Automated Shuttle-Run Detection Application

By:

Amy Ho Kee Young, Ng Mei Xuan Bernadin,
Sherill Goh

Nanyang Girls' High School

Professor Lawrence Wong Wai Choong

National University of Singapore

Automated Shuttle-Run Detection Application

Amy Ho Kee Young, Ng Mei Xuan Bernadin,
Sherill Goh
Nanyang Girls' High School
Professor Lawrence Wong Wai Choong
National University of Singapore

1. Background and Purpose of Research

Field-based fitness tests such as the 4x10m shuttle run speed test are used worldwide in population-based studies, sport centres, and schools as a main indicator of an individual's physical fitness (Vicente-Rodríguez et al., 2011). However, according to research by Hetzler, Lundquist, Kimura and Stickley (2008), the accuracy of such speed test results is compromised when a stopwatch is used to keep time. As such, systematic reviews have identified a need for thorough validity and reliability studies on fitness testing in young populations, particularly in speed or agility tests (Castro-Pinero et al., 2009).

The 4x10m shuttle run speed test involves a runner covering a 4x10m sprint distance while transferring 2 object markers from a distance of 10m away to the start line. The time taken to complete the run is recorded by a digital stopwatch (Mackenzie, 2007).

Using a stopwatch is highly inaccurate because research has shown that reacting to a visual stimulus, in this case being the runner passing the start line, results in a large room for human reaction error (Ng and Chan, 2012). Since the average human reaction time is 200 milliseconds, this causes a possible significant discrepancy of +/- 200ms per reading (Shu, Javed, Tan and Weng, 2006). Furthermore, the shuttle run test itself also requires very accurate timings as even a slight discrepancy of 0.1s or 100ms can change an individual's fitness grade in tests such as the National Physical Fitness Award test ("2013 NAPFA Test Singapore", 2013). The fact that this value is lower than the average human reaction time of 200ms further reinforces that a more accurate timing tool is required for the shuttle run test.

On the other hand, timing gates make use of infrared signals and detectors, which eliminates the possibility of human error and thus provides more accurate and reliable results. However, such a timing system can cost up to $18,000, making it impractical for it to be adopted for everyday sprint tests such as the shuttle run ("Timing or Speed Gates", 2012).

However, due to widespread availability and the increasing computational power of smart phones (Cruz-Cunha and Moreira, 2011), mobile devices have been applied for a variety of uses in recent years, in particular, vision-based applications (Perez, 2014). The aim of this project is to combine the main attributes of the stopwatch and the timing gate to develop a timing application that is both practical and accurate, such that a higher level of accuracy in timing with respect to the shuttle run speed test can be achieved. Throughout the investigation process, experiments have been conducted to test the accuracy of the readings recorded by the application in comparison to the stopwatch.

2. Hypothesis

The built-in camera of a smartphone can be manipulated with a custom computer vision algorithm to more accurately time the shuttle run speed test as compared to using a stopwatch.

3. Methods and Materials

3.1 Algorithm Development

Based on the observation of videos of the shuttle-run exercise taken in an indoor location, an intuitive computer vision based algorithm (refer to Appendix A) was proposed to measure the time taken for the sprint test. The algorithm was formulated such that 3 key frames would be extracted to determine the duration of the shuttle run, when the runner first crosses the start line, returns to the start line, and crosses the start line again. The algorithm was written in Java language involving the OpenCV 2.3.1 software library. Based on the proposed algorithm, focus was placed on the start line, termed as the region of interest (ROI), and change detection between each frame was performed to detect the key frames as shown in Figure 1.

