Project Proposal – Video stabilization

聶從煊 R04631024
National Taiwan University, Department of Bio-Industrial Mechatronics Engineering

1. OBJECTIVES

我常常隨手拍一些影片記錄生活，旅遊時更會認真地拍攝記錄當下，之後會搭配音樂剪輯成一段旅遊影片。而當我在走路、跑步、騎腳踏車時，拿著手機錄影，可想而知拍攝出來的影片畫面晃動非常大，大幅降低了觀看品質。後來在我打算買 GoPro HERO5 時，瞭解到他們的運動相機設計了一項功能─影像穩定，能夠利用演算法有效消除畫面地震動、晃動，讓我對這個功能的原理非常好奇，想深入瞭解並自己實作。

2. RESEARCH METHODS

影像穩定的方法有很多種，我選擇用比較直覺的作法，如 Figure 1 所示。我們設定欲做到穩定的方向有水平、垂直與一軸的旋轉，如 Figure 3 所示。假設當前一格影像影像往下晃動時，用光流法檢測影像各 pixel 的強度隨時間變化的動態，算出影像在水平、垂直與旋轉的移動速度，然後再積分取得變化軌跡。接著使目前此格要截取的範圍也往下移動該位移量。如此對觀看者來說，影像便不會晃動，如 Figure 2 所示。然而這個方法必須犧牲影像外框部份一定的範圍量，以去除修正影像產生的黑邊。上述演算法之流程圖如 Figure 4 所示。

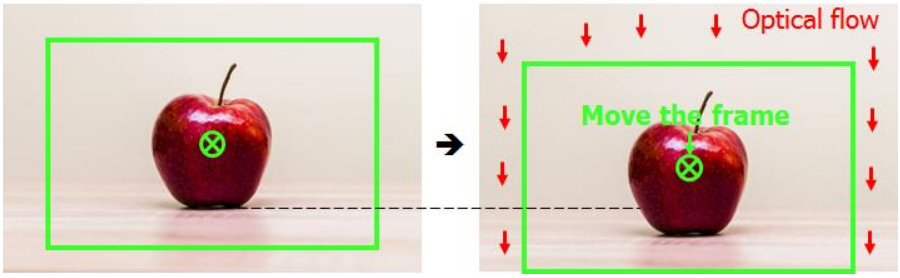

Figure 1 影像穩定方法

Figure 2 觀看者觀看到的影像

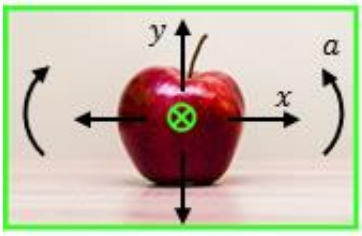

Figure 3 設定能穩定的方向

詳細演算法流程如 Figure 4 所示。第一步，取出前一格 (Frame k-1)與目前此格 (Frame k)之影像，並將彩色影像轉換至灰階以便處理。利用 opencv 函式 ”goodFeaturesToTrack”找出特徵點，並用光流法函式 “calcOpticalFlowPyrLK”得到光流向量。接著換算出水平速度 dx，垂直速度 dy 與旋轉角速度 da，即完成第一步由光流法取得影像動態。第二步，將 dx, dy, da 積分得到水平、垂直、旋轉的軌跡，並用 sliding average window 將軌跡取移動平均，得到平滑化的軌跡。第三步，將各移動平均的軌積與各原軌積相減，即得到影像應該要調整的變化量。故最後將這些變化量分別加上原 dx, dy, da，置於目前此格 (Frame k)之影像即為修正結果。

