Sistema de Automação de Testes para *software* Embarcado com Verificação por Imagem

Anderson Carvalho Morelli[1], **Daniela Marques**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) – Câmpus Hortolândia – São Paulo – SP – Brasil

morelli.anderson@gmail.com, marquesdaniela@ifsp.edu.br

Abstract. *Thesoftware testing phase is important in quality assurance. When the tests are applied in situations where it is necessary to test an embedded-software that has a human-computer interface, these tests are usually performed manually, with the testers operating the DUT (Device Under Test) and verifying that the elements presented on the screen are really expected. There are intelligent camera systems used in HIL (Hardware In the Loop) systems to perform automatic tests on embedded systems that have this characteristic of having a human-computer interface, using the capture of these images and verification with previous images that are used as reference a picture of the expected behavior for the test. The problem is that this equipment is usually very expensive, making access difficult for small companies and startups. The objective of this work is to develop a low-cost test automation system, using conventional webcams, Arduino and open source tools in order to exclude human interaction in the execution of the test cases from the necessary activation to the verification of the result on the screen.*

Resumo. *A fase de teste desoftware é importante na garantia da qualidade. Quando os testes são aplicados em situações onde é necessário testar umsoftware embarcado que possui uma interface humano-computador normalmente estes testes são executados de forma manual, com os testadores operando o DUT (Device Under Test) e verificando se os elementos apresentados na tela são realmente o esperado. Existem sistemas de câmeras inteligentes utilizadas em sistemas de HIL (Hardware In the Loop) para executar os testes automáticos em sistemas embarcados que possúem essa característica de ter uma interface humano-computador, utilizando a captura destas imagens e verificação com imagens anteriores que são utilizadas como referência de uma imagem do comportamento esperado para o teste. O problema é que estes equipamentos costumam ser muito caros, dificultando o acesso para empresas pequenas e startups. O objetivo deste trabalho é desenvolver um sistema de automação de testes de baixo custo, utilizando webcams convencionais, Arduino e ferramentas open source com o intuito de excluir a interação humana na execução dos testescases desde o acionamento necessário quanto a verificação do resultado na tela.*

1. Introdução

De acordo com o relatório da pesquisa, o *"Embedded System Market by Hardware (MPU, MCU, Application-specific Integrated Circuits, DSP, FPGA, and Memories),software*

(Middleware, Operating Systems), System Size, Functionality, Application, Region - Global Forecast to 2025", publicado pela [Markets and Markets 2020], o mercado de sistemas embarcados deverá crescer cerca de 35% no período de 2020 até 2025 passando de US$ 86,5 bilhões em 2020 para USD 116,2 bilhões por 2025, o que representa um crescimento anual de aproximadamente 6%. Mas o que são sistemas embarcados e por qual motivo é um mercado que vem crescendo nos últimos anos e que tem previsão de crescimento para o futuro?

Como qualquer outro*software*, os sistemas embarcados também precisam passar por fases de testes para assegurar uma melhor qualidade. Porém, o problema é que como estes sistemas são encapsulados em um produto suas entradas e saídas norlamente estão ligadas em sensores e atuadores. Por exemplo, caso o produto que vai ser testado seja um sistema que faz gerenciamento da temperatura de uma sala e este sistema possui como entrada um sensor de temperatura e como saída um relé que vai acionar um exaustor. Para testar o produto seria necessário que ter o sensor e o exaustor, e também seria necessário colocar o sensor em uma temperatúra onde é esperado o acionamento do exaustor, isso tudo acaba dificultando a execução dos testes.

Então para garantir um controle melhor o ideal seria ter uma maneira de simular o comportamento do sensor e conseguir medir a saída do sistema, essa técnica chama-se HIL (*Hardware-In-the-Loop*). De acordo com o trabalho de [Botelho 2021]

> "Um *Hardware-In-the-Loop* (HIL) consiste em um sistema retroalimentado que comunica o computador de bordo (ECU) do produto com sua versão virtual ou gêmeo digital. Como deve interagir com a realidade, operar em tempo real torna-se um requisito fundamental. *Hardwares-In-the-Loop* são compostos por versões virtuais de diversos sensores como acelerômetros ou giroscópios, atuadores como motores elétricos, turbinas e bombas, todos conectados ao computador central".

O objetivo deste trabalho é desenvolver um sistema semelhante a um HIL utilizando ferramentas gratuítas e um *hardware* de baixo custo.

2. Trabalho correlato

Testes voltados para sistemas embarcados é uma área pouco explorada no Brasil. Em pesquisas realizadas por palavras-chave como "teste embarcado", "teste display"ou "automação teste interface usuário"em grandes bases de dados brasileiras como o SciELO[1] e o portal de periódicos da CAPES[2], as pesquisas não retornam nenhuma ocorrência de trabalhos no Brasil, foi possível localizar poucos trabalhos buscando por "*UI test automation*"em outros países como Estados Unidos e Índia. Na subseção a seguir é apresentado um trabalho que serviu de inspiração pela similaridade com o trabalho proposto neste artigo.

[1]https://scielo.org

[2]https://www-periodicos-capes-gov-br.ezl.periodicos.capes.gov.br

2.1. *Advanced UI test automation (AUTA) for BIOS validation using OpenCV and OCR*

O trabalho de [Mohammed et al. 2021] sobre o desenvovimento de um sistema de automação de testes focados na interface com usuário (UI - *User Interface*) de um sistema BIOS (*Basic In/Out System*) utiliza um microcontrolador para simular o sistema de *mouse* e de teclado que vai fornecer os *inputs* para a BIOS que está sendo testada e o *output* da BIOS seria a tela do computador que ela está ligada. Contudo, no artigo é instalada uma placa de captura de vídeo ligada em paralelo com essa saída para fazer a captura do sinal de vídeo que a BIOS está enviando ao monitor. Essas imagens capturadas serão utilizadas no processamento de imagem. A Figura 1 mostra a arquitetura do projeto. O artigo correlato e este trabalho utilizam a biblioteca OpenCV. O trabalho correlato mostra algumas

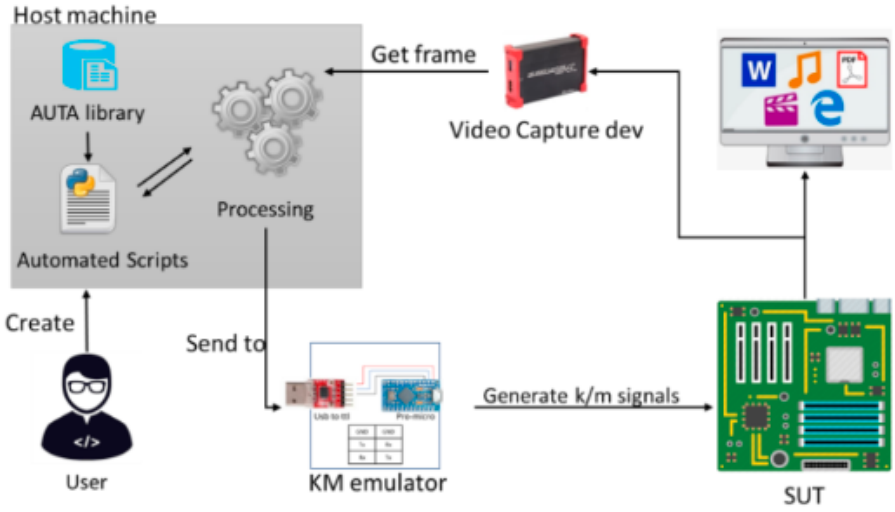

Figura 1. Arquitetura do projeto AUTA. Fonte:[Mohammed et al. 2021]

funções desenvolvidas para testar as imagens: *IAST - Image Analyzing on Selected Text, IATM - Image Analyzing on Text Menu* e *IAPM - Image Analyzing on Pop-up Menu*. Essa abordagem também foi adotada neste trabalho, porém foi desenvolvido somente uma função de detecção de texto que vai comparar o texto lido na imagem com uma *string* que vai ser passada para a função de detecção de *templates* que vai procurar por uma imagem padrão dentro da imagem capturada e por último uma função para detecção de cores. As funções que foram desenvolvidas neste trabalho serão apresentadas mais detalhadamente no tópico 6.2.4.

Para verificar o ganho com tempo de execução dos testes foi criado um *script* de teste com 15 *testcases* aleatórios que demorou 42 minutos para serem executados. Com isso pode-se estimar um tempo de 4 a 6 horas para executar uma validação com 125 *testcases* utilizando este sistema automatizado, contra aproximadamente 48 horas para execução dos mesmos testes de forma manual.Com essa informação foi possível afirmar que em uma validação totalmente automatizada da BIOS demoraria em média um oitavo do tempo da mesma validação sendo executada de forma manual, isso com um nível de precisão de 90%.

3. Referencial Teórico

3.1. Sistemas Embarcados

A definição de um sistema embarcado é "Sistemas embarcados são sistemas de processamento de informações embutidos em um produto fechado"[Marwedel 2021], ou seja, um sistema embarcado é construído para o único propósito da sua aplicação.

