
UNIVERSITà DEGLI STUDI DI PERUGIA

# RELAZIONE

DEL PROGETTO DI INGEGNERIA DEL SOFTWARE 2019/2020

*PROF. ALFREDO MILANI*

# App Telecomando Gesture web

**CRISTIAN COSCI**

# Indice

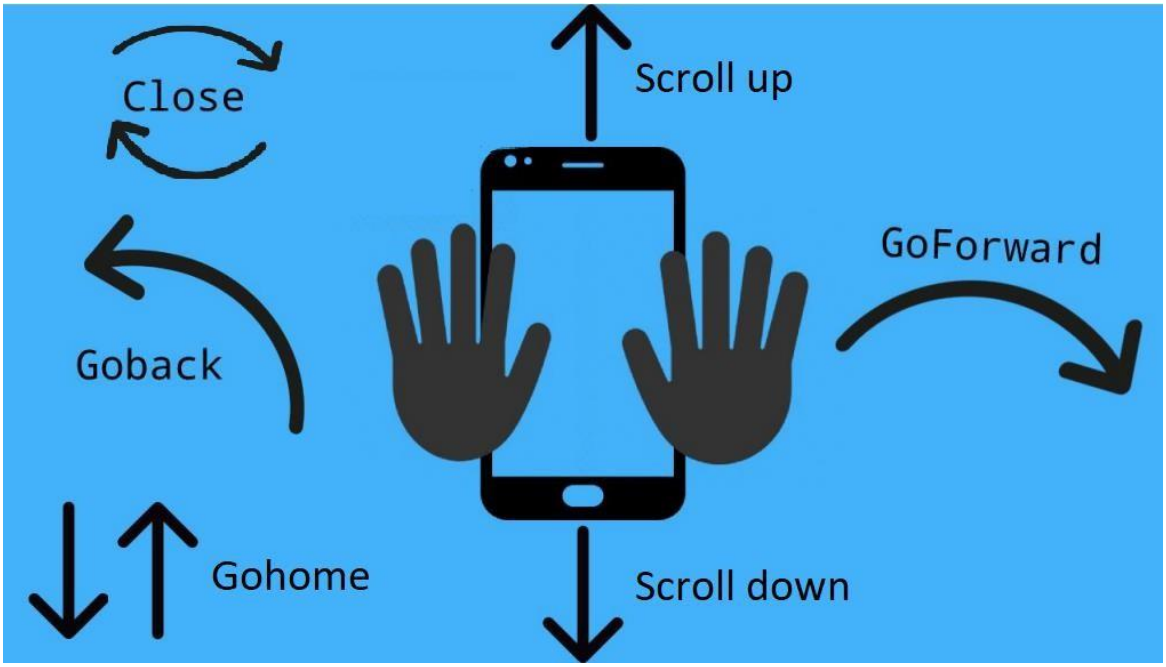
# 1 INTRODUZIONE

*Sommario*

Si descrive l'implementazione di un applicativo android che permette l'esecuzione di azioni: avanti, indietro, home, scroll su, scroll giu su una pagina web visualizzata da un browser in un secondo dispositivo.

## 1.1 Obbiettivi

Implementare un applicativo android che riconosce le gesture effettuate muovendo o ruotando il cellulare con uso dell'accelerometro/bussola/giroscopio, per provocare azioni su pagina visualizzata da un browser terzo, es. start, next, previous, end, home.




## 1.2 Glossario

***GESTURE:*** Con questo termine intendiamo tutti i movimenti del telefono che possono essere riconosciuti dall'accelerometro e dal giroscopio, come per esempio l'inclinazione del dispositivo.

***GIROSCOPIO:*** Serve a misurare l'inclinazione dello smartphone.

***ACCELEROMETRO:*** É un sensore che serve a misurare l'accelerazione del dispositivo rispetto alla caduta a terra.

***SOCKET:*** Un socket, in informatica, indica un'astrazione software progettata per utilizzare delle API standard e condivise per la trasmissione e la ricezione di dati attraverso una rete oppure come meccanismo di IPC. (Concatenazione di ip e porta)

***CLIENT:*** Un client, in informatica, nell'ambito delle reti informatiche e dell'architettura logica di rete detta client-server, indica genericamente una qualunque componente software, presente tipicamente su una macchina host, che accede ai servizi di un'altra componente detta server, attraverso l'uso di determinati protocolli di comunicazione.

***SERVER:*** Un server, in informatica è un componente e sottosistema informatico di elaborazione e gestione del traffico di informazioni che fornisce, a livello logico e fisico, un qualunque tipo di servizio ad altre componenti (solitamente chiamate client) che ne fanno richiesta attraverso una rete di computer, all'interno di un sistema informatico o anche direttamente in locale su un computer.

***BROWSER:*** In informatica il browser è un'applicazione per l'acquisizione, la presentazione e la navigazione di risorse sul web.

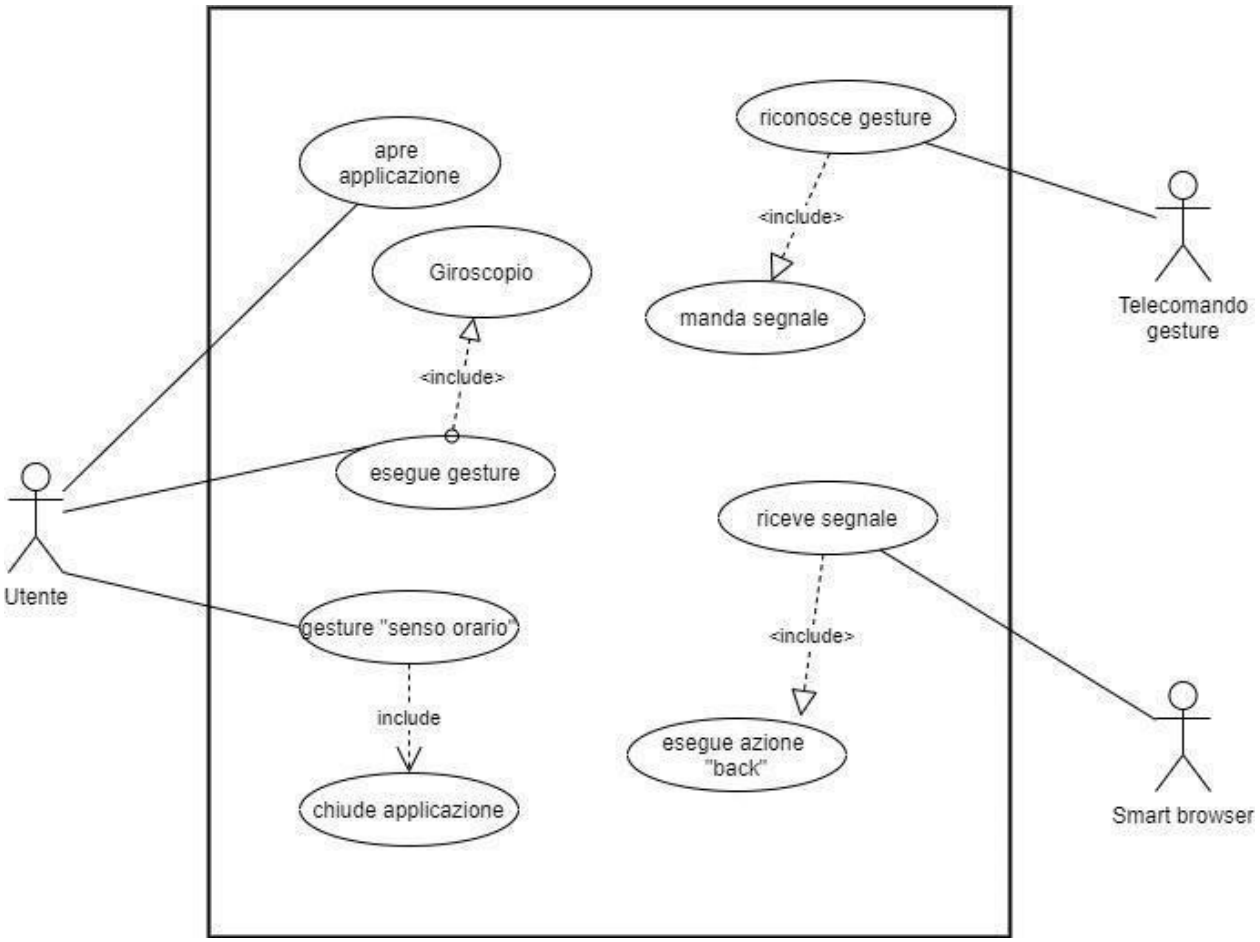
***THREAD:*** Un thread (o thread di esecuzione), in informatica, è una suddivisione di un processo in due o più filoni (istanze) o sottoprocessi che vengno eseguiti concorrentemente da un sistema di elaborazione monoprocessore (multithreading) o multiprocessore o multicore.

