ANEXO II

Códigos

```
% CRIAÇÃO DO BANCO DE IMAGENS
% Esse script salva imagens que serão utilizadas posteriormente para treinamento do
classificador de imagens
% Primeiramente, obtém-se o ROI (região de interesse) usando o ponto da mão fornecido
pelo Kinect.
% Em seguida, a imagem é tratada e salva no diretório.

% Número de imagens para cada gesto
qte_imagens = 200;

% Iniciando o Kinect
[himg, colorVid, depthVid] = iniciar_kinect;

% Contador de imagens
cont = 0;

% Loop 'while' para cada ciclo de leitura do Kinect
while(ishandle(himg))
    [depthMap,depthMetaData] = mostrar_video_profundidade(depthVid);

    % Verifica se é encontrado algum esqueleto
    if sum(depthMetaData.IsSkeletonTracked) > 0
        idx = find(depthMetaData.IsSkeletonTracked);
        % Podem ser encontrados até seis pessoas
        % idx é o índice referente a pessoa encontrada

            if(idx~=0)
             % Obtendo região de interesse a partir da imagem fornecida pelo kinect
             limites = gerar_roi(depthMetaData,idx);

             % Tratando a imagem
             imagem = tratar_imagem(depthMap,limites);

             % Salva a imagem tratada no diretório
             salvar_imagem(imagem,depthMetaData);

             % Interrompe o loop quando são salvas o número de imagens desejadas
             cont = cont + 1;
             if(cont>=qte_imagens)
                 break
             end
        end
    end
end
stop(depthVid);
```

Figura 1 - Gerando imagens para o banco de treinamento.

```
function [himg, colorVid, depthVid] = iniciar_kinect ()
% Função para iniciar o kinect. Retorna himg (auxiliar para tratamento dos dados) e os
streamins de cor e profundidade.

% Desconecta e reseta todos os dispositivos de aquisição de imagens
imaqreset

% Configurações para obtenção dos dados do Kinect
colorVid = videoinput('kinect',1);
depthVid = videoinput('kinect',2);
triggerconfig([colorVid depthVid],'manual');

depthVid.FramesPerTrigger = 1;
colorVid.FramesPerTrigger = 1;
depthVid.TriggerRepeat = inf;
colorVid.TriggerRepeat = inf;

set(getselectedsource(depthVid), 'TrackingMode', 'Skeleton');

start(depthVid);
start(colorVid);

% Retorna objeto de imagem
himg = figure;
end
```

Figura 2 - Função *iniciar_kinect*.

```
function [depthMap,depthMetaData] = mostrar_video_profundidade (depthVid)
% Função que mostra vídeo utilizando o stream de profundidade (depth)
% Devem ser passados como parâmetros [himg colorVid depthVid] que são
% obtidos da seguinte maneira:
% [himg colorVid depthVid] = iniciar_kinect;

trigger(depthVid);
[depthMap,~,depthMetaData] = getdata(depthVid);
imshow(depthMap, [0 4096]);
```

Figura 3 - Função *mostrar_video_profundidade.*

```
function [roi] = gerar_roi (depthMetaData,idx)
% Essa função retorna os limites da região de interesse (mão). Recebe como parâmetros os
metadados fornecidos pelo Kinect e o índice da pessoa localizada. É realizada uma
compensasão do tamanho em função da profundidade da mão (distância da mão até o sensor).

% variável 'mao_direita' recebe o pixel referente à posição da mão
mao_direita = depthMetaData.JointImageIndices(12,:,idx);

% coordenada global da mão
mao_direita_coord_glob = depthMetaData.JointWorldCoordinates(12,:,idx);
profundidade = norm(mao_direita_coord_glob);
raio = round(90-profundidade / 50);

% limites gerados para a região de interesse
roi = round([mao_direita(2)-1*raio; mao_direita(2)+0.25*raio; ...
    mao_direita(1)-0.25*raio; mao_direita(1)+0.75*raio]);
roi(roi<=0) = 1;
end
```