O primeiro grande sistema embarcado que se tem conhecimento é o ACG (*Apollo Guidance Computer*) que era um computador responsável pelo controle das espaçonaves Apollo em suas viagens para a Lua entre os anos 60 e 70 [Lima 2014]. Atualmente podemos ver aplicações de sistemas embarcados em diverssas áreas como o marcapasso, sistemas de segurança de veículos, câmeras inteligentes capazes de detectar princípios de incêndio, inúmeros eletrodomésticos que facilitam o dia a dia das pessoas, até mesmo urnas eletrônicas que auxiliam no processo eleitoral.

Este avanço tecnológico em sistemas embarcados se dá principalmente por três fatores, o aumento do poder de processamento dos microprocessadores, a diminuição constante do tamanho dos componentes eletrônicos e o avanço de pesquisa de gestão de energia destes componentes fazendo com que equipamentos que precisam ficar fora da rede elétrica consigam aumentar o seu tempo de funcionamento em baterias. Entretanto, todo esse avanço causa uma espécie de efeito colateral, estes sistemas embarcados estão cada vez mais complexos e muitas vezes necessitam de alguma interação humano computador, estes fatores acabam deixando o desenvolvimento das aplicações que utilizam sistemas embarcados mais complexas, demandando um tempo maior de desenvolvimento e uma das causas dessa demora é a fase de testes.

3.2. Arduino

De acordo com o trabalho de [Olivera and Zanetti 2015]:

> "Arduino uma plataforma de *hardware open source*, projetada sobre o microcontrolador Atmel AVR, que pode ser programado através de uma linguagem de programação similar a C/C++, permitindo a elaboração de projetos com um conhecimento mínimo ou mesmo nenhum de eletrônica. Foi criado com o objetivo de fornecer uma plataforma fácil de prototipação de projetos interativos, unindo*software* e *hardware*, características da Computação Física".

Para o desenvolvimento do projeto utilizamos o Arduino Uno que utiliza um microcontrolador ATmega328P e suas especificações técnicas estão podem ser virificadas na Tabela 1. Já Figura 2 é uma representação da placa do Arduino Uno com as posssíveis configurações dos seus pinos.

3.3. Visão computacional

De acordo com [IBM 2022], "A visão computacional é uma área da inteligência artificial capaz de permitir que computadores e sistemas obtenham informações através de imagens digitais, vídeos ou outras entradas visuais e que tomem ações com base nessas informações. Se a inteligência artificial permite que os computadores pensem, a visão coputacional permite que eles vejam, observem e entendam"

Tabela 1. Especificações técnicas do Arduino Uno. Fonte:[Arduino 2022]

Microcontrolador	ATmega328P
Tensão de Operação	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limites)	6-20V
Entrada e saída digital	14 (6 com configuração para PWM)
Entradas analógicas	6
Corente máxima I/O Pin	20 mA
Corrente máxima 3.3V Pin	50 mA
Memória flash	32 KB (0.5 KB utilizado pelo bootloader)
SRAM	2 KB
EEPROM	1 KB
Clock	16 MHz
Comprimento	68.6 mm
Largura	53.4 mm

3.3.1. Como as máquinas enxergam as imagens?

De acordo com o trabalho de [Elgendy 2019], "As imagens podem ser representadas como uma função de duas variáveis, X e Y, que definem uma área bidimensional. As imagens digitais são feitas de uma grade de *pixels*. Um *Pixel* é o bloco de construção bruto de uma imagem. Toda imagem é composta por um conjunto de *pixels* em que seus valores representam a intensidade da luz que aparece em um determinado local da nossa imagem.

A Figura 3 mostra uma matriz de tamanho de 32 x 16. Isso significa que as dimensões da imagem são 32 *pixels* de largura e 16 *pixels* de altura. O eixo X começa de 0 a 31 e o eixo Y de 0 a 16. A imagem tem 32×16 = 512 *pixels*. Nesta imagem em tons de cinza, cada *pixel* contém um valor que representa a intensidade da luz neste *pixel* específico. Os valores de*pixel* variam de 0 a 255. Como o valor de *pixel* representa a intensidade da luz, o valor 0 representa *pixels* escuros (preto), 255 é claro (branco) e os valores intermediários representam a intensidade na escala de cinza.

Você pode ver que o sistema de coordenadas da imagem é semelhante ao sistema de coordenadas cartesianas: as imagens são bidimensionais e estão no plano x-y. A origem (0, 0) está no canto superior esquerdo da imagem. Para representar um pixel específico usamos as seguintes notações: “F” como uma função, “x,y” a localização do pixel nas coordenadas x,y. Por exemplo, o pixel localizado em x=9 e y= 14 é branco, isso é representado pela seguinte função: F(9, 14) = 255. Da mesma forma, o pixel (27,7) que fica na roda dianteira do a motocicleta é preta, representada como F(27,7) = 0."

O mesmo princípio acontece para as imagens coloridas, porém em cada pixel ao invés de termos somente um valor de 0 a 255 teremos um *array* de três valores, um para cada canal RGB, onde cada um deles também vai de 0 até 255 a diferença é que com imagens coloridas precisamos misturar os canais (ter valores acima de 0 em mais de um canal) para conseguirmos cores diferentes das cores primárias.

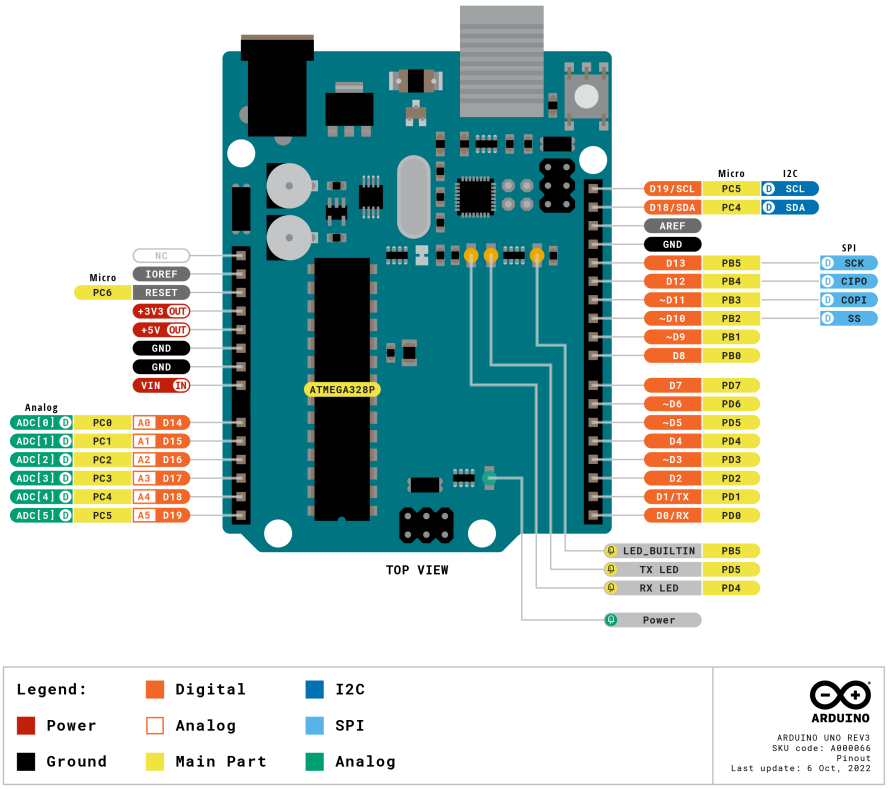

Figura 2. Representação do Arduino Uno com suas configurações de pinos. Fonte:[Arduino 2022]

3.3.2. *Optical Character Recognition* (OCR)

Sistemas OCR transformam uma imagem bidimensional que contem textos impressos por computador ou até mesmos escritos à mão em algo que seja "legível"pelos computadores. Para conseguir este resultado os sistemas OCR realizam diversos processos para trazer uma acertividade melhor possível, estes processo são [Nanonets 2022]: Processamento da imagem, Localização de texto, Segmentação de caracteres, Reconhecimento de caracteres e Pós processamento.

Neste trabalho utilizaremos a *engine Tesseract* que começou à ser desenvolvida pela HP Labs por volta de 1984 e desde 2006 vêm sendo desenvolvida pela Google [Nanonets 2022].

4. Metodologia

A metodologia utilizada para o desenvolvimento deste trabalho foi uma pesquisa com caráter descritivo, para levantar os principais problemas presentes no ambiente de testes de sistemas embarcados que possuem IHM (Interação Humano Computador) quando são utilizados apenas testes manuais. Após isso iniciou-se um processo de pesquisa de carater exploratório para verificar maneiras de automatizar este processo de teste à fim de resolver os problemas levantados na etapa anterior.

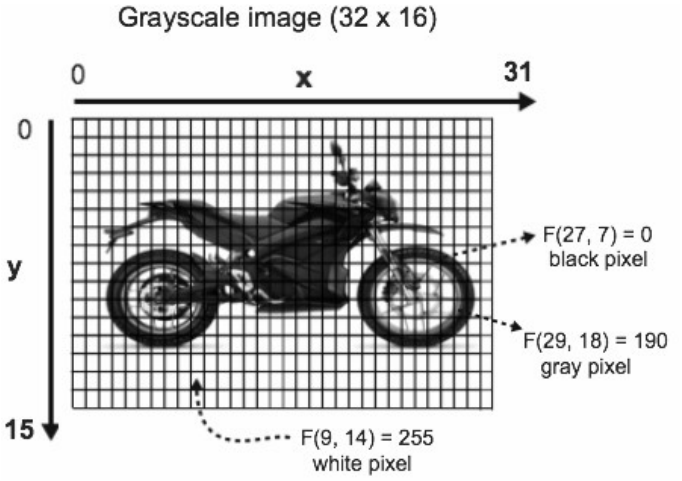

Figura 3. Figura que representa a matriz formada pela imagem. Fonte:[Elgendy 2019]

5. Ferramentas

5.1. Python

Python é uma das linguagens de programação que mais vem crescendo no quesito utilização devido ao fato de ser mais legível se comparada a outras alternativas, o que facilita muito o seu aprendizado e também a manutenção de códigos [Brasil 2015]. Outras características que fazem com que a utilização desa linguagem seja utilizada cada vez mais é o seu suporte avançado para mecanismos de reutilização de código e a vasta opções de bibliotecas (módulos) para as mais diversas finalidades.