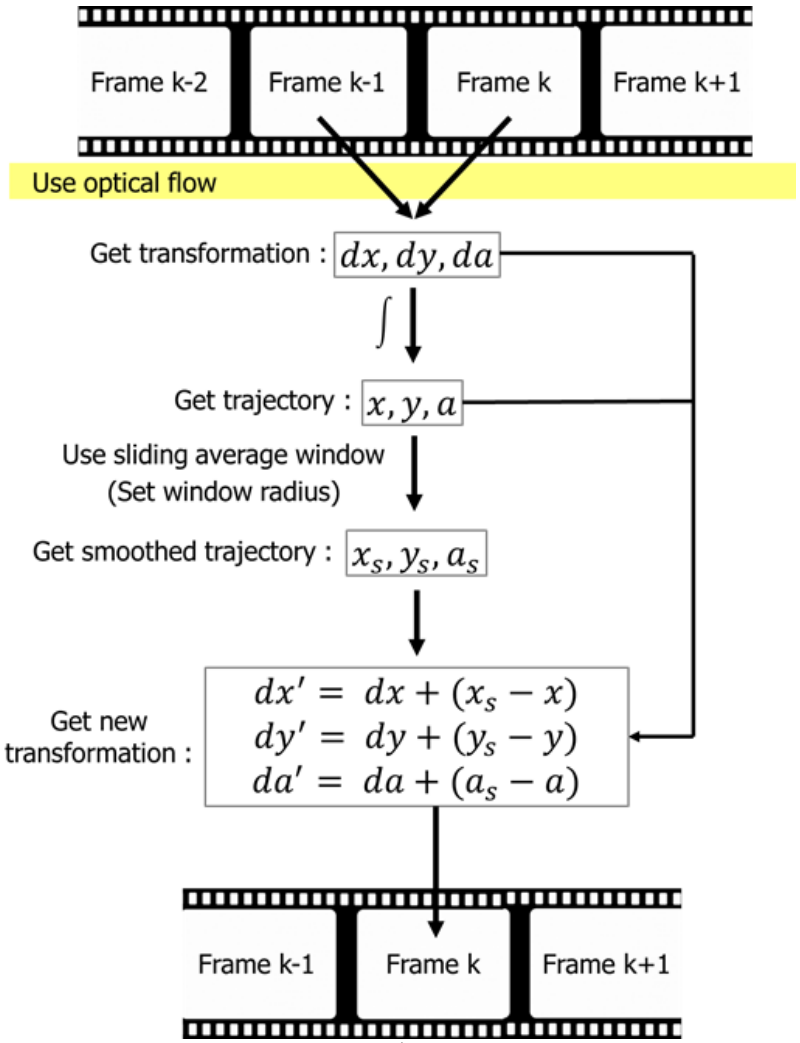

Figure 4 演算法流程圖

核心程式碼如下所示：

第一步：

```
VideoCapture cap(fileName.toStdString());  //Input the video
Mat cur, cur_grey;
Mat prev, prev_grey;

cap >> prev;
cvtColor(prev, prev_grey, COLOR_BGR2GRAY);

// Step 1 - Get previous to current frame transformation (dx, dy, da) for all frames
vector <TransformParam> prev_to_cur_transform; // previous to current

int k=1;
int max_frames = cap.get(CV_CAP_PROP_FRAME_COUNT);
Mat last_T;

while(true) {
  cap >> cur;

  if(cur.data == NULL) {
    break;
  }

  cvtColor(cur, cur_grey, COLOR_BGR2GRAY);

  // Vector from previous to current frame
  vector <Point2f> prev_corner, cur_corner;
  vector <Point2f> prev_corner2, cur_corner2;
  vector <uchar> status;
  vector <float> err;

  goodFeaturesToTrack(prev_grey, prev_corner, 200, 0.01, 30);
  calcOpticalFlowPyrLK(prev_grey, cur_grey, prev_corner, cur_corner, status, err);

  // Decompose Transformation to dx, dy, da
  double dx = T.at<double>(0,2);
  double dy = T.at<double>(1,2);
  double da = atan2(T.at<double>(1,0), T.at<double>(0,0));

 prev_to_cur_transform.push_back(TransformParam(dx, dy, da)); // Put them into vectors
```