3.1.1 Mechanics of the Application

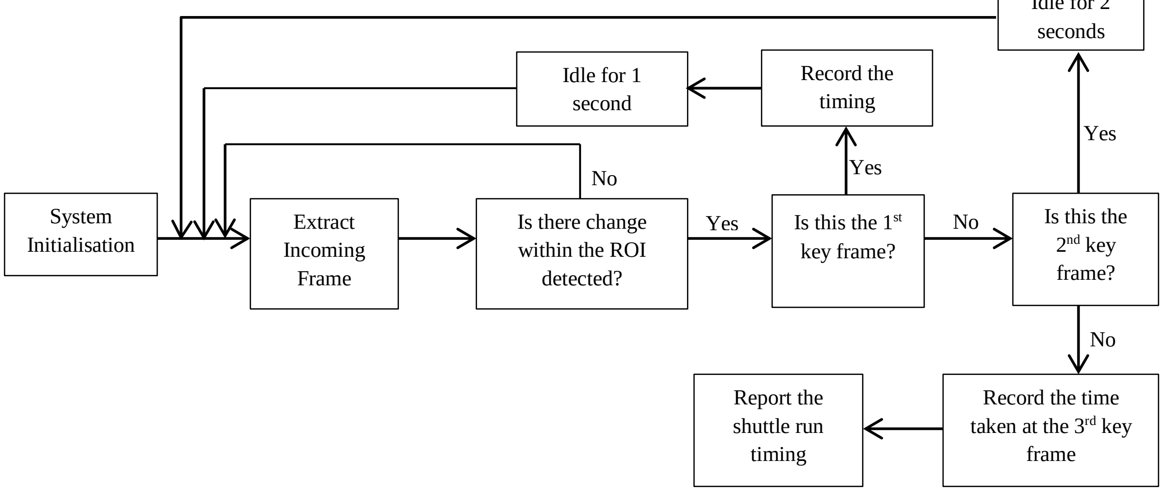
System Initialisation
Extract Incoming Frame
Is there change within the ROI detected?
No
Yes
Is this the 1st key frame?
Yes
Record the timing
Idle for 1 second
No
Is this the 2nd key frame?
Yes
Idle for 2 seconds
No
Record the time taken at the 3rd key frame
Report the shuttle run timing

Figure 1: Flow-chart of the Time Estimation Mechanism

Before the runner starts running, the 'Start' button is clicked. Upon clicking the button, the camera installed on the android device is activated, thereby capturing a video at 30 frames per second. Each frame is then compared with the frame captured before it, and the absolute difference between the two frames is derived in pixels. The absolute difference is calculated in reference to the changes in image in the ROI.

Using the *NonZerooutput = countnonzero(ROI)* method, the number of pixels that have recorded a change are compared to the number of pixels in the stipulated ROI and stored as a ratio. When the ratio passes the ratio threshold (refer to 3.1.2), the application recognizes that the runner has disturbed the ROI and the timer is activated. The runner runs 10 metres to pick up a marker and when the runner returns to the start line to place the marker down, disturbance on the virtual start line is detected and recognized as the first return to the start line. When the runner picks up the second marker and runs across the start line to finish, disturbance is detected again, stopping the timer. The elapsed duration is displayed on the screen. This entire process occurs in real time. Figure 2 below shows the layout of the user interface of the application.

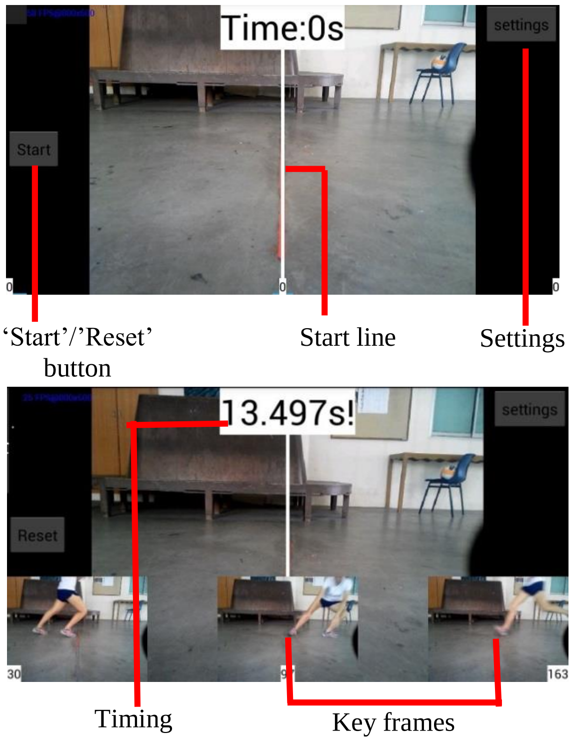

Figure 2: Screenshot of User Interface of Application

3.1.2 Ratio Threshold

As mentioned in 3.1.1, the application compares the number of pixels that have changed from one frame to the next in the ROI to the total number of pixels in the ROI, storing the data as a ratio. A threshold is required to ensure that slight disturbances do not trigger a false result.

In other words, if the threshold is set at 0.7, only ratios above 0.7 would be considered as a disturbance and activate the timer. However, recognizing the fact that different locations with different external factors such as lighting may affect the minimum ratio threshold needed, the 'Settings' button allows users to manually adjust the ratio threshold, thereby increasing or decreasing the sensitivity of the application as necessary.

3.2 Experimental Procedure

Two experiments were conducted to determine the success rate of the application and the accuracy of recorded timings respectively. The Android device with the application was placed on a tripod for stability, and the virtual start line was visually aligned with the actual start line seen through the lenses of the camera.