Optou-se por utilizar Python para o desenvolvimento do projeto devido a familiaridade do autor com a linguagem, sua facilidade de aprendizado e a existência de módulos como PyQt6, opencv-python e o pytesseract que facilitaram o desenvolvimento do*software*.

5.2. QT

Qt é um framework de desenvolvimento de interfaces gráficas *desktop*, embarcada e *mobile*. As plataformas suportadas incluem *Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS* entre outras. Desenvolvido pela empresa norueguesa Trolltech é amplamente utilizado em diversas aplicações pelo mundo por possuir uma vasta quantidade de recursos que facilitam o desenvolvimento de interfaces gráficas e por ser multiplataforma, facilitando a portabilidade de uma aplicação para uma outra plataforma[Qt 2022]. Atualmente está em sua versão QT6, porém foi utilizada a versão QT5 para o desenvolvimento deste trabalho através do módulo *Python PyQt5*[3].

5.3. OpenCV

OpenCV é uma biblioteca *open source* de visão computacional e *Machine Learning* Foi criada para fornecer uma infraestrutura de aplicações voltadas a visão computacioal com o intuíto de facilitar o desenvolvimento de produtos que necessitam utilizar essa tecnologia.

[3]https://pypi.org/project/PyQt6/

Atualmente a biblioteca conta com mais de dois mil e quinhentos algoritmos otimizados para resolver os mais diversos problemas de visão computacional e *Machine Learning* como detecção e reconhecimento de rostos humanos, identificação de objetos, rastreamento de objetos em movimento, criação de modelos 3D de objetos, procura de objetos semelhantes em uma base de dados entre outros [OpenCV 2022]. Neste trabalho foram utilizadas algumas funções da biblioteca para detecção de objetos semelhantes em uma base de dados através do módulo *Python opencv-python*[4].

5.4. Tesseract

Tesseract é uma *engine open source* de OCR que pode ser utilizada via linha de comando ou através de uma API (*Application Programming Interface*) [Tesseract-OCR 2016]. Neste trabalho é utilizado o Tesseract por meio do módulo *Python pytesseract*[5].

5.5. Modelo de desenvolvimento (Iterativo)

De acordo com [Santos 2019], "O desenvolvimento iterativo é exatamente a definição do dicionário, ou seja, desenvolvimento repetitivo. É quando repetimos as mesmas etapas do desenvolvimento de um*software*, até obter o resultado final desejado. No processo iterativo, vamos construindo o*software*, avaliando o mesmo e fazendo ajustes a fim de melhorá-lo. Nunca esperamos que o resultado saia perfeito na primeira fez".

6. Desenvolvimento

6.1. Arquitetura proposta

A arquitetura proposta (Figura 4) ficou semelhante ao trabalho correlato apresentado na Seção 2.1. A principal diferença entre os trabalhos é que no lugar da placa de captura de vídeo foi utilizada uma câmera para capturar as imagens da tela do equipamento que vai ser testado, podendo esta ser uma *webcam* convencional USB ou até mesmo a câmera de um celular que vai ser acessada via IP.

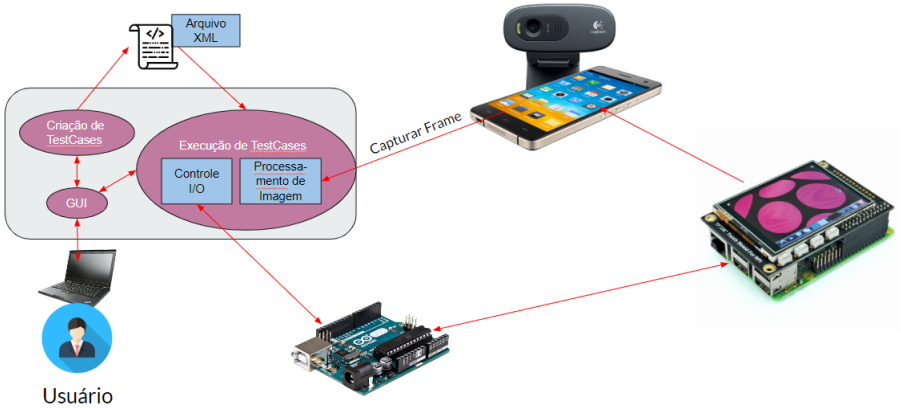

Figura 4. Arquitetura proposta para o sistema de teste. Fonte:Própria

O usuário através da GUI (*Graphics User Interface*) vai criar casos de testes e poder salvá-los em arquivo .xml, com o arquivo .xml. Concluído o usuário vai poder

[4]https://pypi.org/project/opencv-python/
[5]https://pypi.org/project/pytesseract/

executar o *script* que interpreta os *testecases*, o *script* vai executar os *testcases* sequencialmente de acordo com o arquivo .xml executando capturas de imagem da câmera/celular ou interagindo com o Arduino, conforme a descrição do *step* de teste que vai sendo executado em tempo real o script vai exibindo um report passo a passo no console. Após a execução um relatório é gerado em um arquivo texto juntamente com as imagens capturadas durante a execução dos testes.

6.2. Implementação da solução

6.2.1. Desenvolvimento da interfáce gráfica

Inicialmente foi feito um esboço em papel de como seriam as telas do sistema (Figura 5), a ideia era ter uma tela principal onde o usuário escolheria se gostaria de criar um *testcase* ou executar um *testcase*, e cada uma das opções levaria para uma tela diferente.

Porém foi necessário realizar algumas alterações devido ao tempo para finalizar o trabalho. Como por exemplo, não foi possível fazer a tela de execução de testes e por este motivo a tela principal também não foi desenvolvida para o protótipo apresentado. Em seguida foi feito um *mockup* (Figura 6) utilizando a ferramenta do próprio Qt chamada *Qt Designer* onde foi definido os tamanhos e distanciamentos dos componentes das telas. Este *mockup* foi salvo como "UiCriar.ui". Após finalizado o desenvolvimento do *mockup* foi utilizado o comando "pyuic5 UiCriar.ui -o UiMain.py"que gerou o *script python* "UiMain.py"que já funcionava como uma casca de interface mostrando toda a tela executável feita conforme o arquivo .ui porém sem nenhum evento nos botões e nos componente, que foram todos desenvolvidos posteriormente.

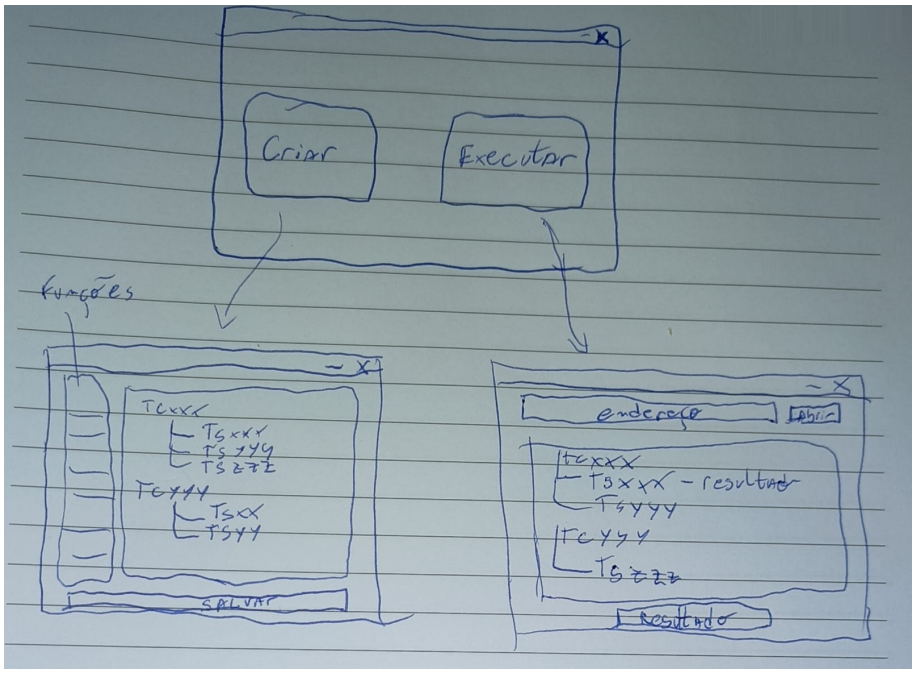

Figura 5. Esboço inicial desenvolvido feito em papel das telas do sistema. Fonte:Própria

