A camera preview with a bounding box like Google goggles

adblogcat.com /a-camera-preview-with-a-bounding-box-like-google-goggles/

This article describes how to draw a bounding box on top of a camera preview to capture part of the image just like Google goggles does. This article mainly involves looking at a SurfaceView for generating a camera view and a View to create a canvas and draw on top of the camera view. Also shows how to use the onTouchInterceptEvent to pass touch events to lower Views. Android 2.2 API needs to be used for this.

After seeing Google goggle's implementation, I thought there would be some API to do this, but I could not find any. I then decided to create my own. I did this for my app: Translanguage OCR.

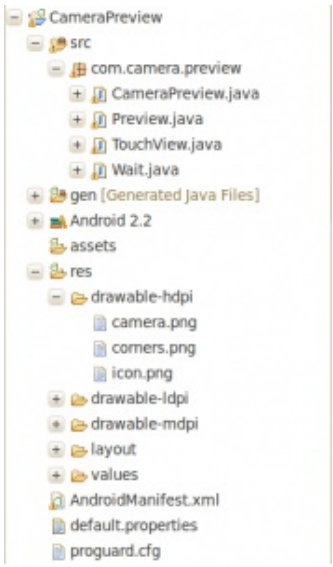
You can download the source HERE
Password is: preview

Create the Project

Create a new android project. Here create the classes: CameraPreview, Preview, TouchView and Wait. Also you need two images, camera.png and corners.png. You can save them from here:

Your folder structure will look something like this:

Open the manifest and add the following permissions:




```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

Now open the main.xml file and add the following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical" android:layout_width="fill_parent"
 android:layout_height="fill_parent">
 <com.camera.preview.Preview android:id="@+id/preview"
 android:layout_width="fill_parent" android:layout_height="fill_parent">
 </com.camera.preview.Preview>
 <com.camera.preview.TouchView android:id="@+id/left_top_view"
 android:layout_width="fill_parent" android:layout_height="fill_parent">
 </com.camera.preview.TouchView>
 <ImageView android:id="@+id/startcamerapreview"
android:src="@drawable/camera"
 android:layout_width="fill_parent" android:layout_height="wrap_content"
 android:layout_alignParentBottom="true"
 android:layout_alignParentRight="true" />
</RelativeLayout>
```

Here we are adding a Preview and a TouchView which we will create so that the camera is used for a camera preview and a View to draw on top of it.

Wait.java

Now we will write to the Wait.java class. This is just a simple thread to wait some period of time. This will be useful for the autofocus.

```
public class Wait {
 public static void oneSec() {
 try {
 Thread.currentThread().sleep(1000);
 }
 catch (InterruptedException e) {
 e.printStackTrace();
 }
 }
 public static void manySec(long s) {
 try {
 Thread.currentThread().sleep(s *
1000);
 }
 catch (InterruptedException e) {
 e.printStackTrace();
 }
 }
}
```

Preview.java

Now open Preview.java and extend SurfaceView and implement SurfaceHolder.CallBack. Three methods will be created: surfaceCreated, surfaceDestroyed, surfaceChanged.

Preview.java is used to open a camera preview. It uses hardware.Camera and a SurfaceView to display the camera's view on the screen. Now to write some code. The constructors and some extra methods:

```
class Preview extends SurfaceView implements SurfaceHolder.Callback {
 private SurfaceHolder mHolder;
 private Camera mCamera;
 private Camera.Parameters mParameters;
 private byte[] mBuffer;
 public Preview(Context context, AttributeSet attrs) {
 super(context, attrs);
 init();
 }
 public Preview(Context context) {
 super(context);
 init();
 }
 public void init() {
 // Install a SurfaceHolder.Callback so we get notified when the
 // underlying surface is created and destroyed.
 mHolder = getHolder();
 mHolder.addCallback(this);
 mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
 }
```