Figura 4 - Função *gerar_roi.*

```
function [imagem_preto_branco] = tratar_imagem(depthMap,limites)
% Esse script utiliza os dados das articulações fornecidos pelo Kinect para gerar a
imagem binarizada para classificação.

% Utilização da imagem de profundidade:
img_profundidade = double(depthMap(limites(1):limites(2),limites(3):limites(4)));

% Normalização da imagem:
min_matriz = min(min(img_profundidade));
img_profundidade = ((img_profundidade)-min_matriz);
max_matriz = max(max(img_profundidade));
img_profundidade =  img_profundidade / max_matriz;

% Criação da imagem em escala de cinza a partir da matriz
img = mat2gray(img_profundidade);

% Transformação em preto e branco utilizando o valor médio da imagem como limiar para
binarização
imagem_preto_branco = im2bw(img,mean(mean(img)));
imagem_preto_branco = imresize(imagem_preto_branco, [70 70]);
end
```

Figura 5 - Função *tratar_imagem.*

```
function salvar_imagem(imagem,depthMetaData)
% Essa função grava a imagem passada como parâmetro no diretório de
% execução. Os metadados são utilizados para nomear o arquivo (o nome da
% imagem salva é referente a data e a hora).

figure(2);
imshow(imagem);
abs_time = depthMetaData.AbsTime;
imwrite(imagem,['img_' num2str(round(abs_time(1))) num2str(round(abs_time(2)))
num2str(round(abs_time(3))) num2str(round(abs_time(4))) num2str(round(abs_time(5)))
num2str(round(100*abs_time(6))) '.png']);
delete(figure(2));
end
```

Figura 6 - Função *salvar_imagem.*

```
% GERAR_DADOS_MAO_2017
% Esse script gera os dados para o treinamento do modelo que reconhece o número de dedos
mostrados.
% Deve ser executado no MATLAB 2017 pois utiliza a função imageDatastore para facilitar
a criação dos dados.
% Criação do banco de imagens (imageDatastore):
rootfolder = 'C:\Users\MARQUES\Desktop\Engenharia\Iniciação
Científica\Códigos\Banco_imagens_mao';
banco_imagens =
imageDatastore(rootfolder,'IncludeSubfolders',1,'LabelSource','foldernames');
% Muda a ordem das imagens de forma aleatória e cria dois novos bancos de imagens (o
primeiro contendo 70% dos dados e o segundo, 30%):
banco_imagens = shuffle(banco_imagens);
[banco_treinamento,banco_teste] = splitEachLabel(banco_imagens,0.7);
% Mostra quatro imagens como exemplo do conteúdo do banco de imagens:
figure();
for i=1:4
    img = readimage(banco_treinamento,i);
    img = imresize(img, [70 70]);
    subplot(2,2,i);
    imshow(img);
    if(banco_treinamento.Labels(i)=='aberta')
        title('ABERTA');
    else
        title('FECHADA');
    end
end

% Alocação das matrizes de treinamento e do vetores de rótulos. Cada imagem é
transformada em uma linha da matriz e cada rótulo refere-se a classe a qual essa linha
pertence. Esses dados foram alocados inicialmente com o número 7 para facilitar a
verificação em caso de falhas na alocação.
matriz_treinamento_mao = 7*ones(length(banco_treinamento.Files),length(img(:)'));
rotulos_treinamento_mao = 7*ones(length(banco_treinamento.Files),1);
matriz_teste_mao = 7*ones(length(banco_teste.Files),length(img(:)'));
rotulos_teste_mao = 7*ones(length(banco_teste.Files),1);
```

```
% Preenchimento da matriz de treinamento e do vetor de rótulos de treinamento:
for i=1:length(banco_treinamento.Files)
    img = readimage(banco_treinamento,i);
    img = imresize(img, [70 70]);
    matriz_treinamento_mao(i,:) = img(:)';
    if(banco_treinamento.Labels(i)=='aberta')
        rotulos_treinamento_mao(i) = 1;
    else
        rotulos_treinamento_mao(i) = 0;
    end
end
% Preenchimento da matriz de teste e do vetor de rótulos de teste:
for i=1:length(banco_teste.Files)
    img = readimage(banco_teste,i);
    img = imresize(img, [70 70]);
    matriz_teste_mao(i,:) = img(:)';
    if(banco_teste.Labels(i)=='aberta')
        rotulos_teste_mao(i) = 1;
    else
        rotulos_teste_mao(i) = 0;
    end
end
% Salvando os dados gerados
save('dados_treinamento_mao','matriz_treinamento_mao','rotulos_treinamento_mao','matriz_
teste_mao','rotulos_teste_mao');
```