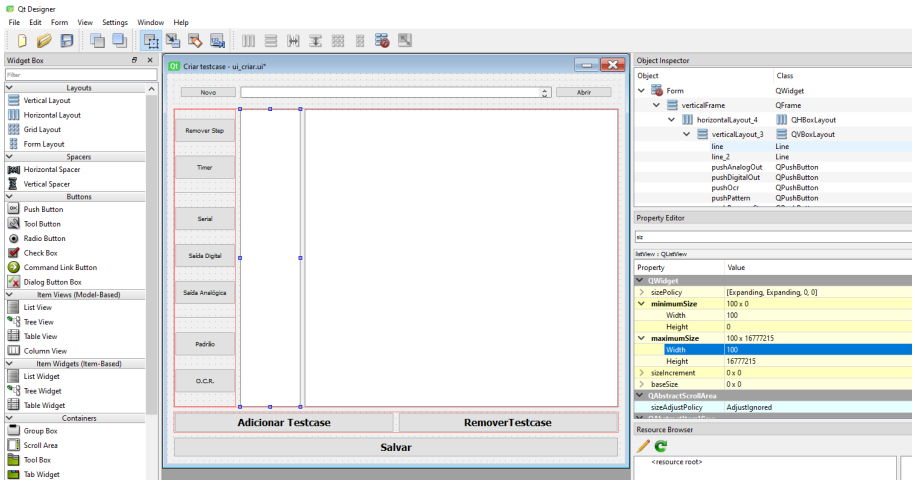

Figura 6. Desenvolvimento do mockup da tela de criação de *tescases* no programa QtDesigner. Fonte:Própria

6.2.2. *Script* de criação de *testcases*

Na Figura 6 é possível verificar como ficou o *layout* final da seção de criação de *testcases*. Nela é possível abrir um arquivo .xml ou criar um arquivo novo de *testcase*. Uma vez criado o arquivo é possível adicionar ou remover um *testcase* nos seus respectivos botões inferiores. Cada *testcase* vai ficar listado na lista do lado esquerdo, ao selecionar um *testcase* os *teststeps* do *testcase* selecionado será exibido. Após carregado um *testcase* também vai ser possível adicionar ou remover um *teststep*. As funcões disponíveis para um *teststep* são:

- *Timer*: valor em milissegundos, este valor vai ser o tempo que o *script* vai ficar "congelado"até o próximo *teststep*;
- *Serial*: *string* com o dado que será enviado para o Arduino realizar em sua comunicação serial;
- Saída Digital: na primeira caixa de mensagem vamos inserir o valor correspondente à porta digital que pretendemos que o Arduino manipule, em seguida é necessário inserir também o valor lógico para a porta em questão (0-1);
- Padrão: será feita a captura de uma imagem, devemos recortar o padrão que gostaríamos de procurar no teste e também marcar o ROI (*Regio of Interest*) que é um conjunto de coordenadas informando onde é esperado que o padrão esteja localizado, dessa maneira caso a imagem esteja deslocada do ROI durante o teste o resultado vai ser de falha;
- *O.C.R*: será capturada uma imagem porém é necessário selecionar o ROI para definir que que área da imagem o texto deve ser exibido.

6.2.3. *Script* de execução de *testcases*

Para a execução dos *testcases*, existe um *script python* para interpretar o arquivo .xml, executar as funções de interface com o Arduino e também fazer a chamada das funções de processamento de imagem de acordo com o que o *step* do *testcase* requisitar, esse *scritp* é o "executar_teste.py"(Anexo A).

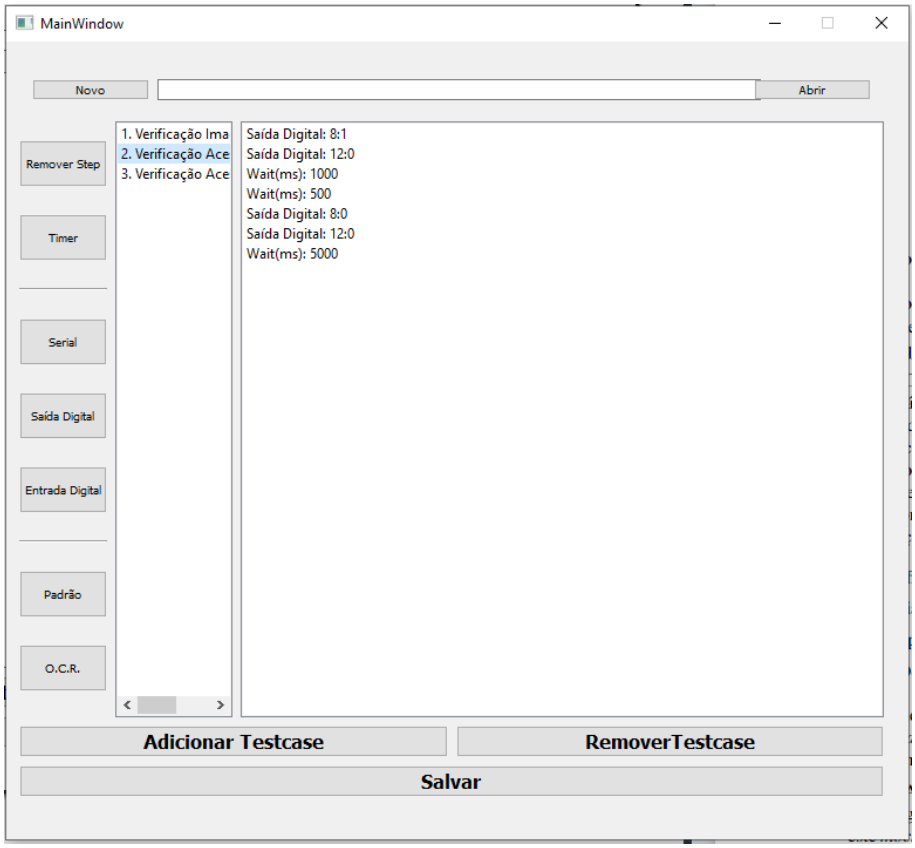

Figura 7. Layout final "ui_main.py". Fonte:Própria

Este *script* é responsável por manipular o XML cujo o endereço precisa ser passado pelas variáveis "fpath"e "fname". O *script* vai inicializar a comunicação com o Arduino utilizando a biblioteca pyFirmata, configurar os pinos que vão ser utilizados neste *testcase* em saídas digitais Depois ele abrirá o arquivo XML e possui dois laços de repetição "for": o primeiro laço para cada *testcase* presente no arquivo e outro laço para cada *teststep* presente em cada *testcase*.

De acordo com a chamada daquele *teststep* em questão uma ação diferente é executada. Se for uma ação do tipo *TIMER* o *script* vai entrar em *sleep* pelo tempo correspondente ao valor da chamada, se for do tipo *DOUT* o script vai escrever na biblioteca pyFirmata o valor e a porta que devem ser modificados, se for do tipo *PATTERN* o *script* vai chamar uma função de outro *script* responsável por fazer a detecção de padrões, inicialmente chamando a função "carregar"de "ImageProcessing.pattern"para fazer o carregamento da imagem que deve ser procurada na captura que vai ser feita através do comando "template = capturar_frame.carregar(str(templates_path + ts.attrib.get('template')))". Após isso, é feito o comando de capturar uma imagem da câmera através do comando "imagem = capturar_frame.capturar()"e por último o *script* chama a função que realiza o teste de detecção de padrões através do comando "resultado = pattern.testar(roi, template, imagem, report_path+'TC'+str(contador_tc)+'STEP'+str(contador_ts))". Caso a chamada seja do tipo OCR é enviado um comando para capturar a imagem "imagem = capturar_frame.capturar()"e em seguida é enviado um comando para executar o texto de leitura de caracteres através do co-

mando "resultado = tesseract_temp.testar(roi, str(ts.attrib.get('texto')), imagem,report_path+'TC'+str(contador_tc)+'STEP'+str(contador_ts))".

Todas as funções tem um retorno booleano caso não seja detectado o que se esperava ou não consiga executar por algum outro motivo. Quando isso ocorre o status do teste é colocado como *FAIL*. Para o caso de não ocorrer nenhum problema na execução e o resultado obtido for igual o esperado anteriomente, o resultado do *step* de teste vai ser de *PASS*. Conforme os testes são executados estes *status* e descrições dos testes em tempo real vão sendo exibidos no console. Após finalizar todos os *testcases*, todos estes resultados estão armazenados em um arquivo texto chamado de Report.txt.

6.2.4. *Scripts* de processamento de imagem

Foram criados três *scripts* para realizar o processamento das imagens de teste.

O primeiro *script* é o "capturar_frame.py"(Anexo B) que possui três funções:

- "capturar_ip"que faz a captura de imagem atrabés de uma requisição Http, para funcionar é preciso instalar no celular alguma aplicativo de câmera IP. Para os testes eu utilizei o "*Ip Webcam*"disponível para **Android** disponível para *download* em [6]. Após deixar ele em execução no aparelho celular que vai fazer a captura das imagens é só inserir o endereço de IP do aparelho celular com um final "/shot.jpg"que a imagem vai ser capurada e armazenada em uma variável temporária. Essa imagem será passada como retorno da função;
- "capturar_usb"que é responsável por fazer a captura da imagem através de uma câmera convencional conectada na porta USB do computador. É necessário passar qual a câmera que deseja capturar (caso conecte mais de uma câmera) e ela também vai passar como retorno a imagem capturada;
- função "carregar"que carrega uma imagem através de um endereço no computador, quando ela é chamada se faz necessário passar o endereço da imagem que se deseja abrir e a imagem vai ser passada como retorno da função para quem a chamou.

