Documentação ImageCV

Componente de Visão Computacional para o App Inventor

Orientadores:
Prof. Dr. Flávio Luiz Coutinho
Prof. Dr. Fábio Nakano

Camila Vieira Bezerril

1. Introdução

A extensão ImageCV está sendo desenvolvida com base na biblioteca de visão computacional BoofCV (BoofCV, 2020), escolhida por não utilizar bibliotecas “nativas” codificadas em linguagem diferente de Java, ser Open Source, estar em constante atualização, e ainda por ter suporte ao Android.

Embora tenha sido cogitado a utilização da biblioteca OpenCV (OpenCV, 2020), geralmente a primeira escolha na área de visão computacional e codificada predominantemente em linguagem C, poderia haver dificuldade de interface com o App Inventor, que é codificado em Java, bem como o ambiente de execução (Android), que utiliza código executável, gerado por compiladores Java. A biblioteca escolhida como alternativa, além de ser bastante similar ao OpenCV, apresenta alguns módulos que permitem a interação entre as duas bibliotecas, embora não sejam utilizados na extensão em questão.

As instruções desta documentação tiveram como base a documentação do próprio App Inventor e da biblioteca BoofCV, disponíveis nos seguintes links:

- How to add a Component (App Inventor): https://docs.google.com/document/u/1/d/1xk9dMfczvjbbwD-wMsr-ffqkTlE3ga0ocCE1KOb2wvw/pub#h.fbmmsngpnimu

- App Inventor Extensions: http://ai2.appinventor.mit.edu/reference/other/extensions.html

- BoofCV: https://boofcv.org/index.php?title=Main_Page

- BoofCV JavaDoc: http://boofcv.org/javadoc/

2. Primeiros Passos

2.1. Configurando o ambiente

Para o desenvolvimento da extensão, primeiro deve-se configurar o ambiente. Para isto, as seguintes ferramentas devem ser instaladas através dos links:

- **Java JDK**

  https://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html

- **Git**

  https://git-scm.com/

- **Apache Ant**

  https://ant.apache.org/bindownload.cgi

Todas as alterações feitas para o App Inventor, tais como extensões, serão feitas localmente por intermédio do Apache Ant: biblioteca Java e ferramenta de linha de comando que assiste na compilação dos arquivos.

Em seguida, as variáveis de ambiente devem ser acertadas nas propriedades do sistema no painel de controle do Windows:

Variável	Valor
ANT_HOME	F:\apache-ant-1.10.7 (Pasta do Apache)
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_111 (Pasta do Java)
ANT_OPTS	-Xmx256M
_JAVA_OPTIONS	-Xmx1024m

CLASSPATH	%ANT_HOME%\lib;%JAVA_HOME%\lib
Path	;%ANT_HOME%\bin;%JAVA_HOME%\bin

Por meio do Git, os arquivos do App Inventor serão clonados:

1. Selecione o local do arquivo
2. Clique com o botão direito e em seguida em Git bash here
3. Na linha de comando digite:

   git clone https://github.com/mit-cml/appinventor-sources.git

2.2. Localização da extensão

Abrindo a pasta App Inventor como um projeto em um Ambiente de Desenvolvimento Integrado (IDE), as extensões por padrão devem ser colocadas no caminho:

appinventor\components\src\com\google\appinventor\components\runtime

2.3. Classe da extensão

2.3.1. *Imports* básicos

Criada a classe da extensão no local adequado, os *imports* para as funcionalidades mais utilizadas do App Inventor são:

```
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.errors.IllegalArgumentError;
import com.google.appinventor.components.runtime.util.AnimationUtil;
import com.google.appinventor.components.runtime.util.ErrorMessages;
import com.google.appinventor.components.runtime.util.HoneycombUtil;
import com.google.appinventor.components.runtime.util.MediaUtil;
import com.google.appinventor.components.runtime.util.SdkLevel;
import com.google.appinventor.components.runtime.util.ViewUtil;

import com.google.appinventor.components.annotations.UsesLibraries;
import com.google.appinventor.components.annotations.UsesPermissions;

import com.google.appinventor.components.runtime.*;
```

Note que há inclusão do *import* UsesLibraries, necessário para o funcionamento correto da extensão, já que serve para posteriormente informar as bibliotecas essenciais que serão utilizadas pela extensão.

2.3.2. Detalhes da extensão

Os detalhes da extensão estão disponíveis na seção:

```
@DesignerComponent(version = 1,
        category = ComponentCategory.EXTENSION,
        description = "Componente para Visao Computacional no App Inventor",
        nonVisible = true,
        iconName = "images/extension.png")
@SimpleObject(external = true)
```

A categoria da classe deve ser, por padrão, EXTENSION.