Figura 7 - Programa *Gerar_dados_mao.*

```
% GERAR DADOS DEDOS
% Esse script gera os dados para o treinamento do modelo que reconhece o número de dedos
mostrados.
% Deve ser executado no MATLAB 2017 pois utiliza a função imageDatastore para facilitar
a manipulação das imagens.

% Criação do banco de imagens (imageDatastore):
rootfolder = 'C:\Users\MARQUES\Desktop\Engenharia\Iniciação
Científica\Códigos\Banco_imagens_dedos';
banco_imagens = imageDatastore(rootfolder,...
    'IncludeSubfolders',1,'LabelSource','foldernames');

% Muda a ordem das imagens de forma aleatória e cria dois novos bancos de imagens (o
primeiro contendo 70% dos dados e o segundo, 30%):
banco_imagens = shuffle(banco_imagens);
[banco_treinamento,banco_teste] = splitEachLabel(banco_imagens,0.7);

% Mostra quatro imagens como exemplo do conteúdo do banco de imagens imds:
figure();
for i=1:4
    img = readimage(banco_treinamento,i);
    img = imresize(img, [70 70]);
    subplot(2,2,i);
    imshow(img);
    switch banco_treinamento.Labels(i)
        case 'um_dedo'
            title(1);
```

```
        case 'tres_dedos'
            title(3);
        case 'cinco_dedos'
            title(5);
    end
end

% Alocação das matrizes de treinamento e do vetores de rótulos. Cada imagem é
transformada em uma linha da matriz e cada rótulo refere-se a classe a qual essa linha
pertence. Esses dados foram alocados inicialmente com o número 7 para facilitar a
verificação em caso de falhas na alocação.
matriz_treinamento_dedos = 7*ones(length(banco_treinamento.Files),length(img(:)'));
rotulos_treinamento_dedos = 7*ones(length(banco_treinamento.Files),1);
matriz_teste_dedos = 7*ones(length(banco_teste.Files),length(img(:)'));
rotulos_teste_dedos = 7*ones(length(banco_teste.Files),1);

% Preenchimento da matriz de treinamento e do vetor de rótulos de treinamento:
for i=1:length(banco_treinamento.Files)
    img = readimage(banco_treinamento,i);
    img = imresize(img, [70 70]);
    % Cada imagem corresponderá a uma linha na matriz de treinamento
    matriz_treinamento_dedos(i,:) = img(:)';
    % O grupo a que esta imagem pertecente é atribuído no trecho abaixo
    switch banco_treinamento.Labels(i)
        case 'um_dedo'
            rotulos_treinamento_dedos(i) = 1;
        case 'tres_dedos'
            rotulos_treinamento_dedos(i) = 3;
        case 'cinco_dedos'
            rotulos_treinamento_dedos(i) = 5;
    end
end

% Preenchimento da matriz de teste e do vetor de rótulos de teste:
for i=1:length(banco_teste.Files)
    img = readimage(banco_teste,i);
    img = imresize(img, [70 70]);
    % Cada imagem corresponderá a uma linha na matriz de treinamento
    matriz_teste_dedos(i,:) = img(:)';
    % O grupo a que esta imagem pertecente é atribuído no trecho abaixo
    switch banco_teste.Labels(i)
        case 'um_dedo'
            rotulos_teste_dedos(i) = 1;
        case 'tres_dedos'
            rotulos_teste_dedos(i) = 3;
        case 'cinco_dedos'
            rotulos_teste_dedos(i) = 5;
    end
end

% Salvando os dados gerados
save('dados_treinamento_dedos','matriz_treinamento_dedos','rotulos_treinamento_dedos','m
atriz_teste_dedos','rotulos_teste_dedos');
```