# 2 ANALISI DEI REQUISITI

- Effettuare azioni di comando su un browser web su un dispositivo terzo a quello in cui vengono impartiti i comandi.
- I vari comandi vengono riconosciuti mediante gesture effettuate sul dispositivo con il ruolo di telecomando.
- I sensori utilizzati per riconoscere le gesture sul dispositivo sono accelerometro e giroscopio.
- Vi sono due applicazioni: una funge da browser in cui vengono eseguite le azioni impartite dall'altro dispositivo; una funge da telecomando e ha lo scopo di riconoscere le gesture effettuate e di inviare i comandi all'app browser.
- Le due app necessitano di comunicare e scambiarsi informazioni e la decisione più ovvia è stata quella di optare per una comunicazione client-server mediante socket. L'applicazione telecomando funge quindi da server in quanto abbiamo pensato che un possibile miglioramento dell'applicazione (con un piccolo aggiornamento) sia quello di poter scegliere quale dispositivo controllare(comandare) tra tutti quelli collegati al telecomando, mentre l'app browser funge da client.
- Mettere a disposizione un'interfaccia grafica in entrambe le applicazioni semplice, minimale e intuitiva per l'utilizzo da parte dell'utente.

# 2.1 Descrizione e diagrammi dei casi d'uso

In fase di revisione dei requisiti, siamo andati a definire con maggior precisione i requisiti funzionali del nostro applicativo tramite l'utilizzo del diagramma di caso d'uso.




### *A)* SPECIFICA DI CASO D'USO: *"apre applicazione"*

Personaggio principale: utente

Precondizioni: nessuna

Scenario principale: L'utente apre entrambe le applicazioni e connette il server con il client

Postcondizioni: Le due applicazioni adesso sono connesse.

### *B)* SPECIFICA DI CASO D'USO: *"esegue gesture"*

Personaggio principale: utente

Precondizioni: aver connesso il client con il server

Scenario principale: L'utente esegue una gesture mettendo in azione il giroscopio.

Postcondizioni: La gesture effettuata provocherà un'azione sul browser

### *C)* SPECIFICA DI CASO D'USO: *"riconosce gesture"*

Personaggio principale: Telecomando gesture

Precondizioni: aver eseguito una gesture

Scenario principale: L'applicazione riesce a riconoscere la gesture effettuata dall'utente e manda un messaggio al Browser.

Postcondizioni: Il browser adesso è in grado di eseguire i comandi perchè ha ricevuto il messaggio.

### *D)* SPECIFICA DI CASO D'USO: *"riceve segnale"*

Personaggio principale: Smart Browser

Precondizioni: aver eseguito una gesture

Scenario principale: Viene letto il messaggio inviato dal telecomando e in base al contenuto di esso viene eseguito il comando.

Postcondizioni: I comandi scelti dall'utente sono stati eseguiti.

# 3 ARCHITETTURA DEL SISTEMA

## 3.1 Scelte architetturali

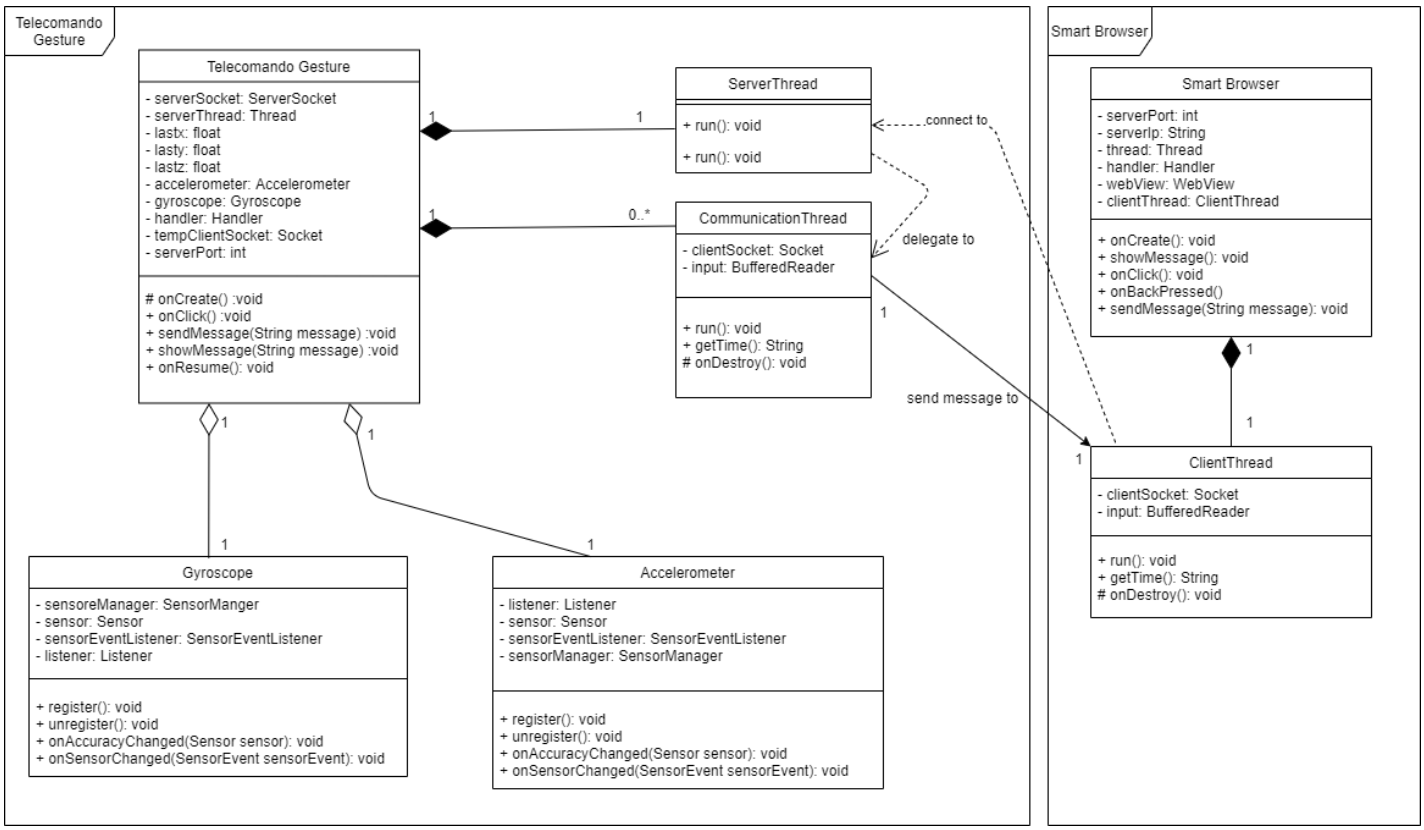
Un'applicazione per dispositivi android può essere scritta in linguaggi come Kotlin e Java. A causa di esperienze pregresse con il linguaggio Java si è deciso di implementare le due applicazioni con l'utilizzo di quest'ultimo. Come IDE, ovvero come ambiente di sviluppo, è stato utilizzato Android Studio.

## 3.2 Architettura di Implementazione

Di seguito sono presentati e descritti i vari diagrammi UML per la progettazione delle due applicazioni (Telecomando Gesture e Smart Browser) relative al progetto.

