平成 28 年度　修士論文

携帯端末上での画像処理による野球の投球速度推定方式

平成 29 年 2 月 16 日

15350928

山口　陽介

指導教員 三浦 元喜 准教授

九州工業大学大学院 工学府 先端機能システム工学専攻

概 要

野球というスポーツにおいて，球速の測定は最も一般的な投球解析の一つである．プロ野球では，試合や練習においてスピードガンによる球速測定が一般的におこなわれているが，測定機器の高価さや測定の難しさなどの理由から，アマチュア野球選手や野球観戦をするファンにとっては一般的なものにはなっていない．

そこで，誰もが手軽に球速の測定を楽しめるようにするために，一般の人々にも広く普及しているスマートフォン上で動作するアプリケーションとしてスピードガンを実現するための手法を提案する．特に本研究では，野球の練習の場面においても手軽に使用できることを目指し，球速測定の際の利便性の向上を目的として開発をおこなった．

ピッチャーからキャッチャーまでの距離を加速度センサとカメラを利用して求め，ボールのリリースの瞬間は画像処理によって検出し，ボールキャッチの音をマイクで検知することによってボールの移動距離と移動時間を取得し，球速の推定をおこなった．

開発したアプリケーションによる球速測定の結果とドップラー方式のスピードガンによる測定結果との比較実験をおこない，球速の測定結果として十分に妥当な球速をリアルタイムに出力することができた．測定中の人による操作をなくし，キャッチャーの後方からのリアルタイムな測定を実現することで，球速測定の利便性を大きく向上させることができた．

目 次

第1章　序論

本論文は野球の投球解析の中でも最も一般的な球速の測定を携帯端末上で画像処理をおこなうことによって実現する手法について論ずるものである．本章では本研究の背景，目的について説明していく．

1.1　研究背景

本研究では，スマートフォン上でコンピュータビジョンの技術を駆使し，野球のピッチャーの投げるボールの速度を推定するアプリケーションを開発する．野球はピッチャーがマウンドからバッターに向けてボールを投げることで勝負が始まるスポーツであり，そのピッチャーの投げるボールの速度はバッターとの対戦結果を決める重要な要素の一つである．プロ野球の観戦時には球場に設置されたスピードガンによる球速の計測及び表示がおこなわれ，プロ野球観戦時のファンの楽しみの一つとなっている．

プロ野球では，練習時においてもスピードガンによる球速測定が一般的におこなわれている． 測定された球速は，ピッチャーのパフォーマンスを判断する上で重要な材料となる．しかし，少年野球や学生野球のようなアマチュア野球の試合や練習の場面においては，球速測定は一般的なものにはなっていない．球場設置のスピードガンによる測定は福岡県の高校野球の試合では，県大会ベスト 8 以上の試合でしか実施されておらず，それ以外の試合や練習時に気軽に利用できるものではない．少年野球や中学野球，草野球等の場面でも同様のことが言える．

近年，スマートフォンは携帯電話としてだけでなく，小型のコンピュータとして利用できるスペックを持つようになっている．スマートフォンに実装されているカメラについても同様で，その機能は年々進化してきている．例えば，2015 年 9 月に発売された iPhone 6s では最大 240fps での撮影が可能となっている．また，カメラ以外にも加速度センサやマイクといった様々なセンサが搭載されており，バッテリーで動作することから屋外で

の利用に適している．このような様々なセンサを搭載しながら携帯性に優れたサイズを持つ端末はスマートフォンをおいて他にない．近年では多くのスマートフォンにおいて最初に購入した時からカメラやマイク，加速度センサといったセンサが搭載されていることが当たり前になっており，これらのセンサを利用した球速推定手法は本研究で実装をおこなったスマートフォン端末以外にも同様のセンサを持つスマートフォンに対して応用が可能である．

現在では，スマートフォンは誰もが当たり前に持っている一般的な端末となってきているため，スマートフォン上で動作するアプリケーションとしてスピードガンを実現することで，球速測定を手軽なものにすることができる．このような理由から，本研究ではスマートフォンを用いた球速推定手法を提案し，実装をおこなう．

1.2 目的

前研究 ([1]，[2]) では横から撮影した動画 (図 1.1) を利用したスピードガンアプリケーションを開発することで，球速測定の可能域を拡大し，投球完了から 5 秒程度の処理速度を実現した．

図 1.1: 横からの投球動画からボールを追跡し，球速を推定する．

野球の試合の場面で 5 秒以内に次の投球を開始することはほとんどないが，練習の場面においては 5 秒以内の短い時間間隔で投球練習をおこなうことも多い．球速測定の際には投球の開始に合わせて測定開始のボタン操作をおこなう必要があり，測定者の操作に慣れが必要であった．また，このアプリケーションを使用して球速の測定をおこなう際には，画面上での距離を取得するために実行画面上のマーカーにピッチャーとキャッチャーが入るような画角になる位置まで離れて測定をおこなうため，ピッチャーとキャッチャー以外に観測者が必要であり，あらかじめピッチャーからキャッチャーまでの距離がわかっている必要があった (図 1.2).

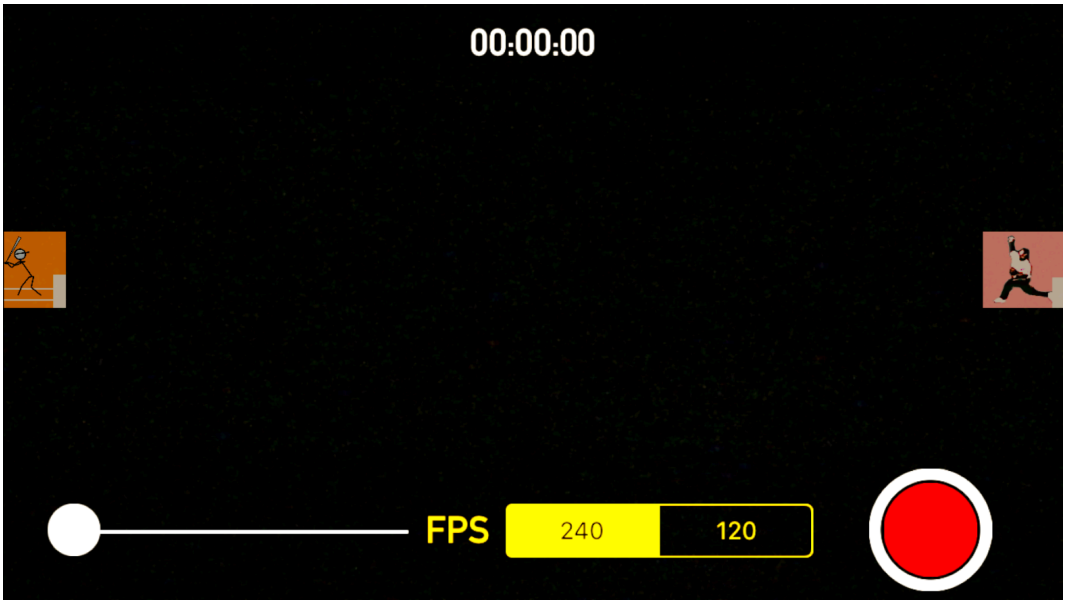


図 1.2: アプリケーションの実行画面．マーカーにピッチャーとキャッチャーを入れるようにして測定をおこなう．

そこで，本研究では，投球練習やキャッチボールのような野球の練習の場面において，より手軽に利用できるアプリケーションを実現するための手法を提案することを目的とし，球速測定の精度の向上，ロバスト性の向上は別研究でおこなう．具体的な技術目標としては，

1. キャッチャーの後ろから測定できる．
2. 球速測定中の人の操作をなくす．

3. リアルタイムに球速を出力できる．

の 3 つの項目を全て満たすことにより，スピードガンアプリケーションの利便性向上を実現する．

投球の練習は，ブルペンのように測定可能なスペースが限られる場所においてもおこなわれるため，キャッチャーの後ろから測定することができれば限られたスペースにおける球速測定が可能になる．前研究において横方向からの球速測定については実現しているため，どの方向からでも測定が可能なスピードガンアプリケーションの実現にもつながる．また，アプリケーションの操作や出力の確認をキャッチャーがおこなうことができるようになり，測定の手間を少なくすることができる．ボールのキャッチを近くで観測することができるため，ボールの到達点を精度よく取得できるという本研究独自のメリットもある．

前研究では，投球に合わせて人が操作をおこなう必要があったが，そのような操作をなくすことができれば，操作方法を覚える必要がなくなり，ピッチャーの投球を普段から見ている人でなくても球速の測定をおこなうことができるようになる．

リアルタイムに測定結果を知ることができればピッチャーは練習時に様々な投球間隔で投球練習をおこなうことができる．次の投球を測定できるようになるまでの処理の時間を待つこともなくなり，投げることに集中することができる．

このような利便性の向上を実現することが本研究の目的である．

第2章　関連研究

プロ野球ではドップラー方式のスピードガンで球速測定がおこなわれているが，野球の投球を分析する研究はその他にも様々な方法でおこなわれてきている．本章では，ドップラー方式のスピードガン以外に，加速度センサを使用する研究，画像処理による研究，レーザー装置を用いた研究，公開されているスマートフォンアプリケーションを使った球速測定，そして我々が以前おこなった研究について紹介する．

2.1　ドップラー方式のスピードガンを用いた手法

球速を測定する代表的な機材として，電磁波や超音波のドップラー効果を利用したスピードガンがある．蔭山らの研究 [3] や勝亦らの研究 [4] など，野球の投球を解析する多くの研究で用いられているのがこのドップラー方式のスピードガンである．投球軌道上で測定をおこなうことができればリアルタイムに高精度な測定結果を出力できる．しかしながら，アマチュアの選手や選手の保護者，観戦するファンにとって，スピードガンを使った球速測定は一般的ではない．これには，ドップラー方式のスピードガンによる測定の 2 つの問題が原因だと考える．

1 つ目はコストの問題である．プロ野球でも使用されるハイグレードタイプのもの[1] から [2] 比較的安価なタイプのスピードガンまで，様々な種類のスピードガンがある．しかし，比較的安価なタイプでも学生をはじめとするアマチュア選手にとってはまだまだ高価であり，ファンや選手が個人で気軽に利用できるものとはいえない．

2 つ目は，測定精度の問題である．ドップラー方式のスピードガンはボールに向けて電磁波を照射し，ボールからの反射波を測定する．ドップラー効果により照射波と反射波では周波数に変化がおきているため，これらの波の周波数を比較することにより球速を計算

[1] ミズノ スピードガン HP-2 2ZM1050: 477,360 円 60m までの距離から測定可能．
[2] BUSHNELL デジタルスピードガン スピードスター V: 30,240 円 27m までの距離から測定可能

している．正しい測定結果を得るためにはボールの軌道上にスピードガンを固定しておく必要があり，ボールの軌道に対して角度があるとその分だけ誤差が生じる．ハイグレードタイプのスピードガン HP-2 の場合，球速 100km/h のボールを測定したとき，ボールの投球軌道からの計測角度が 10 度で 98.48km/h，20 度で 93.69km/h，40 度で 76.60km/h と表示されることが HP-2 取扱説明書 [5] で示されている．

これらの 2 つの問題がスピードガンを利用しての球速測定がアマチュア選手やファンにとって一般的でない原因であると考えられる．

2.2 画像処理を用いた高度な投球分析

投球動画に対し画像処理をおこなうことによって球速やその他の投球に関する分析をおこなう研究はこれまでに数多くおこなわれている．

Hua-Tsung Chen ら [6] の研究では，野球中継の映像から投球フォームの分別をおこなっている．ピッチャーの姿勢を 人の手足と頭の 5 点を利用した Star Skeleton と呼ばれる基準によって投球フォームを分析し，利き腕や投法によってピッチャーの 投球フォームを分別している．この研究では，精度の高いフォーム解析を実現している反面，処理速度を課題としており，野球中継においてリアルタイムに分別をおこなうことはできていない．

子安ら [7] の研究では，ピッチャー後方とキャッチャー後方の 2 台の高速度カメラを用いてマーカーが塗布されたボールを撮影し，ASIFT を用いた特徴点追跡によって，投球軌道全体とボールの回転情報を解析している．複数の方向からの高速度映像を使用することで，精度の高い結果を得ることができるが，使用する機材が高価なため，一般の利用には向かない．

Theobalt ら [8] の研究では，周りを暗くした地下室において光学マーカーを描いたボールを投げ，それをストロボライトを点滅させながら撮影することによって多重露光画像を作成し，投球されたボールの軌道に沿った三次元位置，ボールの初期速度，回転軸，回転周波数を取得し，手の動きの再現をおこなっている．使用するカメラは低コストの商用カメラだが，周りを暗くできる空間を用意する必要がある．

蔭山ら [9] の研究では，光学式モーションキャプチャシステム MAC3D を用いて投球動作中の上胴，下胴回転角，捻転角度，及び各速度を算出し，投球数の投球動作への影響を調査している．MAC3D ではリアルタイムに 3 次元情報の取得が可能であるが，装

置の値段が非常に高価で一般の人々の使用は想定されていない．

これらの研究では，手軽さやリアルタイム性よりも，分析精度の高さに重きをおいている．専用の空間での撮影や高価なビデオカメラを使用しての撮影によって得られた動画を画像処理することによって，投球フォームやボールの回転など，様々な投球分析が高い精度で可能となっている．

一般の人がこれらのシステムによって球速の測定をおこなうには，装置が高価であったり，測定に必要な準備が困難であるという問題がある．また，ビデオカメラで撮った動画をコンピュータに読み込んだ後に投球の分析をおこなうものが多く，その場合は分析するために撮影した動画を研究室に持ち帰る必要があるため，投球をおこなったその場でリアルタイムに球速を知ることはできない．

2.3 加速度センサを用いた手法

Lapinski ら [10] の研究では手，手首，上腕，胸，腰，バットにセンサデバイスを取り付け，投球フォームや打撃フォームの解析をおこなっている．ウェアラブルでワイヤレスなデバイスを使うことにより，コーチや医療関係者が怪我の防止やリハビリーテーション， 練習の指導に利用しやすいシステムとなっている．センサデバイスを体に取り付けて使用する解析においては，装置自体に大きさや重さがあり，デバイスを装着した状態で普段の投球フォームと全く同じ測定結果を得ることは難しい．また，デバイスを装着する位置によっても結果は変わってしまうため，毎回の測定時にはデバイスが正確に同じ位置にある必要がある．取り付ける装置も高価になってしまうため，手軽に利用できるものではない．

スマートフォン内蔵の加速度センサによって簡易的に球速を推定する方法として,「びゅん」[11] というアプリケーションが公開されている． 「びゅん」では，携帯端末をボールを投げるように振り，端末内の加速度センサの値によって球速を推定する．携帯端末を振る場合，端末を投げてしまう危険性がある． また，実際にボールを投げて球速を計測することはできない．

2.4 レーザー装置を用いた手法

斎藤ら [12] の研究では，レーザー装置を使った球速の測定をおこなっている．レーザー装置は 1.0m × 1.44m のアルミ製の枠の下段に 16 個の発光部，上段に 16 個の受光部をそれぞれ 5cm 間隔で並べ，さらに投球方向に 0.45m 離れた位置に同じ設定でレーザーを並べた二重カーテン状の装置である．この二重のレーザーカーテンを通過する時間を計測し，球速を算出している．