O segundo *scritp* é o pattern.py (Anexo C), ele é responsável por procurar a ocorrência de uma forma geométrica baseado em um *template* passado na função com o objetivo de "testar"onde é preciso passar o ROI onde e qual *template* deve ser procurado, a imagem onde deve ser procurado o *template* e o endereço do arquivo de relatório.

Inicialmente é obtida da variável de ROI as definições de posição x,y superior esquerdo e x,y inferior direito, então a imagem onde deve ser procurado o *template* é convertida em escalas de cinza através do "imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)". Após converter a imagem em escala de cinza é aplicada a função "cv2.matchTemplate(imagem_cinza, template, cv2.TM_CCOEFF_NORMED)"essa função é responsável por procurar a ocorrência de uma imagem dentro de outra imagem, utilizando o método de comparação "cv2.TM_CCOEFF_NORMED". Outros métodos de comparação podem ser consultados em [OpenCV.org 2022]. Após a comparação foi definido uma nota de corte (*threshold*)

[6] https://play.google.com/store/apps/details?id=com.pas.webcam&hl=pt_BR&gl=US

no valor de 0.8, para chegar neste valor foi necessário realizar a verificação dos dois *templates* e testar valores onde foi possível ter o resultado de encontrado quando o template estava presente na tela do DUT e também garantir que o resultado da detecção se mantivesse em falso enquanto o template não estava presente na tela do DUT garantindo um bom índice de falso negativo e nennhuma ocorrência de falso positivo nos testes. Após a comparação é feita a verificação se existem pontos onde os templates foram encontrados. Caso existam, o *script* vai traçar um retângulo vermelho em volta de onde foi localizado o *template* e retornar o valor lógico verdadeiro para o resultado. Caso não encontre, o retângulo vermelho não vai ser desenhado e o *script* vai retornar um valor lógico falso no resultado, para ambos os casos o *script* vai desenhar um retângulo azul em volta do ROI definido no *testcase* e vai salvar essa imagem com os retângulos desenhados para efeito de relatório.

Por útimo, o terceiro *scritp* "tesseract.py"(Anexo D) é responsável por fazer a detecção de textos na imagem, antes de executar os testes foi necessário instalar o *Tesseract* no computador onde os *scripts* vão ser executados e passar no *script* o endereço de seu executável (linha 15 do *script*) "ocr.pytesseract.tesseract_cmd = C:\\Program Files\\Tesseract-OCR\\tesseract.exe". Neste momento o *Pytesseract* vai se configurar para o endereço que foi fornecido, também é necessário após a instalação do *software* fazer o *download* do pacote de linguas em português para que a ferramenta consiga detectar as palavras na língua portuguesa. Feito tudo isso o *script* vai verificar as coordenadas do ROI e fazer um recorte (*crop*) do pedaço referente às coordenadas do ROI na imagem capturada. Essa nova imagem recortada é armazenada na variável "cropped". A partir deste momento todas as manipulações vão ser realizadas nessa imagem recortada. Primeiro o *script* faz uma primeira tentativa de encontrar o texto procurado na imagem recortada na linha 23 através da função "ocr.image_to_string(cropped, lang='por')". Caso encontre nessa primeira verificação o texto a função já vai retornar um valor lógico verdadeiro para o resultado do teste traçar um retângulo azul em volta do ROI na imagem original capturada e salvar a imagem para o relatório.

Contudo, na maioria dos casos o texto não era reconhecido nos primeiros testes executados duranrte o desenvolvimento do *script*, por este motivo durante o desenvolvimento foi necessário criar alguns tratamentos na imagem antes de verificar o texto. Diversos fatores como cor ou tamanho do texto podem influenciar na detecção. Para resolver o problema, caso na primeira verificação não seja detectado o texto que se espera, a lógica criada consiste em fazer um laço de repetição para testar todas as combinações de remoção de dois canais de cores ao mesmo tempo da escala RGB e logo em seguida binarizar a imagem utilizando o valor de 127 como limiar entre branco e preto (tudo acima de 127 é considerado como branco e tudo abaixo de 127 é considerado como preto). Isso é feito para melhorar o processamento pois o *scritp* passa a testar cada ponto de forma booleana. Depois da imagem binarizada é feito uma nova busca pelo texto, caso encontre o texto o resultado que vai ser passado como retorno fica com o valor lógico verdadeiro, ao testar todas as combinações RGB o *script* finaliza o laço de repetição, traça o retângulo azul em volta do ROI, salva a imagem e retorna o valor do resultado para o *script* anterior.

6.3. Resultados

Para validar o sistema proposto seria necessário ter um DUT para a ser testado. Para resolver este problema eu foi desenvolvido um*software* em Python utilizando o QT que

simula um sistema de controle de acesso com tela com três telas de navegação, uma principal que fica visível quando nenhum dos dois *inputs* for verdadeiro, uma de acesso negado que é exibida quando o *input* de acesso negado for verdadeiro e uma de acesso concedido que fica visível quando o *input* de acesso concedido for verdadeiro. As três telas podem ser verificadas na Figura 8. Este *software* foi executado em um *notebook* que estava ligado à um teclado externo e os *inputs* se deram através das teclas "A"e "B"do teclado, sendo "A"para o sinal de acesso negado e "B"para o sinal de acesso concedido. O Arduino ligado ao sistema de testes teve duas saídas digitais ligadas às teclas do teclado através de um circuito de relê com optoacoplador para fechar o contato, simulando assim o pressionar das teclas do teclado conforme requisitado pelo *testcase*. Também foi definido para o processo de validação do sistema dois ciclos de teste, um primeiro com um *software* que estaria correto e então deveria passar por todos os testes do *testcase* e um segundo ciclo com o *software* modificado e com todas as telas com algum erro de imagem, devendo assim reprovar os testes durante a execução do *testcase*. Para preparar os testcases foi necessário forçar os *inputs* para fazer a capturar das imagens, recorte dos *templates* e definição dos ROIs dos testes.

Figura 8. Telas do DUT que serão testadas pelo *software*. Fonte:Própria

6.4. Resultados do primeiro ciclo de teste (com o *software* sem erros)

O primeiro ciclo foi executado com o *software* com todos os componentes gráficos corretos, conforme é possível observar na Figura 9 todos os testes obtiveram resultado positivo, não havendo nenhum *testcase* com *status* de "*FAIL*.

6.4.1. *Testcase* 1 - Verificação Imagem em Iddle

O *testcase* 1 vai verificar a tela principal, que contém três linhas de texto, então os *teststeps* vão colocar as duas saídas digitais em nível lógico zero, aguarda 10 segundos e faz 3 testes seguidos de OCR o primeiro procurando pelo texto "CONTROLE", o segundo procurando pelo texto "DE"e o terceiro por "ACESSO". Todos os testes foram aprovados conforme o report da Figura 9, e as imagens obtidas nos testes podem ser verificadas na Figura 10.

6.4.2. *Testcase* 2 - Verificação Acesso Concedido

O *testcase* 2 vai verificar a tela de acesso concedido, que contém uma figura de um cadeado aberto e uma linha de texto abaixo da figura. Então os *teststeps* vão colocar a saída digital de acesso negado em nível lógico zero, a saída digital de acesso concedido em nível lógico um, aguardar 10 segundos e fazer um teste de *pattern* para procurar o template "cadeado_aberto.jpg"que foi capturado anteriormente no processo de criação dos *testcases*, em seguida realiza um teste de OCR procurando pelo texto "ACESSO CONCEDIDO".

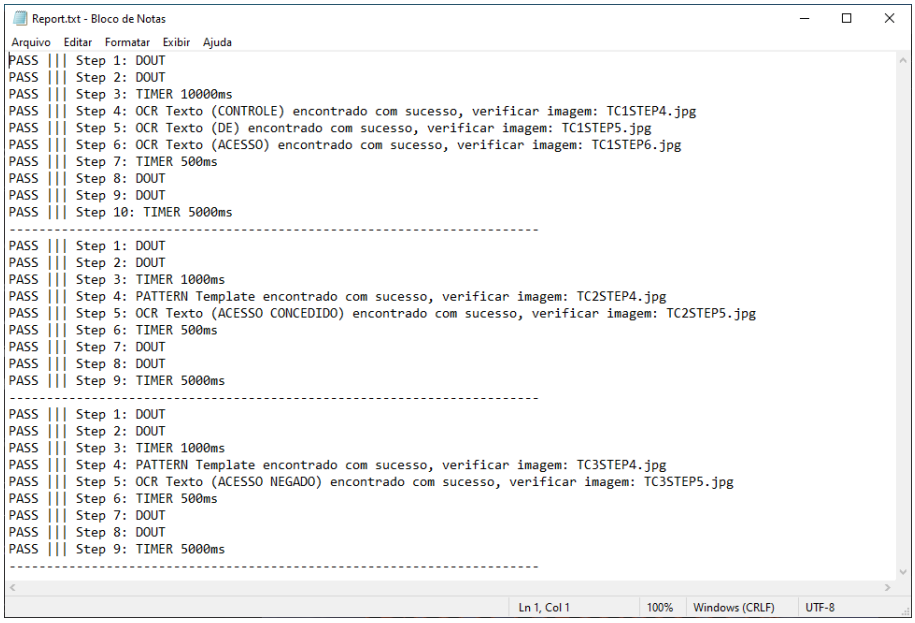

Figura 9. Report da execução dos testes no*software* sem erros de imagem. Fonte:Própria

Figura 10. Imagens capturadas durante o *testcase* 1 do primeiro ciclo de testes. Fonte:Própria

Todos os testes foram aprovados conforme o report da Figura 9, e as imagens obtidas nos testes podem ser verificadas na Figura 11.