第二步：

```
// Step 2 - Accumulate the transformations to get the image trajectory

// Accumulated frame to frame transform
double a = 0;
double x = 0;
double y = 0;

vector <Trajectory> trajectory; // trajectory at all frames

for(size_t i=0; i < prev_to_cur_transform.size(); i++) {
```

```
        x += prev_to_cur_transform[i].dx;
        y += prev_to_cur_transform[i].dy;
        a += prev_to_cur_transform[i].da;

        trajectory.push_back(Trajectory(x,y,a));


    // Smooth out the trajectory using an sliding average window
    vector <Trajectory> smoothed_trajectory; // trajectory at all frames

    for(size_t i=0; i < trajectory.size(); i++) {
        double sum_x = 0;
        double sum_y = 0;
        double sum_a = 0;
        int count = 0;

        for(int j=-SMOOTHING_RADIUS; j <= SMOOTHING_RADIUS; j++) {
            if(i+j >= 0 && i+j < trajectory.size()) {
                sum_x += trajectory[i+j].x;
                sum_y += trajectory[i+j].y;
                sum_a += trajectory[i+j].a;

                count++;
            }
        }

        double avg_a = sum_a / count;
        double avg_x = sum_x / count;
        double avg_y = sum_y / count;

        smoothed_trajectory.push_back(Trajectory(avg_x, avg_y, avg_a));
```

第三步：

```
    // Step 3 - Generate new set of previous to current transform, such that the trajectory ends up being the same as the smoothed trajectory
    vector <TransformParam> new_prev_to_cur_transform;

    // Accumulated frame to frame transform
    a = 0;
    x = 0;
    y = 0;

    for(size_t i=0; i < prev_to_cur_transform.size(); i++) {
        x += prev_to_cur_transform[i].dx;
        y += prev_to_cur_transform[i].dy;
        a += prev_to_cur_transform[i].da;

        // Target - current
        double diff_x = smoothed_trajectory[i].x - x;
        double diff_y = smoothed_trajectory[i].y - y;
        double diff_a = smoothed_trajectory[i].a - a;

        double dx = prev_to_cur_transform[i].dx + diff_x;
        double dy = prev_to_cur_transform[i].dy + diff_y;
```

```
        double da = prev_to_cur_transform[i].da + diff_a;

        new_prev_to_cur_transform.push_back(TransformParam(dx, dy, da));
```

第四步：

```
    // Step 4 - Apply the new transformation to the video
    cap.set(CV_CAP_PROP_POS_FRAMES, 0);
    Mat T(2,3,CV_64F);

    int vert_border = HORIZONTAL_BORDER_CROP * prev.rows / prev.cols; // get the aspect ratio correct

    k=0;
    while(k < max_frames-1) { // Don't process the very last frame, no valid transform
        cap >> cur;

        if(cur.data == NULL) {
            break;
        }

        T.at<double>(0,0) = cos(new_prev_to_cur_transform[k].da);
        T.at<double>(0,1) = -sin(new_prev_to_cur_transform[k].da);
        T.at<double>(1,0) = sin(new_prev_to_cur_transform[k].da);
        T.at<double>(1,1) = cos(new_prev_to_cur_transform[k].da);

        T.at<double>(0,2) = new_prev_to_cur_transform[k].dx;
        T.at<double>(1,2) = new_prev_to_cur_transform[k].dy;

        Mat cur2;

        warpAffine(cur, cur2, T, cur.size());

        cur2 = cur2(Range(vert_border, cur2.rows-vert_border), Range(HORIZONTAL_BORDER_CROP, cur2.cols-
HORIZONTAL_BORDER_CROP));

        // Resize cur2 back to cur size, for better side by side comparison
        cv::resize(cur2, cur2, cur.size());

        // Now draw the original and stablised side by side for comparison
        Mat canvas = Mat::zeros(cur.rows, cur.cols*2+10, cur.type());

        cur.copyTo(canvas(Range::all(), Range(0, cur2.cols)));
        cur2.copyTo(canvas(Range::all(), Range(cur2.cols+10, cur2.cols*2+10)));

        // If too big to fit on the screen, then scale it down by 2
        if(canvas.cols > 1920) {
            cv::resize(canvas, canvas, Size(canvas.cols/2, canvas.rows/2));
        }
        imshow("before and after", canvas);
        k++;
```