最大誤差 2 % という高い精度で測定できるが，この方法を利用する場合，測定装置の用意が難しい．また，測定が可能なのはこの装置の内側を通ったボールに限られる．

2.5 ボールの到達時間からの分析

スマートフォンアプリケーションとして，ピッチャーがボールが投げたらスマートフォンの画面をタップ，キャッチャーがボールを捕球したらもう一度タップ，またはピッチャーがボール投げたらタッチしてボールが捕球されたら指を離す，という操作でボールの到達時間を測り，球速を算出する，という手法のアプリケーション「Easy SpeedGun」[13] が配信されている．

「EasySpeedGun」では，スマートフォンを使って測定ができ，非常に手軽であるという点では優れている．しかし，ピッチャーからボールが離れてキャッチャーまで到達するのにかかる時間は，球速 100km/h で約 0.6 秒，120km/h で約 0.5 秒，150km/h で約 0.4 秒であり，この測定方法ではタッチのタイミングが 0.1 秒ずれると測定結果が大きく変わってしまう．人によってタッチのタイミングも違うため，高い精度は期待できない．

2.6 移動物体を指でなぞるアプリケーションでの分析

野球用ではないが，Android アプリケーション「スピードガン - Speed Gun」[14] では，移動する物体との最短距離を入力し，スマートフォンのカメラからの画像上の移動物体を画面上でタッチし，移動物体の動きに合わせて画面上の移動物体を指で水平になぞることで移動物体の速度を測定することができる．

車のように大きさの大きい物体であれば，指でなぞりやすく，正確になぞることができれば高い精度の結果が得られる可能性がある．しかし，野球の投球に応用する場合，ボールの動きが速く大きさも小さいため，指で追うことは難しい．

2.7 前研究「携帯端末の画像処理による球速測定」

我々が以前おこなった研究 [1][2] では，ベンチやスタンドの位置など，横から撮影した投球動画を画像処理することによって球速を測定するアプリケーションを開発した．アプリケーションの実行画面上にピッチャーとバッターのマーカーを配置しており，そのマーカーにピッチャーとバッターが入る位置に測定者が移動して測定をおこなう．ピッチャーマウンドからバッターボックスまでの距離は野球のルールから既知である．撮影した動画からボールの追跡をおこない，2 つのマーカー間でボールが検出されたフレーム数を数えることにより球速を推定した．動きが複雑であるピッチャーとバッターの領域は処理対象から外している．動画を一度保存してから画像処理をおこなうため，120fps のフレームレートで動画撮影をおこなった場合，動画撮影終了から約 5 秒の処理時間を必要とする． 投球開始のタイミングに合わせて開始ボタンをタップする必要があり，操作の慣れが必要であった．また，白っぽい砂のグラウンドではボールと一体化してしまって測定ができないことがある．

2.8 本研究の位置づけ

本章で上げた関連研究では，高価なハイスピードカメラや特殊な機材を用いたり，分析用に専用の空間を用意する必要があったり，実際の投球シーンとは異なる状態で測定する必要があったりと，投球の分析をおこなうためにかかるコストや手間が大きく，一般の人に向けたものではなかった．手軽に利用できるスマートフォンアプリケーションの場合，精度に問題があり，人間の操作によって大きな誤差が出てしまうものが多い．

本研究では，ボール抽出が難しくなるキャッチャーの後ろ方向からの球速測定に挑戦する．以前の我々の研究では処理の対象から外していたピッチャーの領域についても本研究では画像処理をおこなう．投球練習の場面でも手軽に球速測定がおこなえるように

測定中のボタン操作をなくし，前研究では一体化してボールが消えてしまっていた白っぽい砂グラウンドでも測定をおこなうための工夫についても提案する．

第3章　提案手法

本章では球速測定前の準備から球速を推定し出力するまでの提案手法を説明する．スマートフォン上でリアルタイムに処理をおこなうことを目標としているため，処理速度にかかる時間を考慮して，物体検出器やオプティカルフローといった動体検出でよく用いられている手法は使用せず，簡易的な画像処理に基づく手法を提案する．

3.1　キャッチャーからピッチャーまでの距離推定

本研究では，キャッチャーの後方から球速の測定をおこなう．キャッチャーの後方にスマートフォンを設置して撮影をおこなう場合，画面内で測定地点からピッチャーまでの距離を 2 次元の画像のみから取得することが難しい．そこで我々は，スマートフォンの加速度センサの値を取得し，それを利用して距離の推定をおこなう．スマートフォン のヨー軸，ロール軸，ピッチ軸は図 3.1 のように定義できる．

本研究では，スマートフォンを横向き (ホームボタンが右側に来る向き) で使用することを想定しており，ピッチ角とロール角の値を利用する．距離の測定は，測定対象が地面と接地する点が画面の中央に写るように画面を傾けることでおこなう (図 3.2).

スマートフォンのカメラレンズの高さを h[m]，ロール角を θ_r とすると，対象物と地面の接地点までの距離 l，は次の計算式で求めることができる．

$$l \quad = \quad h \tan \theta_r [m] \tag{3.1}$$

上記の方法から 観測点からキャッチャーまでの距離 l_c，ピッチャーまでの距離 l_p をこの順にそれぞれ測定する．測定時はピッチャーの接地点が画面の中央に映る状態で測定を開始するため，キャッチャーまでの距離を先に測定し，次にピッチャーまでの距離を測定することで，スマートフォンを距離測定終了時の状態のまま動かすことなく測定に移ることができる．

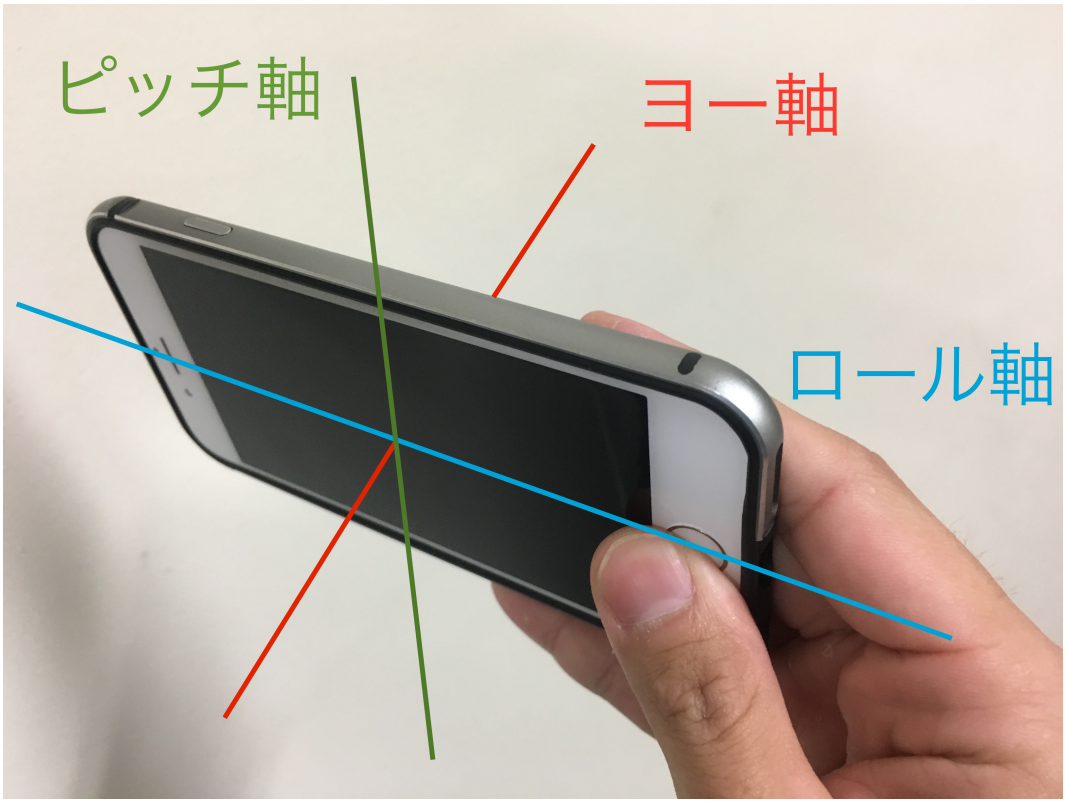


図 3.1: スマートフォンのヨー軸，ロール軸，ピッチ軸の定義

また，キャッチャーまでの距離測定時のピッチ角 θ_{pc} と ピッチャーまでの距離測定時のピッチ角 θ_{pp} を用いて，キャッチャーからピッチャーまでの距離 l_{cp} は，次の計算式で求めることができる．

$$l_{cp} = \sqrt{l_c^2 + l_p^2 - 2l_c l_p \cos\left(|\theta_{pc} - \theta_{pp}|\right)}[m] \tag{3.2}$$

距離測定の概要図を図 3.3 に示す．

この方法で距離を計算することによって，ピッチャーとキャッチャーを結ぶ直線上の位置からでなくてもピッチャーからキャッチャーまでの距離を測定することができる．

実際に投球する際には，ピッチャーはキャッチャー方向にステップして投げるため，リリースポイントはピッチャーの立っていた位置よりもキャッチャー方向にずれ，ボールの移動距離は l_{cp} よりも短くなる．本論文では，ボールの移動距離を $l_{cp} - 1$[m] と仮定して

図 3.2: 距離測定の様子．距離を測りたい対象の接地点を中央に写す．

計算をおこなう．

3.2　ピッチャーの投球探索領域の設定

ここからは画像処理によってリリースポイントを検出する方法を説明していく．リリースポイントが検出されるまでの画像処理の大まかな流れを図 3.4 に示す．

本研究では，キャッチャーの後方から球速の測定をおこなう．キャッチャーの後方からの動画撮影によって，図 3.5 のようなフレーム画像が得られる．このような画像に対して画像処理をおこなっていく．

ピッチャーからキャッチャーまでの距離の測定が完了した時点で，スマートフォン端末からピッチャーまでの距離 l_p が得られており，画面の中央にはピッチャーの接地点が写っている．ピッチャーの画面中央からの高さは，l_p によって一意に決まるはずである．ピッチャーの投球を探索する領域はこのことを利用して設定する．

実験から得られた値を利用し，解像度 1920 × 1080[pixel] の画像において，身長 200 cm 以下の投手が投球時に手を伸ばしても領域内に収まる大きさとして，高さと幅が 300 cm 程度になるように表 3.1 のように領域の大きさを定めた (図 3.6)．

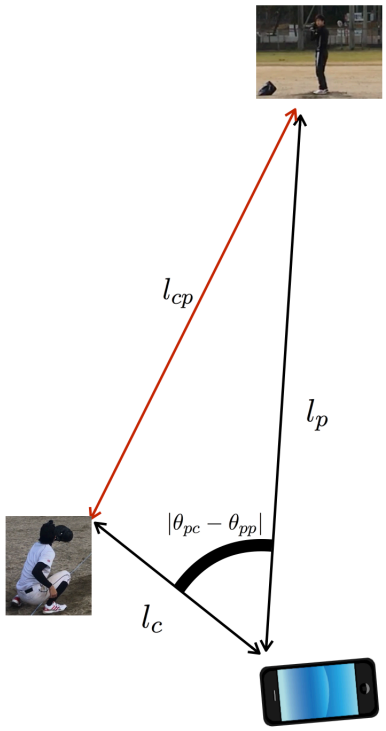


図 3.3: 距離測定の概要図

3.3 投球領域の抽出

ピッチャーからボールがリリースされるまでは，3.2 で設定した領域内の画像処理をおこなう．リリースポイントを検出するための前処理として，投球探索領域から実際に投球動作をおこなっている投球領域を抽出する．

3.3.1 フレーム間差分

投球探索領域内から投手の投球動作を検出するためにフレーム間差分による動体検出をおこなう．連続する 3 フレームを取り出し，グレースケール化した後，1 フレーム目と 2 フレーム目の差分画像と 2 フレーム目と 3 フレーム目の差分画像の論理積をとる (図 3.7，図 3.8)．

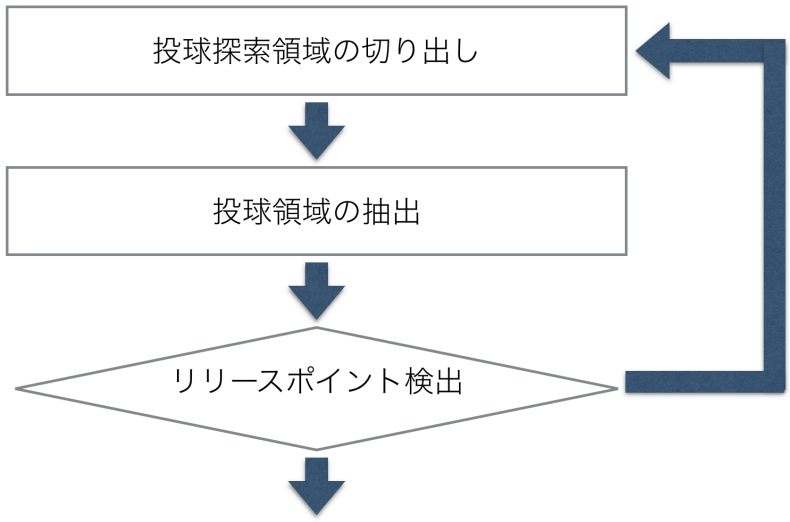


図 3.4: リリースポイント検出までの画像処理の流れ

この処理画像を二値化した後，収縮処理によってノイズを除去し，膨張処理によって動体部分を強調する．処理後の画像は図 3.9 のようになる．

3.3.2 投球領域矩形の取得

3.3.1 までの処理によって，投球していると思われる部分を白の領域として抽出することができる．この白の領域を包括する矩形を取得するために，まず白の領域を連結成分ごとに矩形で囲む．それらの矩形全てを包括する矩形 (図 3.10) を求め，ピッチャーの投球領域とする．

図 3.5: キャッチャーの後方から撮影したフレーム画像

表 3.1: 投球探索領域の設定

距離 [m]	高さ [pixel] × 幅 [pixel]
20 - 25	210 × 210
15 - 20	260 × 260
15 以下	350 × 350

3.4 ボール探索

本研究では，ピッチャーの手からボールが完全に離れた瞬間をリリースポイントと呼ぶことにする．右投手がオーバースロー，またはスリークォーター投法で投球をおこなう場合，投球画像中におけるボールはピッチャーの左上の位置に現れるはずである．そこで，投球領域矩形の左上からボールを探索する．リリース直前からリリース直後にかけて，ピッチャーの動きは大きくなり，投球動作をおこなっている投球領域矩形の面積も大きくなる．この面積の値が設定した最小値を超え，かつ投球矩形内の白の面積が投球領域矩形の面積の 4 分の 1 以上であるとき，リリースポイント検出のためにボールの探索をおこなう．面積の最小値の設定は，本手法を録画された動画に対して実装する際に投球領域と呼べる面積値の最小値を人の目で判断し，適切な値を設定した．