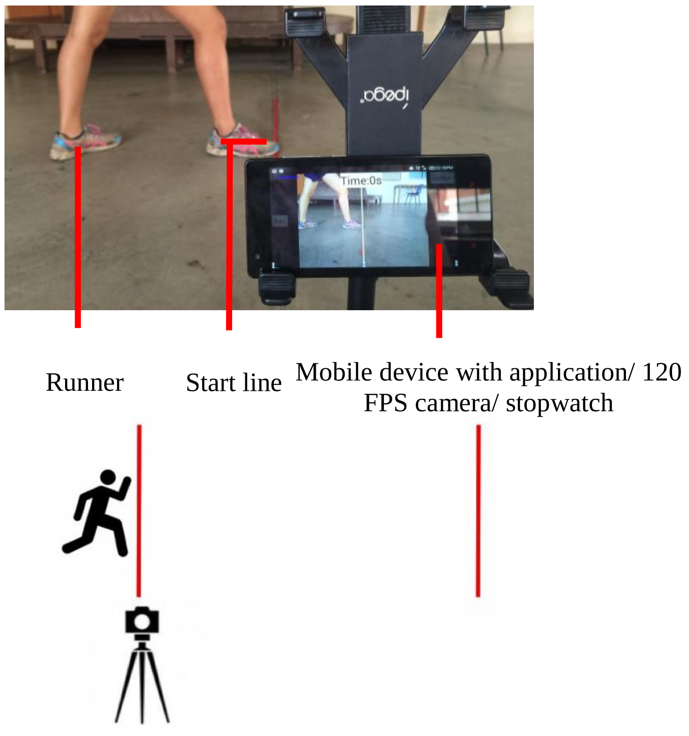

Figure 3: Experimental Set-Up

3.2.1 Experiment 1: Testing the Success Rate of the Application

Through repeated trials of shuttle-run sprint tests by different individuals, the success rate of the application in detecting disturbances when the runner crosses the line was determined to test the reliability and consistency of the application.

3.2.2 Experiment 2: Testing the Accuracy of the Application

The purpose of the second experiment was to determine the absolute accuracy of the application and its relative accuracy with respect to a stopwatch. A high frames per second (FPS) camera (120 FPS) was used as it was the most viable option available. Using the high FPS camera, videos of runs were taken and played back in slow motion to enable manual extraction of the accurate timings. Readings taken from the three timing tools were then compared to determine the accuracy of the application and stopwatch.

4. Results and Discussion

4.1 Experiment 1

This experiment was split into test 1A and 1B as improvements to the application were made.

Table 1: Table of Results for Test 1A

No. of successes	No. of failures			Total runs	Success rate
	1st frame	2nd frame	3rd frame		
33	5	8	4	50	66%

For test 1A, successes were recorded when disturbances for all three key frames were detected. Failures were recorded when the application was unable to detect disturbances in the ROI for any of the key frames. The results from the first round of testing showed that the application had a low success rate of 66%. Upon further observation, the inconsistency of the detection was found to be due to the low sensitivity of the frame difference detection (refer to 3.1.2). Hence, modifications allowing users to manually change the sensitivity were made.

Table 2: Table of Results for Test 1B

No. of successes	No. of failures			Total runs	Success rate
	1st frame	2nd frame	3rd frame		
35	0	0	0	35	100%

In this test, the sensitivity of the improved application was increased to attain a more accurate frame detection, but not to the extent where subtle lighting changes would trigger a false positive result. Hence, a success rate of 100% was achieved, proving the usability of the application, and that the application is as consistent as a stopwatch or light gate.

4.2 Experiment 2: Testing the Accuracy of Recorded Timings

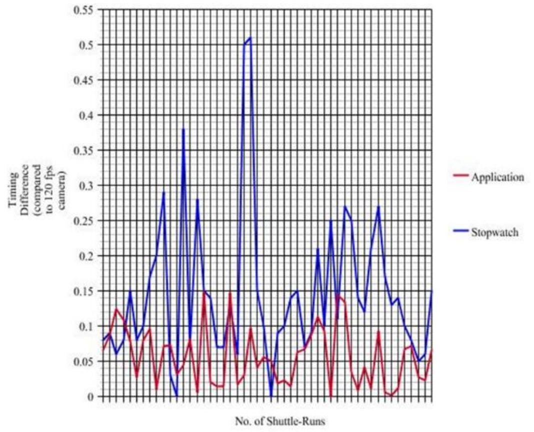

Figure 4: Timing Differences of Stopwatch and Application Per Run

Table 3: Mean and Median Timing Differences of Application and Stopwatch Per Run

	Application	Stopwatch
Mean Difference/s (compared to 120 fps)	**0.0568**	**0.1478**
Median Difference/s (compared to 120 fps)	**0.0530**	**0.1250**
Maximum Difference/s	**0.149**	**0.51**
Minimum Difference/s	**0.00**	**0.00**

