目次

- **はじめに**
  自己紹介
- **課題**
  マルチプラットホームへの対応、イベントトリガーの検出、人間のエミュレート
- **デバイス比較**
  エミュレータとスマホ実機、ビデオキャプチャー機器とWebカメラ
- **画像解析**
  入力画像の加工、領域の分割、矩形の検出、オブジェクトの検出
- **テンプレートマッチング**
  第1世代、第2世代、第3世代
- **日本語OCR**
  クローラーの開発、文字認識用の教師データ、文章解析用の教師データ
- **性能改善**
  処理速度の向上、聴覚のサポート
- **まとめ**

自己紹介

折田 武己 <takemi.orita@levtech.jp>

フリーランスのエンジニアとして、様々な開発プロジェクトに携わってきた。また、エンジニア向け技術研修の講師として、後進の指導に当たることも多い。

どこの開発現場でも直面する喫緊の課題であるテストの自動化については、ライフワークのひとつとして長年に渡り取り組んできた。

本日の講演内容も、これまでの開発経験から得られた知見をベースにしているが、独自の視点からアイデアを練り直し、日々カイゼンを施している。

課題

Webアプリとスマホアプリでは設計思想が根本的に異なる。

	Webアプリ	スマホアプリ
実行環境	Webブラウザー	OS（Android/iOS）
通信プロトコル	HTTP/HTTPS	（任意）
通信データ	テキスト*1	バイナリ、テキスト
プログラミング言語	HTML/CSS/JavaScript	（任意）
データ構造	DOM	（任意）
自動テストツール	Selenium	Appium

Webアプリに比べて、スマホアプリのUIテストが技術的に困難な理由を次の視点から考えてみる。

- マルチプラットホームへの対応
- イベントトリガーの検出
- 人間のエミュレート

*1 画像データや音声データなど、一部にバイナリを利用することもある。

【課題】マルチプラットホームへの対応

日本国内での*Android*と*iOS*のシェアは拮抗しており、どちらか一方だけに特化するのは避けたい。次のような要求を満たすため、既存のツールキットを採用することになる。

- **提供するアプリの*UX/UI*を統一したい**
- **コードベースを共有したい**

Android/*iOS*に両対応したツールキットの多くは、*OS*固有のウィジェットを使わず、等価な挙動のコンポーネントを自前で描画している。その結果、**テキストやボタンなどの汎用的なコンポーネントでさえ、その背後にあるデータ構造は失われ、画面上のピクセルに色の濃淡として塗り潰される。**

スマホアプリの*UI*テストの技術的な障壁は、すべてそれに起因する。**ラスタライズされた画像データの中から、ボタンやリンクなどのイベントトリガーを探し出すことを求められる。**

【課題】イベントトリガーの検出

ラスタライズされた画像データの中から、イベントトリガーを検出するために画像解析技術を応用することになる。しかし、それだけは不十分なことも多い。

普段は何気なく操作している、次のようなウィジェットを画面上のピクセルに格納された*RGB*データから拾い出し、適切な座標でイベントを発行する必要がある。

- テキスト入力フィールド
- チェックボックス
- ラジオボタン
- ドロップダウンリスト
- リンク
- ボタン

Webアプリなら簡単に実現できることが、なぜこんなにも難しいのか？あまりの不条理さに、この最初のステップで脱落してしまうことが多い。

【課題】人間のエミュレート

*UI*テストを自動化するためには、ラスタライズされた画像データから意味を読み取り、適切なアクションを起こす仕掛け・・・つまり人間のような器官が必要不可欠である。

人間をエミュレートするためのソフトウェア構成は次のようになる。

脳	• Python
視覚	• OpenCV
触覚	• Appium
聴覚	• BlueZ

デバイス比較

スマホアプリの*UI*テストを自動化するのに利用可能なデバイスをリストアップすると下記のようになる。

- エミュレータ
- スマホ実機
- ビデオキャプチャー機器
- *Web*カメラ

次のような観点からデバイスの比較検討を行う。

- エミュレータとスマホ実機
- ビデオキャプチャー機器と*Web*カメラ

【比較】エミュレータとスマホ実機

エミュレータと実機の比較結果は次の通り。

	エミュレータ	スマホ実機
利便性	○	×
導入コスト	○	×
管理コスト	○	×
デバイスの互換性	×*1	○
バイナリの互換性	×*2	○
実行速度	△*3	○