Figura 8 - Programa *Gerar_dados_dedos.*

```
% PRINCIPAL_ABERTA_FECHADA - Classificando imagens diretamente do kinect
% Esse script:  carrega os dados de imagens salvos anteriormente; inicia a captura do
sensor Kinect; e classifica a mão entre aberta e fechada.

% Carregando dados para treinamento do modelo
load('dados_treinamento_mao.mat');

% Criação dos modelos para classificação das imagens:
% ver função "svm_multiclasse"
modelo = svm_multiclasse(matriz_treinamento_mao,rotulos_treinamento_mao);

% Calculando a precisão para o banco de treinamento e de teste:
[precisao_trein, matriz_confusao_trein] =
calcular_precisao_mao(modelo,matriz_treinamento_mao,rotulos_treinamento_mao);
[precisao_teste, matriz_confusao_test] =
calcular_precisao_mao(modelo,matriz_teste_mao,rotulos_teste_mao);

% Inicialização do sensor Kinect; ver função "iniciar_kinect"
 [himg, colorVid, depthVid] = iniciar_kinect;

% Loop 'while' para cada imagem obtida pelo Kinect
while(ishandle(himg))
    % Capturando imagens em cores e de profundidade do kinect
    trigger([depthVid colorVid]);
    [depthMap,~,depthMetaData] = getdata(depthVid);
    [colorMap,~,colorMetaData] = getdata(colorVid);
    imshow(colorMap, [0 4096]);

    % Verifica se é encontrado algum esqueleto
    if sum(depthMetaData.IsSkeletonTracked) > 0
        idx = find(depthMetaData.IsSkeletonTracked);
        % idx é o índice referente a pessoa encontrada
        if(idx~=0)
            % Obtendo região de interesse a partir da imagem fornecida pelo kinect
            limites = gerar_roi(depthMetaData,idx);

            % Tratando a imagem
            imagem = tratar_imagem(depthMap,depthMetaData,idx,limites);

            % Classificando a imagem obtida
            classe = classificar_imagem(imagem,modelo);
            if(classe == 1)
                clc;
                disp('FECHADA');
            else
                clc;
                disp('ABERTA');
            end
        end
    end
    break
end
stop(depthVid);
```

Figura 9 - Programa *Principal_aberta_fechada*.

```
% PRINCIPAL_CONTADOR_DEDOS - Classificando imagens diretamente do kinect
% Esse script carrega os dados de imagens salvos anteriormente; inicia a captura do
sensor Kinect; e classifica quantos dedos são mostrados.

% Carregando dados para treinamento do modelo
load('dados_treinamento_dedos.mat');

% Criação dos modelos para classificação das imagens;
% ver função "multisvm_adaptado"
modelo = svm_multiclasse(matriz_treinamento_dedos,rotulos_treinamento_dedos);
[precisao_treinamento, matriz_confusao_treinamento] =
calcular_precisao_dedos(modelo,matriz_treinamento_dedos,rotulos_treinamento_dedos);
[precisao_teste, matriz_confusao_teste] =
calcular_precisao_dedos(modelo,matriz_teste_dedos,rotulos_teste_dedos);

% Inicialização do sensor kinect; ver função "iniciar_kinect"
[himg, colorVid, depthVid] = iniciar_kinect;

% Loop 'while' para cada imagem obtida pelo Kinect
while(ishandle(himg))
    % Capturando imagens em cores e de profundidade do kinect
    trigger([depthVid colorVid]);
    [depthMap,~,depthMetaData] = getdata(depthVid);
    [colorMap,~,colorMetaData] = getdata(colorVid);
    imshow(colorMap, [0 4096]);

    % Verifica se é encontrado algum esqueleto
    if sum(depthMetaData.IsSkeletonTracked) > 0
        idx = find(depthMetaData.IsSkeletonTracked);
        % Podem ser encontrados até seis pessoas
        % idx é o índice referente a pessoa encontrada
        if(idx~=0)
            % Obtendo região de interesse a partir da imagem fornecida pelo Kinect
            limites = gerar_roi(depthMetaData,idx);

            % Tratando a imagem
            imagem = tratar_imagem(depthMap,limites);

            % Classificando tal imagem com base nos modelos criados
            classe = classificar_imagem(imagem,modelos);

            % Mostrando na tela o valor obtido
            switch classe
                case 1
                    clc; disp('1');
                case 2
                    clc; disp('3');
                case 3
                    clc; disp('5');
            end
        end
    end
end
stop(depthVid);
```

