), int(y)), 2, color)
```

digits.py

```
#!/usr/bin/env python

'''
This module was found as a sample in OpenCV Library. Some good part of it was removed
or modified.

SVM, KNearest, and MLP digit recognition.

Following preprocessing is applied to the dataset:
 - Moment-based image deskew (see deskew())
 - Digit images are split into 4 10x5 cells and 16-bin
   histogram of oriented gradients is computed for each
   cell
 - Transform histograms to space with Hellinger metric (see [1] (RootSIFT))

[1] R. Arandjelovic, A. Zisserman
    "Three things everyone should know to improve object retrieval"
    http://www.robots.ox.ac.uk/~vgg/publications/2012/Arandjelovic12/arandjelovic12.pdf

Usage:
   digits.py
'''


# Python 2/3 compatibility
from __future__ import print_function
```

```
27
28  # built-in modules
29  from multiprocessing.pool import ThreadPool
30
31  import cv2
32
33  import numpy as np
34  from numpy.linalg import norm
35
36  # local modules
37  from common import clock, mosaic
38
39  from sklearn.neural_network import MLPClassifier
40  from sklearn.neighbors import KNeighborsClassifier
41  from sklearn import datasets, svm, metrics
42  from sklearn.naive_bayes import GaussianNB
43  from sklearn.naive_bayes import MultinomialNB
44  from sklearn.naive_bayes import BernoulliNB
45
46  import pickle
47
48  w_SZ = 10
49  h_SZ = 20
50  CLASS_N = 10
51  DIGITS_FN = 'digits-5000-AM.png'
52
53  def split2d(img, cell_size, flatten=True):
54      h, w = img.shape[:2]
55      sx, sy = cell_size
56      cells = [np.hsplit(row, w//sx) for row in np.vsplit(img, h//sy)]
57      cells = np.array(cells)
58      if flatten:
59          cells = cells.reshape(-1, sy, sx)
60      return cells
61
62  def load_digits(fn):
63      print('loading "%s" ...' % fn)
64      digits_img = cv2.imread(fn, 0)
65      digits = split2d(digits_img, (w_SZ, h_SZ))
66      labels = np.repeat(np.arange(CLASS_N), len(digits)/CLASS_N)
67      return digits, labels
68
69  def deskew(img):
70      m = cv2.moments(img)
71      if abs(m['mu02']) < 1e-2:
72          return img.copy()
73      skew = m['mu11']/m['mu02']
74      M = np.float32([[1, skew, -0.5*h_SZ*skew], [0, 1, 0]])
75      img = cv2.warpAffine(img, M, (w_SZ, h_SZ), flags=cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR)
76      return img
77
78  def evaluate_model(model, digits, samples, labels):
79      resp = model.predict(samples)
80      err = (labels != resp).mean()
81      print('error: %.2f %%' % (err*100))
82
83      confusion = np.zeros((10, 10), np.int32)
84      for i, j in zip(labels, resp):
85          confusion[int(i), int(j)] += 1
86      print('confusion matrix:')
87      print(confusion)
88      print()
89
90      vis = []
91      for img, flag in zip(digits, resp == labels):
92          img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
93          if not flag:
94              img[...,:2] = 0
95          vis.append(img)
96      return mosaic(25, vis)
97
98  def preprocess_hog(digits):
99      samples = []
100     for img in digits:
101         gx = cv2.Sobel(img, cv2.CV_32F, 1, 0)
102         gy = cv2.Sobel(img, cv2.CV_32F, 0, 1)
103         mag, ang = cv2.cartToPolar(gx, gy)
104         bin_n = 16
105         bin = np.int32(bin_n*ang/(2*np.pi))
106         bin_cells = bin[:10,:5], bin[10:,:5], bin[:10,5:], bin[10:,5:]
107         mag_cells = mag[:10,:5], mag[10:,:5], mag[:10,5:], mag[10:,5:]
```

```
108             hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in zip(bin_cells, mag_cells)]
109             hist = np.hstack(hists)
110
111             # transform to Hellinger kernel
112             eps = 1e-7
113             hist /= hist.sum() + eps
114             hist = np.sqrt(hist)
115             hist /= norm(hist) + eps
116
117             samples.append(hist)
118         return np.float32(samples)
119
120
121     if __name__ == '__main__':
122         print(__doc__)
123         print()
124
125         digits, labels = load_digits(DIGITS_FN)
126
127         print('preprocessing...')
128         # shuffle digits
129         rand = np.random.RandomState(321)
130         shuffle = rand.permutation(len(digits))
131         digits, labels = digits[shuffle], labels[shuffle]
132
133         digits2 = list(map(deskew, digits))
134         samples = preprocess_hog(digits2)
135
136         train_n = int(0.9*len(samples))
137         cv2.imshow('test set', mosaic(25, digits[train_n:]))
138         digits_train, digits_test = np.split(digits2, [train_n])
139         samples_train, samples_test = np.split(samples, [train_n])
140         labels_train, labels_test = np.split(labels, [train_n])
141
142         trainModel=True
143
144         if trainModel:
145             knnModel = KNeighborsClassifier(n_neighbors=4)
146             print('training k-Nearest Neighbor...')
147             knnModel.fit(samples_train,labels_train)
148             print('saving KNearest as "digits_knn.dat"...')
149             pickle.dump(knnModel, open('digits_knn.pkl', 'wb'))
150         else:
151             knnModel = pickle.load(open('digits_knn.pkl', 'rb'))
152         vis = evaluate_model(knnModel, digits_test, samples_test, labels_test)
153         cv2.imshow('KNearest test', vis)
154
155         if trainModel:
156             svmModel = svm.SVC(C=2.67, gamma=5.383)
157             print('\ntraining Support Vector Machine...')
158             svmModel.fit(samples_train, labels_train)
159             print('saving SVM as "digits_svm.dat"...')
160             pickle.dump(svmModel, open('digits_svm.pkl', 'wb'))
161         else:
162             svmModel = pickle.load(open('digits_svm.pkl', 'rb'))
163         vis = evaluate_model(svmModel, digits_test, samples_test, labels_test)
164         cv2.imshow('SVM test', vis)
165
166         if trainModel:
167             mlpModel = MLPClassifier(tol=0.000010, verbose=False, random_state=1, hidden_layer_sizes=(380,),
                    learning_rate_init=0.01, alpha=0.00001)
168             print('\ntraining Multi-Layer Percepton...')
169             mlpModel.fit(samples_train,labels_train)
170             print('saving MLP as "digits_mlp.dat"...')
171             pickle.dump(mlpModel, open('digits_mlp.pkl', 'wb'))
172             # print(mlpModel)
173         else:
174             mlpModel = pickle.load(open('digits_mlp.pkl', 'rb'))
175         vis = evaluate_model(mlpModel, digits_test, samples_test, labels_test)
176         cv2.imshow('MLP test', vis)
177
178         cv2.waitKey(0)
```