The above figures show the results obtained from a total of 50 shuttle runs. Using the timings extracted from the 120 FPS camera as the "true timings", the time differences between the timings recorded by the application, stopwatch and the 120 FPS camera were calculated to investigate their accuracy. (refer to Appendix B)

The results show that the application is significantly more accurate than the stopwatch. The mean and median differences between timings recorded by the stopwatch and 120 FPS camera are 0.1478s and 0.1250s respectively, which are slightly lesser than the average human reaction error of approximately 0.200s as determined by research (Shu et al., 2006). The mean and median differences between timings recorded by the application and the 120 FPS camera are significantly lower at 0.0568s and 0.0530s respectively, which proves that the application is more accurate than the stopwatch in timing the shuttle-run.

From Figure 4, it can be inferred that timings recorded by the stopwatch are prone to major fluctuations in accuracy, as the differences between timings recorded by the stopwatch and timings recorded by the 120 FPS camera range from 0.00s to 0.51s. This is due to the human reaction error involved when a stopwatch is being used. Since the usage of the application does not require human reaction, the fluctuations of timing differences between the application and the 120 FPS camera are much less significant in comparison, ranging from 0.00s to 0.149s. This proves that the application is more consistent in accuracy than the stopwatch.

However, the highest difference between the timing recorded by the application and that of the 120 FPS camera is 0.149s. Although it is not as high as that of the stopwatch, this value is rather significant. Future work can be done to reduce such fluctuations by adjusting the sensitivity of the application or editing the algorithm to further increase its accuracy.

5. Conclusion and Recommendations for Future Work

In conclusion, the application is able to replace the digital stopwatch to time the shuttle run sprint test, as it produces more accurate results as compared to a stopwatch, and is a more viable option as compared to a timing gate. The application is downloadable onto any android-based device and only requires a tripod or a stable surface to operate ideally, which makes it less costly, more portable and more convenient than the timing gate. Hence, the application merges the ideal characteristics of both the stopwatch and timing gate, allowing individuals to obtain a more accurate gauge of their fitness and health. This application can replace the stopwatch in timing the shuttle run during the NAPFA test, allowing for more accurate results without significant costs.

During the period of experimentation, possibilities of using the frame disturbance detection algorithm for the purpose of timing other runs emerged. For example, the algorithm could be manipulated to stop the timer when a disturbance is detected at a different time (instead of the current third time) to accurately time other variations of runs such as single sprints, repeat sprints, running back and forth through the same gate, and multiple people sprinting in different lanes. Furthermore, the accuracy of the application in timing runs can be improved by allowing for quicker processing and detection of key frames through further modifications of the code. In the absence of the tripod to ensure stability of the Android device, the possibility of adding video or image stabilization into the algorithm could be explored to automatically compensate for the pan and tilt of the camera. With the new algorithm, accurate timings may still be derived despite the unstable camera view when the device is hand-held. Lastly, the application could also be extended to other operating systems such as the Apple IOS to ensure maximum outreach such that a wider audience can have access to the timing application.

6. References

2013 NAPFA Test Singapore. (2013). Retrieved November 6, 2014, from http://www.pullupbarsg.com/napfa-test-singapore-requirements-standards-chart/

Castro-Pinero, J., Artero, E.G., Espana-Romero, V., Ortega, F.B., Sjostrom, M., Suni, J., and Ruiz, J.R. (2009). Criterion-related validity of field-based fitness tests in youth: A systematic review. British Journal of Sports Medicine, 44(13), 934–943. Retrieved December 24, 2014, from http://bjsm.bmj.com/content/44/13/934.full.pdf+html

Cruz-Cunha, M., and Moreira, F. (2011). Building Mobile Sensor Networks Using Smartphones and Web Services. In *Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts* (Vol. 2, p. 517).