3. RESULTS AND DISCUSSION

3.1 GUI 介面與操作

如 Figure 5 所示為 GUI 介面設計與操作指示。首先點選 File, open 選取欲處理之檔案，繳交的作業資料夾中附帶 7 種不同晃動的影片供測試用。接著可以設定兩個參數，分別為 Smoothing radius 與 Horizontal border crop。Smoothing radius 為 Sliding average window 之參數，為前後用來作平均的 Frames 數量。數值愈大則輸出影像愈穩定流暢，但過大會使軌跡有嚴重的偏移。Horizontal border crop 則為裁切影像穩定後周圍產生的黑屏區。晃動愈大的影像黑屏區愈明顯，會影響觀看，故需要調整適當大小值作裁切。最後按下 Start 即開始作穩定處理。

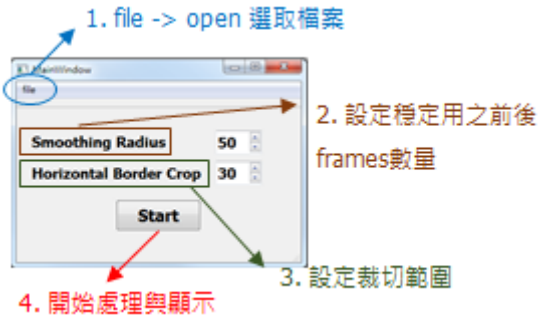

Figure 5 GUI 介面

如 Figure 6 所示為影像處理過程，作業附帶的影像皆大約需 5 秒處理時間。若 Frames 之數量愈多，或空間解析度愈大，需要處理時間將更久。

```
Starting C:\Users\Lab301\Documents\build-VideoStablizer-Desktop_Qt_5_7_1_MinGW_32bit-Debug\debug\VideoStablizer.exe...
Frame: 1/418 - good optical flow: 68
Frame: 2/418 - good optical flow: 76
Frame: 3/418 - good optical flow: 82
Frame: 4/418 - good optical flow: 82
Frame: 5/418 - good optical flow: 78
Frame: 6/418 - good optical flow: 66
Frame: 7/418 - good optical flow: 80
Frame: 8/418 - good optical flow: 73
Frame: 9/418 - good optical flow: 68
```