Figura 10 - Programa *Principal_contador_dedos*.

```
function [modelos] = svm_multiclasse(BancoTreinamento,RotulosTreinamento)
% Essa função cria um modelo para classificação dado um conjunto de
% treinamento e seus grupos correspondentes. Utiliza um SVM de classe única
% para classificação multiclasse.

% Conta a quantidade de classe:
u=unique(RotulosTreinamento);
numClasses=length(u);

% Cria um modelo para cada classe:
for k=1:numClasses
    G1vAll=(RotulosTreinamento==u(k));
    modelos(k) = svmtrain(BancoTreinamento,G1vAll);
end
```

Figura 11 - Função *svm_multiclasse.*

```
function [classe] = classificar_imagem(imagem, modelo)
% Esse script recebe a imagem já tratada e o modelo a ser utilizado na classificação. O
retorno é a classe prevista pelo modelo.

% Transformação da imagem em uma matriz de double's.
imagem = double(imagem);

% Classificação
for k=1:length(modelo)
    if(svmclassify(modelo(k),imagem(:)'))
        break;
    end
end
classe = k;
end
```

Figura 12 - Função *classificar_imagem.*

```
% PRINCIPAL_ANGULO_BRACO
%
% Calculando o ângulo formado entre o braço e o antebraço.
% Esse código obtém as posições das articulações da pessoa identificada e
% calcula o ângulo formado entre o braço e o antebraço.

% Inicialização do sensor kinect; ver função "iniciar_kinect"
[himg, colorVid, depthVid] = iniciar_kinect;

%buffer para suavizar as saídas de ângulo e porcentagem
buffer = 5;
vetor_angulo = 180*ones(1,buffer);

while(ishandle(himg))
    % Capturando imagens em cores e de profundidade do kinect
    trigger([depthVid colorVid]);
    [depthMap,~,depthMetaData] = getdata(depthVid);
    [colorMap,~,colorMetaData] = getdata(colorVid);
    imshow(colorMap, [0 4096]);

    % Verifica se é encontrado algum esqueleto
    if sum(depthMetaData.IsSkeletonTracked) > 0
        idx = find(depthMetaData.IsSkeletonTracked);
        % Podem ser encontrados até seis pessoas
        % idx é o índice referente a pessoa encontrada

        skeletonJoints = depthMetaData.JointImageIndices(:,:,idx);

        if(idx~=0)
            % Obtendo região de interesse a partir da imagem fornecida pelo Kinect
            [angulo_cotovelo, vetor_angulo] =
calcular_angulo(depthMetaData,idx,vetor_angulo);

            desenhar_braco(skeletonJoints);

            clc;
            disp(angulo_cotovelo);
        end
    end
end
stop(depthVid);
```

Figura 13 - Programa *Principal_angulo_braco.*

```
function [angulo_cotovelo, vetor_angulo] = calcular_angulo
(depthMetaData,idx,vetor_angulo)
% Função que retorna o ângulo formando pelo cotovelo esquerdo e vetor que contem a
% sequência dos últimos ângulos calculados.
% vetor_angulo é um vetor auxiliar para o cálculo que deve ser passado como parâmetro;
% possui o tamanho configurado na variável buffer no programa principal; sua função é
% suavizar a saída do ângulo calculado; o resultado final é a média dos elementos
% contido no vetor_angulo.

% Posições referentes ao ombro, cotovelo e pulso capturadas pelo Kinect.
pos_glob_ombro = depthMetaData.JointWorldCoordinates(5,:,idx);
pos_glob_cotovelo = depthMetaData.JointWorldCoordinates(6,:,idx);
pos_glob_pulso = depthMetaData.JointWorldCoordinates(7,:,idx);

% Estes vetores representam o braço e o antebraço:
vetor1 = pos_glob_ombro-pos_glob_cotovelo;
vetor2 = pos_glob_pulso-pos_glob_cotovelo;

% Cálculo do ângulo atual:
ang_cotovelo  = radtodeg(atan2(norm(cross(vetor1,vetor2)),dot(vetor1,vetor2)));

% Atualização do vetor: é retirado o primeiro elemento (o mais antigo) e
% adicionado o novo ângulo
vetor_angulo = [vetor_angulo(2:end) ang_cotovelo];

% O ângulo de saída será a média de todos os elementos do vetor, suavizando
% possíveis mudanças bruscas.
angulo_cotovelo = mean(vetor_angulo);
end
```