Hetzler, R.K., Lundquist, K.M., Kimura, I.F., and Stickley, C.D. (2008). Reliability and Accuracy of Handheld Stopwatches Compared with Electronic Timing in Measuring Sprint Performance. Journal of Strength and Conditioning Research, 22(6), 1969-1976. Retrieved December 4, 2014, from http://www.researchgate.net/publication/23446051_Reliability_and_accuracy_of_handheld_stopwatches_compared_with_electronic_timing_in_measuring_sprint_performance

Mackenzie, B. (2007). Shuttle Run Test. Retrieved December 28, 2014, from http://www.brianmac.co.uk/runtest.htm

Ng, A., and Chan, A. (2012). Finger Response Times to Visual, Auditory and Tactile Modality Stimuli. *International MultiConference of Engineers and Computer Scientists 2012, 2*(-). Retrieved December 28, 2014, from International Association of Engineers.

Perez, S. (2014, April 1). Mobile App Usage Increases In 2014, As Mobile Web Surfing Declines. TechCrunch. Retrieved Deccember 27, 2014, from http://techcrunch.com/2014/04/01/mobile-app-usage-increases-in-2014-as-mobile-web-surfing-declines/

Shu, M., Javed, S., Tan, J., and Weng, N. (2006, January 1). Fingertip Reaction Time. Retrieved December 28, 2014, from http://hypertextbook.com/facts/2006/reactiontime.shtml

Timing or Speed Gates. (2012, January 1). Retrieved December 28, 2014, from http://www.topendsports.com/testing/timing-gates.htm

Vicente-Rodríguez, G., Rey-López, J., Ruíz, J., Jiménez-Pavón, D., Bergman, P., Ciarapica, D., ... Ortega, F. (2011). Interrater Reliability and Time Measurement Validity of Speed–Agility Field Tests in Adolescents. Journal of Strength and Conditioning Research, 0(0), 2059-2063. Retrieved November 24, 2014, from http://www.helenastudy.com/files/13_Vicente-Rodriguez-JSCR-2011.pdf