Figure 6 計算處理中

如 Figure 7 所示，圖左顯示原影片，圖右顯示穩定後影片。

Figure 7 結果顯示

其中的 Smoothing radius 若調整不當，如 Figure 8 所示為 Smoothing radius = 150 時，將產生異常的軌跡導致黑屏。

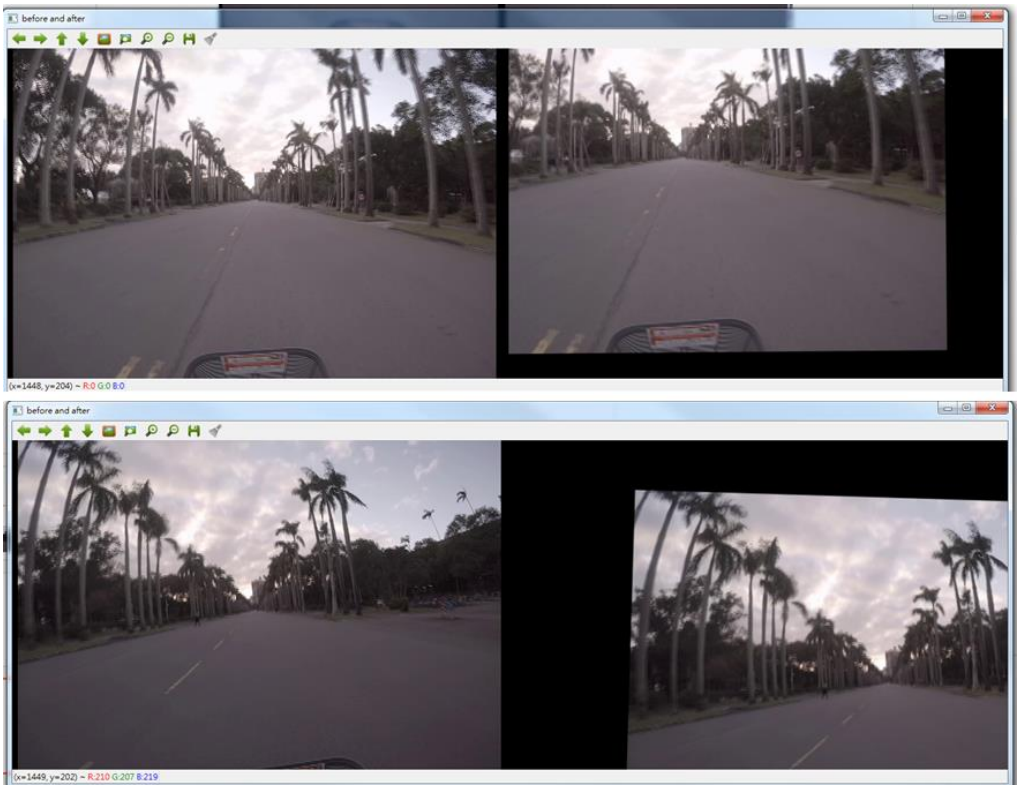

Figure 8 影片 ”Shaky_slow02”之 Smoothing radius = 150 之結果截圖

除此之外，此程式可以輸出原影像與處理後的影像之水平、垂直與旋轉三軸的軌跡與軌跡變化量，供後續分析使用，如 Figure 9 所示。

Figure 9 輸出之原影像與處理後影像分析數據

3.2 上下左右晃動影像

如 Figure 10 所示為資料夾中的影片 ”UpDownShake_slow”處理結果截圖。原影片有上下晃動的情況，其晃動頻率較慢。雖然其晃動幅度大，但在晃動頻率慢的情況下，處理後有不錯的結果，影片中的河馬皆於中央位置。調整適當的 Horizontal border crop 可以將大幅晃動產生的黑屏消除。

Figure 10 ”UpDownShake_slow”處理之結果截圖

如 Figure 11 所示為資料夾中的影片 ”UpDownShake_fast”處理結果截圖。原影片有上下晃動的情況，而其晃動頻率較快。其晃動頻率明顯較快，穩定效果不佳，仍有震動的感覺，即使調大 Smoothing radius 至 200 效果仍不佳。

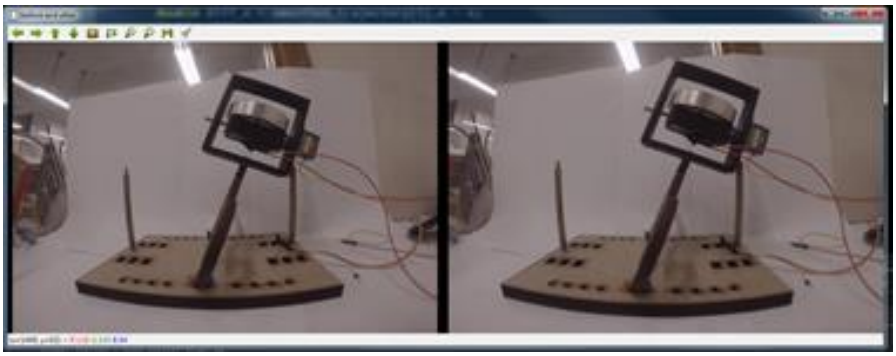

Figure 11 ”UpDownShake_fast”處理之結果截圖

如 Figure 12 所示為資料夾中的影片 ”Shaky_slow”處理結果截圖。原影片為騎乘自行車時手持攝影機拍攝，影像晃動主要包含上下及左右。其晃動頻率中等，處理後效果不錯，明顯看得出穩定後的品質較佳。