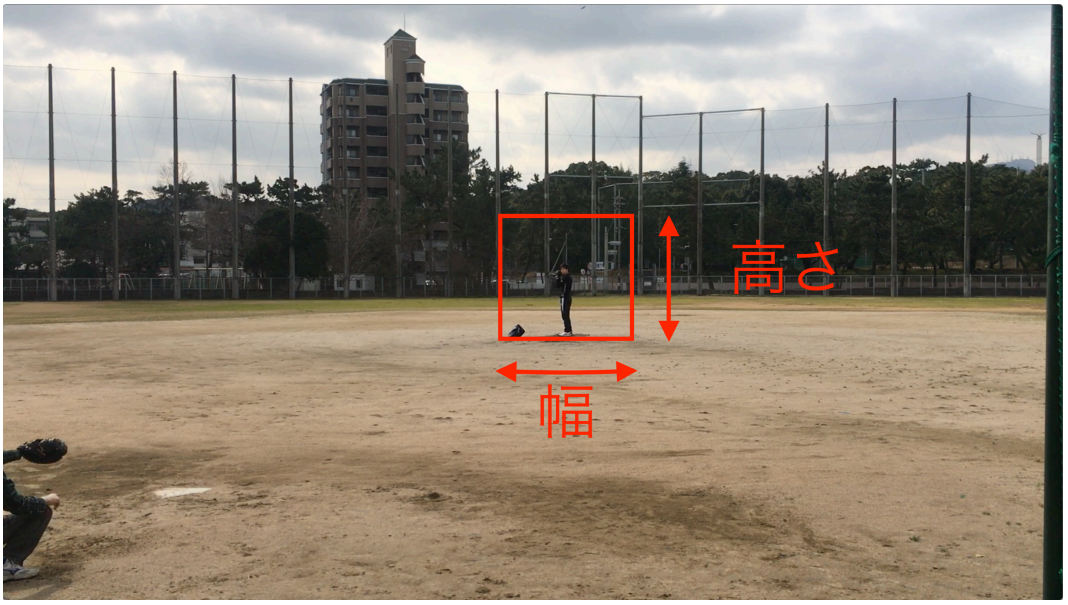


図 3.6: 投球の探索領域の設定

ボール探索領域は投球矩形の始点座標から投球矩形の幅の 3 分の 1 を一辺とする正方形とする．この方法で投球矩形を求めた場合，足元のみが動いている時に足元周りでボール探索をしてしまう場合やノイズによって大きく離れた矩形を含んだ投球矩形になる場合が出てきてしまうため，投球矩形に対する白の部分の割合の最小値と投球矩形の面積の最小値を設定し，それ以外の時にはボール探索をおこなわない．

この正方形の中からボールとなる白の部分を検出する．元のグレースケール画像からこの正方形部分を切り取り，二値化する．二値化した画像から輪郭抽出をおこない，輪郭が検出された時，ボールを検出したと判断する．ただし，大きすぎたり小さすぎる輪郭が検出された場合はボールとは判断しない．砂のグラウンドのようなボールと背景との輝度値の差が小さい環境で測定する場合は，そのグラウンドの色の輝度値を閾値として設定することもできる．リリースを検出したフレームの画像を図 3.11 に示す．

3.5 ボールキャッチ検出

後ろからの球速測定では，ボールキャッチの数フレーム前からボール画像はキャッチャーに隠れるなどして見えなくなってしまう．そこで，スマートフォンのマイクからボール

(a)1 フレーム目　(b)2 フレーム目　(c)3 フレーム目

図 3.7: 連続 3 フレーム画像

(a) フレーム 1 とフレーム 2 の差分画像　(b) フレーム 2 とフレーム 3 の差分画像　(c)2 枚の差分画像の論理積画像

図 3.8: 差分画像と論理積画像

キャッチ時の音を検知することで，ボールキャッチの検出をおこなう．測定はキャッチャーの近くでおこなうため，ボールキャッチの音は容易に検知できる．リリースポイントを検出した時間からボールキャッチを検出するまでの時間をボールの移動時間とする．

ボールの移動時間 t_b と ピッチャーとキャッチャーの間の距離 l_{cp} から，ボールの平均時速 v は次の計算式から求めることができる．

$$v = \frac{l_{cp} - 1}{t_b} \times 3.6[km/h] \tag{3.3}$$

最後に，ここまでの球速推定手法のまとめとして，処理の流れを図 3.12 に示す．

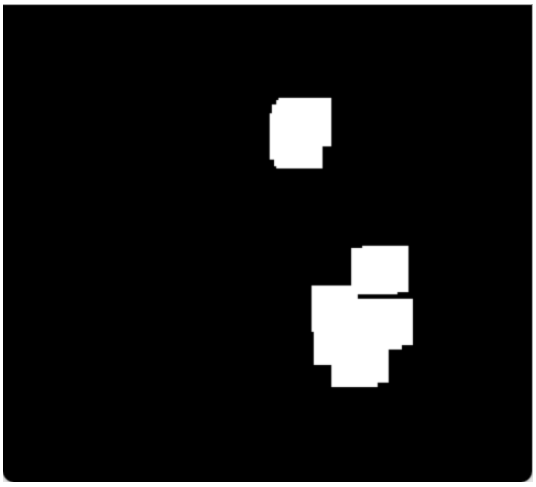
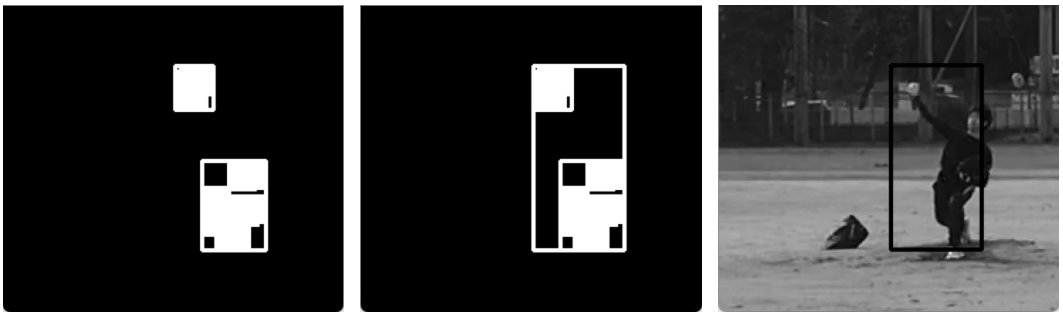


図 3.9: 収縮，膨張後の二値画像



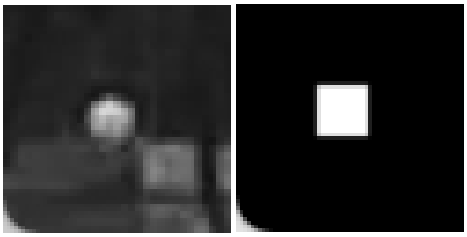
(a) 連結成分ごとの矩形を取得する．

(b) 全ての矩形を包括する矩形を求める．

(c) フレーム画像と矩形の対応

図 3.10: 投球領域矩形の取得

(a) 処理画像



(b) 投球矩形の左上のボール画像　(c) ボールの二値化後画像

図 3.11: リリース検出時の画像

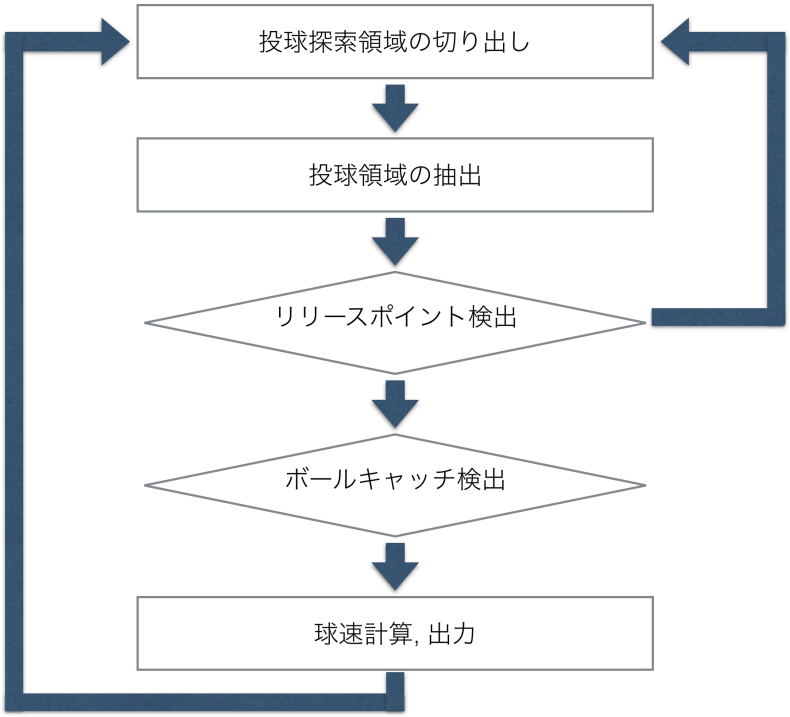


図 3.12: 球速測定の流れ

第4章　球速測定手法のスマートフォンへの実装

本章では，3 章で述べた手法をシステムに実装する方法について説明する．ソースコードの一部を使った説明もおこなうが，ソースコード全文は付録として掲載する．説明のために，変数名等は，ソースコード全文とは異なる名前に変更しているものもある．

4.1　開発プラットフォーム

アプリケーションの開発プラットフォームには Apple 社製 MacBook (Retina，12-inch, Early 2015，1.2GHz，RAM 8GB) を用いた．

使用したソフトウェアは表 4.1 のとおりである．

表 4.1: 開発に使用したソフトウェア

ソフトウェア	バージョン	配布元
Xcode	8.2	https://developer.apple.com/jp/xcode//
OpenCV	3.1.0	http://opencv.org/downloads.html
CocoaPods		http://cocoapods.org

Xcode[15] は Apple 社が提供するエディタ，ライブラリ，コンパイラ，デバッガ，シミュレータが統合された開発環境である．macOS，iOS，watchOS 等 のアプリケーションを開発できる．

Xcode にも Apple 独自の画像処理ライブラリが付属する．しかし，本研究では Apple ライブラリの利用を内蔵カメラからの画像入力，処理中・処理後のディスプレイへの画像出力を主とし，コアとなる画像解析は OpenCV[16] のフレームワークを使用した．OpenCV は macOS 以外に Windows や Linux の上でも広く使われ，実績もあり，Android 端末への実装も可能である．

iOS アプリケーションの開発言語は swift であり，一方で OpenCV の開発言語は C++ で記述されているため，開発言語が異なる．このままでは iPhone のカメラで取得した画像を OpenCV で解析することができない．しかし，CocoaPods[17] を使用することで，Objective-C の文脈から OpenCV の C++ ライブラリを自由に呼び出せるようになる．CocoaPods を使用した iOS アプリケーションへの OpenCV の導入方法は付録にて紹介する．

2017 年 2 月現在，swift の文脈から OpenCV のコードを直接呼び出すことはできない．しかし，Objective-C のコードであれば swift から呼び出すことができる．Objective-C で OpenCV を呼び出すクラスを作成し，swift のコードに追加することで，swift からも OpenCV を利用可能となる．

Xcode，OpenCV，CocoaPods はいずれもフリーなソフトウェアであり，インターネットからダウンロードし，インストール，実行することができる．

一般に，iPhone アプリケーションの開発は，

1. macOS 上の Xcode でプログラム開発をおこない，
2. その動作を Xcode のデバッガ，シミュレータで確認後，
3. 開発したプログラムを iPhone へ転送する

という手順になる．本研究もこれにしたがう．

4.2 実装のターゲット

本研究では，iPhone6s をターゲットとして，システムの実装をおこなった．

2017 年 1 月の株式会社ウェブレッジによる調査 [18] ではスマートフォン市場における iPhone のシェア率が 60% を超えているという結果がある．本論文で使用した，iPhone6s(2015 年 9 月発売) のハードウェアスペックを表 4.2 に示す．CPU クロック等，一部のデータは未公表であり，スマートフォンのスペックの調査をおこなっている専門誌 [19] による推定値（*で表示）を引用している．

比較のため，iPhone7，Android 端末 Xperia Z3 のハードウェアスペックも示している．

iPhone6s において本手法を実装することができれば，iPhone7 や Android 機種など，他のスマートフォンにおいても同手法を実装できる可能性が高い．

表 4.2: iPhone6s，iPhone7，Xperia Z3 Compact スペック比較（*は推定値）

端末名	iPhone6s	iPhone7	Xperia Z3
CPU クロック	1.84 GHz*	2.34GHz*	2.5GHz
メモリ	2GB*	2GB*	3GB
動画撮影	2160p/30fps, 1080p/120fps, 720p/240fps	2160p/30fps, 1080p/120fps, 720p/240fps	2160p/30fps, 1080p/60fps, 720p/120fps

4.3 加速度センサによる距離測定機能

本手法ではスマートフォンに搭載されている加速度センサの値を使用して距離の測定をおこなう．swift で加速度センサを扱うために，CoreMotion フレームワークを使用する．加速度センサを有効にするには，次のようにインスタンスの CMMotionManager を生成し，値の取得を開始する．値の更新周期は devicemotionUpdateInterval に設定する．

```
myMotionManager = CMMotionManager()
myMotionManager.deviceMotionUpdateInterval = 0.1
myMotionManager.startDeviceMotionUpdates()
```

iPhone の姿勢のオイラー角 (ロール角，ピッチ角，ヨー角) は，次のようにして取り出すことができる．

```
roll = motionData.attitude.roll
pitch = motionData.attitude.pitch
yaw = motionData.attitude.yaw
```

ここで取得したセンサの値にはブレがあるため，1 周期前の加速度値を pastRoll，pastPitch，pastYaw に保存しておき，ノイズ除去のためにローパスフィルタをかけた値 currentRoll，currentPitch，currentYaw を測定に使用する．

```
currentRoll = pastRoll * 0.9 + motionData.attitude.roll * 0.1
currentPitch = pastPitch * 0.9 + motionData.attitude.pitch * 0.1
currentYaw = pastYaw * 0.9 + motionData.attitude.yaw * 0.1
```

4.4 画像処理の実装

測定中の人の手による操作をなくすために，画像処理によるリリースポイントの検出をおこない，リリースが検出されたことを測定開始の合図とする．iPhone6s は，表 4.2 に示したカメラ性能を持つが，投球後 0.1 秒以内に計測結果が表示できることを本研究のリアルタイムの目標とし，その目標時間内に画像処理をおこなうことができるフォーマットとして，フレーム画像の解像度を 1920 × 1080 [pixel]，最大フレームレートを 60fps に設定した．

カメラからの画像の取得には，AVFoundation フレームワークを使用する．カメラから取得した毎フレームの画像に対して画像処理をおこなうために， フレームのキャプチャーデータを保持している CMSampleBuffer を，swift で画像処理をおこなえる UIImage 型に変換する必要がある．そのために CMSampleBuffer を UIImage 型に変換する関数 imageFromSampleBuffer を作成し，以下のように実行する．

```
let uiimage = self.imageFromSampleBuffer(sampleBuffer)
```