## 3.2.1 Diagramma di Classe

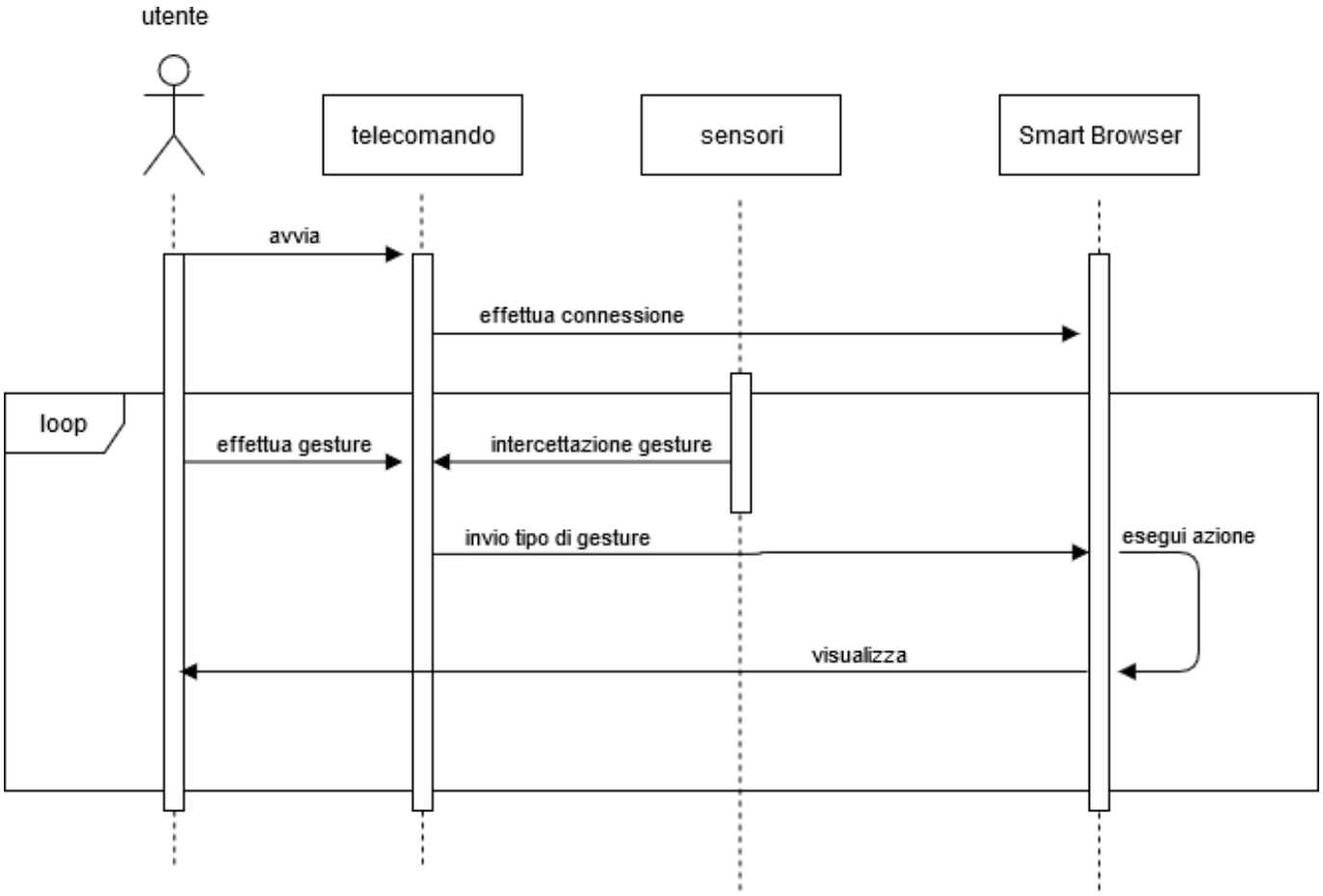
Si descrivono le classi presenti nella logica del nostro applicativo.



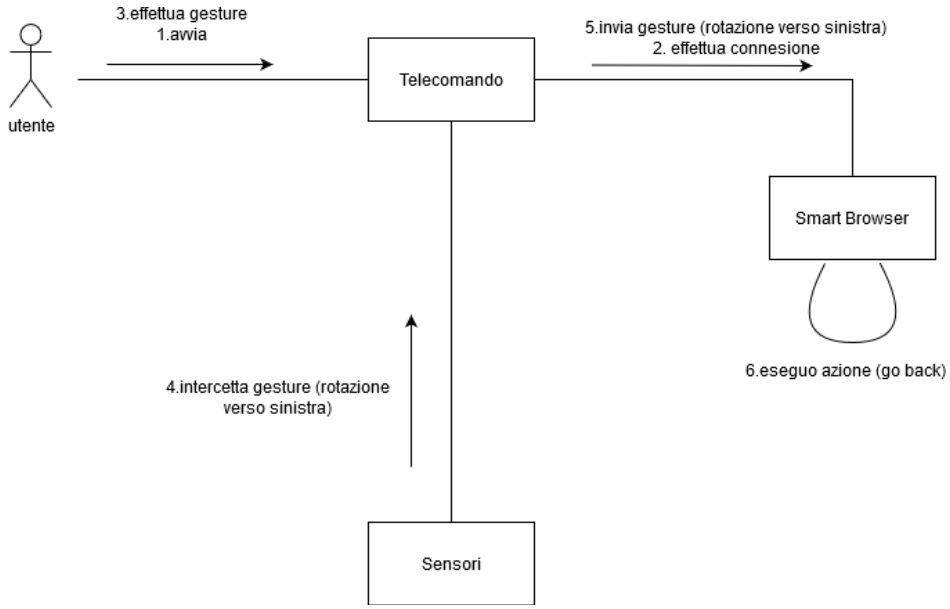

Distinguiamo 2 parti principali: le classi relative all'applicativo "Telecomando Gesture" e quelle relative allo "Smart Browser".

Come possiamo intuire deve esservi una comunicazione continua tra i due applicativi, ela scelta per noi migliore era mediante socket in comunicazione "client-server", quindi entrambi gli applicativi non potranno funzionare senza un thread relativo al server (per il telecomando) e un thread relativo al client (per il browser). Vi sarà un'associazione di dipendenza tra i due thread dato che dovrà essere effettuata una connessione, in seguito a quest'ultima verrà creato un nuovo thread (delegazione) che si occuperà della comunicazione con il relativo client. Ovviamente avremo anche le due classi relative ai sensori Giroscopio e Accelerometro in un'associazione di aggregazione in quanto nella nostra visione sono considerate come una componente della classe telecomando.

### 3.2.2 Diagramma di Sequenza e di Collaborazione



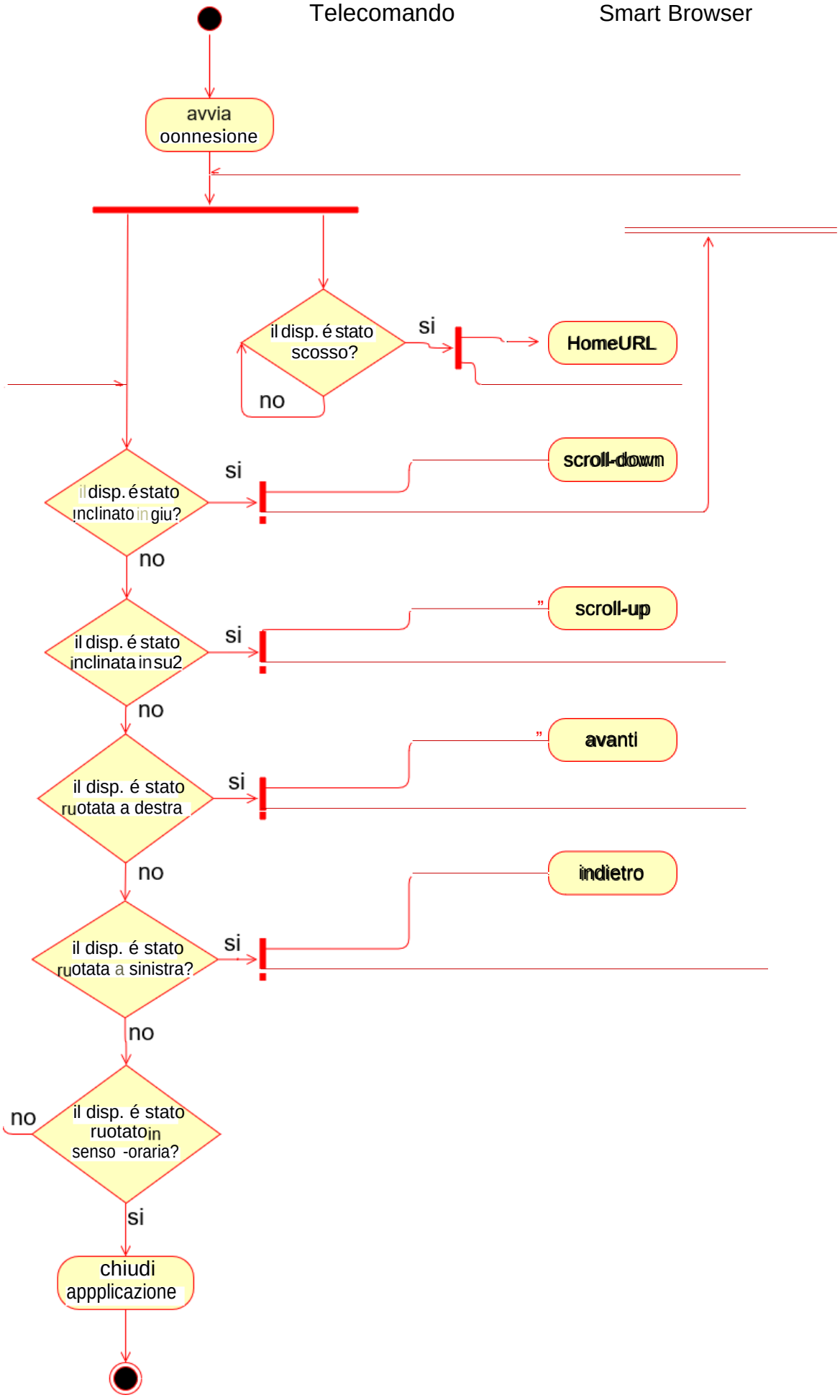

Come rappresentato il nostro sistema mette in gioco varie componenti: il nostro attore principale si interfaccia con una componente ”telecomando", che una volta avviato e connesso eseguirà un loop nel quale andrà a rilevare le gesture effettuate, inviandolo quindi alla componente “smart browser”, dove in base al segnale ricevuto effettua una precisa azione, come può essere il go-back.