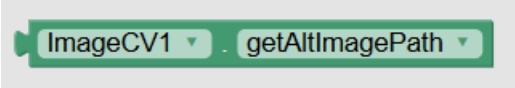
As extensões por padrão, não podem ser componentes visíveis (design do App Inventor) para o usuário, atuando apenas na parte de programação por blocos.

2.3.3. Propriedades do App Inventor na extensão

Os métodos que poderão ser vistos pelo usuário para a programação por blocos, em geral, possuem propriedades que aparecerem antes da inicialização destes e definem as formas que poderão ser utilizados. Tais como:

@SimpleProperty define o método seguinte como um bloco de propriedade simples de alguma outra função ou método, portanto atua como um parâmetro, como no exemplo a seguir:

```
/** Metodo getter para imagem alterada pela extensao */
@SimpleProperty(category = PropertyCategory.BEHAVIOR)
public String getAltImagePath(){
```



@SimpleFunction define o método seguinte como um bloco de função que receberá parâmetros a serem definidos no mesmo.

```
@SimpleFunction(description = "Substitui tons de uma cor por outra através do RGB de cada pixel")
public void altRGB(String picturePath){
```

2.4. Compilando a extensão

Para compilar a extensão, é necessário clicar com o botão direito e em seguida em git bash here dentro da pasta appinventor para entrar na linha de comando do git.

Há dois comandos essenciais:

- ant clean: limpa build anteriores (sempre utilizar antes de ant extensions)
- ant extensions: compila arquivos e gera arquivo aix da extensão

Após a conclusão correta da compilação (sinalizada pela mensagem: BUILD SUCCESSFUL), um arquivo aix é gerado e está localizado na pasta appinventor\components\build\extensions podendo então ser aplicado a qualquer projeto do App Inventor por meio da seção extension no ambiente de design deste.

3. Entendendo a extensão

3.1. Pacotes do BoofCV

A biblioteca BoofCV é disponibilizada de duas formas: uma delas apresenta as dependências colocadas de forma a serem facilmente adicionadas pelas IDE's por meio de comandos. Já a outra é através de arquivos jar, que devido à plataforma para o qual a extensão é desenvolvida (App Inventor), é a forma utilizada para aplicação da biblioteca na extensão.

A versão da biblioteca selecionada é a 0.27, podendo ainda ser modificada para a 0.28. As versões mais recentes não são utilizadas devido à incompatibilidade do App Inventor, que apresenta suporte para o Java 7, com a biblioteca, desenvolvida para o Java 8-11 a partir da versão 0.29.

A versão 0.27 pode ser baixada no seguinte link:

https://sourceforge.net/projects/boofcv/files/v0.27/boofcv-v0.27libs.zip/download

As demais versões estão presentes no site da própria biblioteca.

Os pacotes utilizados pela extensão atualmente são: ---

3.1.1. Configuração dos pacotes

Após serem baixados, os pacotes que serão utilizados devem ser colocados na pasta appinventor\lib. Estes podem ser colocados em uma subpasta com nome a ser escolhido.

Para que seja compilado devidamente, o arquivo build.xml, localizado dentro da pasta appinventor\components, deve ser modificado.

A seguinte formatação deve ser colocada na seção CopyComponentLibraries target (Há um comentário com uma indicação: <!-- Add extension libraries here -->) para cada pacote adicionado à pasta lib que será utilizado:

<copy toFile=”${public.deps.dir}/nomeSimplificadoDoArquivoJar.jar”

file=”${lib.dir}/SubpastaDaBibliotecaNoLib/nomeDoArquivoJar.jar” />

Se estiver correto, após a compilação os pacotes serão copiados da pasta lib para a pasta appinventor\build\components\deps.

Por último, é necessário alterar o arquivo da classe da extensão de forma a informar quais bibliotecas serão utilizadas durante a extensão, etapa no qual já está feita para os arquivos Jar que são atualmente utilizados.

Debaixo dos detalhes da extensão, a seguinte formatação deve estar colocada:

@UsesLibraries(libraries = "library1.jar," + “library2.jar,” + … + “libraryN.jar”)

3.2. Funcionalidade básica da extensão

3.2.1. Caminho da imagem alterada

O método seguinte é um método getter para o caminho da imagem alterada da extensão para que esta possa ser utilizada em outras partes do código pelo usuário

através da programação por blocos, funcionando como um parâmetro para outros métodos.

```
/** Metodo getter para imagem alterada pela extensao */
@SimpleProperty(category = PropertyCategory.BEHAVIOR)
public String getAltImagePath(){
    return altImagePath;
}
```