これによって，UIImage 型のフレーム画像を取得できる．次にこの画像を OpenCV に渡すのだが，OpenCV での画像処理は cv::Mat 型の画像に対して処理をおこなうため，以下のように UIImage 型の画像を cv::Mat 型に変換する．

```
cv::Mat matImage
UIImageToMat(uiimage, matImage);
```

これによって，OpenCV に画像を渡して画像処理をおこなうための準備が完了した．

4.4.1 リリースポイント検出

ここからは，リリースポイントの検出処理の実装について説明していく．まず最初に，投球探索領域の切り出しをおこなう．imageFromSmpleBuffer によって得られた UIImage 型の画像 をピッチャーとの距離によって設定された大きさに切り取る．切り取る座標，大きさは cropRect に格納する．

```
let uiimage = self.imageFromSampleBuffer(sampleBuffer)
let cropImage: UIImage = clipImage(uiimage, rect: cropRect)
```

次に投球領域の抽出をおこなう．ここからは OpenCV に画像を渡して処理するため，UIImageToMat によって cv::Mat 型に変換する．処理時間の短縮のために，フレーム間差分処理をおこなう前にフレーム画像をグレースケールに変換しておく．

```
UIImageToMat(cropImage, matImage);
cv::Mat grayMat1;
cv::cvtColor(matImage, grayMat1, cv::COLOR_BGR2GRAY);
```

フレーム画像が 3 枚得られたらフレーム間差分処理を開始する．フレーム 1 と フレーム 2，フレーム 2 と フレーム 3 でそれぞれフレーム間差分処理をおこない，得られた 2 枚の差分画像の論理積をとり，処理画像を diffMat とする．

```
cv::absdiff(grayMat1, grayMat2, sub12);
cv::absdiff(grayMat2, grayMat3, sub23);
cv::bitwise_and(sub12, sub23, diffMat);
```

diffMat を二値化し，平滑化する．ここでは収縮処理を 1 回おこなってノイズを除去し，その後膨張処理を 10 回おこなうことによって投球シルエットを強調する．これによって次のラベリング処理に必要な二値画像が得られる．

```
cv::adaptiveThreshold(diffMat, diffMat, 255,
    cv::ADAPTIVE_THRESH_GAUSSIAN_C, cv::THRESH_BINARY_INV, 9, 2);
cv::erode(diffMat, diffMat, cv::Mat(), cv::Point(-1,-1), 1);
cv::dilate(diffMat, diffMat, cv::Mat(), cv::Point(-1,-1), 10);
```

投球シルエットが強調された二値画像に対して，白の連結成分の面積と外接矩形を取得する．ここではラベリングを実行する．ラベリングをおこなうこと自体は本研究には関係ないが，ラベリングをおこなう際に面積値と外接矩形を取得することができるのでそれを利用する．得られた外接矩形を包括する矩形 rect2 を投球領域とする．

```
cv::Mat LabelImg;
cv::Mat stats;
cv::Mat centroids;//重心(x,y) (CV_64FC1)
int nLab = cv::connectedComponentsWithStats(src, LabelImg, stats,
    centroids);
cv::Mat labelImage(src.size(), CV_32S);
cv::connectedComponents(src, labelImage, 8);
cv::Mat dst(src.size(), CV_8UC3);

CvRect rect1;
```

```
CvRect rect2;

for (int i = 1; i < nLab; ++i) {
        int *param = stats.ptr<int>(i);
        areaLabeling += param[cv::ConnectedComponentsTypes::CC_STAT_AREA];

        int x = param[cv::ConnectedComponentsTypes::CC_STAT_LEFT];
        int y = param[cv::ConnectedComponentsTypes::CC_STAT_TOP];
        int height = param[cv::ConnectedComponentsTypes::CC_STAT_HEIGHT];
        int width = param[cv::ConnectedComponentsTypes::CC_STAT_WIDTH];

        rect1 = cvRect(x, y, width, height);
        if(i == 1){
                rect2 = rect1;
        }else{
                rect2 = cvMaxRect(&rect1, &rect2);
        }
}
```

次にボールの探索をおこなう．ボールの探索は毎回はおこなわず，投球領域の面積が設定した最小値以上でかつ白の領域の面積が投球領域の面積の 4 分の 1 以上の時のみボールの検出をおこなう．投球領域の面積の最小値は測定者からピッチャーまでの距離によって表 4.3 のように設定した．

表 4.3: ボール探索をおこなう投球領域面積の最小値

距離 [m]	面積の最小値 [pixel]
20 - 25	8000
15 - 20	12000
15 以下	22000

ボールの検出は，二値画像から白のピクセルの集まりを cv::findContours 関数によって検出することによっておこなう．ボール探索矩形の中から白いボールを検出するために，輝度値による二値化をおこなう．この時の閾値は，輝度値の最大を 255，最小を 0 とした時，150 の輝度値を設定する．輝度値が 150 を超えるピクセルは輝度値を 255(黒)，150 以下のピクセルは輝度値を 0(白) に設定し，二値化する．砂のグラウンドのような環境で測定をおこなう場合，背景とボールの輝度値の差が小さく，ボールを検出することが困難であった．そこで，本システムでは，閾値の調整機能として，タップした点からその

場所の輝度値を取得し，閾値とする機能を実装した．タップした座標を取得し，その座標から 20 × 20[pixel] の大きさの矩形を取り出し，その中の最大の輝度値を閾値とする．

4.5 ボールキャッチ音の検出

ボールキャッチの際には「パン」というボールとミットがぶつかる音がする．その音を iPhone のマイクで取得し，ボールキャッチの合図とする．swift で音を検知するために，AudioToolBox フレームワークを使用する．マイクからの音のレベルは -0.10 を最大とする負の値で表現され，最大値を取得する mPeakPower と，平均値を取得する mAveragePower の 2 種類の取得方法がある．本研究ではキャッチの瞬間の音を検出するために mPeakPower の値を使用した．ボールキャッチをマイクで検知する実験において，キャッチ音の mPeakPower の値は全て -4.0 より大きい値であったため，mPeakPower の値が -4.0 を超えた瞬間をボールキャッチの時間と判断し，リリースの瞬間からキャッチまでの時間を得る．以上で球速測定のすべての処理を実装できた．

```
var levelMeter = AudioQueueLevelMeterState()
var currentPowerLevel = levelMeter.mAveragePower
```

4.6 球速の出力方法

本システムでは，スマートフォンのディスプレイへの出力のみではなく，音声による出力もおこなう．これによって測定者がスマートフォンの画面を常に監視して球速の確認をおこなう必要がなくなる．音声の読み上げには AVSpeechSynthesizer を使用する．読み上げるテキストとして，String 型にした球速を AVSpeechUtterance でセットし，speakUtterance で読み上げをおこなう．

```
let speedText : String = String(format:"%.0f キロ", speed)
let synthesizer = AVSpeechSynthesizer()
let utterance = AVSpeechUtterance(string: speedText)
synthesizer.speakUtterance(utterance)
```

4.7 アプリケーションを使用した球速測定手順

iPhone6s に実装したアプリケーションの実行画面を図 4.1 に示す．距離測定のためのボタンを右下に配置している．上から順に，観測地点からキャッチャーまでの距離を取得する 1.Set Catcher Position ボタン，観測地点からピッチャーまでの距離を取得する 2.Set Pitcher Position ボタン，取得した距離をリセットする Reset ボタンを配置した．

画面の右上には測定モードの ON/OFF スイッチを配置した．測定モードが on になっている間，球速測定の処理を実行し続ける．

画面の左上にはボールと背景の色が近い時に使用する GroundColor スイッチを配置した．GroundColor スイッチが ON になっている時はタッチした座標の周り 20×20[pixel] の領域の最大の輝度値を閾値に設定する．GroundColor スイッチが OFF の時はデフォルトの閾値 150 が設定される．

画面左下には distanceCP ラベルと distanceLabel を配置した．distanceCP ラベルには，キャッチャーまでの距離を測定していない時には “Set Catcher Position!”，キャッチャーまでの距離を測定した後は “Set Pitcher Position!”，ピッチャーまでの距離を測定した後はキャッチャーからピッチャーまでの距離 distanceCP が表示される．distanceLabel には観測地点から画面の中央に写っている地点までの距離が表示される．

開発したアプリケーションを使って球速測定をおこなう手順は，以下のようになる．

1. キャッチャーの足元の接地点がアプリケーション実行画面の中央に来るようにスマートフォンを傾け，1.Set Catcher Position ボタンをタップし， 観測地点からキャッチャーまでの距離を測定する．

2. ピッチャーの足元の接地点がアプリケーション実行画面の中央に来るようにスマートフォンを傾け，2.Set Pitcher Position ボタンをタップし，観測地点からピッチャーまでの距離を測定する．

3. ピッチャーまでの距離を測定した時のままの状態で，測定モードのスイッチを ON にする．

測定モードを ON にしている間は常にリリースポイントの検出がおこなわれ，検出されたことが球速測定開始の合図になっているので，測定中にアプリケーションの操作を

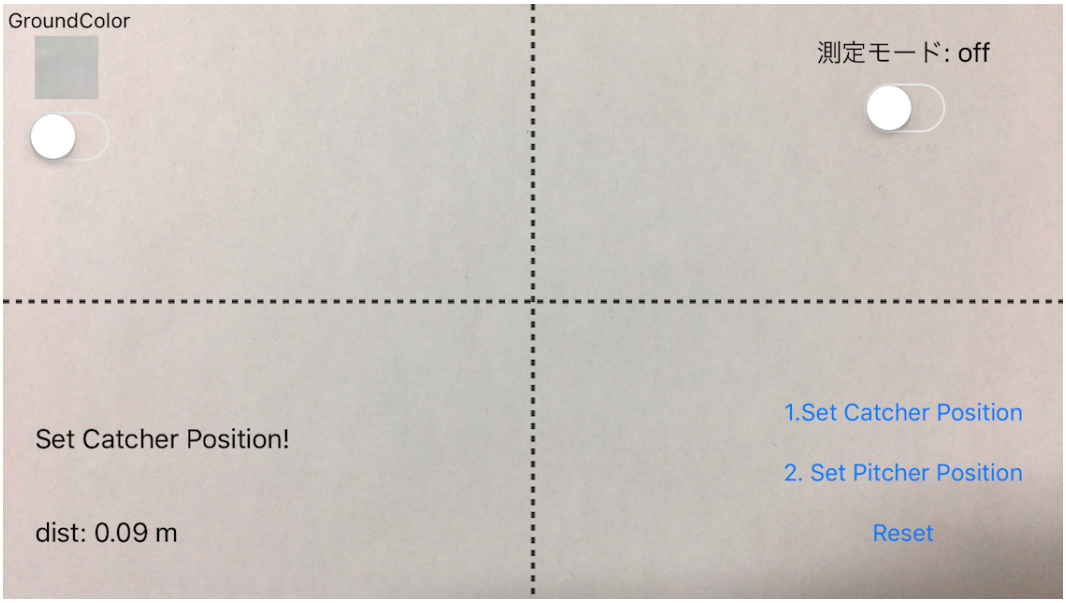


図 4.1: アプリケーションの実行画面

おこなうことはない．そのため，スマートフォンを手に持って操作する必要はなく，三脚でスマートフォンを固定することを想定している．

前研究で球速測定をおこなう際にはピッチャーからバッターまでの距離をメジャーで測定し，ピッチャーとバッターがマーカー内に収まる画角に測定者が移動して実験をおこない，観測者はピッチャーの投球動作に合わせて測定開始ボタンを押す操作を投球毎におこなう必要があった．

それに対して，本システムでは，距離の測定をスマートフォンを動かすだけでおこなうことができ，測定準備のために観測者が移動することはない．投球の開始は画像処理によって自動的に検出するため，測定モードを ON にする以外の操作を測定中におこなう必要もない．測定のための準備に関しても，測定中の操作に関しても，本システムが利便性を大きく向上させることは明らかである．

第5章　実験と結果

本章では，実装したアプリケーションを使用して，実際のピッチャーの投球に対しておこなった球速の測定実験について説明する．

5.1　距離測定実験

提案手法の球速測定の計算には，加速度センサを利用して測定した距離を使用する．そのため，距離の測定実験をおこない，どのくらいの測定誤差が出るのかを調査する．

加速度センサを利用した距離測定を 1 メートルから 20 メートルまでの距離でおこない，メジャーで測った実際の距離と比較した．測定は，高さ 1.65 メートルの位置にレンズが来るように三脚に iPhone を固定しておこなった．距離測定実験の結果を表 5.1 に示す．

誤差の平均は 0.4 m，誤差の標準偏差は 0.09 と，ばらつきの小さい測定結果を得ることができた．20m 以内の距離測定においては，誤差率 10 % 以内で測定をおこなうことができている．球速は，移動距離に比例するため，距離測定における 10 % 以内の誤差は，球速の結果においても 10 % 以内の誤差になると考えられる．

5.2　球速測定実験

提案手法を実装したスピードガンアプリケーションの球速測定の精度を検証するために，ドップラー方式のスピードガンによる球速測定との比較実験をおこなった．

実験は曇りの日のグラウンドにて，キャッチャーの後方の位置でスマートフォンを固定し，ピッチャーが連続して投げた 30 球のボールの球速測定を，開発したアプリケーションによっておこなった．同時にドップラー方式のスピードガン Bushnell スピードスター V[20] を使った球速の測定をおこない，結果を比較する．Bushnell スピートスター V のスペックを表 5.2 に示す．

表 5.1: 距離測定実験の結果

実際の対象物までの距離 [m]	加速度センサを利用した測定値 [m]	誤差 [m]	誤差率 [%]
1.00	0.94	-0.06	6.0
2.00	1.97	-0.03	1.5
3.00	2.96	-0.04	1.3
4.00	4.02	+0.02	0.5
5.00	5.05	+0.05	1.0
6.00	6.10	+0.10	1.7
7.00	7.11	+0.11	1.6
8.00	8.13	+0.13	1.6
9.00	9.26	+0.26	2.8
10.00	10.26	+0.26	2.6
11.00	11.32	+0.32	2.9
12.00	12.86	+0.86	7.2
13.00	13.92	+0.92	7.1
14.00	14.56	+0.56	4.0
15.00	15.52	+0.52	3.5
16.00	16.40	+0.40	2.5
17.00	17.70	+0.70	4.1
18.00	18.90	+0.90	5.0
19.00	19.78	+0.78	4.1
20.00	21.50	+1.50	7.5

5.3 結果と考察

ドップラー方式のスピードガンでは，ボタンを押している間，電磁波を出し続けて，速度の測定をおこない，ボタンを押している間に測定された速度の中での最大の速度，すなわち初速度を出力するのに対して，開発したアプリケーションが出力するのはピッチャーの手からボールが離れてからボールがキャッチャーのミットに届くまでの平均速度である．そのため，得られる測定結果は開発アプリケーションの方が遅くなる．永田らの研究 [21] によれば，キャッチャーミットにボールが到達する直前の速度である終速度は，ボールリリース直後の速度である初速度に比べ，一般的に 8 km/h から 12 km/h 程度遅くなる．そのため，初速度と平均速度を比較した場合，4 km/h から 6 km/h 程度平均速度の方が遅い数値になると予想される．