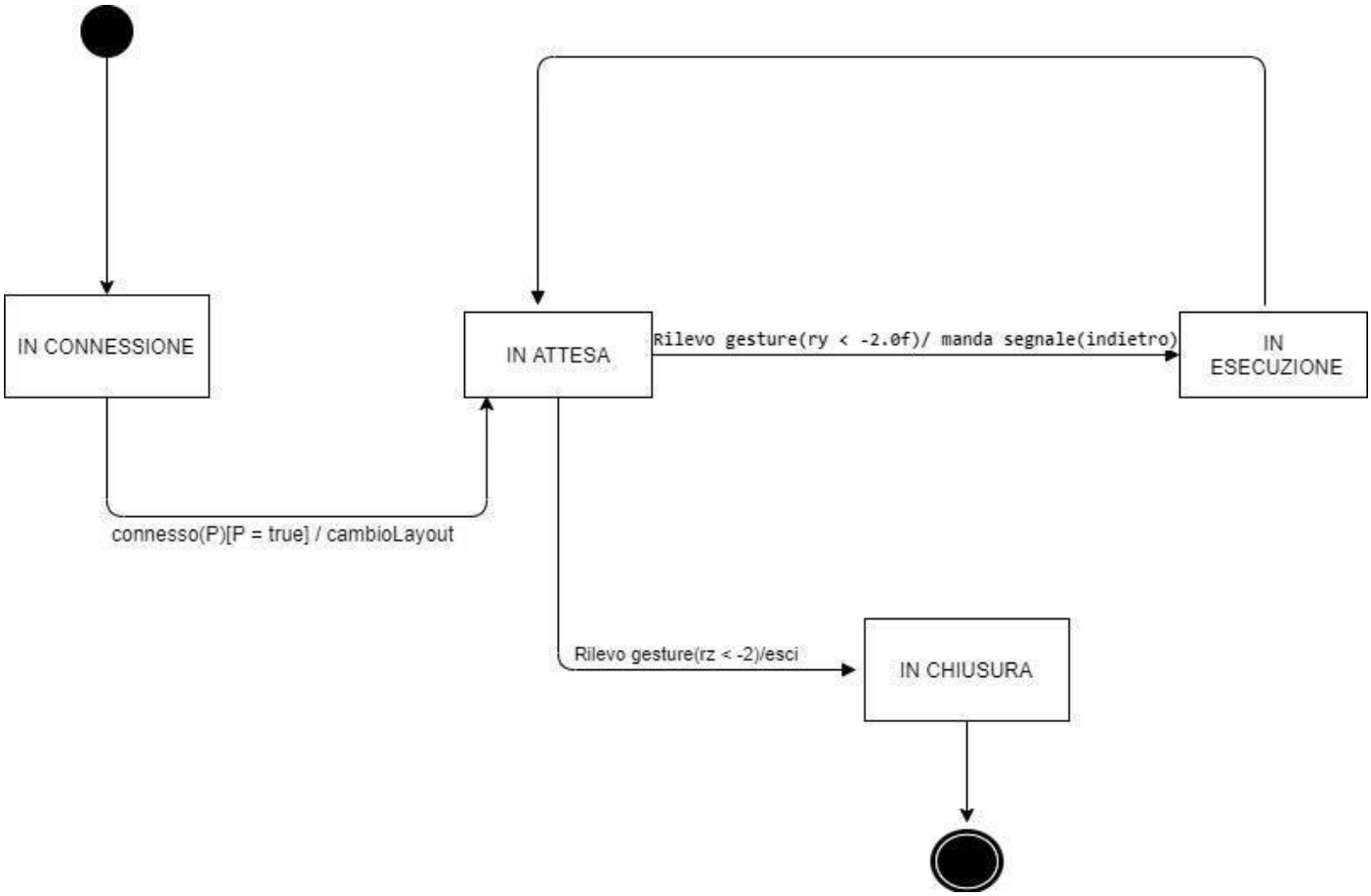

### 3.2.3 Diagramma di Attività

L'utente avvia il telecomando, dopo aver effettuato la connesione può gia iniziare ad usare l'app telecomando.

Già dalla prima gesture i sensori rilevano il tipo di movimento effettuato, inviandolo allo smart browser che esegue l'azione richiesta, tutto ciò viene ripetuto finche non viene deciso di chiudere l'app attraverso la rotazione in senso orario del telecomando.


Telecomando
Smart Browser
avvia oonnesione
il disp. é stato scosso?
si
no
HomeURL
il disp. é stato inclinato in giu?
si
scroll-down
no
il disp. é stato inclinata in su2
si
scroll-up
no
il disp. é stato ruotata a destra
si
avanti
no
il disp. é stato ruotata a sinistra?
si
indietro
no
il disp. é stato ruotato in senso -oraria?
no
si
chiudi appplicazione

### 3.2.4 Diagramma di Stato




L'applicazione Telecomando Gesture si trova prima in uno stato di "connessione", ovvero quando cerca di connettersi con lo Smart Browser, una volta che si sono connessi passa in uno stato di "attesa", aspetta cioè che l'utente compia qualche gesture valida, e nel caso ne esegua una (diversa dal giro in senso orario) passa allo stato di "esecuzione", invia cioè un messaggio allo Smart Browser, per poi ritornare allo stato di attesa. Qualora l'utente esegua la gesture "giro orario" l'applicazione andrà in uno stato di "chiusura" per poi terminare.

# 4 Realizzazione ed Implementazione

Si passa ora alla descrizione del codice sorgente relativo ai due applicativi “Telecomando Gesture” e “Smart browser”. Il codice è scritto in linguaggio di programmazione Java.

## 4.1 Codice sorgente e spiegazione

### *4.1.1 Telecomando Gesture*

Qui di seguito è possibile vedere gli import effettuati nella classe del Telecomando, saranno gli stessi utilizzati anche nell'altro applicativo relativo al Browser, ma per ovviare a ripetizioni li riporteremo solo una volta.

```
import android.os.Build;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import android.graphics.Color;
import android.os.Handler;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;
```

Possiamo vedere i vari import relativi ai Buffer per inviare e ricevere messaggi tra i due applicativi, e possiamo vedere anche gli import necessari all'implementazione di una connessione client-server.

Qui di seguito vediamo invece l'implementazione della classe principale per il Telecomando, la classe "MainActivity" ovvero il corpo vero e proprio del programma, cioè la funzione che avrà il controllo dell'esecuzione.

Possiamo vedere tra le dichiarazioni e le inizializzazioni, quelle relative al server, ai sensori giroscopio e accelerometro e al layout, nello specifico ad una textView che sarà a schermo fino al momento di una connessione con un client.

```java
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private ServerSocket serverSocket;
    private Socket tempClientSocket;
    Thread serverThread = null;
    public static final int SERVER_PORT = 3003; //porta alla quale effettuare connessione al server
    private LinearLayout msgList;
    private Handler handler;
    private int greenColor;

    private boolean waspiano = true;
    private boolean notFirstTime = true;
    private float lastx,lasty,lastz,differencex,differencey,differencez, max=15.0f; //assi x, y, z per
sensori di movimento
    private Accelerometer accelerometer; //oggetto che permette l'utilizzo dell'accelerometro per
rilevare le gesture
    private Gyroscope gyroscope; //oggetto che permetto l'utilizzo del giroscopio per rilevare le
gesture

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //settaggio layout all'apertura dell'app
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setTitle("Telecomando gesture");
        greenColor = ContextCompat.getColor(this, R.color.colorAccent);
        handler = new Handler();
        msgList = findViewById(R.id.msgList);

        accelerometer = new Accelerometer(this);
        gyroscope = new Gyroscope(this);
        //metodo che riconosce la gesture "shake" mediante l'accelerometro e l'asse z, inviando poi il
messaggio al client
        accelerometer.setListener(new Accelerometer.Listener() {
            @Override
            public void onTranslation(float tx, float ty, float tz) {
                if(notFirstTime){
                    differencez=Math.abs(lastz-tz);
                    if(differencez > max){
                        sendMessage("shake");
                    }
                }
                lastz=tz;
                notFirstTime=true;
            }
        });
```