7. Appendices

7.1 Appendix A: Code for Timing Application

7.1.1 MainActivit.Java

```
package com.example.ShuttleRun;

import java.util.ArrayList;
import java.util.List;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewFrame;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;

import Utils.SettingsDialog;
import Utils.ShuttleRunUtils;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import android.widget.TextView;

import com.example.opticalflow.R;

public class MainActivity extends ActionBarActivity implements
            CvCameraViewListener2, OnClickListener {
      private static final String TAG = "OCVSample::Activity";
      private Mat mRgba, mResizedRgba, mMaskImg, firstFrame, curFrame;
      MatOfPoint corner;
      int startRun, midRun, lastRun = 0;
      int ImgW = 320;
      int ImgH = 240;
      long startTime,midTime,endTime=0;
      int mMaskCount, count, EventCount = 0;
      ImageView iv,ivStart,ivMid,ivEnd;
      TextView tvStart,tvMid,tvEnd,tvTime;
      Bitmap bmp = null;
      boolean START, INITIALIZED = false;
      Button bStart,bSettings;
      Mat Merged;
      Mat MorphROI;
      ShuttleRunUtils utils = new ShuttleRunUtils(ImgH, ImgW, this);
      Mat prevframe;
      double sensitivityRatio=0.7;
      private CameraBridgeViewBase mOpenCvCameraView;
      private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
            @Override
            public void onManagerConnected(int status) {

      if(status==LoaderCallbackInterface.SUCCESS){


      mOpenCvCameraView.enableView();

      mOpenCvCameraView.enableFpsMeter();

     mOpenCvCameraView.getDrawingTime();
                        corner = new MatOfPoint();



                  }

            }
      };

      void InitialMask() {
```

```
		mMaskImg = new
Mat().zeros(ImgH, ImgW, CvType.CV_8UC1);
		Point p1 = new Point(ImgW / 2, 0);
		Point p2 = new Point(ImgW / 2,
ImgH);
		Scalar s = new Scalar(255, 255,
255);
		Core.line(mMaskImg, p1, p2, s,
5);
		// Imgproc.cvtColor(mMaskImg,
mMaskImg, Imgproc.COLOR_BGRA2GRAY);
		// Imgproc.threshold(mMaskImg,
mMaskImg, 50, 255,
		// Imgproc.THRESH_BINARY);
		Imgproc.threshold(mMaskImg,
mMaskImg, 50, 50, Imgproc.THRESH_BINARY);
		mMaskCount =
Core.countNonZero(mMaskImg);
	}

	void getFirstFrame() {
		firstFrame =
utils.ResizeFrame(mRgba);
		// Size kernelsize= new
Size(21,21);
		//
Imgproc.GaussianBlur(firstFrame, firstFrame,
kernelsize, 5.0);
	}

	@Override
	protected void onCreate(Bundle
savedInstanceState) {

	super.onCreate(savedInstanceState);

	requestWindowFeature(Window.FEATU
RE_NO_TITLE);

	getWindow().addFlags(WindowManager.
LayoutParams.FLAG_KEEP_SCREEN_ON);

	setContentView(R.layout.activity_main);
		mOpenCvCameraView =
(CameraBridgeViewBase)
findViewById(R.id.surface_view);
		iv = (ImageView)
findViewById(R.id.imageView1);
		ivStart = (ImageView)
findViewById(R.id.iv_start);
		ivMid = (ImageView)
findViewById(R.id.iv_mid);
		ivEnd = (ImageView)
findViewById(R.id.iv_end);
		tvStart = (TextView)
findViewById(R.id.tv_start);
		tvMid = (TextView)
findViewById(R.id.tv_mid);
		tvEnd = (TextView)
findViewById(R.id.tv_end);
		tvTime = (TextView)
findViewById(R.id.tv_time);

	mOpenCvCameraView.setCvCameraView
Listener(this);

		bStart = (Button)
findViewById(R.id.button1);
		bSettings = (Button)
findViewById(R.id.b_setting);
		bStart.setOnClickListener(this);

	bSettings.setOnClickListener(this);
	}

	@Override
	public void onResume() {
		super.onResume();

	OpenCVLoader.initAsync(OpenCVLoade
r.OPENCV_VERSION_2_4_3, this,

	mLoaderCallback);
	}

	public void showOnImageView() {
		runOnUiThread(new Runnable()
{

			@Override
			public void run() {


	iv.setImageBitmap(utils.ConvertMattoBit
map(MorphROI));

			}
		});

	}

	public void onDestroy() {
		super.onDestroy();
		if (mOpenCvCameraView != null)

	mOpenCvCameraView.disableView();
	}

	public void onCameraViewStarted(int
width, int height) {
		mRgba = new Mat(height, width,
CvType.CV_8UC3);
		prevframe= new Mat(height,
width, CvType.CV_8UC3);

	}

	public void onCameraViewStopped() {
		mRgba.release();
	}
```

```
//Mat prevframe= new Mat(height, width, CvType.CV_8UC3);
	public Mat onCameraFrame(CvCameraViewFrame inputFrame) {

		mRgba = inputFrame.rgba();


		if (START) {
			START = false;
			InitialMask();
			getFirstFrame();
			// showOnImageView();
			INITIALIZED = true;
		}
		if (INITIALIZED) {
			processFrame();
			showOnImageView();
			getFirstFrame();
		}

	System.out.println("firstRunCount:" + startRun + " midRunCount:"
					+ midRun + " LastRunCount:" + lastRun);
		System.out.println("frame count:" + count);
		// Draw line
		Point p3 = new Point(mRgba.cols() / 2, 0);
		Point p4 = new Point(mRgba.cols() / 2, mRgba.rows());
		Scalar s = new Scalar(255, 255, 255);
		Core.line(mRgba, p3, p4, s, 5);
		// ///////////////
		return mRgba;
	}

	void processFrame() {

		if(startTime>0){

			runOnUiThread(new Runnable() {
				@Override
				public void run() {
					long curTime = System.currentTimeMillis(); // Get the start Time
					long diff=curTime-startTime;

	tvTime.setText(""+diff/1000.0f+"s");
				}
			});
		}

		curFrame = utils.ResizeFrame(mRgba);
		Mat diff = new Mat(firstFrame.rows(), firstFrame.cols(), CvType.CV_8UC3);
		Core.absdiff(curFrame, firstFrame, diff);
		Scalar mean = Core.mean(diff);
		List<Mat> channels = new ArrayList<Mat>();
		Core.split(diff, channels);

		double thres0 = 2 * mean.val[0];
		if (thres0 < 18.0)
			thres0 = 18.0;
		double thres1 = 2 * mean.val[1];
		if (thres1 < 18.0)
			thres1 = 18.0;
		double thres2 = 2 * mean.val[2];
		if (thres2 < 18.0)
			thres2 = 18.0;


	Imgproc.threshold(channels.get(0), channels.get(0), thres0, 50,

	Imgproc.THRESH_BINARY);

	Imgproc.threshold(channels.get(1), channels.get(1), thres1, 50,

	Imgproc.THRESH_BINARY);

	Imgproc.threshold(channels.get(2), channels.get(2), thres2, 50,

	Imgproc.THRESH_BINARY);
		Merged = new Mat(diff.rows(), diff.cols(), CvType.CV_8UC3);
		Core.merge(channels, Merged);
		Imgproc.cvtColor(Merged, Merged, Imgproc.COLOR_BGRA2GRAY);
		Imgproc.threshold(Merged, Merged, 1, 50, Imgproc.THRESH_BINARY);
		Imgproc.threshold(Merged, Merged, 1, 255, Imgproc.THRESH_BINARY);
		Core.add(Merged, mMaskImg, Merged);
		Imgproc.erode(Merged, Merged, new Mat(), new Point(-1, -1), 1);
		Imgproc.dilate(Merged, Merged, new Mat(), new Point(-1, -1), 1);
		int curCount = Core.countNonZero(Merged);
		double dRatio = (double)curCount / mMaskCount;
	//	double dRatio = curCount / (Merged.width()*Merged.height());