開発したアプリケーションによる測定結果とドップラー方式のスピードガンによる測定結果の比較を表 5.3 に示す．

表 5.2: Bushnell スピードスター V のスペック

ボールの測定可能速度	時速 16 − 177km
測定精度 ※ 正面測定の場合	± 時速 1 km
測定精度 ※ 角度がある場合	± 時速 2km 以上
サイズ	幅 109 ×奥行 152 ×高さ 213mm
重量	539 g

処理時間に関しては，カメラからの画像は最大 60fps で取得しており，毎フレームでの球速測定処理は次のフレームまでの時間内に終えることができている．体感としては，ボールのキャッチとほぼ同時に球速の出力ができており，本論文で目標の一つとしてあげていた, リアルタイム性に関しては達成できた．

測定結果に関しては，初速度と平均速度の違いから，全体的に開発アプリケーションの方がドップラー方式のスピードガンに比べて遅い球速が得られた．ドップラー方式のスピードガンによる測定においても $\pm 1km/h$ 程度の測定誤差があり，距離測定の際にも誤差があることも考慮すると，今回の実験では球速の測定結果としておおよそ妥当と言える結果を得ることができた．

連続 30 球の測定において，開発アプリケーションによる測定では 10 回，スピードガンでは 4 回計測結果が得られなかった．ドップラー方式のスピードガンでは，電磁波の跳ね返りを受け取れない場合に計測結果を得ることができないが，開発アプリケーションでは，ボールキャッチの音が小さくマイクが音を検知できなかった場合や，リリースポイントの検出が正しくおこなえていない場合に，遅すぎる，または速すぎる球速を出力してしまい，正しい計測結果が得ることができなかった．リリースポイントの検出が正しくおこなえていないのは，次のような場合である．

- 風などによって探索領域内においてピッチャー以外の背景に動きがある場合
- 太陽の光の当たり方によってボール以外に白い物体が検出される場合
- スマートフォンの揺れによって画像全体に動きが生じた場合

また，測定中に強い風が吹いている間はスマートフォンが揺れてしまうため，球速の測定をおこなうことがほとんどできず，本アプリケーションは風に弱いということがわかっている．今回の実験結果は強い風が吹いていない場面における測定によって得られたものである．本手法ではフレーム間差分によるピッチャーの投球領域検出をおこなっ

ているため，風による揺れがスマートフォンに少しでも伝わると画像全体に動きが生じてしまう．風が吹いている間は正しい計測結果を得ることができなかった．本手法はカメラを固定することを前提としているため，画像全体が動くことによって全体にノイズが生じた場合，ノイズ全体を含む矩形が投球領域と判断され，ボールの検出をおこなう．もし，ボール検出矩形内に白っぽく写るものが見つかった場合は，ボールキャッチ検出が始まる．このような状態になった場合は，ボールキャッチを検出するまでボールキャッチ検出処理をし続けるので，拍手をするなどして，キャッチ音をマイクに聞かせることでリリースポイント検出に戻すことが可能である．

前研究の横からの動画撮影による球速測定では，$\pm 4km/h$ 未満の精度で測定をおこなうことができていたので，それと比較すると測定の精度は高くない．しかし，そのことを考慮しても使いたくなるだけの利便性を本アプリケーションでは実現している．距離の測定はメジャー等を使用せずに，その場でスマートフォンを動かすだけでおこなうことができ，測定モードのスイッチを ON にするだけで，球速の測定，結果の出力までを全て自動でおこなう．音声で結果を出力することで，測定者が画面を見ている必要もない．前研究では追跡することができなかった白っぽい砂のグラウンドでの測定をおこなうための機能も実装した．

本研究では，測定精度よりも測定の利便性に重きを置いて開発をおこなったため，リアルタイム性を重視し，簡易的な画像処理によるリリース検出をおこなっているが，今後の研究ではリアルタイム性を維持しつつリリースポイントの検出精度の向上にも取り組む必要がある．測定精度についてはまだまだ改善が必要であるが，本研究の目標であった，キャッチャーの後方から測定すること，測定中の人の操作をなくすこと，リアルタイムに球速を出力することを達成でき，スマートフォンを用いた球速測定アプリケーションの利便性を大きく向上した．

表 5.3: 開発したアプリケーションとドップラー方式のスピードガンの測定結果の比較 (—では測定結果が得られなかった．)

開発したアプリケーションによる測定結果 $[km/h]$	スピードスター V による測定結果 $[km/h]$	誤差 $[km/h]$	誤差率 [%]
93	102	-9	8.8
101	99	+2	2.0
97	104	-7	6.7
—	101	—	—
100	101	-1	1.0
88	89	-1	1.1
—	73	—	—
77	80	-3	3.8
—	74	—	—
—	80	—	—
—	83	—	—
78	—	—	—
—	94	—	—
85	—	—	—
81	—	—	—
84	84	0	0.0
—	94	—	—
78	—	—	—
—	101	—	—
96	102	-6	5.9
84	84	0	0.0
75	82	-7	8.5
85	85	0	0.0
85	79	+6	7.6
71	79	-8	10.1
—	80	—	—
—	81	—	—
85	86	-1	1.2
81	86	-5	5.8
81	85	-4	4.7

第6章　まとめ

野球の球速の測定が一般的でない理由の一つとしてコストの問題があったが，近年広く普及しているスマートフォン上で動作するスピードガンアプリケーションとして実装することで，少年野球や高校野球などのアマチュア野球の場面でも球速の測定，比較を低コストでおこなうことができる．特に本研究では，横からの動画撮影による球速測定をおこなう前研究に比べて，野球の練習の場面においてより手軽に利用できることを研究の目的とし，次の 3 つの項目を達成目標として開発をおこなった．

1. キャッチャーの後ろから測定できる．
2. 球速測定中の人の操作をなくす．
3. リアルタイムに球速を出力できる．

本論文では，キャッチャーの後方から撮影した投球画像に対して画像処理をおこなうことによって，野球のピッチャーの投球シーンからリリースポイントを検出し，ボールキャッチの音を検出するまでの時間から速度を自動推定する手法を提案した．キャッチャーの後方から測定をおこなう上で困難であったピッチャーまでの距離の測定は，加速度センサを用いておこなった．キャッチャーの完全に真後ろからではなくてもピッチャーの足元の接地点を画面の中央に写すことができれば，ピッチャーまでの距離を測定することができ，球速の測定をおこなうことができる．

投球の開始から終了までを自動で検出するため，測定時の人の操作はなく，音声によるフィードバックを採用することで，画面を見て出力を確認する必要もない．測定者なしでの測定もおこなうことができるため，壁当てのように一人でおこなう練習の場面においても利用可能である．

本研究では，カメラのフォーマットとして 60fps の動画撮影を採用し，次のフレームを取得するまでの時間内に球速測定の処理を全ておこない，リアルタイムに球速を出力することができた．

球速の測定精度は, 横から撮影した動画からの球速測定に比べて低い結果となったが, 測定目標としていた 3 つの項目全てを達成することができ，スマートフォンを用いた球速測定の利便性を大きく向上することができた.

キャッチャーの後方からの動画を使用する本手法は，投球フォームの解析など，球速測定以外の場面でも応用が期待できる．球速測定の精度は横からの動画を使用した時に比べ低いため，リリースポイントの検出精度の向上や測定環境で起こりうるノイズへの対策をしていくことで，測定精度を向上させていくことが今後の課題である.

また，本手法はリリースポイントの探索範囲を変更することで，テニスのサーブやソフトボールの投球のような他のスポーツにも応用が可能である.

謝辞

修士論文を完成するにあたり，ご指導ご教授くださりました三浦元喜准教授に御礼申し上げます．卒業研究の際にご指導くださり，学会発表の経験のなかった私に国際学会での発表の機会を与えてくださった木村広准教授に御礼申し上げます．また，輪講や中間発表においてご指導やご教授を下さりました情報セクションの先生方に御礼申しあげます．加えて，本論文のデータ収集実験や測定実験において，実験に協力していただきました伊藤学歩さん，木村哦竜さん，前田祐樹さん，松井政憲さん，サーベイ輪講や日々の研究活動などで共に勉学に励んでくださった三浦研究室，情報セクションの皆様にお礼を述べたいと思います．最後に，私の意思を尊重してくださり大学院進学を応援して頂き，経済面や生活面において，ご支援をして頂いた家族に心から感謝申し上げます．

発表論文リスト

- Yosuke Yamaguchi, Motoki Miura: Real-time Analysis of Baseball Pitching using Image Processing on Smartphone, Proceedings of 20th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES2016), Vol. 96, York, UK, pp. 1059-1066, September 2016.
- 山口 陽介，三浦 元喜：携帯端末の画像処理を用いた野球の球速のリアルタイム推定，情報処理学会インタラクション 2016，東京，pp. 692-693，2016 年 3 月.
- 山口 陽介，三浦 元喜：携帯端末の画像処理による微小オブジェクトの追跡，第 23 回インタラクティブシステムとソフトウェアに関するワークショップ, 日本ソフトウェア科学会 WISS 2015，日出，大分，pp. 135-136，2015 年 12 月.
- Yosuke Yamaguchi, Keiichi Uehara, and Hiroshi Kimura: Tracking small object by image processing on smart phone, 2015 International Conference on Informatics, Electronics & Vision (ICIEV), Fukuoka, pp. 1-4, June 2015.

参考文献

[1] Yosuke Yamaguchi, Keiichi Uehara, and Hiroshi Kimura. Tracking small object by image processing on smart phone. In *Informatics, Electronics & Vision (ICIEV), 2015 International Conference on*, pp. 1–4, 2015.

[2] Yosuke Yamaguchi and Motoki Miura. Real-time analysis of baseball pitching using image processing on smartphone. In *Proceedings of 20th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES2016), Vol. 96*, pp. 1059–1066, 2016.

[3] 蔭山雅洋, 岩本峰明, 杉山敬, 水谷未来, 金久博昭, 前田明. 大学野球投手における体幹の伸張-短縮サイクル運動および動作が投球速度に与える影響. 体育学研究 Vol. 59, pp. 189–201, 2014.

[4] 勝亦陽一, 長谷川伸, 川上泰雄, 福永哲夫. 投球速度と筋力および筋量の関係. スポーツ科学研究 3, pp. 1–7, 2006.

[5] 株式会社トーアスポーツマシーン. Sports radar gun professional sports radar 取扱説明書 model hp-2. https://media.mizuno.com/ /media/Files/cojp/customer/faq/baseball/2zm-1050.pdf.

[6] Takanori Hashimoto, Yuko Uematsu, and Hideo Saito. Generation of see-through baseball movie from multi-camera views. In *Multimedia Signal Processing (MMSP), 2010 IEEE International Workshop on*, pp. 432–437, 2010.

[7] 子安大士, 沼田洋行, 前川仁, 永見智行, 彼末一之. 複数カメラによる実投球ボールの画像解析. FIT2011（第 10 回情報科学技術フォーラム), pp. 249–250, 2011.

[8] Christian Theobalt, Irene Albrecht, Jrg Haber, Marcus Magnor, and Hans-Peter Seidel. Pitching a baseball - tracking high-speed motion with multi-exposure images. In *ACM Transactions on Graphics*, pp. 540–547, 2004.

[9] 蔭山雅洋, 和田智仁, 前田明. 野球投手における投球数の増加が投球速度と投球動作に及ほす影響. 日本体育学会大会予稿集 第 62 回大会, p. 149, 2011.

[10] Michel Lapinski, Eric Berkson, Thomas Gill, Mike Reinold, and Joseph A. Paradiso. A distributed wearable, wireless sensor system for evaluating professional baseball pitchers and batters. In *2009 International Symposium on Wearable Computers*, pp. 131–138, 2009.

[11] KEINS. びゅん. https://itunes.apple.com/jp/app/byun/id639458550?mt=8.

[12] 斎藤健治, 仰木裕嗣, 井上伸一, 市川浩, 山岸正克, 宮地力, 高井省. 手首で計測した加速度による投球スピードの推定. 体育学研究 第 47 巻 第 1 号, pp. 41–51, 2002.

[13] uenoma. Easy speedgun. https://itunes.apple.com/jp/app/easy-speedgun/id451039156?mt=8.

[14] Smart Tools co. スピードガン：speed gun. https://play.google.com/store/apps/details?id=kr.sira.speed&hl=ja.

[15] Apple. xcode. https://developper.apple.com/xcode/.

[16] OpenCV. http://opencv.org/.

[17] Danielle Tomlinson Orta Therox Ben Asher, Dimitris Koutsogiorgas and many others. The CocoaPods Dev Team with contributions from many. cocoapods. https://cocoapods.org/.

[18] 株式会社ウェブレッジ. スマートフォン・シェアランキング (top10). https://webrage.jp/techblog/sp_share/.

[19] GSMARENA. Gsmarena.com. http://www.gsmarena.com/.

[20] Ltd. Bushnell Outdoor Products Japan Co. スピードガン スピードスター v. http://www.bushnell.jp/speedsterv.html.

[21] 永田大貴, 大石惇喜, 樫山文音, 早瀬亮, 南美穂子. 一般化加法モデルを用いたストレートにおけるコンタクト確率の解析 - ノビの正体とは？ -. SAJ2015 -スポーツアナリティクスジャパン 2015-, 2015.