Figura 14 - Função *calcular_angulo.*

```
function desenhar_braco(skeletonJoints)
% Essa função desenha no vídeo as posições do braço e do antebraço

% Posições das articulações
posicao_ombro = skeletonJoints(5,:);
posicao_cotovelo = skeletonJoints(6,:);
posicao_pulso = skeletonJoints(7,:);

% Primeira linha
X1 = [posicao_ombro(1,1) posicao_cotovelo(1,1)];
Y1 = [posicao_ombro(1,2) posicao_cotovelo(1,2)];
line(X1,Y1, 'LineWidth', 5, 'LineStyle', '-', 'Marker', '+', 'Color', 'r');

% Segunda linha
X1 = [posicao_cotovelo(1,1) posicao_pulso(1,1)];
Y1 = [posicao_cotovelo(1,2) posicao_pulso(1,2)];
line(X1,Y1, 'LineWidth', 5, 'LineStyle', '-', 'Marker', '+', 'Color', 'g');
end
```

Figura 15 - Função *desenhar_braco.*

```
% Testes com PCA
% Calcula a precisão para o modelo gradualmente a medida em que são
% adicionadas dimensões

load('dados_treinamento_dedos.mat');

[coeff,matriz_treinamento_pca,~,~,~,~] = pca(matriz_treinamento_dedos);
figure;
scatter3(matriz_treinamento_pca(:,1),matriz_treinamento_pca(:,2),matriz_treinamento_pca(
:,3),[],rotulos_treinamento_dedos(:),'LineWidth',2)
axis equal; xlabel('1st Principal Component'); ylabel('2nd Principal Component');
zlabel('3rd Principal Component')
figure;
scatter(matriz_treinamento_pca(:,1),matriz_treinamento_pca(:,2),[],rotulos_treinamento_d
edos(:),'LineWidth',2)
axis equal; xlabel('1st Principal Component'); ylabel('2nd Principal Component')

matriz_treinamento_pca = matriz_treinamento_dedos*coeff(:,1:3);
matriz_teste_pca = matriz_teste_dedos*coeff(:,1:3);
figure;
scatter3(matriz_teste_pca(:,1),matriz_teste_pca(:,2),matriz_teste_pca(:,3),[],rotulos_te
ste_dedos(:),'LineWidth',2)
axis equal; xlabel('1st Principal Component'); ylabel('2nd Principal Component')
zlabel('3rd Principal Component')
figure;
scatter(matriz_teste_pca(:,1),matriz_teste_pca(:,2),[],rotulos_teste_dedos(:),'LineWidth
',2)
axis equal; xlabel('1st Principal Component'); ylabel('2nd Principal Component')

matriz_treinamento_pca = matriz_treinamento_mao;
matriz_teste_pca = matriz_teste_mao;

n = size(coeff,2);
for i = 1:n
    try
        matriz_treinamento_pca = matriz_treinamento_dedos*coeff(:,1:i);
        matriz_teste_pca = matriz_teste_dedos*coeff(:,1:i);
        modelo = svm_multiclasse(matriz_treinamento_pca,rotulos_treinamento_dedos);
        [precisao_treinamento(i), ~] = calcular_precisao_dedos(modelo);
        [precisao_teste(i), ~] = calcular_precisao_dedos(modelo);
    catch
        precisao_treinamento(i) = precisao_treinamento(i-1);
        precisao_teste(i) = precisao_teste(i-1);
        continue
    end
end

vetor = 1:n;
figure; p = plot(vetor,precisao_treinamento,vetor,precisao_teste);
axis([1 5 0 100]); legend('Treinamento','Teste'); xlabel('Dimensões');
ylabel('Precisão [%]');
set(p(1),'LineWidth',4,'Color','b'); set(p(2),'LineWidth',4,'Color','g');
grid on;
```

Figura 16 - Calculando a precisão dos modelos utilizando PCA.