This is a pretty standard way of getting a Holder to hold the camera preview. The purpose of this view is to get the view onto the screen and be able to take the picture and save it. For this we need another method:

```
 public Bitmap getPic(int x, int y, int width, int height) {
 System.gc();
 Bitmap b = null;
 Size s = mParameters.getPreviewSize();
 YuvImage yuvimage = new YuvImage(mBuffer, ImageFormat.NV21, s.width, s.height,
null);
 ByteArrayOutputStream outStream = new ByteArrayOutputStream();
 yuvimage.compressToJpeg(new Rect(x, y, width, height), 100, outStream); // make
JPG
 b = BitmapFactory.decodeByteArray(outStream.toByteArray(), 0, outStream.size());
 if (b != null) {
 //Log.i(TAG, "getPic() WxH:" + b.getWidth() + "x" + b.getHeight());
 } else {
 //Log.i(TAG, "getPic(): Bitmap is null..");
 }
 yuvimage = null;
 outStream = null;
 System.gc();
 return b;
 }
```

This method is the one that will get an area of the image, or a set of coordinates in the image, and turn it into a Bitmap. The CameraPreview class will explain how the set of coordinates is taken. The reason API 2.2 is needed is because of the YuvImage class. This class is crucial for what we are trying to do here. YuvImage is used, compressed to jpeg and turned to bitmap.

In order to keep the image, we need to set up a buffer. This will ensure that when the image is captured, the buffer is big enough to hold the data.

```
 private void updateBufferSize() {
 mBuffer = null;
 System.gc();
 // prepare a buffer for copying preview data to
 int h = mCamera.getParameters().getPreviewSize().height;
 int w = mCamera.getParameters().getPreviewSize().width;
 int bitsPerPixel =
 ImageFormat.getBitsPerPixel(mCamera.getParameters().getPreviewFormat());
 mBuffer = new byte[w * h * bitsPerPixel / 8];
 //Log.i("surfaceCreated", "buffer length is " + mBuffer.length + "
bytes");
 }
```

Now the methods we need to Override:

```
 public void surfaceCreated(SurfaceHolder holder) {
 // The Surface has been created, acquire the camera and tell it where to draw.
 try {
 mCamera = Camera.open(); // WARNING: without permission in Manifest.xml, crashes
 }
 catch (RuntimeException exception) {
 //Log.i(TAG, "Exception on Camera.open(): " + exception.toString());
 Toast.makeText(getContext(), "Camera broken, quitting :
(",Toast.LENGTH_LONG).show();
 // TODO: exit program
 }
 try {
 mCamera.setPreviewDisplay(holder);
 //updateBufferSize();
 mCamera.addCallbackBuffer(mBuffer); // where we'll store the image data
 mCamera.setPreviewCallbackWithBuffer(new PreviewCallback() {
 public synchronized void onPreviewFrame(byte[] data, Camera c) {
 if (mCamera != null) { // there was a race condition when onStop() was called..
 mCamera.addCallbackBuffer(mBuffer); // it was consumed by the call, add it back
 }
 }
 });
 } catch (Exception exception) {
 //Log.e(TAG, "Exception trying to set preview");
 if (mCamera != null){
 mCamera.release();
 mCamera = null;
 }
 // TODO: add more exception handling logic here
 }
 }
```

When the surface is created, this method is called. Here we have to open the hardware.Camera and set a Preview display (thus camera preview), set the buffer and start receiving data. Using hardware.Camera is sometimes annoying because you have to be sure to release the camera or you will have problems with your application or even other applications that need to access the camera after you are done. I remember having to reboot my phone a few times because I was not releasing the camera.

When SurfaceDestroyed gets called, we stop the preview let go of all the resources:

```
 public void surfaceDestroyed(SurfaceHolder holder) {
 // Surface will be destroyed when we return, so stop the preview.
 // Because the CameraDevice object is not a shared resource, it's
very
 // important to release it when the activity is paused.
 //Log.i(TAG,"SurfaceDestroyed being called");
 mCamera.stopPreview();
 mCamera.release();
 mCamera = null;
 }
```