付 録A　OpenCVのインストール

OpenCV のインストールは，OS X 用のパッケージ管理システム Homebrew[**?**] を利用した．

ターミナルを立ち上げ，Homebrew のコマンドを入力してインストールする．OpenCV の情報を確認するためには，ターミナルを開き，次のコマンドを実行する．

```
$ brew info opencv
```

これで最新のバージョンやライセンスなど，OpenCV に関する情報を見ることができる．OpenCV をインストールする前に，OpenCV が必要とするライブラリをインストールする必要がある．次のコマンドを実行する．

```
$ sudo port install plgconfig zlib
```

このコマンドが完了したら，OpenCV のインストールをおこなう．OpenCV のインストールには，次のコマンドを実行する．

```
$ brew install opencv
```

これで mac で OpenCV を利用する準備ができた．

付 録B　CocoaPodsのインストール

iPhone のアプリケーション開発に OpenCV を導入するために Objective-C のライブラリ管理ツール CocoaPods を利用した．CocoaPods のインストールには，Ruby 言語用のパッケージ管理システム RubyGems を利用する．ターミナルを開き，次のコマンドを実行する．

```
$ sudo gem install cocoapods
$ pod setup
```

これで，CocoaPods を利用する準備ができた．

CocoaPods を使い，iPhone のアプリケーション開発に OpenCV を導入するには次のような手順でおこなう．

1. Xcode で iOS プロジェクトを作成
2. Xcode を一旦終了
3. pod init を実行し，Podfile という名前のファイルを作成
4. Podfile に pod 'OpenCV' を記述
5. 同ディレクトリで pod install を実行
6. Xcode で .xcworkspace のファイルを開く
7. Linked Frameworks and Libraries の Pods_projectname.frameworks を削除
8. Linked Frameworks and Libraries に /Pods/OpenCV/opencv2.framework を追加
9. プロジェクトに Objective-C のクラスを追加
10. 追加した Objective-C クラスの拡張子を .m から .mm に変更

作成した Objective-C クラスに OpenCV のコードを書くことで，OpenCV のライブラリを iOS から利用できる．

付 録C　アプリケーションの全ソースコード

C.1　ViewController.swift

```
1  //
2  //  ViewController.swift
3  //  iPhoneSG6
4  //  Written in swift 2
5  //  Created by Yamaguchi Yosuke on 2017/01/19.
6  //  Copyright ^^c2^^a9 2017 Yamaguchi Yosuke. All rights reserved.
7  //
8
9  import UIKit
10 import AVFoundation
11 import AssetsLibrary
12 import CoreMotion
13 import AudioToolbox
14
15 private func AudioQueueInputCallback(
16         inUserData: UnsafeMutablePointer<Void>, inAQ: AudioQueueRef,
              inBuffer: AudioQueueBufferRef, inStartTime:
             UnsafePointer<AudioTimeStamp>, inNumberPacketDescriptions
             : UInt32, InPacketDescs: UnsafePointer<
             AudioStreamPacketDescription>){}
17
18 class ViewController: UIViewController,
       AVCaptureVideoDataOutputSampleBufferDelegate{
19         private var selectLayer:CALayer!
20         private var touchLastPoint:CGPoint!
21
22         var queue: AudioQueueRef!
23         var Timer: NSTimer!
```

```
24
25          var detector: OpenCVWrapper!
26
27          var mySession: AVCaptureSession!
28          var myCamera: AVCaptureDevice!
29          var myVideoInput: AVCaptureDeviceInput!
30          var myVideoOutput: AVCaptureVideoDataOutput!
31          var detectFlag: Bool = false
32          var isReleased: Bool = false
33          var isRecording: Bool = false
34          var isFinished: Bool = false
35          //var fileOutput: AVCaptureMovieFileOutput!
36          var myMotionManager: CMMotionManager!
37          var HEIGHT = 1.65//height of camera lenz
38          var pastPitch:Double = 0.0
39          var pastRoll:Double = 0.0
40          var pastYaw:Double = 0.0
41          var distC:Double = 0.0
42          var distP:Double = 0.0
43          var distCP:Double = 18.44//distance between Pitcher &
                Catcher
44          var pitchC:Double = 0.0
45          var cropXRate: Double = 7/16.0
46          var cropYRate: Double = 1/4.0
47          var cropWRate: Double = 2/16.0
48          var cropHRate: Double = 1/4.0
49          let orgWidth: Double = 1920.0
50          let orgHeight: Double = 1080.0
51          var cropX: Double = 0.0
52          var cropY: Double = 0.0
53          var cropW: Double = 0.0
54          var cropH: Double = 0.0
55          var tapLocation: CGPoint = CGPoint()
56          var groundX: CGFloat = 1920 / 2.0
57          var groundY: CGFloat = 1080 / 2.0 + 100
58          var gotGroundColor: Bool = false
59          var startTime: NSDate!// = NSDate()
60          var currentTime: NSDate!
```

```
61          var finishTime: NSDate!// = NSDate()
62          var takasa: Float = 1.0
63          let desiredFPS = 60.0
64          //      var track_x : CGFloat = 0.0
65          //      var track_y : CGFloat = 0.0
66          var processTime: NSDate!
67          var pastProcessTime: NSDate!
68
69          //var filePath: String!
70          @IBOutlet weak var distanceCPLabel: UILabel!
71          @IBOutlet weak var distanceLabel: UILabel!
72          @IBOutlet weak var detectModeLabel: UILabel!
73          @IBOutlet weak var myImageView: UIImageView!
74          @IBOutlet weak var speedLabel: UILabel!
75          @IBOutlet weak var groundColorView: UIImageView!
76
77  //      @IBOutlet weak var peak: UITextField!//textfield of sound
        level
78  //      @IBOutlet weak var average: UITextField!
79
80          @IBAction func intensitySwitch(sender: UISwitch) {
81                  if (sender.on ) {
82                          detector.setIntensity(true)
83                  } else {
84                          detector.setIntensity(false)
85                  }
86          }
87
88
89          @IBAction func detectSwitch(sender: UISwitch) {
90                  if ( sender.on ) {
91                          cropX = orgWidth/2
92                          cropY = orgHeight/2
93                          detectFlag = true
94                          if(distCP < 15){
95                                  cropW = 350
96                                  cropH = 350
97                          }else if(distCP < 20){
```

```
98                              cropW = 260
99                              cropH = 260
100                         }else{
101                             cropW = 210
102                             cropH = 210
103                         }
104                         /*let synthesizer = AVSpeechSynthesizer()
105                         let utterance = AVSpeechUtterance(string: "
                                オン")
106                         utterance.pitchMultiplier = takasa + 0.5
107                         synthesizer.speakUtterance(utterance)*/
108                         myMotionManager.stopDeviceMotionUpdates()
109                         self.startUpdatingVolume()
110                         detectModeLabel.text = "測定モード:␣on"
111                 } else {
112                         detectFlag = false
113                         myMotionManager.startDeviceMotionUpdates()
114                         detectModeLabel.text = "測定モード:␣off"
115                         self.stopUpdatingVolume()
116                 }
117         }
118
119         @IBAction func setCatcher(sender: AnyObject) {
120                 self.distC = self.HEIGHT * tan(self.pastRoll)
121                 self.pitchC = self.pastPitch
122                 self.distanceCPLabel.text = String(format:"Set␣
                        Pitcher␣Position!")
123                 /*let synthesizer = AVSpeechSynthesizer()
124                 let utterance = AVSpeechUtterance(string: "捕手の位
                        置を設定しました。投手の位置を設定してください")
125                 utterance.pitchMultiplier = takasa
126                 synthesizer.speakUtterance(utterance)*/
127         }
128
129         @IBAction func setPitcher(sender: AnyObject) {
130                 self.distP = self.HEIGHT * tan(self.pastRoll)
131                 self.distCP = sqrt(distC*distC + distP*distP - 2 *
                        distP * distC * cos(fabs(pitchC - self.pastPitch
```

```
                )))
132             print(self.distC)
133             print(self.pitchC)
134             print(pastYaw)
135             print(fabs(pitchC-self.pastPitch)*180/M_PI)
136             distanceCPLabel.text = String(format:"distCP: %.2f m
                    ", self.distCP)
137             detector.setDistance(distCP); //距離をセット
138             /*let synthesizer = AVSpeechSynthesizer()
139             let utterance = AVSpeechUtterance(string: "距離の設
                    定が完了しました。測定モードにしてください。")
140             utterance.pitchMultiplier = takasa
141             synthesizer.speakUtterance(utterance)*/
142         }
143         @IBAction func reset(sender: AnyObject) {
144             distC = 0.0
145             distP = 0.0
146             distCP = 0.0
147             pitchC = 0.0
148
149             self.distanceCPLabel.text = String(format:"Set
                    Catcher Position!")
150             /*let synthesizer = AVSpeechSynthesizer()
151             let utterance = AVSpeechUtterance(string: "リセット
                    しました。捕手の位置を設定してください")
152             utterance.pitchMultiplier = takasa
153             synthesizer.speakUtterance(utterance)*/
154
155         }
156
157         // make UIImage from sampleBuffer
158         func captureImage(sampleBuffer:CMSampleBufferRef) -> UIImage
                {
159
160             //  image from Sampling Buffer
161             let imageBuffer:CVImageBufferRef =
                    CMSampleBufferGetImageBuffer(sampleBuffer)!
162
```

```
163                 // lock pixel buffer base adress
164                 CVPixelBufferLockBaseAddress(imageBuffer,
                        CVPixelBufferLockFlags(rawValue: CVOptionFlags
                        (0)))
165
166                 let baseAddress:UnsafeMutablePointer<Void> =
                        CVPixelBufferGetBaseAddressOfPlane(imageBuffer,
                        0)
167
168                 let bytesPerRow:Int = CVPixelBufferGetBytesPerRow(
                        imageBuffer)
169                 let width:Int = CVPixelBufferGetWidth(imageBuffer)
170                 let height:Int = CVPixelBufferGetHeight(imageBuffer)
171
172                 let colorSpace:CGColorSpaceRef =
                        CGColorSpaceCreateDeviceRGB()
173
174                 //let bitsPerCompornent:Int = 8
175                 // swift 2
176                 let newContext:CGContextRef = CGBitmapContextCreate(
                        baseAddress, width, height, 8, bytesPerRow,
                        colorSpace, CGImageAlphaInfo.PremultipliedFirst.
                        rawValue|CGBitmapInfo.ByteOrder32Little.rawValue
                        )!
177
178                 let imageRef:CGImageRef = CGBitmapContextCreateImage
                        (newContext)!
179                 let resultImage = UIImage(CGImage: imageRef, scale:
                        1.0, orientation: UIImageOrientation.Right)
180
181                 return resultImage
182             }
183
184
185             func captureOutput(captureOutput: AVCaptureOutput!,
                    didOutputSampleBuffer sampleBuffer: CMSampleBuffer!,
                    fromConnection connection: AVCaptureConnection!) {
186                 //print("captureOutput:didOutputSampleBuffer:
```

```
        fromConnection)")
187             if connection.supportsVideoOrientation {
188                 connection.videoOrientation =
                        AVCaptureVideoOrientation.LandscapeRight
189             }
190             dispatch_async(dispatch_get_main_queue(), {
191                 let image = self.imageFromSampleBuffer(
                        sampleBuffer)
192                 if self.detectFlag {//is on detect mode
193                     if(self.isReleased == false){
194                         self.myImageView.image =
                                self.detectRelease(image)
195                     }else{
196                         self.myImageView.image =
                                image//self.trackBall(
                                image)
197                         if(self.isFinished == true){
198                             self.finishTime =
                                    NSDate()
199                             let spentTime = self
                                    .finishTime.
                                    timeIntervalSinceDate
                                    (self.startTime)
200                             let speed: Double =
                                    (self.distCP - 1)
                                     * 3.6 /
                                    spentTime
201                             let speedText :
                                    String = String(
                                    format:"%.0f km/h
                                    ", speed)
202                             //print(speedText)
203                             let speedText2 :
                                    String = String(
                                    format:"%.0f キロ
                                    ", speed)
204                             self.speedLabel.text
                                     = speedText
```

```
205                                         let synthesizer =
                                                AVSpeechSynthesizer
                                                ()
206                                         let utterance =
                                                AVSpeechUtterance
                                                (string:
                                                speedText2)
207                                         if(speed >= 100){
208                                                 utterance.
                                                        pitchMultiplier
                                                        = self.
                                                        takasa +
                                                        0.5
209                                         }else {
210                                                 utterance.
                                                        pitchMultiplier
                                                        = self.
                                                        takasa
211                                         }
212                                         synthesizer.
                                                speakUtterance(
                                                utterance)
213
214                                         self.isFinished =
                                                false
215                                         self.isReleased =
                                                false
216                                 }
217                                 /* self.processTime = NSDate
                                        ()
218                                 print(self.processTime.
                                        timeIntervalSinceDate(
                                        self.pastProcessTime))
219                                 self.pastProcessTime = self.
                                        processTime*/
220                         }
221                 } else {
222                         self.myImageView.image = image
```

```
223                                 self.groundColorView.image = self.
                                        setGroundPosition(image)
224                             }
225                         })
226             }
227
228             func imageFromSampleBuffer(sampleBuffer: CMSampleBufferRef)
                    -> UIImage {
229                     let imageBuffer: CVImageBufferRef =
                            CMSampleBufferGetImageBuffer(sampleBuffer)!
230                     CVPixelBufferLockBaseAddress(imageBuffer,
                            CVPixelBufferLockFlags(rawValue: CVOptionFlags
                            (0)))    // Lock Base Address
231                     let baseAddress = CVPixelBufferGetBaseAddressOfPlane
                            (imageBuffer, 0)  // Get Original Image
                            Information
232                     let bytesPerRow = CVPixelBufferGetBytesPerRow(
                            imageBuffer)
233                     let width = CVPixelBufferGetWidth(imageBuffer)
234                     let height = CVPixelBufferGetHeight(imageBuffer)
235
236                     let colorSpace = CGColorSpaceCreateDeviceRGB()  //
                            RGB ColorSpace
237                     let bitmapInfo = (CGBitmapInfo.ByteOrder32Little.
                            rawValue | CGImageAlphaInfo.PremultipliedFirst.
                            rawValue)
238                     let context = CGBitmapContextCreate(baseAddress,
                            width, height, 8, bytesPerRow, colorSpace,
                            bitmapInfo)
239                     let imageRef = CGBitmapContextCreateImage(context!)
                            // Create Quarts image
240
241                     CVPixelBufferUnlockBaseAddress(imageBuffer,
                            CVPixelBufferLockFlags(rawValue: CVOptionFlags
                            (0)))     // Unlock Base Address
242
243                     let resultImage: UIImage = UIImage(CGImage: imageRef
                            !)
```

```
244
245                 return resultImage
246         }
247
248
249         func prepareVideo() {
250                 //                  let desiredFPS = 60.0
251                 //                  let height:Int = 1080
252                 let width:Int32 = 1920
253                 //     Full HD (iPhone6)
254                 mySession = AVCaptureSession()
255                 //mySession.sessionPreset =
                        AVCaptureSessionPreset1920x1080
256                 let devices = AVCaptureDevice.devices()
257
258                 for device in devices {
259                         if (device.position ==
                            AVCaptureDevicePosition.Back) {
260                                 myCamera = device as!
                                    AVCaptureDevice
261                         }
262                 }
263                 do {
264                         myVideoInput = try AVCaptureDeviceInput(
                            device: myCamera)
265                         if (mySession.canAddInput(myVideoInput)) {
266                                 mySession.addInput(myVideoInput)
267                         } else {
268                                 print("cannot add input to session")
269                         }
270
271                         myVideoOutput = AVCaptureVideoDataOutput()
272                         myVideoOutput.videoSettings = [
                            kCVPixelBufferPixelFormatTypeKey : Int(
                            kCVPixelFormatType_32BGRA)]
273                         myVideoOutput.setSampleBufferDelegate(self,
                            queue:dispatch_get_main_queue())
274                         myVideoOutput.alwaysDiscardsLateVideoFrames
```

```
                    = true
275                 if (mySession.canAddOutput(myVideoOutput)) {
276                         mySession.addOutput(myVideoOutput)
277                 } else {
278                         print("cannot add output to session"
                            )
279                 }
280
281                 /* // preview background
282                 let myVideoLayer =
                        AVCaptureVideoPreviewLayer(session:
                        mySession)
283                 myVideoLayer.frame = view.bounds
284                 myVideoLayer.videoGravity =
                        AVLayerVideoGravityResizeAspectFill
285                 view.layer.insertSublayer(myVideoLayer,
                        atIndex:0)
286                 */
287             } catch let error as NSError {
288                 print("cannot use camera \(error)")
289             }
290             /*
291             // lock the device
292             // swift 2
293             let videoDevice = AVCaptureDevice.defaultDevice(
                    withMediaType: AVMediaTypeVideo)
294             let videoInput = try! AVCaptureDeviceInput.init(
                    device: videoDevice)
295             captureSession.addInput(videoInput)*/
296             //                print(videoDevice?.formats)
297             var bestFormat: AVCaptureDeviceFormat? = nil
298             var bestFrameRateRange: AVFrameRateRange? = nil
299             //                var bestPixelArea: Int32 = 0
300
301             for format in myCamera!.formats {
302                 let dims: CMVideoDimensions =
                        CMVideoFormatDescriptionGetDimensions((
                        format as AnyObject).formatDescription)
```

```
303                         //                          let pixelArea: Int32
                                = dims.width * dims.height
304                         let ranges = (format as AnyObject).
                                videoSupportedFrameRateRanges as! [
                                AVFrameRateRange];
305                         for range in ranges {
306                                 if (range.minFrameRate <= desiredFPS
                                        && desiredFPS <= range.
                                        maxFrameRate && range.
                                        maxFrameRate <= desiredFPS + 20
                                        && dims.width >= width){
307                                         bestFormat = format as?
                                                AVCaptureDeviceFormat
308                                         bestFrameRateRange = range
309                                         //
                                                bestPixelArea = pixelArea
310                                 }
311                         }
312                 }
313                 do {
314                         try myCamera!.lockForConfiguration()
315
316                         myCamera!.activeFormat = bestFormat
317                         myCamera!.activeVideoMinFrameDuration =
                                bestFrameRateRange!.minFrameDuration
318                         myCamera!.activeVideoMaxFrameDuration =
                                bestFrameRateRange!.minFrameDuration
319                 }
320                 catch let error as NSError {
321                         print(error)
322                 }
323                 print(myCamera!.activeFormat.
                        videoSupportedFrameRateRanges)
324                 print(myCamera!.activeFormat)
325                 myCamera!.unlockForConfiguration()
326         }
327
```

```
328         func detectRelease(image: UIImage) -> UIImage {
329                 print("now Detecting")
330                 //print(image.size)
331                 var rect :CGRect?
332
333                 let origin = CGPoint(x: cropX - cropW/2, y: cropY -
                        cropH)//position coordinate
334
335                 let size = CGSize(width: cropW, height: cropH)  //
                        crop size
336                 rect = CGRect(origin: origin, size: size)   //make
                        rect from origin andsize
337                 //print(rect!.size)
338                 let cropImage: UIImage = clipImage(image, rect: rect
                        )!
339                 isReleased = detector.detect(cropImage)
340
341                 if(isReleased == true){
342                         startTime = NSDate()//remember the start
                                time
343                         //                          track_x = origin.x
344                         //                          track_y = origin.y
345                         let synthesizer = AVSpeechSynthesizer()
346                         let utterance = AVSpeechUtterance(string: "は
                                い")
347                         utterance.pitchMultiplier = takasa
348                         synthesizer.speakUtterance(utterance)
349                         print("detect!release")
350                         self.speedLabel.text = "---"
351                 }
352
353                 processTime = NSDate()
354                 let spentTime = processTime.timeIntervalSinceDate(
                        pastProcessTime)
355                 print(spentTime)
356                 pastProcessTime = processTime
357
358                 return cropImage
```

```
359         }
360         /*
361         func trackBall(image: UIImage) -> UIImage {
362         print("now Tracking")
363         var rect :CGRect?
364         let searchPoint : CGPoint = detector.getSearchPoint()
365         let searchArea : CGSize = detector.getSearchArea()
366         let origin = CGPoint(x: track_x + searchPoint.x, y: track_y
                + searchPoint.y)//position coordinate
367         let size = CGSize(width: searchArea.width, height:
                searchArea.height)  //crop size
368         rect = CGRect(origin: origin, size: size)   //rect from
                origin and size
369         print(origin)
370         print(rect!.size)
371         let cropImage: UIImage = clipImage(image, rect: rect)!
372         isFinished = detector.afterDetected(cropImage)
373         if(isFinished == true){
374         finishTime = NSDate()
375         let spentTime = finishTime.timeIntervalSinceDate(startTime)
376         let speed: Double = distCP * 3.6 / spentTime
377         let speedText : String = String(format:"%.0f km/h", speed)
378         print(speedText)
379         let speedText2 : String = String(format:"%.0f キロ", speed)
380         speedLabel.text = speedText
381         let synthesizer = AVSpeechSynthesizer()
382         let utterance = AVSpeechUtterance(string: speedText2)
383         if(speed >= 100){
384         utterance.pitchMultiplier = takasa + 0.5
385         }else {
386         utterance.pitchMultiplier = takasa
387         }
388         synthesizer.speakUtterance(utterance)
389
390         isFinished = false
391         isReleased = false
392         }
393         return cropImage
```

```
394         }*/
395
396         func setGroundPosition(image: UIImage) -> UIImage {
397                 //print("Enterd setGroundPosition")
398                 var rect :CGRect?
399                 let origin = CGPoint(x: groundX, y: groundY)//
                        position coordinate
400                 let size = CGSize(width: 20, height: 20)  //crop
                        size
401                 rect = CGRect(origin: origin, size: size)
402                 let cropImage: UIImage = clipImage(image, rect: rect
                        )!
403                 detector.getGroundIntensity(cropImage)
404                 return cropImage
405         }
406
407         /**
408         crop UIImage
409
410         - parameter image:crop source image(UIImage)
411         - parameter rect:position coordinate and size (CGRect)
412         - returns: result image(UIImage)
413         */
414         func clipImage(image: UIImage?, rect: CGRect?) -> UIImage? {
415
416                 guard let originalImage = image else {
417                         return nil
418                 }
419
420                 guard let rct = rect else {
421                         return image
422                 }
423
424                 //crop cgimage from source image, and make cgimage
                        cropped by setted value
425
426                 let cripImageRef = CGImageCreateWithImageInRect(
                        originalImage.CGImage!, rct)
```

```
427
428                 guard let imgrf = cripImageRef else {
429                         return image
430                 }
431
432                 //CGImage to UIImage
433                 let crippedImage = UIImage(CGImage: imgrf)
434
435                 return crippedImage
436
437         }
438
439
440         func motionAnimation(motionData: CMDeviceMotion?, pp: Double
                    , pr: Double, py: Double, error: NSError?) {
441                 if let motion = motionData {
442                         //low pass filter
443                         self.pastPitch = pp * 0.9 + motion.attitude.
                                pitch * 0.1
444                         self.pastRoll = pr * 0.9 + motion.attitude.
                                roll * 0.1
445                         self.pastYaw = py * 0.9 + motion.attitude.
                                yaw * 0.1
446
447                         self.distanceLabel.text = String(format:"
                                dist: %.2f m", self.HEIGHT * tan(fabs(
                                self.pastRoll)))
448                         //self.distanceCPLabel.text = String(format
                                :"dist: %.2f m", self.HEIGHT * fabs(tan(
                                self.pastRoll)))
449                         if(self.distC == 0.0){
450                                 self.distanceCPLabel.text = String(
                                        format:"Set Catcher Position!")
451                         }
452                 }
453         }
454
455         func drawLine() -> UIImage {
```

```
456
457                 let size = view.bounds.size
458                 UIGraphicsBeginImageContextWithOptions(size, false,
                        1.0)
459                 let drawPath = UIBezierPath()
460
461                 drawPath.moveToPoint(CGPoint(x: self.view.frame.
                        width/2, y: 0))
462                 drawPath.addLineToPoint(CGPoint(x: self.view.frame.
                        width/2, y: self.view.frame.height))
463                 drawPath.moveToPoint(CGPoint(x: 0, y: self.view.
                        frame.height/2))
464                 drawPath.addLineToPoint(CGPoint(x: self.view.frame.
                        width, y: self.view.frame.height/2))
465                 drawPath.lineWidth = 2.0
466                 drawPath.setLineDash([4.0, 4.0], count: 2, phase:
                        0.0)//[点, 空白]
467
468                 drawPath.stroke()
469
470                 let img = UIGraphicsGetImageFromCurrentImageContext
                        ()
471                 UIGraphicsEndImageContext()
472                 return img!
473         }
474
475         func startUpdatingVolume(){
476                 var dataFormat = AudioStreamBasicDescription(
                        mSampleRate: 44100.0, mFormatID:
                        kAudioFormatLinearPCM, mFormatFlags:
                        AudioFormatFlags(kLinearPCMFormatFlagIsBigEndian
                        | kLinearPCMFormatFlagIsSignedInteger |
                        kLinearPCMFormatFlagIsPacked), mBytesPerPacket:
                        2, mFramesPerPacket: 1, mBytesPerFrame: 2,
                        mChannelsPerFrame: 1, mBitsPerChannel: 16,
                        mReserved: 0)
477
478                 var audioQueue: AudioQueueRef = nil
```

```
479                 var error = noErr
480                 error = AudioQueueNewInput(&dataFormat,
                        AudioQueueInputCallback as
                        AudioQueueInputCallback, UnsafeMutablePointer(
                        Unmanaged.passUnretained(self).toOpaque()), .None
                        , .None, 0, &audioQueue)
481                 if error == noErr {
482                         self.queue = audioQueue
483                 }
484                 AudioQueueStart(self.queue, nil)
485
486                 //enable level meter
487                 var enabledLevelMeter: UInt32 = 1
488                 AudioQueueSetProperty(self.queue,
                        kAudioQueueProperty_EnableLevelMetering, &
                        enabledLevelMeter, UInt32(sizeof(UInt32))) //
                        UInt32(MemoryLayout<UInt32>.size))
489                 self.Timer = NSTimer.scheduledTimerWithTimeInterval
                        (1/60.0, target: self, selector: #selector(
                        ViewController.detectVolume(_:)), userInfo: nil,
                        repeats: true)
490                 self.Timer.fire()
491         }
492
493         func stopUpdatingVolume(){
494                 self.Timer.invalidate()
495                 self.Timer = nil
496                 AudioQueueFlush(self.queue)
497                 AudioQueueStop(self.queue, false)
498                 AudioQueueDispose(self.queue, true)
499         }
500
501         @objc func detectVolume(timer: NSTimer) -> (Float){
502                 //get level
503                 var levelMeter = AudioQueueLevelMeterState()
504                 var propertySize = UInt32(sizeof(
                        AudioQueueLevelMeterState))
505
```

```
506                 AudioQueueGetProperty(self.queue,
                        kAudioQueueProperty_CurrentLevelMeterDB, &
                        levelMeter, &propertySize)
507                 //self.peak.text = "".stringByAppendingFormat("%.2f
                        ", levelMeter.mPeakPower)
508                 //self.average.text = "".stringByAppendingFormat
                        ("%.2f", levelMeter.mAveragePower)
509                 if(isReleased==true){
510                         if levelMeter.mPeakPower >= -4.0 {
511                                 isFinished = true
512                         }
513                 }
514                 return levelMeter.mPeakPower
515         }
516
517         override func viewDidLoad() {
518                 super.viewDidLoad()
519                 //make MotionManager
520                 myMotionManager = CMMotionManager()
521                 // set update freqency
522                 myMotionManager.deviceMotionUpdateInterval = 0.1
523                 // start getting acceleration
524
525                 let handler:CMDeviceMotionHandler = {
526                         (motionData: CMDeviceMotion?, error: NSError
                            ?) -> Void in
527                         self.motionAnimation(motionData, pp: self.
                            pastPitch, pr:self.pastRoll, py: self.
                            pastYaw, error: error)
528                 }
529
530                 myMotionManager.startDeviceMotionUpdatesToQueue(
                        NSOperationQueue.mainQueue(), withHandler:
                        handler)
531
532                 let drawImage = drawLine()
533                 let drawView = UIImageView(image: drawImage)
534                 view.addSubview(drawView)
```

```
535
536                 pastProcessTime = NSDate()
537
538                 detector = OpenCVWrapper()
539                 prepareVideo()
540                 mySession.startRunning()
541         }
542
543         // release memory
544         override func viewDidDisappear(animated: Bool) {
545                 // camera stop and release memory
546                 mySession.stopRunning()
547
548                 for output in mySession.outputs {
549                         mySession.removeOutput(output as?
                                AVCaptureOutput)
550                 }
551
552                 for input in mySession.inputs {
553                         mySession.removeInput(input as?
                                AVCaptureInput)
554                 }
555                 mySession = nil
556                 myCamera = nil
557         }
558
559         override func didReceiveMemoryWarning() {
560                 super.didReceiveMemoryWarning()
561                 print("did␣Receive␣Memory␣Warning")
562         }
563
564         override func touchesBegan(touches: Set<UITouch>, withEvent
                event: UIEvent?) {
565                 // get touch ivent
566                 let touch = touches.first
567                 // get the position coordinate of touch
568                 tapLocation = touch!.locationInView(self.view)
569                 groundX = tapLocation.x * 1920 / 667 - tapLocation.x
```

```
                    * 1920 / 667 % 1
570                 groundY = tapLocation.y * 1080 / 350 - tapLocation.y
                        * 1080 / 350 % 1
571
572                 print(tapLocation)
573         }
574
575 }
```