エミュレータはあくまでも擬似環境に過ぎないので、*UI*テストは（可能な限り）スマホ実機で行うべきである。

*1 スマホ実機を完璧にエミュレートしているわけではないので、画面の解像度以外のパラメータはあまり当てにならない。
*2 *Android*エミュレータは*ARM*プロセッサ用の命令コードをそのまま使っているが、*iOS*シミュレータは*x86*プロセッサ用の命令コードに変換している。
*3 *Android*エミュレータは遅くて使い物にならないが、*iOS*シミュレータは実用的な速度で動作する。

【比較】ビデオキャプチャー機器とWEBカメラ

ビデオキャプチャー機器とWebカメラの比較結果は次の通り。

	ビデオキャプチャー機器	Webカメラ
利便性	○	○
汎用性	△	○*1
画像品質	○	△*2
音声品質	○	△*3
導入コスト	×*4	○
管理コスト	×*5	○
実行速度	○	○

*1 AndroidでもiOSでも、まったく同じ機器構成のまま扱うことができる。
*2 Webカメラのレンズで映像データを取り込むので周囲の環境に大きく影響される。
*3 Webカメラのマイクで音声データを取り込むので周囲の環境に大きく影響される。
*4 MHL（HDMI）の信号は著作権保護されており、PCに直接取り込むことはできないため、パススルー可能なスプリッターを経由する必要がある。
*5 たくさんのケーブルを接続するため、配線の取り回しに苦労する。
また、プロダクトの将来的な供給体制にも不安が残る。

画像解析

実機とWebカメラを使ったUIテストの実施にあたり、下記の目標を掲げる。

- （デバッグ用ではなく）リリース用のバイナリを使う
- （スマホだけではなく）タブレットにも対応する
- （座標を決め打ちせず）オブジェクトを動的に検出する

この目標を実現するためには、レンダリングされた画像データを解析し、画面に表示されているオブジェクトを自動判別する必要がある。そこで次のようなアプローチを採用する。

- 入力画像の加工
- 領域の分割
- 矩形の検出
- オブジェクトの検出

【画像解析】入力画像の加工

*Web*カメラからの入力画像は（ビデオキャプチャー機器のそれと比べて）理想的とは言い難い。そのため、次のような前処理を行う。

1. ノイズ除去・・・照明の映り込みなどを除去
2. 色調補正・・・明度や彩度の調整
3. 射影変換・・・画面をトリミングし、台形から矩形に変換
4. サイズ変換・・・*VGA*または*XGA*にサイズを縮小
5. 色分解・・・フルカラー、グレースケール変換、2値変換

【画像解析】領域の分割

特定のシンボルを検出し、現在表示されている画面を特定するところから画像解析作業はスタートする。その上で、画面を任意の領域に分割し、それぞれにドリルダウンする。

- **ステータス領域**
  スコアやアイテム、パンくずリストなどを表示する。
- **メイン領域**
  アプリのコンテンツを表示する。
- **サブ領域**
  画面遷移のトリガーやティッカーなどパネルを表示する。
- **ポップアップ領域**
  ダイアログボックスなどをオーバーレイ表示する。

※面遷移することによって、画面のレイアウト自体も大きく変化することが多い。

【画像解析】矩形の検出

分割された領域をさらにドリルダウンする際、*OpenCV*の矩形検出機能が大いに役立つ*1 。

ユーザーに情報を表示するためのパネルは、入れ子状態になった矩形*2に内包されているのが一般的*3だからである。

※絶対座標や相対座標を使って、オブジェクトを検出するのは（自動テストの開発資産の寿命を縮める*4ので）避けるべきである。

*1 処理を高速化するため、グレースケールまたは2値画像で矩形検出を行う。
*2 *OpenCV*は円や多角形、直線など様々な機能を備えている。
*3 既存の検出器で対応できない場合、オリジナルの検出器を作成することもできる。
*4 画面レイアウトが変更になった場合に自動追従できない。

【画像解析】オブジェクトの検出

テストスクリプトからダイアログボックスを操作する際、次のようなオブジェクトの座標を動的に検出する必要がある。

- テキスト入力フィールド
- チェックボックス
- ラジオボタン
- ドロップダウンリスト
- リンク
- ボタン

*Web*アプリの場合には、*DOM*というオープンな技術基盤があるため、*JavaScript* や *Selenium* を使って検出できる。

スマホアプリの場合には、後述するテンプレートマッチングを使うことによってオブジェクトを検出する。

テンプレートマッチング

テンプレートマッチングとは、ターゲット画像中に含まれるオブジェクトを検出する技術である。主な用途は次の通り。

- アサーション
  表示されている画面を判定する。
- バリデーション
  コンポーネントの状態を取得する。
- ロケーション
  コンポーネントの座標を取得する。