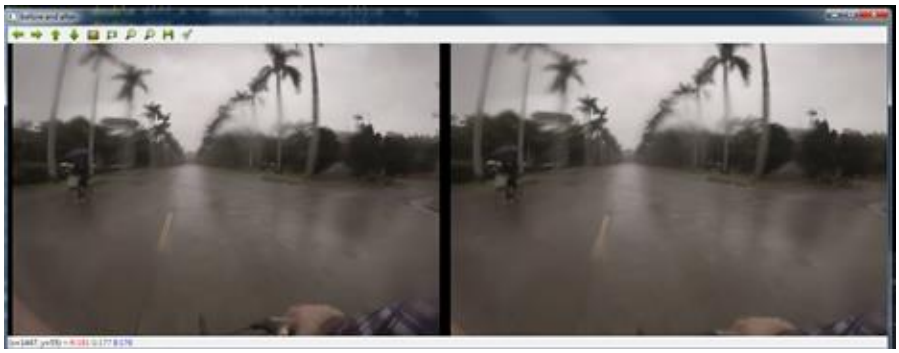

Figure 12 ”Shaky_slow”處理之結果截圖

3.3 旋轉晃動影像

如 Figure 13 所示為資料夾中的影片 ”RotatedShake_fast”處理結果截圖，另外影片 ”UpDownShake_fast ”處理結果與 Figure 13 類似。影片 ”RotatedShake_fast”的影像有快速的旋轉搖晃，而 ”UpDownShake_fast ”則只有上下

快速搖晃，在旋轉方向上的搖晃則非常微小。比較兩種晃動頻率的影片處理後的結果，旋轉晃動頻率在相對較快的情況下，主觀來看仍然有不錯的穩定結果，與晃動頻率較慢的結果相似。

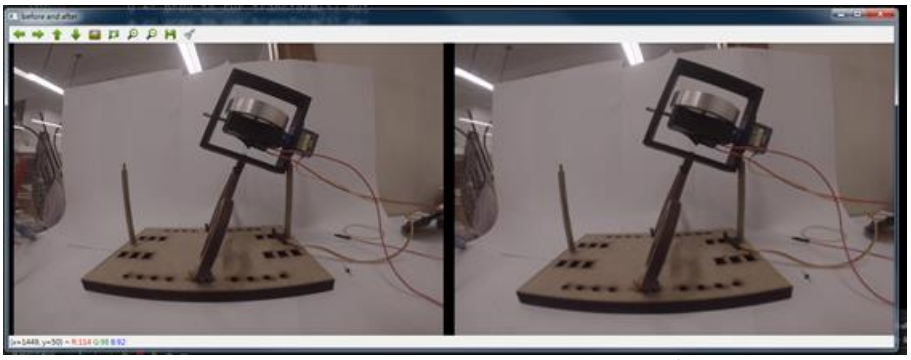

Figure 13 ”RotatedShake_fast”與 ”UpDownShake_fast ”處理之結果截圖

從軌跡的數據分析來看，兩者的旋轉方向軌跡分別如 Figure 14, Figure 15 所示。可以看到 ”RotatedShake_fast”原軌跡有明顯晃動，”UpDownShake_fast ”則相對較少，頻率與震幅皆較低。由結果得知，兩者在同樣的預設處理參數 Smoothing radius = 50 與 Horizontal border crop = 30 情況下，輸出的穩定後軌跡相當類似，皆有不錯的穩定流暢化較果。