3.2.2. Imagem após ser fotografada

O método abaixo trata a imagem assim que a foto é tirada pelo componente da câmera do App Inventor e seu caminho se torna disponível para uso.

```
/** Método para converter imagem (tratada como drawable)
    recebida pela câmera em bitmap - retorna Bitmap */
public Bitmap converteDrawableToBitmap(String picturePath) {
    //recebe path pelo usuario na programação por blocos

    Drawable drawable;
    try {
        drawable = MediaUtil.getBitmapDrawable(form, picturePath);
    } catch (IOException ioe) {
        Log.e("Image", "Unable to load " + picturePath);
        drawable = null;
    }

    int scaledWidth = (int) (form.deviceDensity() * drawable.getIntrinsicWidth());

    int scaledHeight = (int) (form.deviceDensity() * drawable.getIntrinsicHeight());

    Bitmap scaledBitmap = Bitmap.createScaledBitmap(((BitmapDrawable) drawable).getBitmap(),
            scaledWidth, scaledHeight, false);

    return scaledBitmap;
}
```

O caminho da imagem é recebido e um drawable[1] é gerado a partir disto, utilizando uma classe do próprio App Inventor (MediaUtil) que trata as variantes de uma imagem.

Em seguida um bitmap[2] em escala é criado, evitando possíveis problemas gerados pelas imagens rotacionadas etc. e retornado.

A conversão de drawable para bitmap ocorre para que os processos envolvendo visão computacional sejam aplicados.

---

[1] Para exibir imagens estáticas no seu app, é possível usar a classe Drawable e as subclasses relacionadas para desenhar formas e imagens. Um Drawable é uma abstração geral para algo que pode ser desenhado.

[2] O bitmap é um conjunto de pixels (também chamados de “pontos”) que carregam uma informação de cor, e é formado pela união dos pixels.

Os *imports* necessários para o funcionamento desse método são:

```
import android.graphics.drawable.Drawable;
import android.graphics.Bitmap;
import android.graphics.drawable.BitmapDrawable;
import android.util.Log;
import java.io.IOException;
```

3.2.3. Salvando imagem alterada

No método a seguir o bitmap que passou por algum processo e foi alterado é comprimido, convertido para uma jpeg e salvo na galeria do celular por meio de um objeto file.

```
public void salvaAltImage(Bitmap bitmap) {

    String root = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES).toString();
    File myDir = new File(root + "/AppInventorALTERADAS/");
    if (!myDir.exists()) {
        myDir.mkdirs(); //cria diretório appinventoralteradas
    }
    Random generator = new Random();
    int n = 10000;
    n = generator.nextInt(n);
    String fname = "AltImage_" + n + ".jpg";  //nome da imagem alterada
    File file = new File(myDir, fname);

    if (file.exists())
        file.delete();
    try {
        FileOutputStream out = new FileOutputStream(file);
        bitmap.compress(Bitmap.CompressFormat.JPEG, 90, out);
        // int quality = 90
        //If you compress() to a PNG, the quality value is ignored.
        //If you compress() to a JPEG, then it is used to control JPEG compression,
        //which commonly uses a 0-100 range to express the desired quality.
        //However, that does not mean that a value of 0 gives you a photo that takes no space.
        out.flush();
        out.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

Por fim, nesse mesmo método, o caminho da imagem alterada é salvo para posterior uso, como exibição desta no aplicativo, entre outros.

```
//SALVANDO PATH DA IMAGEM ALTERADA
altImagePath = root + "/AppInventorALTERADAS/AltImage_" + n + ".jpg";
```

Todas as imagens que passam por processos da extensão são salvas no mesmo caminho: pictures/AppInventorALTERADAS

Os *imports* necessários para o funcionamento desse método são:

```
import java.io.File;
import java.io.FileOutputStream;
import java.util.Random;
import android.os.Environment;
```

3.3. Funções de visão computacional

3.3.1. Não relacionadas à biblioteca BoofCV

3.3.2. Relacionadas à biblioteca BoofCV

Note que muitas classes da biblioteca BoofCV utilizam o tipo BufferedImage[3] para processar as imagens, porém esta classe não é suportada pelo Android. Alternativamente, a biblioteca apresenta classes que possibilitam a conversão de bitmap para tipos de imagens em que a mesma trabalha. Estas classes estão presentes no pacote jar boofcv-android-0.27.

1. Baixando a extensão e configurando arquivos

A extensão (.aix) pode ser baixada no link: ----

Também há um arquivo compactado com a classe da extensão e junto a ela os arquivos build.xml para substituir aquele contido em appinventor/components, pasta lib contendo os pacotes jar (somente aqueles que são atualmente utilizados) e o arquivo aix da extensão atualmente desenvolvida no link: ---

2. Referências

BoofCV. Título: Main Page. BoofCV, 2020. Disponível em: https://boofcv.org/index.php?title=Main_Page. Acesso em: 21/01/2020.

OpenCV. Título: About. OpenCV, 2020. Disponível em: https://opencv.org/about/. Acesso em: 21/01/2020.

[3] A interface Image é a que modela o comportamento de objetos que representam imagens em Java. A classe BufferedImage é uma implementação de Image que corresponde a imagens representadas por uma sequência de pixels armazenada inteiramente na memória.