Possiamo vedere poi il settaggio dei due sensori mediante le classi Accelerometer e Gyroscope (che spiegheremo in seguito). Il funzionamento principale consiste nel confrontare i valori dei sensori mediante gli assi del piano cartesiano e se nota dei cambiamenti (in base alla sensibilità impostata da noi) rileva il movimento specifico. Possiamo vedere che l'asse z relativo all'accelerometro andrà ad identificare uno "shake" del dispositivo, mentre lo stesso asse relativo al giroscopio andrà a rilevare una rotazione

(che se effettuata in senso orario andrà a mandare il segnale per chiudere il dispositivo). Gli assi x e y invece sono relativi all'inclinazione e alla rotazione del telefono verso destra o sinistra (lungo il lato veritcale) del dispositivo.

```
//metodo che riconosce le gesture di rotazione e inclinazione mediante il giroscopip e gli assi x, y,
z, inviando poi il messaggio al client
        gyroscope.setListener(new Gyroscope.Listener() {
            @Override
            public void onRotation(float rx, float ry, float rz) {
                if(rx>2.0f && waspiano){//telefono inclinato in giu
                    waspiano=false;
                    sendMessage("giu");
                }else if(rx<-2.0f  && waspiano){//telefono inclinato in su
                    waspiano=false;
                    sendMessage("su");
                } else if(ry>2.0f  && waspiano){//ruoto il telefono verso destra
                    waspiano=false;
                    sendMessage("destra");
                } else if(ry<-2.0f  && waspiano){//ruoto il telefono verso sinistra
                    waspiano=false;
                    sendMessage("sinistra");
                } else if((rx<=1.0f)&&(rx>=-1.0f)&&(ry>=-1.0f)&&(ry<=1.0f)&&(rz<=1.0f)&&(rz>=-1.0f)&&
(!waspiano)){
                    waspiano=true;
                    //sendMessage("il telefono è sul tavolo");
                } else if(rz<-2.0f  && waspiano) {//ruoto il telefono in senso orario
                    waspiano = false;
                    sendMessage("chiudi");
                    try { //prima di chiudere l'app attendo 1 secondo
                        TimeUnit.SECONDS.sleep(1);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    //in base alla versione android del dispositivo vi è un diverso metodo di chiusura
dell'app. Nel caso di
                    //android superiore alla verione LOLLIPOP l'applicazione viene chiusa e rimossa dai
task, altrimenti viene solo chiusa
                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                        finishAndRemoveTask();
                    } else{
                        finish();
                    }
                }
            }
        });
```

Possiamo vedere qui la funzione dedicata all'invio dei segnali (messaggi) al client e viceversa. Il metodo come già accennato precedentemente fa utilizzo di uno stream come canale di comunicazione.

```
    private void sendMessage(final String message) { //funzione per inviare messaggi al client mediante
bufferWriter
        try {
            if (null != tempClientSocket) {
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        PrintWriter out = null;
                        try {
                            out = new PrintWriter(new BufferedWriter(
                                    new OutputStreamWriter(tempClientSocket.getOutputStream())),
                                    true);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        out.println(message);
                    }
                }).start();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Qui invece abbiamo l'implementazione del Server nella classe Telecomando. Come possiamo vedere andremo a definire il metodo "run( )" che costituirà il codice da eseguire per il thread relativo al server. Andiamo inizialmente a creare un Server Socket (concatenazione di ip e porta) per permettere di connettersi ad un client. Viene poi definita la classe CommunicationThread che servirà per instanziare un thread per ogni client di comunicazione una volta connesso, permettendo quindi a più client di connettersi al server.

```
public void run() {
    Socket sockt;
        serverSocket = new ServerSocket(SERVER_PORT);
    } catch (IOException e) {
        e.printStackTrace();
    }
    if (       != serverSocket) {
        while (!Thread.currentThread().isInterrupted()) {
            socket = serverSocket.accept();
            CommunicationThread commThread = new CommunicationThread(socket);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
public CommunicationThread(Socket clientSocket) {
        this.input = new BufferedReader(new
InputStreamReader(this.clientSocket.getInputStream()));
    } catch (IOException e) {
        e.printStackTrace();
        showMessage("Error Connecting to Client!!", Color.RED);
        TimeUnit.SECONDS.sleep(2);
    } catch (InterruptedException e) {
        e.printStackTrace();
    handler.post(new Runnable() {
        public void run() {
        }
    });
}
```

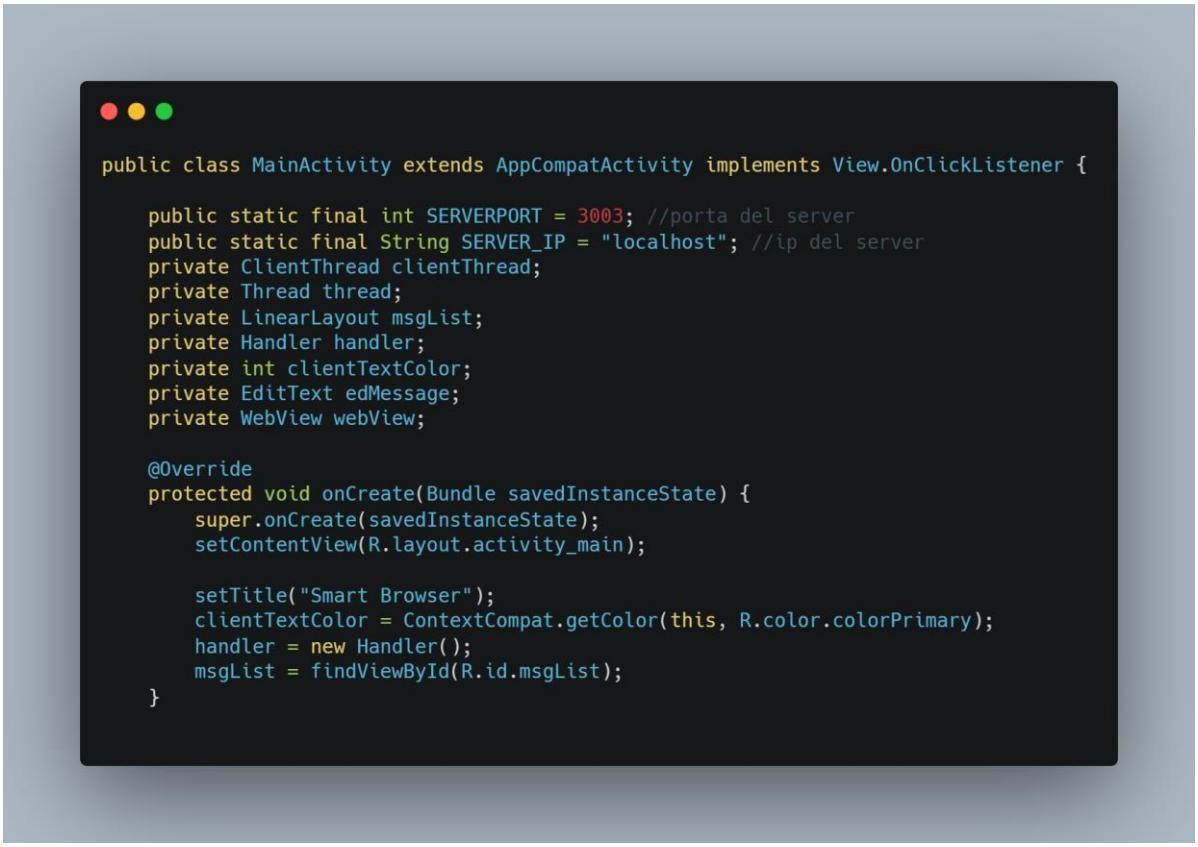
```
while (!Thread.currentThread().isInterrupted()) {
    try {
        String read = input.readLine();
        if (null == read || "Disconnect".contentEquals(read)) {
            read = "Client Disconnected";
            showMessage("Client : " + read, greenColor);
            break;
        }
        showMessage("Client : " + read, greenColor)
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Passiamo qui alla classe relativa al Giroscopio (l'implementazione è uguale a quella dell'accelerometro quindi per evitare ripetizioni riporteremo il codice solo di questo sensore). Possiamo vedere che andiamo ad utilizzare l'interfaccia Listener per la raccolta dei dati dai relativi sensori.

```java
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;

public class Gyroscope {
    public interface Listener{
        void onRotation(float rx, float ry, float rz); //sensori sui tre assi del piano cartesiano
    }

    private Listener listener;

    public void setListener (Listener l){ //listener per la raccolta dei valori relativi ai movimenti
del dispositivo
        listener=l;
    }

    private SensorManager sensorManager;
    private Sensor sensor;
    private SensorEventListener sensorEventListener;

    Gyroscope(Context context){
        sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
        sensor = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
        sensorEventListener = new SensorEventListener() {
            @Override
            public void onSensorChanged(SensorEvent sensorEvent) {
                if(listener!=null){

 listener.onRotation(sensorEvent.values[0],sensorEvent.values[1],sensorEvent.values[2]);
                }
            }

            @Override
            public void onAccuracyChanged(Sensor sensor, int i) {

            }
        };
    }
```