6.4.3. *Testcase* 3 - Verificação Acesso Negado

O *testcase* 3 vai verificar a tela de acesso negado, que contém uma figura de um cadeado fechado e uma linha de texto abaixo da figura, então os *teststeps* vão colocar a saída digital de acesso negado em nível lógico um, a saída digital de acesso concedido em nível lógico zero, aguarda 10 segundos e faz um teste de *pattern* para procurar o template "cadeado_fechado.jpg"que foi capturado anteriormente no processo de criação dos *testcases*, em seguida realiza um teste de OCR procurando pelo texto "ACESSO NEGADO". Todos os testes foram aprovados conforme o report da Figura 9, e as imagens obtidas nos testes podem ser verificadas na Figura 12.

6.5. Resultados do segundo ciclo de teste (com o *software* com erros)

O segundo ciclo foi executado com o *software* com problemas nos componentes gráficos de cada uma das telas, conforme é possível observar na Figura 13 em cada um dos *testcases* houve um erro, então todos eles obtiveram resultado negativo em um de seus *teststeps*, fazendo com que todos os *testcase* fiquem com status de "*FAIL*".

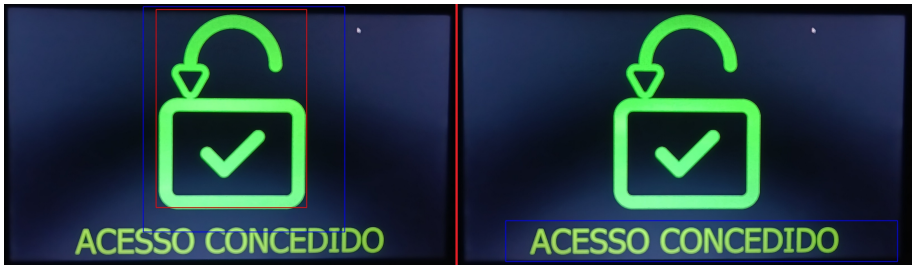

Figura 11. Imagens capturadas durante o *testcase* 2 do primeiro ciclo de testes. Fonte:Própria

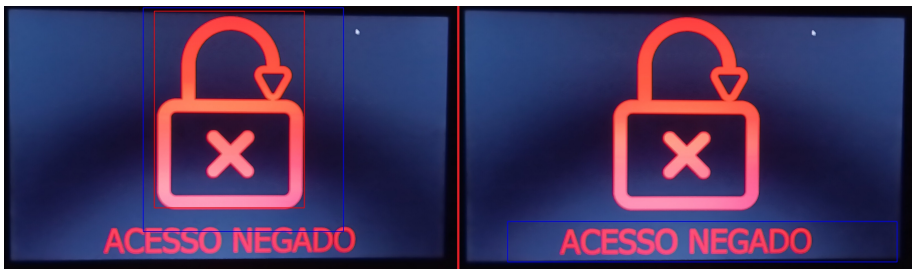

Figura 12. Imagens capturadas durante o *testcase* 3 do primeiro ciclo de testes. Fonte:Própria

6.5.1. *Testcase* 1 - Verificação Imagem em Iddle

Os *steps* de teste foram iguais o do primeiro ciclo de *software* porém neste *software* do DUT está com a palavra "ACESSO"alinhada à esquerda, e não centralizada como deveria ser. Então o *teststep* 6 apresentou o resultado de "FAIL"pois não encontrou a palavra "ACESSO"dentro do ROI definido no *testcase*. As imagens obtidas no report podem ser verificadas na Figura 14.

6.5.2. *Testcase* 2 - Verificação Acesso Concedido

Os *steps* de teste foram iguais o do primeiro ciclo de *software* porém neste *software* do DUT está com a imagem do cadeado fechado no lugar da imagem com o cadeado aberto como deveria. Então o *teststep* 4 apresentou o resultado de "FAIL"pois não encontrou o template "cadeado_aberto.jpg"dentro do ROI definido no *testcase*. As imagens obtidas no report podem ser verificadas na Figura 15.

6.5.3. *Testcase* 3 - Verificação Acesso Negado

Os *steps* de teste foram iguais o do primeiro ciclo de *software* porém neste *software* do DUT está com a imagem do cadeado aberto no lugar da imagem com o cadeado fechado como deveria. Então o *teststep* 4 apresentou o resultado de "FAIL"pois não encontrou o template "cadeado_fechado.jpg"dentro do ROI definido no *testcase*. As imagens obtidas no report podem ser verificadas na Figura 16.

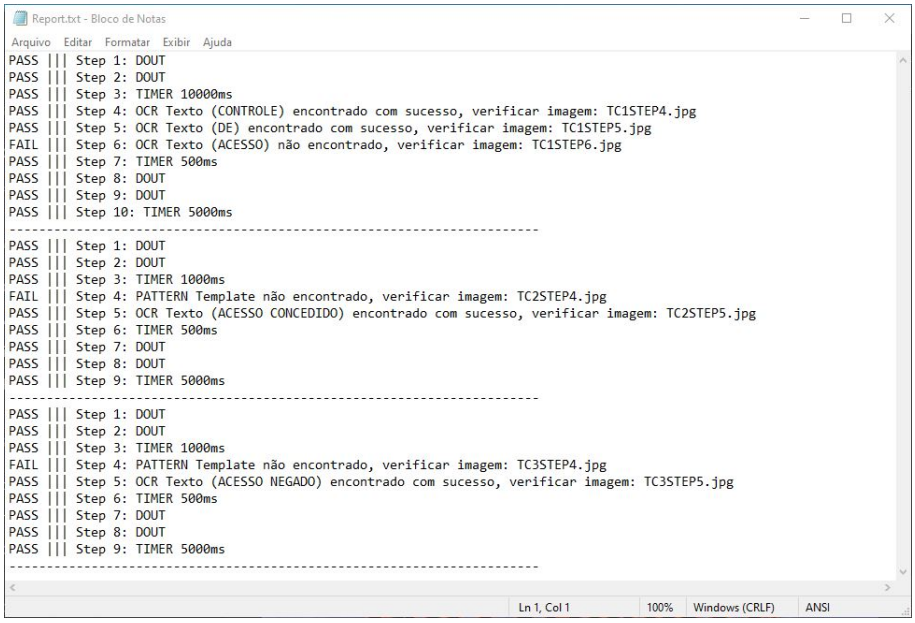

Figura 13. Report da execução dos testes no*software* sem erros de imagem. Fonte:Própria

Figura 14. Imagens capturadas durante o *testcase* 1 do segundo ciclo de testes. Fonte:Própria

7. Conclusão

O objetivo deste trabalho era desenvolver um sistema de testes semelhantes aos *Hardwares in the Loop* que temos disponíveis no mercado utilizando *hardware* de baixo custo e bibliotecas e *softwares open source*. Este objetivo foi atingido e com base nos testes executados foi possível concluir que o sistema consegue encontrar os textos e os *templates* corretamente, garantindo uma certa repetibilidade. Essa repetibilidade foi verificada durante a execução dos testes eu executei o *testcase*, que foram executados por vinte vezes e que atingiram uma taxa de 10% de resultados com falso negativo (testes onde a imagem estava correta, porém algum componente da tela não foi detectado pelo teste) e nenhum resultado falso positivo (testes onde o sistema detecataria um componente como presente na hora da execução porém sem ele estar presente na tela). Principalmente a ausência de resultados falso positivo foram surpreendentes e passam confiança no sistema, os casos de falso negativo existem pontos de melhoria no sistema que estão propostos na seção 7.1 e podem melhorar estas características.

O conhecimento obtido na disciplina de Metodologia de Pesquisa Científica e Tecnológica e também na disciplina de Interação Humano-Computador possibilitou uma pesquisa mais eficaz para iniciar o desenvolvimento do trabalho, as disciplinas de Engenharia de*software*, Arquitetura de*software*, Qualidade de*software*, Projeto de Sistemas I/II ga-

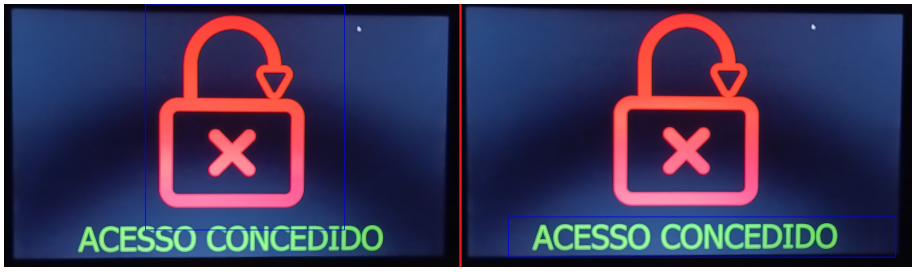

Figura 15. Imagens capturadas durante o *testcase* 2 do segundo ciclo de testes. Fonte:Própria

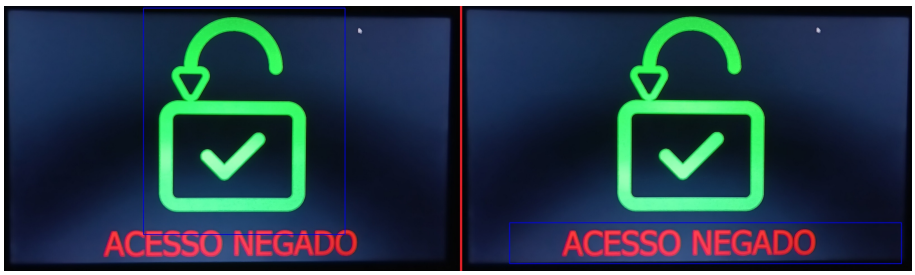

Figura 16. Imagens capturadas durante o *testcase* 3 do segundo ciclo de testes. Fonte:Própria

rantiu um melhor planejamento do sistema e também as disciplinas mais práticas como Linguagem de Programação I/II/III, Redes de Computadores e Serviços de Rede foram essenciais para auxiliar no desenvolvimento do protótipo proposto.