```

```
                System.out.println("Ratio:" + curCount + "maskRatio" + mMaskCount+" dRatio:"+dRatio);
                MorphROI = new Mat(Merged, utils.createROIrect());
                if (dRatio > 1.5 && EventCount == 2 &&count>midRun+30) {

                    int lastCount = Core.countNonZero(MorphROI);
                    double lastRatio = (double)lastCount / mMaskCount;
                //  double lastRatio = lastCount / MorphROI.rows()*MorphROI.height();
                    Log.i("Last ratio:", lastRatio+"");
                    if (lastRatio >= sensitivityRatio) {

        runOnUiThread(new Runnable() {


        @Override
                                public void run() {

        endTime = System.currentTimeMillis(); // Get the start Time

        ivEnd.setImageBitmap(utils.ConvertMattoBitmap(curFrame));

        tvEnd.setText(""+count);

        long diff=endTime-startTime;

        tvTime.setText(""+diff/1000.0f+"s!");

        startTime=0;
                                }
                            });

                        lastRun = count;
                        EventCount++;

                    }
                }

                if (dRatio > 1.5  && EventCount == 1 &&count>startRun+15) {

                    int midCount = Core.countNonZero(MorphROI);
                    double midRatio = (double)midCount / mMaskCount;
                //  double midRatio = midCount /(MorphROI.rows()*MorphROI.height());
                    Log.i("mid ratio:", midRatio+"");
                    if (midRatio >= sensitivityRatio) {

        runOnUiThread(new Runnable() {


        @Override
                                public void run() {

        midTime = System.currentTimeMillis(); // Get the start Time

        ivMid.setImageBitmap(utils.ConvertMattoBitmap(curFrame));

        tvMid.setText(""+count);


                                }
                            });

                        midRun = count;
                        EventCount++;

                    }
                }

                if (dRatio > 1.5  && EventCount == 0 && count > 5) {

                    int firstCount = Core.countNonZero(MorphROI);
                    double firstRatio = (double)firstCount / mMaskCount;
                //  double firstRatio = firstCount / (MorphROI.rows()*MorphROI.height());
                    Log.i("first ratio:", firstRatio+"");
                    if (firstRatio >= sensitivityRatio) {

        runOnUiThread(new Runnable() {


        @Override
                                public void run() {

        startTime = System.currentTimeMillis(); // Get the start Time

        ivStart.setImageBitmap(utils.ConvertMattoBitmap(curFrame));

        tvStart.setText(""+count);
                                }
                            });
```

```
                                        EventCount++;
                                        startRun        =
count;
                              }
                    }

                    count++;
          }

          void ResetCounts() {
                    count = 0;
                    EventCount = 0;
                    startRun= midRun= lastRun = 0;
                    startTime=0;
                    runOnUiThread(new   Runnable()
{

                              @Override
                              public void run() {


          ivMid.setImageResource(R.drawable.abc_
ab_bottom_transparent_light_holo);

          tvMid.setText("0");

          ivStart.setImageResource(R.drawable.abc
_ab_bottom_transparent_light_holo);

          tvStart.setText("0");


          ivEnd.setImageResource(R.drawable.abc_
ab_bottom_transparent_light_holo);

          tvEnd.setText("0");

          tvTime.setText("0s");
                              }
                    });


          }

          @Override
          public void onClick(View v) {
                    if (v.getId() == R.id.button1) {
                              if
(bStart.getText().toString().equals("Start")) {
                                        START = true;

          bStart.setText("Reset");
                              } else {

          bStart.setText("Start");
                                        START = false;
                                        INITIALIZED
= false;
                                        ResetCounts();
                              }
                                        /*
                                         *
ConvertMattoBitmap();                            int
mMaskCount=Core.countNonZero(mMaskImg);
                                         *
System.out.println("Mask Count:"+mMaskCount);
                                         */
                              }else
if(v.getId()==R.id.b_setting){
                                        final       SettingsDialog
sDialog=new SettingsDialog(this);
                                        sDialog.show();

          sDialog.sensitivtyControl.setProgress((int)
(((sensitivityRatio-0.2)*100)));

          sDialog.sensitivtyControl.setSecondaryPr
ogress((int) (((sensitivityRatio-0.2)*100)));
                                        Log.i("progress     now",
sensitivityRatio+"");

          sDialog.sensitivtyControl.setOnSeekBarC
hangeListener(new OnSeekBarChangeListener() {
                                                  //int
progressChanged = 0;


                                                  public        void
onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser){

          sensitivityRatio          =           (double)
progress/100+0.2;

          Log.i("progress", progress+"");
                                                  }

                                                  public        void
onStartTrackingTouch(SeekBar seekBar) {
                                                            //
TODO Auto-generated method stub
                                                  }

                                                  public        void
onStopTrackingTouch(SeekBar seekBar) {

                                                  }
                                        });

          sDialog.bOK.setOnClickListener(new
OnClickListener(){

                                                  @Override
                                                  public        void
onClick(View v) {

          sDialog.dismiss();

                                                  }});
                              }
```

```
	}

}
```