On SurfaceChanged, we set the parameters for the camera, which in this case is landscape. At first I initially set the preview size by using some algorithm from google that didn't work well, so I just removed it and let it grab the current one set, and it seems to work very well. This is also where updateBufferSize() is called so that the buffer size is set for the next picture to be taken and we don't get some error for running out of memory. And another method to get the camera parameters.

```
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
 //Log.i(TAG, "Preview: surfaceChanged() - size now " + w + "x" + h);
 // Now that the size is known, set up the camera parameters and begin
 // the preview.
 try {
 mParameters = mCamera.getParameters();
 mParameters.set("orientation","landscape");
 for (Integer i : mParameters.getSupportedPreviewFormats()) {
 //Log.i(TAG, "supported preview format: " + i);
 }
 List<Size> sizes = mParameters.getSupportedPreviewSizes();
 for (Size size : sizes) {
 //Log.i(TAG, "supported preview size: " + size.width + "x" + size.height);
 }
 mCamera.setParameters(mParameters); // apply the changes
 } catch (Exception e) {
 // older phone - doesn't support these calls
 }
 //updateBufferSize(); // then use them to calculate
 Size p = mCamera.getParameters().getPreviewSize();
 //Log.i(TAG, "Preview: checking it was set: " + p.width + "x" + p.height); //
DEBUG
 mCamera.startPreview();
 }
 public Parameters getCameraParameters(){
 return mCamera.getParameters();
 }
```

Finally to set autoFocus and Flash we have two methods:

```
 public void setCameraFocus(AutoFocusCallback autoFocus){
 if
(mCamera.getParameters().getFocusMode().equals(mCamera.getParameters().FOCUS_MODE_AUTO)
||

mCamera.getParameters().getFocusMode().equals(mCamera.getParameters().FOCUS_MODE_MACRO))
{
 mCamera.autoFocus(autoFocus);
 }
 }
 public void setFlash(boolean flash){
 Toast.makeText(Preview.this.getContext(), "Flash is: "+mParameters.getFlashMode(),
Toast.LENGTH_LONG).show();
 if (flash){
 mParameters.setFlashMode(Parameters.FLASH_MODE_TORCH);
 mCamera.setParameters(mParameters);
 }
 else{
 mParameters.setFlashMode(Parameters.FLASH_MODE_OFF);
 mCamera.setParameters(mParameters);
 }
 }
```

TouchView.java

Things get progressively more difficult. This class is pretty important and the class that will create a canvas to draw on top of the camera preview. It implements View.
I will create Drawables for the corners of the bounding box and lines (Paint) to connect these boxes.

The constructor initializes our boxes and lines. The onDraw method draws them on top of the camera preview with the positions updated.
This update happens when the method invalidate() is called. Then the touch event method takes care of moving the bounding box.
First lets look at the data objects we need.

```
public class TouchView extends View {
 private Drawable mLeftTopIcon;
 private Drawable mRightTopIcon;
 private Drawable mLeftBottomIcon;
 private Drawable mRightBottomIcon;
 private boolean mLeftTopBool = false;
 private boolean mRightTopBool = false;
 private boolean mLeftBottomBool = false;
 private boolean mRightBottomBool = false;

 // Starting positions of the bounding box
 private float mLeftTopPosX = 30;
 private float mLeftTopPosY = 120;
 private float mRightTopPosX = 150;
 private float mRightTopPosY = 120;
 private float mLeftBottomPosX = 30;
 private float mLeftBottomPosY = 200;
 private float mRightBottomPosX = 150;
 private float mRightBottomPosY = 200;
 private float mPosX;
 private float mPosY;
 private float mLastTouchX;
 private float mLastTouchY;
 private Paint topLine;
 private Paint bottomLine;
 private Paint leftLine;
 private Paint rightLine;
 private Rect buttonRec;

 private int mCenter;
 private static final int INVALID_POINTER_ID = -1;
 private int mActivePointerId =
INVALID_POINTER_ID;
 // you can ignore this for this code
 private ScaleGestureDetector mScaleDetector;
 private float mScaleFactor = 1.f;
```

Paints will draw lines, Drawables are the corners, floats are the locations and Booleans are so that we can only move one corner at a time.