7.1. Trabalhos futuros

- Durante a execução do trabalho foi notado que ter uma espécie de bancada com pontos de fixação do DUT e da câmera ajudariam bastante, principalemnte nos testes de *template*. Como o teste utiliza uma imagem prévia de *template*, interessante que o ângulo da câmera em relação ao DUT e a distância entre os dois sejam as mesmas na hora de execução dos testes, isso faria com que a repetibilidade do sistema aumentasse.
- Desenvolver um local fechado para evitar a entrada de luz externa durante a execução dos testes e também criar um sistema de iluminação interno controlável para garantir sempre que as condições de luz sejam as mais próximas possivel em cada execução dos *testcases*.
- Conforme é possível ver na Figura 6, a ideia inicial era que o sistema possuísse três telas com dois ambientes separados de interface com o usuário, tanto na criação de *testcases*, quanto na sua execução. Desenvolver e melhorar a interface com o usuário seria um trabalho interessante à ser pensado para o futuro.
- Pesquisar e desenvolver novas *features* de teste, como teste de cores e também criar uma espécie de "Modo Manual"onde existiria um painel que permita acionar as saídas do Arduino para facilitar na hora de acionar manualmente o DUT para capturar as imagens de template e definir os ROIs.
- No quarto tópico do trabalho correlato descrito na seção 2.1, que discorre sobre resultados e análises, é relatado inicialmente que a precisão dos testes de interpretação de texto alcançaram 90,32%. Porém é relatado um problema com três

casos específicos onde foi detectada uma dificuldade de detecção do texto. Entre o caractere "I"(i maiúsculo) e o caractere "1"(número um) pode-se perceber que nem sempre o resultado interpretado com a função era o que realmente estava sendo exibido na tela, o mesmo problem foi encontrado entre o caractere "O"(o maiúsculo) e o "0"(número zero) e também entre o "U"(u maiúsculo) e "V"(v maiúsculo). Para resolver este problema os autores do artigo criaram uma função que gera todas as combinações possíveis para estes caracteres suspeitos, é possível verificar a representação dessa função na Figura 17. Um trabalho futuro válido seria verificar a existência deste problema neste trabalho aqui apresentado e também tratá-lo para que não apresente resultados errados na execução dos *testcases* poderia ser um trabalho futuro.

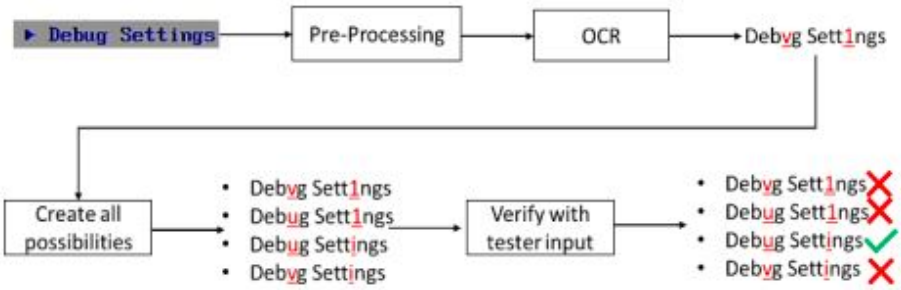

Figura 17. Exemplo resultados aleatórios para caracteres parecidos. Fonte:[Mohammed et al. 2021]

Referências

Arduino (2022). Especificações técnicas. https://store.arduino.cc/products/arduino-uno-rev3?_gl=1*1251qbr*_ga*MTkwMjkwNTU3NC4xNjY3NzY0ODU1*_ga_NEXN8H46L5*MTY2OTU4OTU3NC40LjEuMTY2OTU4OTU4MC4wLjAuMA.. [Online; Acessado em 27 de novembro de 2022].

Botelho, R. (2021). O que é hardware-in-the-loop (hil)? https://opencadd.com.br/o-que-e-hardware-in-the-loop-hil/. [Online; Acessado em 9 de junho de 2022].

Brasil, P. (2015). Aprenda mais. https://wiki.python.org.br/PythonBrasil. [Online; Acessado em 21 de junho de 2022].

Elgendy, M. (2019). The computer vision pipeline, part 2: input images. https://freecontent.manning.com/the-computer-vision-pipeline-part-2-input-images/. [Online; Acessado em 28 de novembro de 2022].

IBM (2022). What is computer vision? https://www.ibm.com/topics/computer-vision. [Online; Acessado em 28 de novembro de 2022].

Lima, T. (2014). Agc – o primeiro grande sistema embarcado. https://www.embarcados.com.br/agc-primeiro-grande-sistema-embarcado/. [Online; Acessado em 9 de junho de 2022].

Markets and Markets (2020). Embedded system market by hardware (mpu, mcu, application-specific integrated circuits, dsp, fpga, and memories), software (middleware, operating systems), system size, functionality, application, region - global forecast to 2025. https://www.marketsandmarkets.com/Market-Reports/embedded-system-market-98154672.html. [Online; Acessado em 9 de junho de 2022].

Marwedel, P. (2021). *Embedded System Design*. Springer, 4th edition.

Mohammed, E. A. A., Mustapa, M., Rahim, H., and Norizan, M. N. (2021). Advanced ui test automation (auta) for bios validation using opencv and ocr. *Indonesian Journal of Electrical Engineering and Computer Science*, 23(3):1350–1356. Disponível em: <http://ijeecs.iaescore.com/index.php/IJEECS/article/view/24438>. Acesso em: 9 de Junho de 2022.

Nanonets (2022). How to ocr with tesseract, opencv and python. https://nanonets.com/blog/ocr-with-tesseract/. [Online; Acessado em 28 de novembro de 2022].

Olivera, C. and Zanetti, H. (2015). *Arduino Descomplicado – Aprenda com projetos de eletrônica e programação*. Saraiva, 1st edition.

OpenCV (2022). About. https://opencv.org/about/. [Online; Acessado em 22 de junho de 2022].

OpenCV.org (2022). Template matching. https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html. [Online; Acessado em 28 de novembro de 2022].

Qt (2022). The future is written with qt. https://doc.qt.io/. [Online; Acessado em 22 de junho de 2022].

Santos, D. (2019). Desenvolvimento iterativo e incremental. https://codejourney.com.br/desenvolvimento-iterativo-e-incremental/. [Online; Acessado em 21 de junho de 2022].

Tesseract-OCR (2016). Tesseract user manual. https://tesseract-ocr.github.io/tessdoc/. [Online; Acessado em 22 de junho de 2022].