## *4.1.2 Smart Browser*

Passiamo ora a mostrare il codice del secondo applicativo, eviteremo le ripetizioni e mostreremo solo le parti più importanti e caratterizzanti. Iniziamo con l'inizializzazione di ip e porta del server (in questo caso l'ip è settato su "localhost" per permettere di testare i due applicativi dallo stesso emulatore, ma è possibile cambiare ip con quello del dispositivo ospite e testarlo anche con due dispositivi diversi connessi alla stessa rete in locale).

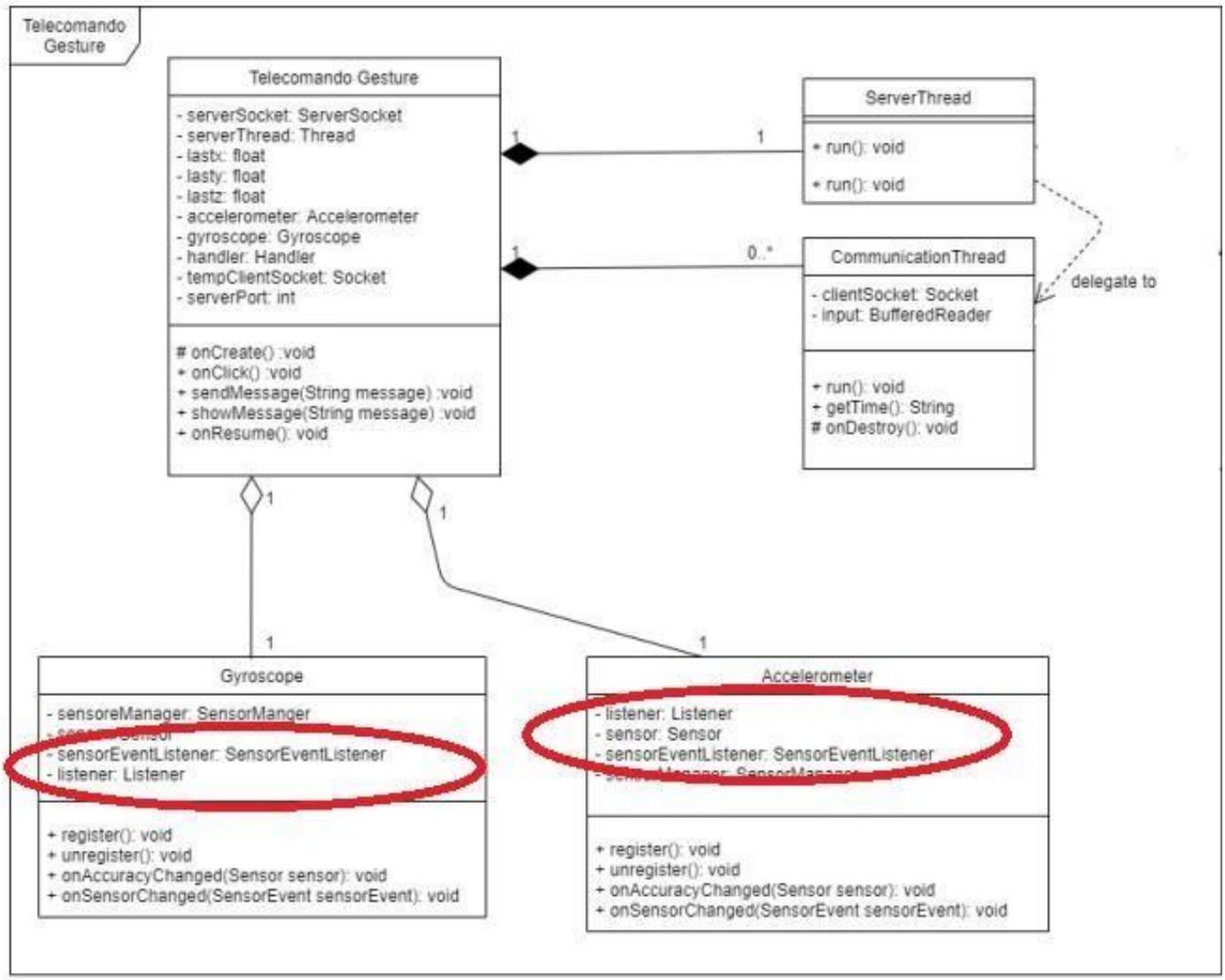


```java
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    public static final int SERVERPORT = 3003; //porta del server
    public static final String SERVER_IP = "localhost"; //ip del server
    private ClientThread clientThread;
    private Thread thread;
    private LinearLayout msgList;
    private Handler handler;
    private int clientTextColor;
    private EditText edMessage;
    private WebView webView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        setTitle("Smart Browser");
        clientTextColor = ContextCompat.getColor(this, R.color.colorPrimary);
        handler = new Handler();
        msgList = findViewById(R.id.msgList);
    }
```

Andiamo infine a mostrare la parte in cui vengono eseguite le azioni in base alla gesture effettuata dal dispositivo Telecomando. Avendo a visualizzazione una WebView andremo quindi a impartire comandi relativi alla specifica classe. Infine nelle ultime linee di codice vediamo come viene eseguita l'impartizione del comando di chiusura dell'applicazione in base alla diversa versione del sistema operativo.