The constructors set the line colors, size, etc. It also sets the positions of the paints.

```
public TouchView(Context context){
super(context);
init(context);
}
public TouchView(Context context, AttributeSet attrs){
super (context,attrs);
init(context);
}
public TouchView(Context context, AttributeSet attrs, int defStyle) {
super(context, attrs, defStyle);
init(context);
}
private void init(Context context) {
// I need to create lines for the bouding box to connect
topLine = new Paint();
bottomLine = new Paint();
leftLine = new Paint();
rightLine = new Paint();
setLineParameters(Color.WHITE,2);

// Here I grab the image that will work as the corners of the bounding
// box and set their positions.
mLeftTopIcon = context.getResources().getDrawable(R.drawable.corners);

mCenter = mLeftTopIcon.getMinimumHeight()/2;
mLeftTopIcon.setBounds((int)mLeftTopPosX, (int)mLeftTopPosY,
mLeftTopIcon.getIntrinsicWidth()+(int)mLeftTopPosX,
mLeftTopIcon.getIntrinsicHeight()+(int)mLeftTopPosY);
mRightTopIcon = context.getResources().getDrawable(R.drawable.corners);
mRightTopIcon.setBounds((int)mRightTopPosX, (int)mRightTopPosY,
mRightTopIcon.getIntrinsicWidth()+(int)mRightTopPosX,
mRightTopIcon.getIntrinsicHeight()+(int)mRightTopPosY);
mLeftBottomIcon = context.getResources().getDrawable(R.drawable.corners);
mLeftBottomIcon.setBounds((int)mLeftBottomPosX, (int)mLeftBottomPosY,
mLeftBottomIcon.getIntrinsicWidth()+(int)mLeftBottomPosX,
mLeftBottomIcon.getIntrinsicHeight()+(int)mLeftBottomPosY);
mRightBottomIcon =
context.getResources().getDrawable(R.drawable.corners);
mRightBottomIcon.setBounds((int)mRightBottomPosX, (int)mRightBottomPosY,
mRightBottomIcon.getIntrinsicWidth()+(int)mRightBottomPosX,
mRightBottomIcon.getIntrinsicHeight()+(int)mRightBottomPosY);
// Create our ScaleGestureDetector
mScaleDetector = new ScaleGestureDetector(context, new ScaleListener());
}
private void setLineParameters(int color, float width){
topLine.setColor(color);
topLine.setStrokeWidth(width);
bottomLine.setColor(color);
bottomLine.setStrokeWidth(width);
leftLine.setColor(color);
leftLine.setStrokeWidth(width);
rightLine.setColor(color);
rightLine.setStrokeWidth(width);

}
```