A. Anexo A - Script de execução de *testcases*

```
import xml.etree.cElementTree as ET
from enum import Enum
import time
import pyfirmata
from ImageProcessing import capturar_frame
from ImageProcessing import tesseract_temp
from ImageProcessing import pattern

try:
    placa = pyfirmata.Arduino("COM4")
except:
    print('Arduino n o conectado.')
    arduino_conectado = False
else:
    arduino_conectado = True
fpath = '../aTeste/'
templates_path = fpath+'templates/'
report_path = fpath+'report/'

fname = 'apresentacao.xml'
tree = ET.parse(fpath+fname)

class TiposTeste(Enum):
    TIMER = 'TIMER'
    SERIAL = 'SERIAL'
    DOUT = 'DOUT'
    DIN = 'DIN'
    PATTERN = 'PATTERN'
    OCR = 'OCR'

lista_portas = []
root = tree.getroot()
if arduino_conectado:
    if root is not None:
        for tc in root:
            for ts in tc:
                if str(ts.attrib.get('name')) == str(TiposTeste.
                   DOUT.value):
                    if str(ts.attrib.get('name'))+str(ts.attrib.
                       get('name')) not in lista_portas:
                        placa.digital[int(ts.attrib.get('porta')
                           )].mode = pyfirmata.OUTPUT
    it = pyfirmata.util.Iterator(placa)
    it.start()

if root is not None:
    arquivo_report = open(report_path+"Report.txt","w+")
    contador_tc = 0
```

```
46      for tc in root:
47          contador_tc += 1
48          print("Testcase " + str(contador_tc) + ': ' + tc.attrib.
               get('name'))
49          contador_ts = 0
50          for ts in tc:
51              time.sleep(0.02)
52              contador_ts += 1
53              complemento=''
54              resultado_teste = 'PASS ||| '
55              if str(ts.attrib.get('name')) == str(TiposTeste.
                   TIMER.value):
56                  #print(ts.attrib.get('name') + ' --- ' + ts.text
                       )
57                  complemento = ' ' + ts.text + 'ms'
58                  time.sleep(int(ts.text)/1000)
59              elif str(ts.attrib.get('name')) == str(TiposTeste.
                   DOUT.value):
60                  #print(ts.attrib.get('name') + ' --- ' +'Porta:
                       ' + ts.attrib.get('porta') + ' Valor: '+ ts.
                       text)
61                  if arduino_conectado:
62                      placa.digital[int(ts.attrib.get('porta'))].
                           write(int(ts.text))
63                      complemento = ' Porta: ' + ts.attrib.get('
                           porta') + ' Valor: ' + ts.text
64              elif str(ts.attrib.get('name')) == str(TiposTeste.
                   SERIAL.value):
65                  pass
66              elif str(ts.attrib.get('name')) == str(TiposTeste.
                   DIN.value):
67                  pass
68
69
70              elif str(ts.attrib.get('name')) == str(TiposTeste.
                   PATTERN.value):
71                  roi = [int(ts.attrib.get('left')), int(ts.attrib
                       .get('top')), int(ts.attrib.get('right')),
                       int(ts.attrib.get('botton'))]
72                  imagem = capturar_frame.capturar()
73                  template = capturar_frame.carregar(str(
                       templates_path + ts.attrib.get('template')))
74                  resultado = pattern.testar(roi, template, imagem
                       , report_path+'TC'+str(contador_tc)+'STEP'+
                       str(contador_ts))
75                  if resultado == False:
76                      resultado_teste = 'FAIL ||| '
77                      complemento = ' Template n o encontrado,
                           verificar imagem: ' + 'TC'+str(
```

```
                        contador_tc)+'STEP'+str(contador_ts)+'.
                        jpg'
78                  else:
79                      complemento = ' Template encontrado com
                        sucesso, verificar imagem: ' + 'TC'+str(
                        contador_tc)+'STEP'+str(contador_ts)+'.
                        jpg'
80
81              elif str(ts.attrib.get('name')) == str(TiposTeste.
                    OCR.value):
82                  roi = [int(ts.attrib.get('left')), int(ts.attrib
                        .get('top')), int(ts.attrib.get('right')),
                        int(ts.attrib.get('botton'))]
83                  imagem = capturar_frame.capturar()
84                  resultado = tesseract_temp.testar(roi, str(ts.
                        attrib.get('texto')), imagem,report_path+'TC'
                        +str(contador_tc)+'STEP'+str(contador_ts))
85                  if resultado == False:
86                      resultado_teste = 'FAIL ||| '
87                      complemento = ' Texto (' + str(ts.attrib.get
                        ('texto')) + ') n o encontrado,
                        verificar imagem: ' + 'TC' + str(
                        contador_tc) + 'STEP' + str(contador_ts)
                        + '.jpg'
88                  else:
89                      complemento = ' Texto (' + str(ts.attrib.get
                        ('texto')) + ') encontrado com sucesso,
                        verificar imagem: ' + 'TC' + str(
                        contador_tc) + 'STEP' + str(contador_ts)
                        + '.jpg'
90              #Exibe o resultado do teststep
91              resultado_final =resultado_teste + "Step " + str(
                    contador_ts) + ': ' + ts.attrib.get('name') +
                    complemento
92              print(resultado_final)
93              arquivo_report.write(resultado_final + "\n")
94          print('---------------------------------')
95          arquivo_report.write('---------------------------------
                ')
96
97      from shutil import make_archive
98      make_archive('Report', 'zip', fpath)
99  }
```

Algoritmo 1. Código fonte de executar_teste.py

B. Anexo B - Script de captura de imagens para teste

```
1  import requests
2  import cv2
```

```
import numpy as np
#import imutils

def capturar():
    # Replace the below URL with your own. Make sure to add "/
       shot.jpg" at last.
    url = "http://192.168.15.61:8080/shot.jpg"

    # While loop to continuously fetching data from the Url
    img_resp = requests.get(url)
    img_arr = np.array(bytearray(img_resp.content), dtype=np.
       uint8)
    img = cv2.imdecode(img_arr, -1)
    #img = imutils.resize(img, width=1000, height=1800)
    return img
def carregar(path):
    return cv2.imread(path, 0)
```

Algoritmo 2. Código fonte de capturar_frame.py

C. Anexo C - Script de verificação de *template* nas imagens capturadas

```
import numpy as np
import cv2
from itertools import combinations

from PIL import Image

def testar(roi, template, imagem, report_path):
    resultado = False
    #print(template_path)
    #template = cv2.imread(template_path, 0)
    w, h = template.shape[::-1]

    roi_upper_left_x = roi[0]
    roi_upper_left_y = roi[1]
    roi_lower_right_x = roi[2]
    roi_lower_right_y = roi[3]

    ### GRAVAR TEMPLATE ###
    #cropped_image = imagem[roi_upper_left_y:roi_lower_right_y,
       roi_upper_left_x:roi_lower_right_x]
    #cv2.imshow('aaa',cropped_image)
    #cv2.imwrite(report_path+'cadeado_aberto.jpg', cropped_image
       )
    ########################

    imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
    res = cv2.matchTemplate(imagem_cinza, template, cv2.
       TM_CCOEFF_NORMED)
```

```
    threshold = 0.8
    loc = np.where(res >= threshold)
    for pt in zip(*loc[::-1]):
        if (pt[0] >= roi_upper_left_x and pt[1] >=
           roi_upper_left_y and pt[0] + w <= roi_lower_right_x
           and pt[1] + h <= roi_lower_right_y):
            cv2.rectangle(imagem, pt, (pt[0] + w, pt[1] + h),
               (0, 0, 255), 2)
            resultado = True
            break
        else:
            resultado = False
    cv2.rectangle(imagem, (roi_upper_left_x, roi_upper_left_y),
       (roi_lower_right_x, roi_lower_right_y), (255, 0, 0),2)
    cv2.imwrite(report_path +'.jpg', imagem)

    return resultado
```

Algoritmo 3. Código fonte de pattern.py

D. Anexo D - Script de verificação de texto nas imagens capturadas

```
import pytesseract as ocr
import numpy as np
import cv2
#import capturar_frame
from itertools import combinations
from PIL import Image

def remove_quebra_linha(value):
    return ' '.join(value.splitlines())

def testar(roi, texto, img, report_path):
    resultado = False
    vetor_passar = roi
    combinacoes = combinations([0, 1, 2], 2)
    ocr.pytesseract.tesseract_cmd = 'C:\\Program Files\\
       Tesseract-OCR\\tesseract.exe'
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    imagem = Image.fromarray(img_rgb)


    left = vetor_passar[0]
    top = vetor_passar[1]
    right = vetor_passar[2]
    bottom = vetor_passar[3]

    cropped = imagem.crop((left, top, right, bottom))

    phrase = ocr.image_to_string(cropped, lang='por')
```

```
    #cropped.save('cadeado_aberto.jpg')
    #print('---- '+texto)
    #print(phrase)
    phrase = remove_quebra_linha(phrase)
    if phrase.strip() == texto:
        resultado = True

    if resultado == False:
        contador =1
        for i in combinacoes:
            contador+=1
            npimagem = np.asarray(cropped).astype(np.uint8)
            # diminui o dos ruidos antes da binariza o
            #npimagem[:, :, 0] = 0 # zerando o canal R (RED)
            npimagem[:, :, i[0]] = 0
            #cv2.imshow('kkk', npimagem)
            npimagem[:, :, i[1]] = 0
            #cv2.imshow('kkk2', npimagem)

            # atribui o em escala de cinza
            im = cv2.cvtColor(npimagem, cv2.COLOR_RGB2GRAY)

            # aplica o da truncagem bin ria para a
               intensidade
            # pixels de intensidade de cor abaixo de 127 ser o
               convertidos para 0 (PRETO)
            # pixels de intensidade de cor acima de 127 ser o
               convertidos para 255 (BRANCO)
            # A atrubi o do THRESH_OTSU incrementa uma
               an lise inteligente dos nivels de truncagem

            ret, thresh = cv2.threshold(im, 127, 255, cv2.
               THRESH_BINARY | cv2.THRESH_OTSU)

            # reconvertendo o retorno do threshold em um objeto
               do tipo PIL.Image
            binimagem = Image.fromarray(thresh)

            phrase = ocr.image_to_string(binimagem, lang='por')
            phrase = remove_quebra_linha(phrase)
            #binimagem.show('aaaa')
            #print(phrase)
            if phrase.strip() == texto:
                resultado = True
    cv2.rectangle(img, (left, top), (right, bottom), (255, 0, 0)
       , 2)
    cv2.imwrite(report_path + '.jpg', img)
    return resultado
```

Algoritmo 4. Código fonte de tesseract.py

Documento Digitalizado Público

Anexo I - Artigo final

Assunto: Anexo I - Artigo final
Assinado por: Daniela Marques
Tipo do Documento: Projeto
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Daniela Marques**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 20/12/2022 10:41:57.

Este documento foi armazenado no SUAP em 20/12/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1192714
Código de Autenticação: ce419c2d26