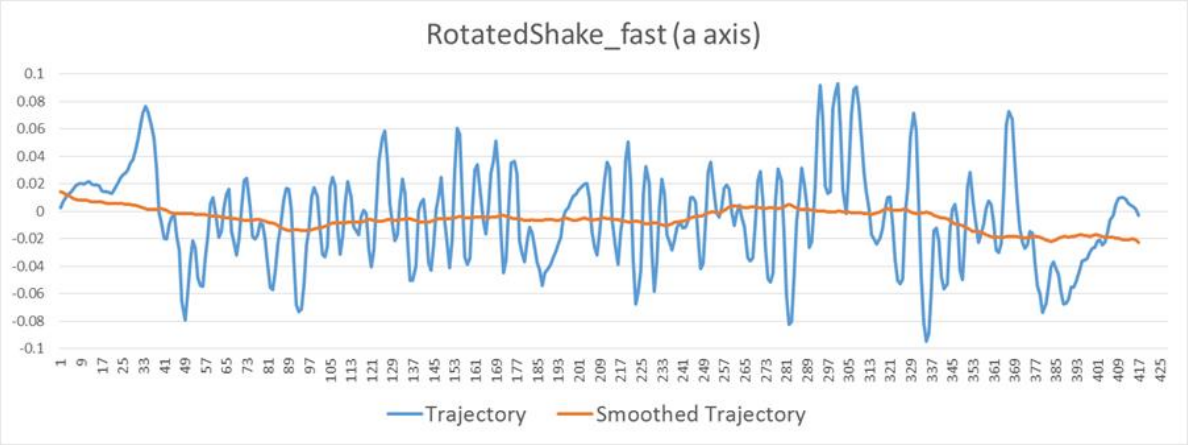

Figure 14 ”RotatedShake_fast”之旋轉方向軌跡

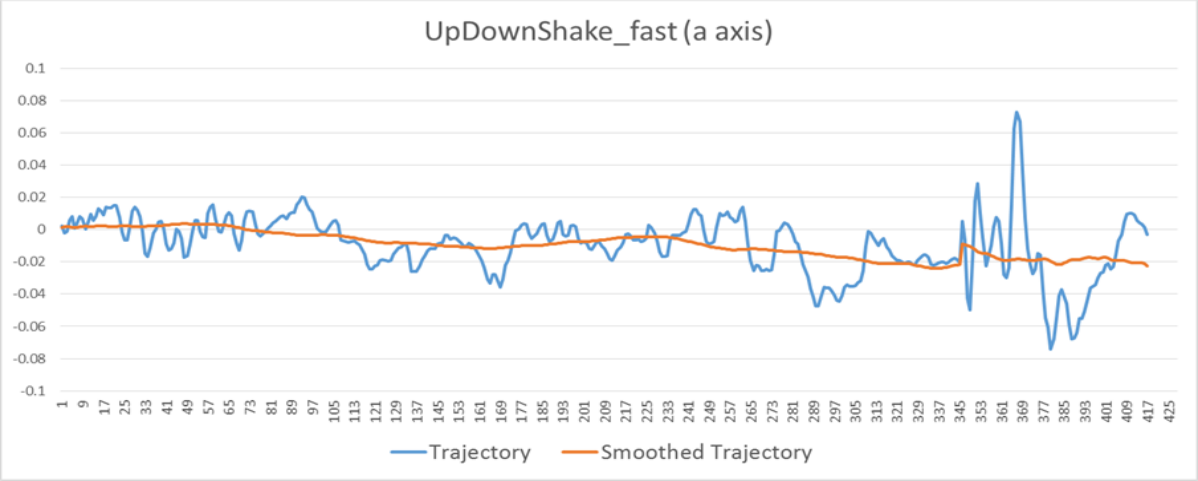

Figure 15 ”UpDownShake_fast ” 之旋轉方向軌跡

4. CONCLUSION

此影像穩定方式能簡單且有效地將不同晃動的影像穩定。由結果觀察與比較，此影像穩定的演算法對於左右、上下及旋轉的晃動，皆能有效處理。然而對於上下晃動的影像，晃動頻率較慢時才有良好的效果，晃動過快則較不理想，處理後仍然能看到晃動的現象。對於旋轉晃動的影像，在測試影片的晃動頻率中，不論較慢或較快的旋轉晃動，皆有不錯的處理結果，處理後影像相對流暢許多。此設計的程式能夠輸出原影像與處理後的影像之水平、垂直與旋轉三軸的軌跡與軌跡變化量，供後續分析使用。

REFERENCES

[1] GOPRO TIPS,“GoPro HERO5 Stabilization Examples,” 16 November 2016, < https://havecamerawilltravel.com/gopro/gopro-hero5-video-stabilization/> (20 December 2017).

[2] Wikipedia,“ 動 態 預 測 ,” 22 May 2017, <https://zh.wikipedia.org/wiki/%E5%8B%95%E6%85%8B%E9%A0%90%E6%B8%AC> (20 December 2017).