これまでに開発したテンプレートマッチングは次のように進化してきた。それぞれの特性に応じた利用が肝要である。

- 【第*1*世代】画像の照合
- 【第2世代】特徴点の照合
- 【第3世代】文字列の照合

【テンプレートマッチング】第1世代

第1世代のテンプレートマッチングでは、画像データを使ってオブジェクトの検出を行う。

```
# OKボタンのクリック（テンプレート画像のパスを間接指定）
button = self.container.find_image(":ok_button")
button.click()
```

OpenCVに用意されたメソッドを呼び出すだけなので、とても簡単に実装できる。しかし、次のような課題がある。

- 背景画像が変化する場合に対応できない*1
- ボタンの色が状態に応じて変化する場合に対応できない*2
- テンプレート画像を事前に用意する必要がある*3

*1 テンプレート画像に背景を含まないようにトリミングすることで回避できる。
*2 それぞれの状態ごとにテンプレート画像を用意すれば回避できる。ただし、照合回数もその分だけ増える。
*3 入力画像に合わせてリサイズし、背景画像を含まないようにトリミングするのは思ったよりも手間がかかる。

【テンプレートマッチング】第2世代

第2世代のテンプレートマッチングでは、**特徴点データ**を使ってオブジェクトの検出を行う。

```
# OKボタンのクリック（特徴点データのパスを間接指定）
button = self.container.find_feature(":ok_button")
button.click()
```

*OpenCV*が独自実装したロイヤリティ不要のアルゴリズム***ORB***を使うことで、テンプレート画像に比べて大幅なデータ容量の軽量化に成功した。しかし、次のような課題がある。

- **テンプレート画像に比べて直感的ではない***1*
- **特徴点データを生成するツールが必要である***2*
- **特徴点データを事前に用意する必要がある***3*

*1 特徴点データの中身を見ても、一体何に対応するものかが人間には理解できない。
*2 テンプレート画像から特徴点データを自動生成できるようにする。
*3 入力画像に合わせてリサイズし、適切にトリミングした画像ファイルを保存しておけば、（必要に応じて）特徴点データが生成されるようにする。

【テンプレートマッチング】第3世代

第3世代のテンプレートマッチングでは、**文字列**を使ってオブジェクトの検出を行う。

```
# OKボタンのクリック（文字列を直接指定）
button = self.container.find_string("OK")
button.click()
```

OCRを使うことで、文字列を引数にしてオブジェクトが検出できるようになった。事前準備が一切不要なので、生産性や保守性が向上する。しかし、次のような課題がある。

- 文字列を含まないオブジェクトには対応できない*1
- 日本語に対応したOCRが必要である*2
- OCRの呼び出しによるオーバーヘッドが発生する*3

*1 第1世代、第2世代のテンプレートマッチングを利用する。
*2 画面遷移のトリガーは文字列が含まれること多いので、OCRを利用するメリットは大きい。
*3 対象領域の画像データと判定結果をキャッシュすることにより2回目以降は高速化できる。

日本語OCR

第3世代のテンプレートマッチングに必要不可欠なのが、日本語OCRである。その調達方法について検討する。

	コスト	機能性	可用性	拡張性
市販プロダクト	×	○	×*1	×
Webサービス	○	×*2	△*3	×
OSSライブラリ	○	△*4	○	○
自作ライブラリ	△*5	○	○	○

日本語OCRに求めるのは次のような仕様である。

- アンダーラインやドロップシャドウに対応
- 手書き文字は対象外（日英フォントのみ）

せっかくなので、日本語OCRを自作してみた。

*1 対応するOSが限定され、ランタイムの配布にも制約がある。
*2 画像データのアップロードが必要になるため、レスポンスが遅くなる。
*3 ダウンタイムなくサービスを利用し続けられるのか不透明である。
*4 英語のように単語間にスペースが存在する前提のものが多いので、改造範囲が広い。
*5 ディープラーニングを応用すれば、実装すべきコード量は極めて少ない。

【日本語OCR】クローラーの開発

日本語OCRの教師データ（文字認識用、文章解析用）を入手するために専用のクローラー *1 を開発する。

	利用目的
Selenium	*HTMLファイルとスクリーンショットの取得*
Beautiful Soup	*HTMLファイルからDOM構造を解析し、コンテンツを抽出*
OpenCV	スクリーンショットに含まれるコンテンツ領域の切り出し
MongoDB	画像データとメタ情報、テキスト情報の管理