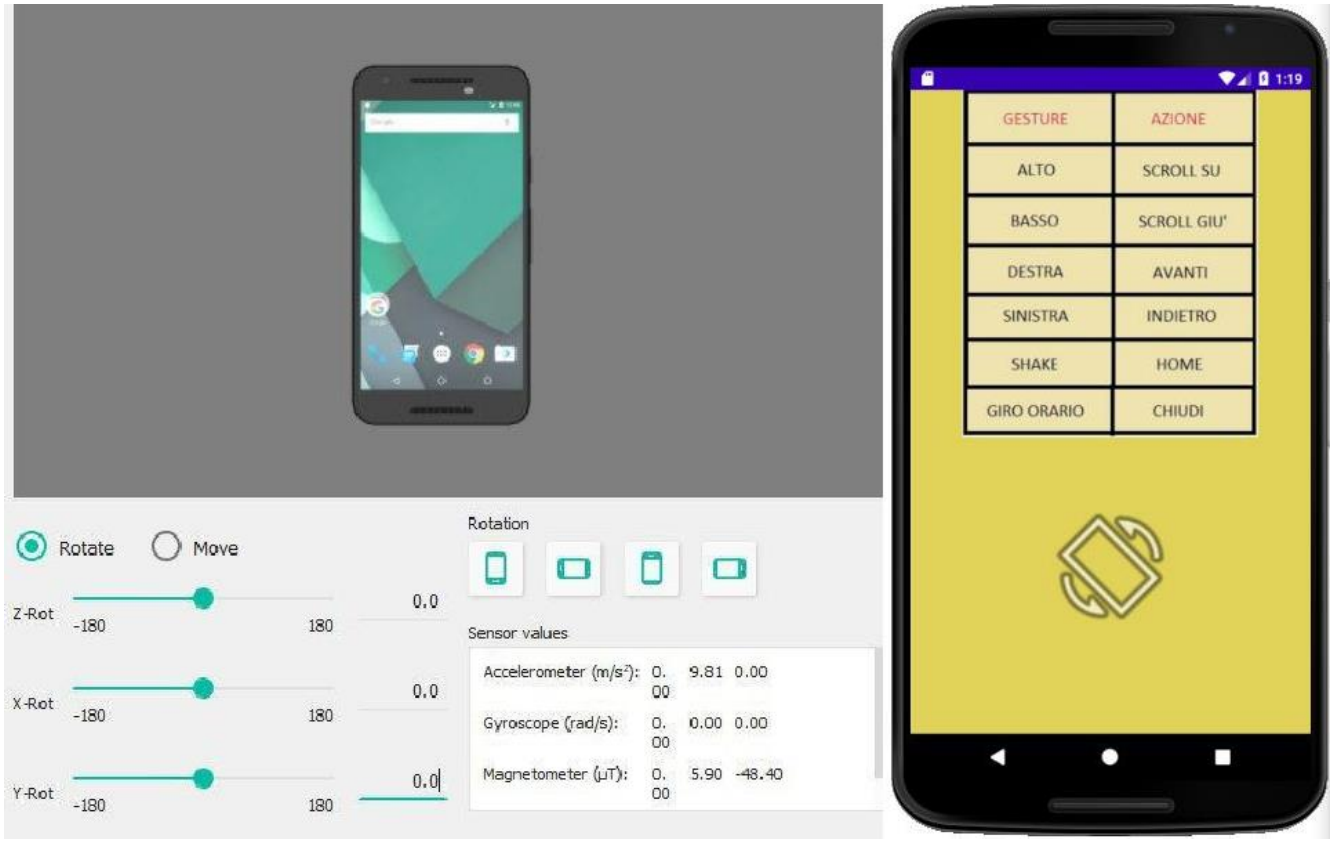
# 5 Design Pattern

Il design pattern che abbiamo utilizzato è l'Observer, poichè avevamo il problema di capire quando la gesture veniva effettuata per poter mandare un messaggio al Browser. Sostanzialmente il pattern si basa su uno o più oggetti, chiamati osservatori o observer, che vengono registrati per gestire un evento che potrebbe essere generato dall'oggetto "osservato", che può essere chiamato soggetto. Per poter risolvere il problema abbiamo usato il giroscopio e l'accelerometro come oggetti "osservati", in modo tale da essere in grado di capire quando cambiassero stato. Una volta ottenuto questo segnale controlliamo se la gesture sia valida. Come possiamo vedere anche dall'implementazione abbiamo utilizzato oggetti di tipo **Listener e SensorEventListener.**

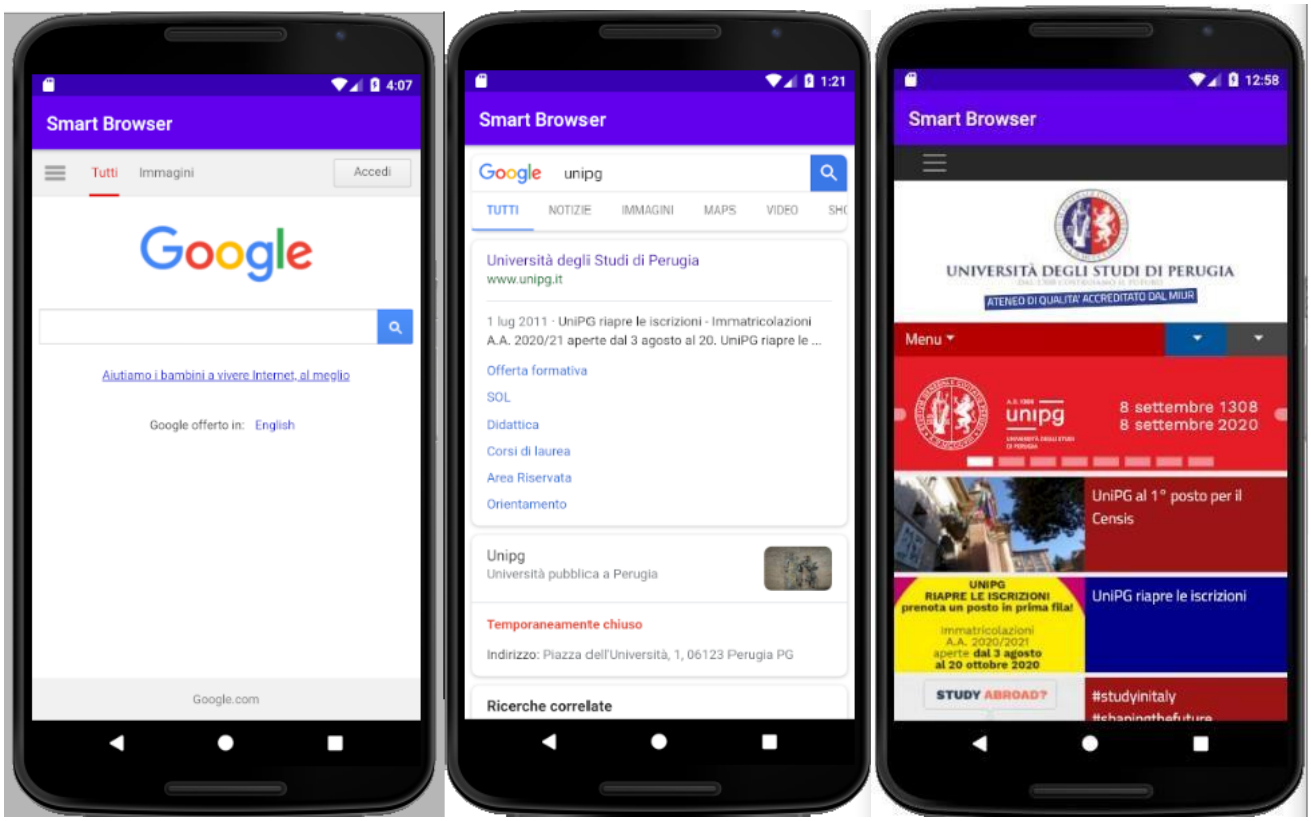


# 6 Test

Dopo aver avviato e connesso entrambe le nostre app procediamo nel testare le gesture che possiamo rilevare partendo dalla seguente situazione:

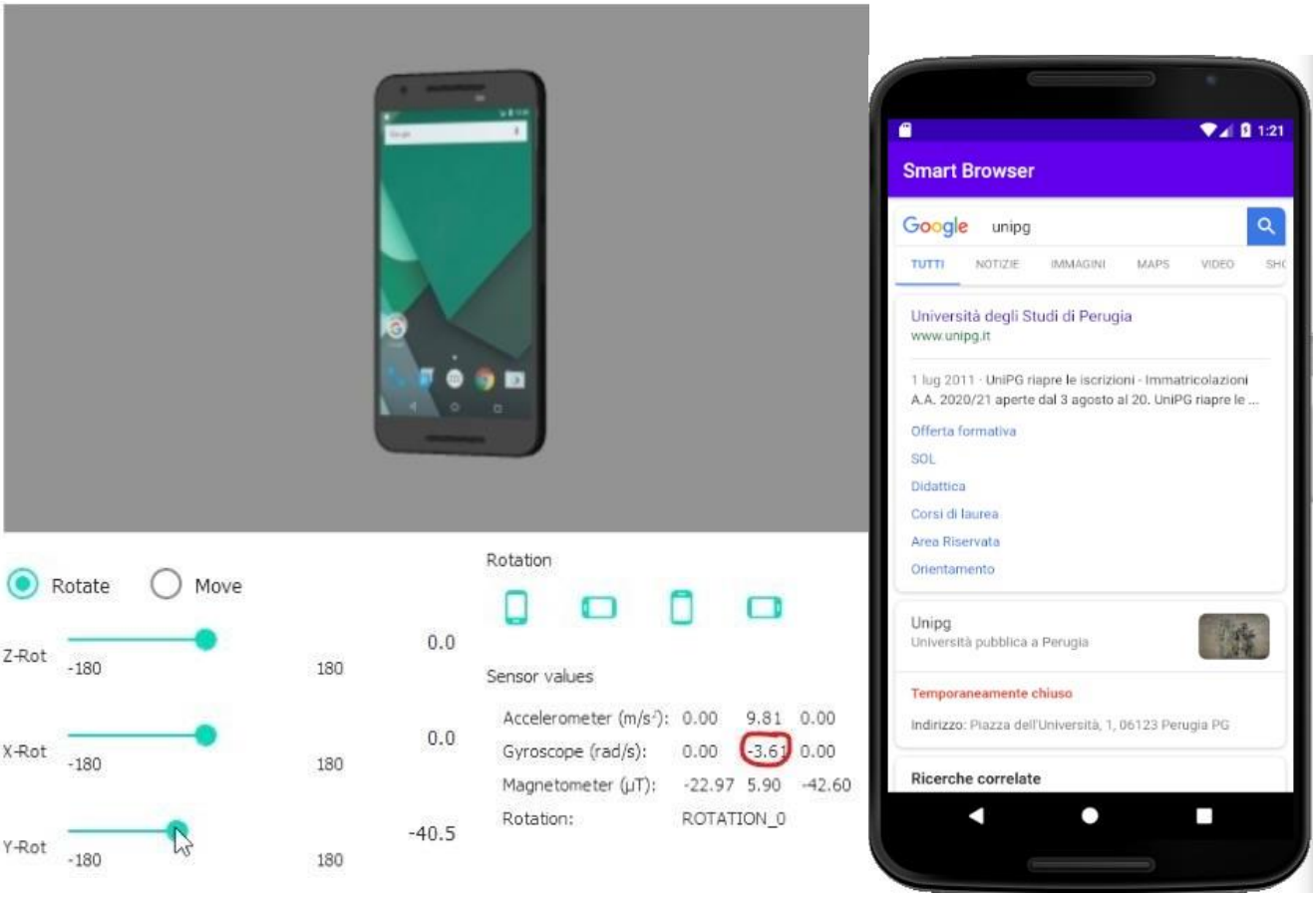


I sensori del nostro telecomando non rilevano nessun movimento, sono in attesa di una gesture.