C.2 AppDelegate.swift

```
 1 //
 2 //  AppDelegate.swift
 3 //  iPhoneSG6
 4 //
 5 //  Created by Yamaguchi Yosuke on 2017/01/19.
 6 //  Copyright ^^c2^^a9 2017 Yamaguchi Yosuke. All rights reserved.
 7 //
 8
 9 import UIKit
10
11 @UIApplicationMain
12 class AppDelegate: UIResponder, UIApplicationDelegate {
13
14         var window: UIWindow?
15
16
17         func application(_ application: UIApplication,
               didFinishLaunchingWithOptions launchOptions: [
               UIApplicationLaunchOptionsKey: Any]?) -> Bool {
18             // Override point for customization after
                   application launch.
19             return true
20         }
21
22         func applicationWillResignActive(_ application:
               UIApplication) {
23             // Sent when the application is about to move from
```

```
                active to inactive state. This can occur for
                certain types of temporary interruptions (such as
                 an incoming phone call or SMS message) or when
                the user quits the application and it begins the
                transition to the background state.
24              // Use this method to pause ongoing tasks, disable
                timers, and invalidate graphics rendering
                callbacks. Games should use this method to pause
                the game.
25          }
26
27          func applicationDidEnterBackground(_ application:
                UIApplication) {
28              // Use this method to release shared resources, save
                 user data, invalidate timers, and store enough
                application state information to restore your
                application to its current state in case it is
                terminated later.
29              // If your application supports background execution
                , this method is called instead of
                applicationWillTerminate: when the user quits.
30          }
31
32          func applicationWillEnterForeground(_ application:
                UIApplication) {
33              // Called as part of the transition from the
                background to the active state; here you can undo
                 many of the changes made on entering the
                background.
34          }
35
36          func applicationDidBecomeActive(_ application: UIApplication
                ) {
37              // Restart any tasks that were paused (or not yet
                started) while the application was inactive. If
                the application was previously in the background,
                 optionally refresh the user interface.
38          }
```

```
39
40          func applicationWillTerminate(_ application: UIApplication)
                {
41                  // Called when the application is about to terminate
                        . Save data if appropriate. See also
                        applicationDidEnterBackground:.
42          }
43
44
45  }
```