OnDraw draws the bounding box on top of the camera preview. invalidate() needs to be called every time an update happens. This is done in the Event Listeners.

```
public boolean onTouchEvent(MotionEvent ev) {
final int action = ev.getAction();
boolean intercept = true;
switch (action) {
case MotionEvent.ACTION DOWN: {
```

```
 final float x = ev.getX();
 final float y = ev.getY();
 // in CameraPreview we have Rect rec. This is passed here to return
 // a false when the camera button is pressed so that this view ignores
 // the touch event.
 if ((x >= buttonRec.left) && (x <=buttonRec.right) && (y>=buttonRec.top) &&
(y<=buttonRec.bottom)){
 intercept = false;
 break;
 }
 // is explained below, when we get to this method.
 manhattanDistance(x,y);
 // Remember where we started
 mLastTouchX = x;
 mLastTouchY = y;
 mActivePointerId = ev.getPointerId(0);
 break;
 }
 case MotionEvent.ACTION_MOVE: {
 final int pointerIndex = ev.findPointerIndex(mActivePointerId);
 final float x = ev.getX();
 final float y = ev.getY();
 //Log.i(TAG,"x: "+x);
 //Log.i(TAG,"y: "+y);
 // Only move if the ScaleGestureDetector isn't processing a gesture.
 // but we ignore here because we are not using ScaleGestureDetector.
 if (!mScaleDetector.isInProgress()) {
 final float dx = x - mLastTouchX;
 final float dy = y - mLastTouchY;
 mPosX += dx;
 mPosY += dy;
 invalidate();
 }
 // Calculate the distance moved
 final float dx = x - mLastTouchX;
 final float dy = y - mLastTouchY;
 // Move the object
 if (mPosX >= 0 && mPosX <=800){
 mPosX += dx;
 }
 if (mPosY >=0 && mPosY <= 480){
 mPosY += dy;
 }
 // while its being pressed n it does not overlap the bottom line or right line
 if (mLeftTopBool && ((y+mCenter*2) < mLeftBottomPosY) && ((x+mCenter*2) <
mRightTopPosX)){
 if (dy != 0){
 mRightTopPosY = y;
 }
 if (dx != 0){
 mLeftBottomPosX = x;
 }
 mLeftTopPosX = x;//mPosX;
 mLeftTopPosY = y;//mPosY;
 }
 if (mRightTopBool && ((y+mCenter*2) < mRightBottomPosY) && (x >
(mLeftTopPosX+mCenter*2))){
 if (dy != 0){
 mLeftTopPosY = y;
 }
 if (dx != 0){
 mRightBottomPosX = x;
 }
 mRightTopPosX = x;//mPosX;
 mRightTopPosY = y;//mPosY;
 }
```

```
 if (mLeftBottomBool && (y > (mLeftTopPosY+mCenter*2)) && ((x +mCenter*2) <
mRightBottomPosX)){
 if (dx != 0){
 mLeftTopPosX = x;
 }
 if (dy != 0){
 mRightBottomPosY = y;
 }
 mLeftBottomPosX = x;
 mLeftBottomPosY = y;
 }
 if (mRightBottomBool && (y > (mLeftTopPosY+mCenter*2)) && (x >
(mLeftBottomPosX+mCenter*2) )){
 if (dx != 0){
 mRightTopPosX = x;
 }
 if (dy != 0){
 mLeftBottomPosY = y;
 }
 mRightBottomPosX = x;
 mRightBottomPosY = y;
 }
 // Remember this touch position for the next move event
 mLastTouchX = x;
 mLastTouchY = y;
 // Invalidate to request a redraw
 invalidate();
 break;
 }
 case MotionEvent.ACTION_UP: {
 // when one of these is true, that means it can move when onDraw is called
 mLeftTopBool = false;
 mRightTopBool = false;
 mLeftBottomBool = false;
 mRightBottomBool = false;
 //mActivePointerId = INVALID_POINTER_ID;
 break;
 }
 case MotionEvent.ACTION_CANCEL: {
 mActivePointerId = INVALID_POINTER_ID;
 break;
 }
 case MotionEvent.ACTION_POINTER_UP: {
 // Extract the index of the pointer that left the touch sensor
 final int pointerIndex = (action & MotionEvent.ACTION_POINTER_INDEX_MASK)
 >> MotionEvent.ACTION_POINTER_INDEX_SHIFT;
 final int pointerId = ev.getPointerId(pointerIndex);
 if (pointerId == mActivePointerId) {
 // This was our active pointer going up. Choose a new
 // active pointer and adjust accordingly.
 final int newPointerIndex = pointerIndex == 0 ? 1 : 0;
 mLastTouchX = ev.getX(newPointerIndex);
 mLastTouchY = ev.getY(newPointerIndex);
 mActivePointerId = ev.getPointerId(newPointerIndex);
 }
 break;
 }
 }
 return intercept;
 }
```

Notice that I am returning intercept boolean. Why? Because I am using onInterceptTouchEvent on the CameraPreview.java class. This allows me to pass the touch event to lower Views of the GUI.

In order to identify which corner is being pressed I used simple math. I used the Manhattan distance to calculate