このクローラーによって自動収集された下記の情報が日本語OCRの教師データとなる。

- *HTMLファイルに含まれるテキスト情報*
- スクリーンショットに含まれる画像データ

文字認識用は、フォントの種類やスタイルを切り替えて、複数のスクリーンショットを同時に取得する。

*1 基本的な仕組みはWebページのスクレイパーそのものである。

【日本語OCR】文字認識用の教師データ

クローラーによって入手した画像データを1文字ごとに切り出すことによって教師データを生成する。

	利用目的
MongoDB	画像データとメタ情報の管理
OpenCV	画像データから文字の切り出し
Chainer *1	深層学習ライブラリ

文字単位の画像切り出し作業は意外と大変である。お薦めなのは下記のようなWebページを動的に生成し、クローラーに収集させることである。

- HTMLのTABLEタグを使って1マスに1個の文字を配置
- TABLEタグの罫線は非表示（border=0）
- ページャーを使って1ページあたりの文字数を制限

*1 TensorFlowなどの他の使い慣れた深層学習ライブラリでも構わない。

【日本語OCR】文章解析用の教師データ

クローラーによって入手したテキスト情報を形態素解析したものを（必要に応じて*1）*Word2Vec* で機械学習させる。

	利用目的
MongoDB	テキスト情報の管理
MeCab	テキスト情報の形態素解析
Chainer *2	深層学習ライブラリ

文字単体の認識ができるようになっても、促音や濁音、拗音などの区別は難しい。次のようにして変換精度を向上させる。

- **辞書データを用意する** *3
- **情報源を厳選する** *4

*1 ボタンやリンクに含まれる名詞だけではなく、まとまった文章を扱うとき。
*2 TensorFlowなどの他の使い慣れた深層学習ライブラリでも構わない。
*3 最長一致する単語を調べる際に用いる。専門用語の辞書があると変換精度が向上する。
*4 テキスト情報に誤字脱字が含まれていると学習結果にも悪影響がある。コンテンツをしっかりと校閲しているWebサイトの利用が好ましい。

性能改善

これまでに説明した技術を組み合わせれば、スマホアプリの*UI*テストを自動化するための準備は揃うはずである。

- **画像解析**
  ***Web*カメラ経由で入手した画像を解析する。**
- **テンプレートマッチング**
  イベントのトリガーを動的に検出する。
- **日本語*OCR***
  画面に表示された文字データを取得する。

さらなる性能改善や機能拡張を目論む場合、次のアプローチを採用してみてはどうだろう。

- **処理速度の向上**
- **聴覚のサポート**

【性能改善】処理速度の向上

アクション性の高いゲームでは、*60fps*でレンダリングされる画面に対しての即応性が求められる。処理を高速化するためには、下記のような工夫を適切に組み合わせ、計算量を減らす必要がある。

- 対象領域のクリッピング
- 照合対象のカスケード
- 判定結果のキャッシュ
- フレームバッファの差分抽出

それでもなお処理落ちが発生してしまう場合、次のような選択肢を考慮しておく。

- 入力画像のサイズ縮小
- 画像処理エンジンのクラスター化

【性能改善】聴覚のサポート

ゲームの場合、プレイヤーの没入感を高めるためにサウンドが果たす役割は大きい。自動テストツールに聴覚が備わると応用範囲も拡大するが、次のような検証が必要になる。

- *ON/OFF*の切り替え
- ボリュームの制御
- ビジュアルとサウンドの同期確認

*USB*マイクやライン入力を利用することもできるが（イヤホン端子のないスマホも増えているので）*Bluetooth*スピーカーに擬態して音声データを取り込む*1のがスマートである。

- フィルターを使った周波数帯域の分離*2
- 波形データからの特徴点抽出*3

*1 *Linux*であれば *BlueZ* などのプロトコルスタックが使える。
*2 フーリエ変換とローパスフィルターといったシンプルな組み合わせでも十分に使える。
*3 音声データの場合、スナップショットではなく、ストリームとして扱う必要がある。

まとめ

スマホアプリの*UI*テストを自動化する際、どのようにしてスクリーンに表示されている情報を取り込むのか？ 身近にある「*Web*カメラ」というデバイスを活用することで、次のようなメリットが得られる。

- 低コストでハードウェアを揃えられる
- *OS*標準のドライバをそのまま使える
- *OpenCV*から映像を直接制御できる

*OSS*の普及によって、画像解析や音声解析、ディープラーニングといったパワフルなツールを個人レベルでも使えるようになった。日本語*OCR*のような複雑なソフトウェアをほんのわずかなコードで実現できたことに自分でも驚いた。

案ずるより産むが易し・・・思考実験に留めるより、実際にアクションを起こすことが重要である。

ご静聴ありがとうございました。