7.1.2 Shuttle Run Utils

```
package Utils;

import org.opencv.android.Utils;
import org.opencv.core.Core;
import org.opencv.core.CvException;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;

import android.app.Activity;
import android.graphics.Bitmap;
import android.util.Log;

public class ShuttleRunUtils {
	int ImgH;
	int ImgW;
	Activity a;

	public ShuttleRunUtils(int imgH, int imgW,
Activity a) {
		this.ImgH = imgH;
		this.ImgW = imgW;
		this.a = a;

	}

	public Mat DrawLine(Mat mat) {
		Mat mMaskImg = new
Mat().zeros(ImgH, ImgW, CvType.CV_8UC1);
		Point p1 = new Point(ImgW / 2, 0);
		Point p2 = new Point(ImgW / 2,
ImgH);
		Scalar s = new Scalar(255, 255,
255);
		Core.line(mat, p1, p2, s, 5);
		return mMaskImg;
	}

	public Mat ResizeFrame(Mat mat) {
		Mat mResizedRgba = new
Mat(ImgH, ImgW, CvType.CV_8UC3);
		Size sz = new Size(ImgW, ImgH);
		Imgproc.resize(mat,
mResizedRgba, sz);

		return mResizedRgba;
	}

	public Bitmap ConvertMattoBitmap(Mat
mat) {
		Bitmap bmp = null;
		try {
			bmp =
Bitmap.createBitmap(mat.cols(), mat.rows(),

	Bitmap.Config.ARGB_8888);
			Utils.matToBitmap(mat,
bmp);


		} catch (CvException e) {
			Log.d("Exception",
e.getMessage());
		}
		return bmp;
	}

	public Rect createROIrect(){
		Rect ROIrect = new Rect();
		ROIrect.x=ImgW/2;
		ROIrect.y=0;
		ROIrect.width=ImgW/10;
		ROIrect.height=ImgH;
		return ROIrect;
	}

}
```

7.1.3 Settings Dialog

```
package Utils;

import android.app.Dialog;
import android.content.Context;
import android.os.Bundle;
import android.widget.Button;
import android.widget.SeekBar;

import com.example.opticalflow.R;

public class SettingsDialog extends Dialog {
	public static SeekBar sensitivtyControl;
	public static Button bOK;
	public SettingsDialog(Context context) {
		super(context);
		// TODO Auto-generated
constructor stub
	}

	@Override
	protected void onCreate(Bundle
savedInstanceState) {
		// TODO Auto-generated method
stub

	super.onCreate(savedInstanceState);

	setContentView(R.layout.dialog_settings);

		sensitivtyControl = (SeekBar)
findViewById(R.id.seekBar1);
```

```
                    bOK=(Button)
findViewById(R.id.b_ok);


/*

          sensitivtyControl.setOnSeekBarChangeLis
tener(new OnSeekBarChangeListener() {
                              int progressChanged = 0;

                              public              void
onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser){

          progressChanged = progress;
                              }

                              public              void
onStartTrackingTouch(SeekBar seekBar) {
                                        // TODO Auto-
generated method stub
                              }

                              public              void
onStopTrackingTouch(SeekBar seekBar) {

                              }
                    });
                    */
          }



}
```