when the user presses the screen, which of the corners it is close to and move it.

```
 // Where the screen is pressed, calculate the distance closest to one of the 4
corners
 // so that it can get the pressed and moved. Only 1 at a time can be moved.
 private void manhattanDistance(float x, float y) {
 double leftTopMan = Math.sqrt(Math.pow((Math.abs((double)x-
(double)mLeftTopPosX)),2)
 + Math.pow((Math.abs((double)y-(double)mLeftTopPosY)),2));
 double rightTopMan = Math.sqrt(Math.pow((Math.abs((double)x-
(double)mRightTopPosX)),2)
 + Math.pow((Math.abs((double)y-(double)mRightTopPosY)),2));
 double leftBottomMan = Math.sqrt(Math.pow((Math.abs((double)x-
(double)mLeftBottomPosX)),2)
 + Math.pow((Math.abs((double)y-(double)mLeftBottomPosY)),2));
 double rightBottomMan = Math.sqrt(Math.pow((Math.abs((double)x-
(double)mRightBottomPosX)),2)
 + Math.pow((Math.abs((double)y-(double)mRightBottomPosY)),2));
 //Log.i(TAG,"leftTopMan: "+leftTopMan);
 //Log.i(TAG,"RightTopMan: "+rightTopMan);
 if (leftTopMan < 50){
 mLeftTopBool = true;
 mRightTopBool = false;
 mLeftBottomBool = false;
 mRightBottomBool = false;
 }
 else if (rightTopMan < 50){
 mLeftTopBool = false;
 mRightTopBool = true;
 mLeftBottomBool = false;
 mRightBottomBool = false;
 }
 else if (leftBottomMan < 50){
 mLeftTopBool = false;
 mRightTopBool = false;
 mLeftBottomBool = true;
 mRightBottomBool = false;
 }
 else if (rightBottomMan < 50){
 mLeftTopBool = false;
 mRightTopBool = false;
 mLeftBottomBool = false;
 mRightBottomBool = true;
 }
 }
```

Then I need the setters and getters for the corner positions and finally a method for invalidating the view.

```
 public float getmLeftTopPosX(){
 return mLeftTopPosX;
 }
 public float getmLeftTopPosY(){
 return mLeftTopPosY;
 }
 public float getmRightTopPosX(){
 return mRightTopPosX;
 }
 public float getmRightTopPosY(){
 return mRightTopPosY;
 }
 public float getmLeftBottomPosX() {
 return mLeftBottomPosX;
 }
 public float getmLeftBottomPosY() {
 return mLeftBottomPosY;
 }
 public float getmRightBottomPosY() {
 return mRightBottomPosY;
 }
 public float getmRightBottomPosX() {
 return mRightBottomPosX;
 }
 public void setRec(Rect rec) {
 this.buttonRec = rec;
 }
 // calls the onDraw method, I used it in my app Translanguage
OCR
 // because I have a thread that needs to invalidate, or redraw
 // you cannot call onDraw from a thread not the UI thread.
 public void setInvalidate() {
 invalidate();

 }
```

If you have followed me all this way, then lets move to the final class.

CameraPreview.java

This is the class that takes care of putting everything together and this is why its the last class to be explained.

```
public class CameraPreview extends Activity implements SensorEventListener
{
 private Preview mPreview;
 private ImageView mTakePicture;
 private TouchView mView;
 private boolean mAutoFocus = true;
 private boolean mFlashBoolean = false;
 private SensorManager mSensorManager;
 private Sensor mAccel;
 private boolean mInitialized = false;
 private float mLastX = 0;
 private float mLastY = 0;
 private float mLastZ = 0;
 private Rect rec = new Rect();
 private int mScreenHeight;
 private int mScreenWidth;
 private boolean mInvalidate = false;
 private File mLocation = new File(Environment.
 getExternalStorageDirectory(),"test.jpg");
 @Override
 protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
//Log.i(TAG, "onCreate()");
requestWindowFeature(Window.FEATURE_NO_TITLE);
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
// display our (only) XML layout - Views already ordered
setContentView(R.layout.main);

// the accelerometer is used for autofocus
mSensorManager = (SensorManager) getSystemService(Context.
SENSOR_SERVICE);
mAccel = mSensorManager.getDefaultSensor(Sensor.
TYPE_ACCELEROMETER);
// get the window width and height to display buttons
// according to device screen size
DisplayMetrics displaymetrics = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(displaymetrics);
mScreenHeight = displaymetrics.heightPixels;
mScreenWidth = displaymetrics.widthPixels;
// I need to get the dimensions of this drawable to set margins
// for the ImageView that is used to take pictures
Drawable mButtonDrawable = this.getResources().
getDrawable(R.drawable.camera);
mTakePicture = (ImageView) findViewById(R.id.startcamerapreview);
// setting where I will draw the ImageView for taking pictures
LayoutParams lp = new LayoutParams(mTakePicture.getLayoutParams());
lp.setMargins((int)((double)mScreenWidth*.85),
(int)((double)mScreenHeight*.70) ,
(int)((double)mScreenWidth*.85)+mButtonDrawable.
getMinimumWidth(),
(int)((double)mScreenHeight*.70)+mButtonDrawable.
getMinimumHeight());
mTakePicture.setLayoutParams(lp);
// rec is used for onInterceptTouchEvent. I pass this from the
// highest to lowest layer so that when this area of the screen
// is pressed, it ignores the TouchView events and passes it to
// this activity so that the button can be pressed.
rec.set((int)((double)mScreenWidth*.85),
(int)((double)mScreenHeight*.10) ,
(int)((double)mScreenWidth*.85)+mButtonDrawable.getMinimumWidth(),
(int)((double)mScreenHeight*.70)+mButtonDrawable.getMinimumHeight());
mButtonDrawable = null;
mTakePicture.setClickable(true);
mTakePicture.setOnClickListener(flashListener);
// get our Views from the XML layout
mPreview = (Preview) findViewById(R.id.preview);
mView = (TouchView) findViewById(R.id.left_top_view);
mView.setRec(rec);
}
// this is the autofocus call back
private AutoFocusCallback myAutoFocusCallback = new AutoFocusCallback(){
public void onAutoFocus(boolean autoFocusSuccess, Camera arg1) {
//Wait.oneSec();
mAutoFocus = true;
}};
```