C.3 OpenCVWrapper.h

```
1  //
2  //  OpenCVWrapper.h
3  //  RealtimeHumandetect
4  //
5  //  Created by Yamaguchi Yosuke on 2017/01/19.
6  //  Copyright ^^c2^^a9 2017 Yamaguchi Yosuke. All rights reserved.
7  //
8
9  //#ifdef __cplusplus
10 //#import <opencv2/imgcodecs/ios.h>
11 #import <Foundation/Foundation.h>
12 //#import <opencv2/opencv.hpp>
13 //#endif
14 #import <UIKit/UIKit.h>
15
16 @interface OpenCVWrapper : NSObject
17
18 // funciton to get opencv version
19 //+(NSString * ) openCVVersionString;
20 -(CGPoint)getSearchPoint;
21 -(CGSize)getSearchArea;
22 -(void)setIntensity: (bool) setintensity;
23 -(void)setDistance : (double) distCP;
24 - (bool) detect: (UIImage *) image;
25 -(void) getGroundIntensity: (UIImage *)image;
```

```
26 - (bool) afterDetected: (UIImage *) image;
27 @end
```

C.4 OpenCVWrapper.mm

```
1  //
2  //  OpenCVWrapper.m
3  //  iPhoneSG
4  //
5  //  Created by Yamaguchi Yosuke on 2017/01/19.
6  //  Copyright ^^c2^^a9 2017 Yamaguchi Yosuke. All rights reserved.
7  //
8
9  #import "OpenCVWrapper.h"
10 #import <opencv2/opencv.hpp>
11 #import <opencv2/imgcodecs/ios.h>
12 #import <iostream>
13
14 //#import <opencv2/highgui/ios.h>
15
16 using namespace std;
17
18 @implementation OpenCVWrapper
19 static double ball_roi_rate = 1/3.0;
20 int frame_cnt = 0;
21 int release_frame_number = 0;
22 bool isReleased = false;
23 bool isUseIntensity = false;
24 //bool isFinished = false;
25 int search_x = 0;
26 int search_y = 0;
27 int past_x = 0;
28 int past_y = 0;
29 int search_height = 10;
30 int search_width = 10;
31 double distanceCP = 18.44;
32 double FPS = 0.0;
33 int intensity = 150;
```

```
34 int lose_cnt = 0;
35 int minArea = 8000;
36
37 //image valiables
38 cv::Mat frame1, frame2, frame3;
39 cv::Mat gframe1, gframe2, gframe3;
40 cv::Mat sub12, sub23, diff;
41 cv::Mat dst_frame, showImg;
42 cv::Mat ball_template, frameBall, groundFrame, gframeBall;
43
44 -(void)setIntensity: (bool) setintensity{
45         isUseIntensity = setintensity;
46 }
47
48
49 -(void)setDistance : (double) distCP{
50         frame_cnt = 0;
51         distanceCP = distCP;
52         if(distCP < 15){
53                 minArea = 22000;
54         }else if(distCP < 20){
55                 minArea = 12000;
56         }else{
57                 minArea = 8000;
58         }
59 }
60
61 -(void) getGroundIntensity: (UIImage *)image{
62         intensity = 150;
63         UIImageToMat(image, groundFrame);
64         cv::Mat gray;
65         cv::cvtColor(groundFrame, gray, cv::COLOR_BGR2GRAY);
66
67         for(int y = 0 ; y < gray.rows; y++){
68                 for(int x = 0 ; x < gray.cols; x++){
69                         if(gray.data[y * gray.cols + x] > intensity
                            ){
70                                 intensity = gray.data[y * gray.cols
```

```
                                        + x];
71                              }
72                      }
73              }
74              //std::cout << "intencity: " << intensity << std::endl;
75
76              /*groundFrame.forEach<uchar>([](unsigned char &x) -> void {
77                      if(x > intensity) intensity = x;
78               });*/
79
80      }
81
82      - (bool) detect: (UIImage *) image{
83              std::cout << "fcnt:"<<frame_cnt<<std::endl;
84              frame_cnt++;
85              if(frame_cnt == 1){
86      //              isFinished = false;
87                      //UIImage to cv::Mat
88                      UIImageToMat(image, frame1);
89                      //gray scale
90                      cv::Mat gray;
91                      cv::cvtColor(frame1, gframe1, cv::COLOR_BGR2GRAY);
92              }else if(frame_cnt == 2){
93                      UIImageToMat(image, frame2);
94                      cv::cvtColor(frame2, gframe2, cv::COLOR_BGR2GRAY);
95              }else{
96                      UIImageToMat(image, frame3);
97                      cv::cvtColor(frame3, gframe3, cv::COLOR_BGR2GRAY);
98
99                      cv::absdiff(gframe1, gframe2, sub12);
100                     //sub23 = gframe2 - gframe3;
101                     cv::absdiff(gframe2, gframe3, sub23);
102                     cv::bitwise_and(sub12, sub23, diff);
103                     cv::Mat andFrame = diff.clone();
104                     cv::threshold(diff, diff, 5, 255, cv::THRESH_BINARY
                            );//binarize
105                     //cv::adaptiveThreshold(diff, diff, 255, cv::
                            ADAPTIVE_THRESH_GAUSSIAN_C, cv::THRESH_BINARY_INV
```

```
    , 9, 2);
106             cv::erode(diff, diff, cv::Mat(), cv::Point(-1,-1),
                    1);
107             cv::Mat erdimg = diff.clone();
108             cv::dilate(diff, diff, cv::Mat(), cv::Point(-1,-1),
                    10);
109             cv::Mat src = diff.clone();
110
111             cv::Mat LabelImg;//label image (CV_32SC1 or CV_16UC1
                    )
112             cv::Mat stats;//bounding box and area
113             cv::Mat centroids;//(x,y) (CV_64FC1)
114             int nLab = cv::connectedComponentsWithStats(src,
                    LabelImg, stats, centroids);
115
116
117             cv::Mat bin = src.clone();
118             //cv::threshold(src, bin, 0, 255, cv::THRESH_BINARY
                    | cv::THRESH_OTSU);
119
120             // make label image (※CV_32S or CV_16U)
121             cv::Mat labelImage(src.size(), CV_32S);
122             cv::connectedComponents(bin, labelImage, 8);
123
124             // draw result
125             cv::Mat dst(src.size(), CV_8UC3);
126
127             CvRect rect1;
128             CvRect rect2;
129             int ball_x = 0;
130             int ball_y = 0;
131             int ball_width = 0;
132             int ball_height = 0;
133             int areaLabeling = 0;
134             //set ROI
135             for (int i = 1; i < nLab; ++i) {
136                     int *param = stats.ptr<int>(i);
137                     areaLabeling += param[cv::
```

```
                    ConnectedComponentsTypes::CC_STAT_AREA];
138
139                 int x = param[cv::ConnectedComponentsTypes::
                    CC_STAT_LEFT];
140                 int y = param[cv::ConnectedComponentsTypes::
                    CC_STAT_TOP];
141                 int height = param[cv::
                    ConnectedComponentsTypes::CC_STAT_HEIGHT
                    ];
142                 int width = param[cv::
                    ConnectedComponentsTypes::CC_STAT_WIDTH];
143
144                 rect1 = cvRect(x, y, width, height);
145
146                 if(i == 1){
147                         rect2 = rect1;
148                         /*if(nLab == 2){
149                                 cv::rectangle(dst, rect2, cv
                                    ::Scalar(0, 255, 0), 2);
150                                 cv::rectangle(showImg, rect2
                                    , cv::Scalar(0, 255, 0),
                                    2);
151                         }*/for debug
152                 }else{
153                         rect2 = cvMaxRect(&rect1, &rect2);
154                         /*if(i == nLab-1){
155                                 cv::rectangle(dst, rect2, cv
                                    ::Scalar(0, 255, 0), 2);
156                                 cv::rectangle(showImg, rect2
                                    , cv::Scalar(0, 255, 0),
                                    2);
157                         }*/for debug
158                 }
159                 ball_x = rect2.x;
160                 ball_y = rect2.y;
161                 ball_width = rect2.width;
162                 ball_height = rect2.height;
163         }
```

```
164                 int area = ball_height * ball_width;
165                 //std::cout<<"area: "<<area<<std::endl;
166                 if(ball_width == 0 || ball_height == 0){
167                         ball_height = 10;
168                         ball_width = 10;
169                 }else if (ball_width > ball_height){
170                         gframe1 = gframe2.clone();
171                         gframe2 = gframe3.clone();
172                         return false;
173                 }else if(area >= minArea && (areaLabeling >= area /
                        4)){
174                         //white detect from pitching area
175                         if( 0 <= ball_x && 0 <= ball_width*
                            ball_roi_rate && ball_x + ball_width*
                            ball_roi_rate <= gframe2.cols && 0 <=
                            ball_y && 0 <= ball_width*ball_roi_rate
                            && ball_y + ball_width*ball_roi_rate <=
                            gframe2.rows) {
176                                 gframe1 = gframe2.clone();
177                                 gframe2 = gframe3.clone();
178                                 return false;
179                         }
180                         cv::Mat work_img2(gframe2, cv::Rect(ball_x,
                            ball_y, ball_width*ball_roi_rate,
                            ball_width*ball_roi_rate));
181
182                         int th = 150;
183                         if(isUseIntensity == true){
184                                 th = intensity;
185                         }else{
186                                 th = 150;
187                         }
188                         std::cout<<"intensity"<<th<<std::endl;
189
190                         cv::threshold(work_img2, work_img2, th + 1,
                            255, cv::THRESH_BINARY);
191                         cv::erode(work_img2, work_img2, cv::Mat(),
                            cv::Point(-1,-1), 1);
```

```
192                         cv::dilate(work_img2, work_img2, cv::Mat(),
                                cv::Point(-1,-1), 3);
193
194                         cv::Mat forfind = work_img2.clone();
195                         std::vector<std::vector<cv::Point>>contours;
196                         cv::Mat contourImage(forfind.size(), CV_8U,
                                cv::Scalar(255));
197                         cv::findContours(forfind, contours,
                                CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
198                         std::cout << "the number of contours: "<<
                                contours.size()<<std::endl;
199                         //cv::Mat contoursMat = contours;
200
201                         for(int i = 0; i < contours.size(); i++){
202                                 size_t count = contours[i].size();
203                                 if(count<10 || count > 70) continue;
204 /*                              std::cout << "count:" << count<< std
        ::endl;
205                                 cv::Point2d point;
206                                 cv::Moments mom = cv::moments(
                                        contours[i]);
207                                 point.x = mom.m10/mom.m00 + ball_x;
208                                 point.y = mom.m01/mom.m00 + ball_y;
209                                 //cv::circle(showImg, point, 10,
                                        CV_AA);
210                                 cv::Rect brect = cv::boundingRect(
                                        contours[i]);
211                                 //                              cv::
                                        rectangle(work_img2, brect.x,
                                        brect.y, cv::Scalar(255, 255,
                                        255), 2, CV_AA);
212                                 //ball_template = cv::Mat (gframe2,
                                        cv::Rect(ball_x+brect.x, ball_y+
                                        brect.y, brect.height, brect.
                                        width));
213                                 //std::cout << "detected !" << std::
                                        endl;
214 */                              isReleased = true;
```

```
215 /*                                  search_x = ball_x + brect.x - brect.
        width*2;
216                                     search_y = ball_y + brect.y - brect.
                                            height*2;
217                                     search_width = brect.width * 5;
218                                     search_height = brect.height * 5;
219                                     past_x = ball_x + brect.x;
220                                     past_y = ball_y + brect.y;
221 */                                  return true;
222                             }
223                     }
224                     gframe1 = gframe2.clone();
225                     gframe2 = gframe3.clone();
226             }
227             return isReleased;
228 }
229
230 -(CGPoint)getSearchPoint{
231             CGPoint p; //= cv::Point2d(search_x, search_y);
232             p.x = search_x;
233             p.y = search_y;
234             return p;
235 }
236
237 -(CGSize)getSearchArea{
238             CGSize s; //= cv::Point2d(search_x, search_y);
239             s.width = search_width;
240             s.height = search_height;
241             return s;
242 }
243
244 - (bool) afterDetected: (UIImage *) image{
245             UIImageToMat(image, frameBall);
246             cv::Mat gray;
247             cv::cvtColor(frameBall, gframeBall, cv::COLOR_BGR2GRAY);
248             //int ball_x = search_x;//search area x
249             //int ball_y = search_y;//search area y
250 //          int ball_width = search_width;
```

```
251 //        int ball_height = search_height;
252           int x_diff;
253           int y_diff;
254           //cv::Mat roiball2(frameBall, cv::Rect(ball_x, ball_y,
                  ball_width, ball_height));// x,y,w,h
255           //cv::Mat work_img2 = roiball2.clone();
256           cv::Mat work_img2 = gframeBall.clone();
257           cv::threshold(work_img2, work_img2, intensity + 1, 255, cv::
                  THRESH_BINARY);
258           cv::erode(work_img2, work_img2, cv::Mat(), cv::Point(-1,-1),
                   1);
259           cv::dilate(work_img2, work_img2, cv::Mat(), cv::Point
                  (-1,-1), 3);
260           cv::Mat forfind = work_img2.clone();
261           std::vector<std::vector<cv::Point>>contours;
262           cv::Mat contourImage(forfind.size(), CV_8U, cv::Scalar
                  (255));
263           cv::findContours(forfind, contours, CV_RETR_EXTERNAL,
                  CV_CHAIN_APPROX_NONE);
264
265           if(contours.size() == 0){
266                   if(lose_cnt == 0){
267                           lose_cnt++;
268                           return false;
269                   }else{
270                           isReleased = false;
271                           lose_cnt = 0;
272                           frame_cnt = 0;
273                           return true;
274                   }
275           }
276
277           for(int i = 0; i < contours.size(); i++){
278                   size_t count = contours[i].size();
279                   if(count<10 || count > 70) continue;
280                   std::cout << "count:" << count<< std::endl;
281                   cv::Point2d point;
282                   cv::Moments mom = cv::moments(contours[i]);
```

```
283                     point.x = mom.m10/mom.m00;// + search_x;
284                     point.y = mom.m01/mom.m00;// + search_y;
285                     //cv::circle(showImg, point, 10, CV_AA);
286                     cv::Rect brect = cv::boundingRect(contours[i]);
287                     x_diff = search_x + brect.x - past_x;
288                     y_diff = search_y + brect.y - past_y;
289                     past_x = search_x + brect.x;
290                     past_y = search_y + brect.y;
291 //                  std::cout << "detected !" << std::endl;
292
293                     search_x = search_x + brect.x - brect.width*2 +
                            x_diff;
294                     search_y = search_x + brect.y - brect.height*2 +
                            y_diff;
295                     search_width = brect.width * 5;
296                     search_height = brect.height * 5;
297             }
298
299             //std::cout<<"width"<<frameBall.rows<<"height"<<frameBall.
                    cols<<std::endl;
300             std::cout<<"nowFrame:"<<frame_cnt<< std::endl;
301
302             return false;
303 }
304
305
306 @end
```