The onCreate method sets the accelerometer so it can be used to do autofocus for the camera. It also gets the screen's width and height so it can place the imageviews in the same spot for the different screen sizes that android devices have. Then we create the autofocus method.

The code below has a method called getRatio(). This method is what translates from screen size to pixels so that the bounding rectangle grabs the right part of the image.

Below after that is the onClick listener previewListener(below the flashListener). This is the method that takes the picture. It opens a thread to take and save the picture. Notice that it calls Preview.getPic(), which we talked about

before. This is the heart of saving the image in the sd card using savePhoto() which has not been explained or showed yet.

```
 // with this I get the ratio between screen size and pixels
 // of the image so I can capture only the rectangular area of the
 // image and save it.
 public Double[] getRatio(){
 Size s = mPreview.getCameraParameters().getPreviewSize();
 double heightRatio = (double)s.height/(double)mScreenHeight;
 double widthRatio = (double)s.width/(double)mScreenWidth;
 Double[] ratio = {heightRatio,widthRatio};
 return ratio;
 }
 // I am not using this in this example, but its there if you want
 // to turn on and off the flash.
 private OnClickListener flashListener = new OnClickListener(){
 @Override
 public void onClick(View v) {
 if (mFlashBoolean){
 mPreview.setFlash(false);
 }
 else{
 mPreview.setFlash(true);
 }
 mFlashBoolean = !mFlashBoolean;
 }
 };
 // This method takes the preview image, grabs the rectangular
 // part of the image selected by the bounding box and saves it.
 // A thread is needed to save the picture so not to hold the UI
thread.
 private OnClickListener previewListener = new OnClickListener() {
 @Override
 public void onClick(View v) {
 //if (mAutoFocus){
 mAutoFocus = false;
 mPreview.setFlash(false);
 //mPreview.setCameraFocus(myAutoFocusCallback);
 Wait.oneSec();
 Log.i("TESTESTEST","Taking picture");
 Thread tGetPic = new Thread( new Runnable() {
 public void run() {
 Double[] ratio = getRatio();
 int left = (int) (ratio[1]*(double)mView.getmLeftTopPosX());
 // 0 is height
 int top = (int) (ratio[0]*(double)mView.getmLeftTopPosY());
 int right = (int)(ratio[1]*(double)mView.getmRightBottomPosX());
 int bottom = (int)(ratio[0]*(double)mView.getmRightBottomPosY());
 savePhoto(mPreview.getPic(left,top,right,bottom));
 mAutoFocus = true;
 }
 });
 tGetPic.start();
 //}
 boolean pressed = false;
 if (!mTakePicture.isPressed()){
 pressed = true;
 }
 }
 };
 // just to close the app and release resources.
 @Override
 public boolean onKeyDown(int keyCode, KeyEvent event) {
 if (keyCode == KeyEvent.KEYCODE_BACK){
 finish();
```

```
}
return super.onKeyDown(keyCode, event);
}

private boolean savePhoto(Bitmap bm) {
FileOutputStream image = null;
try {
image = new FileOutputStream(mLocation);
} catch (FileNotFoundException e) {
e.printStackTrace();
}
bm.compress(CompressFormat.JPEG, 100, image);
if (bm != null) {
int h = bm.getHeight();
int w = bm.getWidth();
//Log.i(TAG, "savePhoto(): Bitmap WxH is " + w + "x" + h);
} else {
//Log.i(TAG, "savePhoto(): Bitmap is null..");
return false;
}
return true;
}
```

Below is the onInterceptTouch(). This is what allows the take picture button to be pressed and the TouchView onTouchEvents to be ignored.

```
public boolean onInterceptTouchEvent(MotionEvent ev) {
final int action = ev.getAction();
boolean intercept = false;
switch (action) {
case MotionEvent.ACTION_UP:
break;
case MotionEvent.ACTION_DOWN:
float x = ev.getX();
float y = ev.getY();
// here we intercept the button press and give it to this
// activity so the button press can happen and we can take
// a picture.
if ((x >= rec.left) && (x <= rec.right) && (y>=rec.top) &&
(y<=rec.bottom)){
intercept = true;
}
break;
}
return intercept;
}


// mainly used for autofocus to happen when the user takes a picture
// I also use it to redraw the canvas using the invalidate() method
// when I need to redraw things.
public void onSensorChanged(SensorEvent event) {
if (mInvalidate == true){
mView.invalidate();
mInvalidate = false;
}
float x = event.values[0];
float y = event.values[1];
float z = event.values[2];
if (!mInitialized){
mLastX = x;
mLastY = y;
mLastZ = z;
mInitialized = true;
}
```

```
 float deltaX = Math.abs(mLastX - x);
 float deltaY = Math.abs(mLastY - y);
 float deltaZ = Math.abs(mLastZ - z);
 if (deltaX > .5 && mAutoFocus){ //AUTOFOCUS (while it is not autofocusing)
 mAutoFocus = false;
 //mPreview.setCameraFocus(myAutoFocusCallback);
 }
 if (deltaY > .5 && mAutoFocus){ //AUTOFOCUS (while it is not autofocusing)
 mAutoFocus = false;
 //mPreview.setCameraFocus(myAutoFocusCallback);
 }
 if (deltaZ > .5 && mAutoFocus){ //AUTOFOCUS (while it is not autofocusing)
*/
 mAutoFocus = false;
 //mPreview.setCameraFocus(myAutoFocusCallback);
 }
 mLastX = x;
 mLastY = y;
 mLastZ = z;
 }
 // extra overrides to better understand app lifecycle and assist debugging
 @Override
 protected void onDestroy() {
 super.onDestroy();
 //Log.i(TAG, "onDestroy()");
 }
 @Override
 protected void onPause() {
 super.onPause();
 //Log.i(TAG, "onPause()");
 mSensorManager.unregisterListener(this);
 }
 @Override
 protected void onResume() {
 super.onResume();
 mSensorManager.registerListener(this, mAccel,
SensorManager.SENSOR_DELAY_UI);
 //Log.i(TAG, "onResume()");
 }
 @Override
 protected void onRestart() {
 super.onRestart();
 //Log.i(TAG, "onRestart()");
 }
 @Override
 protected void onStop() {
 super.onStop();
 //Log.i(TAG, "onStop()");
 }
 @Override
 protected void onStart() {
 super.onStart();
 //Log.i(TAG, "onStart()");
 }
 public void onAccuracyChanged(Sensor sensor, int accuracy) {
 // TODO Auto-generated method stub
 }
```

The onSensorChanged takes care of not only causing the autofocus but also invalidating (calling the onDraw()) the TouchView when a boolean is set to true.

Finally we have the Activity lifecycle methods that release resources (camera and sensors) onPause() and registers them onStart(). That is it guys. Try it out and make some cool apps.