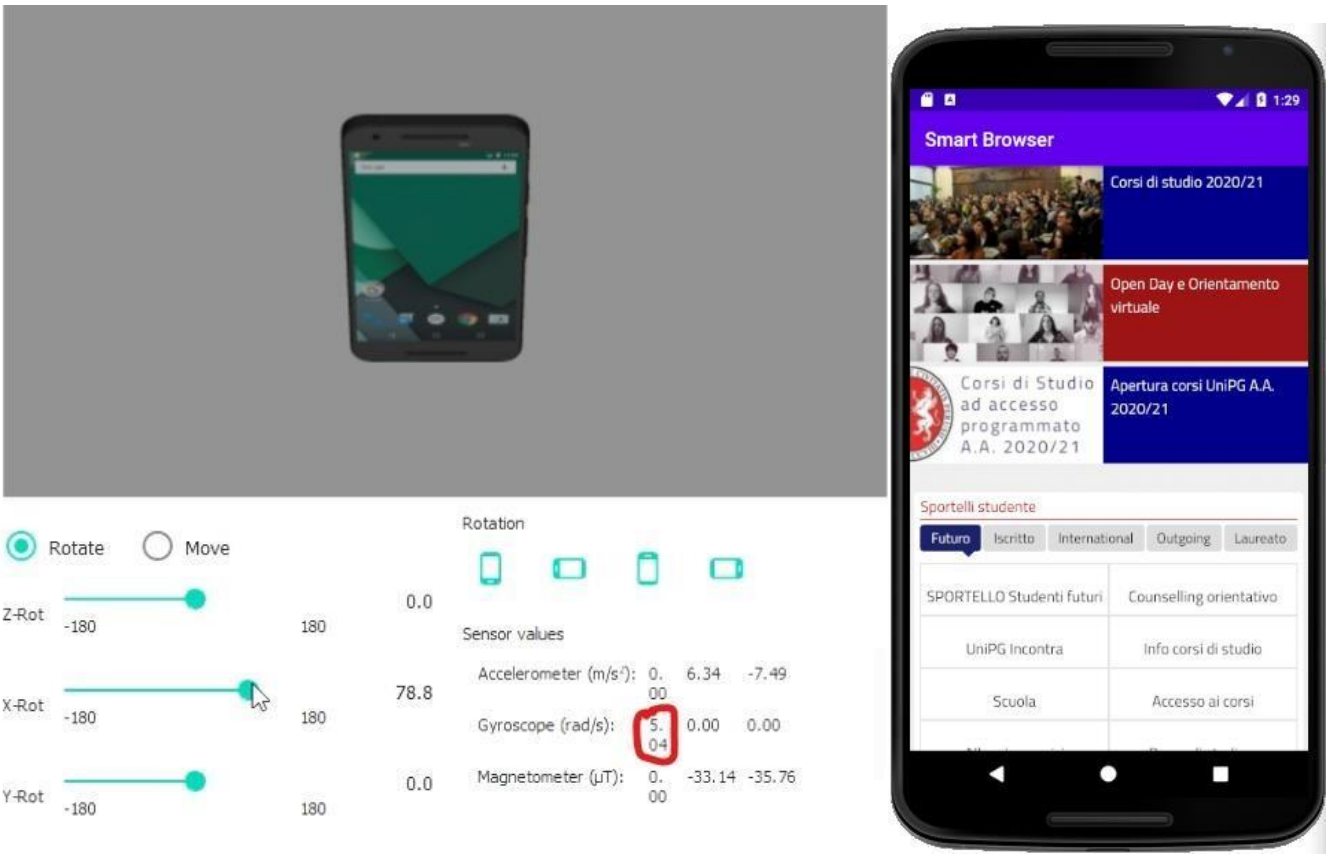

Con l'app smart browser abbiamo effettuato una ricerca per poter verificare ed accertarci dei cambiamenti che verrano effettuati con il telecomando.

## GESTURE SINISTRA (INDIETRO)



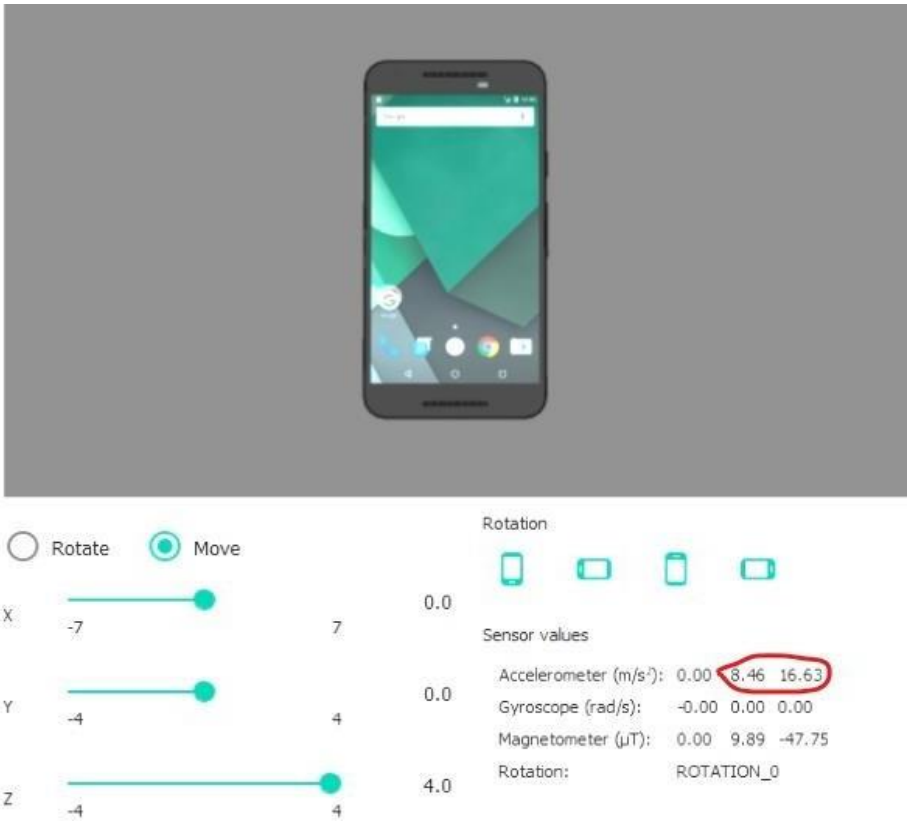

Come previsto la gesture “sinistra”, corrispondente all'azione indietro, ci ha riportato indietro alla ricerca effettuata

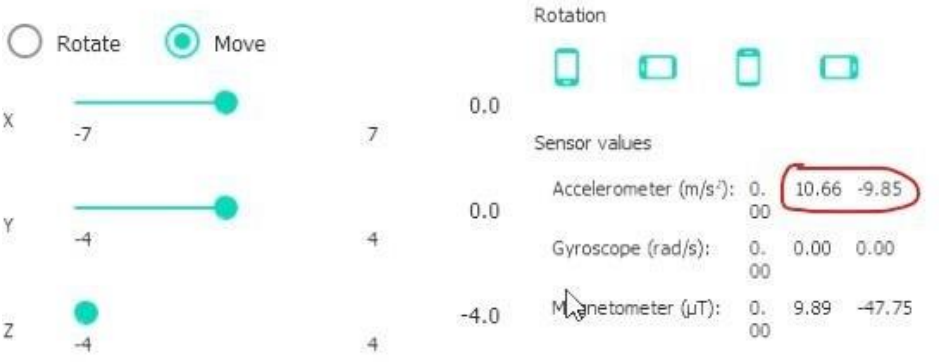
## GESTURE BASSO (SCROLL DOWN)




la gesture “basso” ha fatto scorrere la pagina in giu come pianificato.

# SHAKE (HOME)

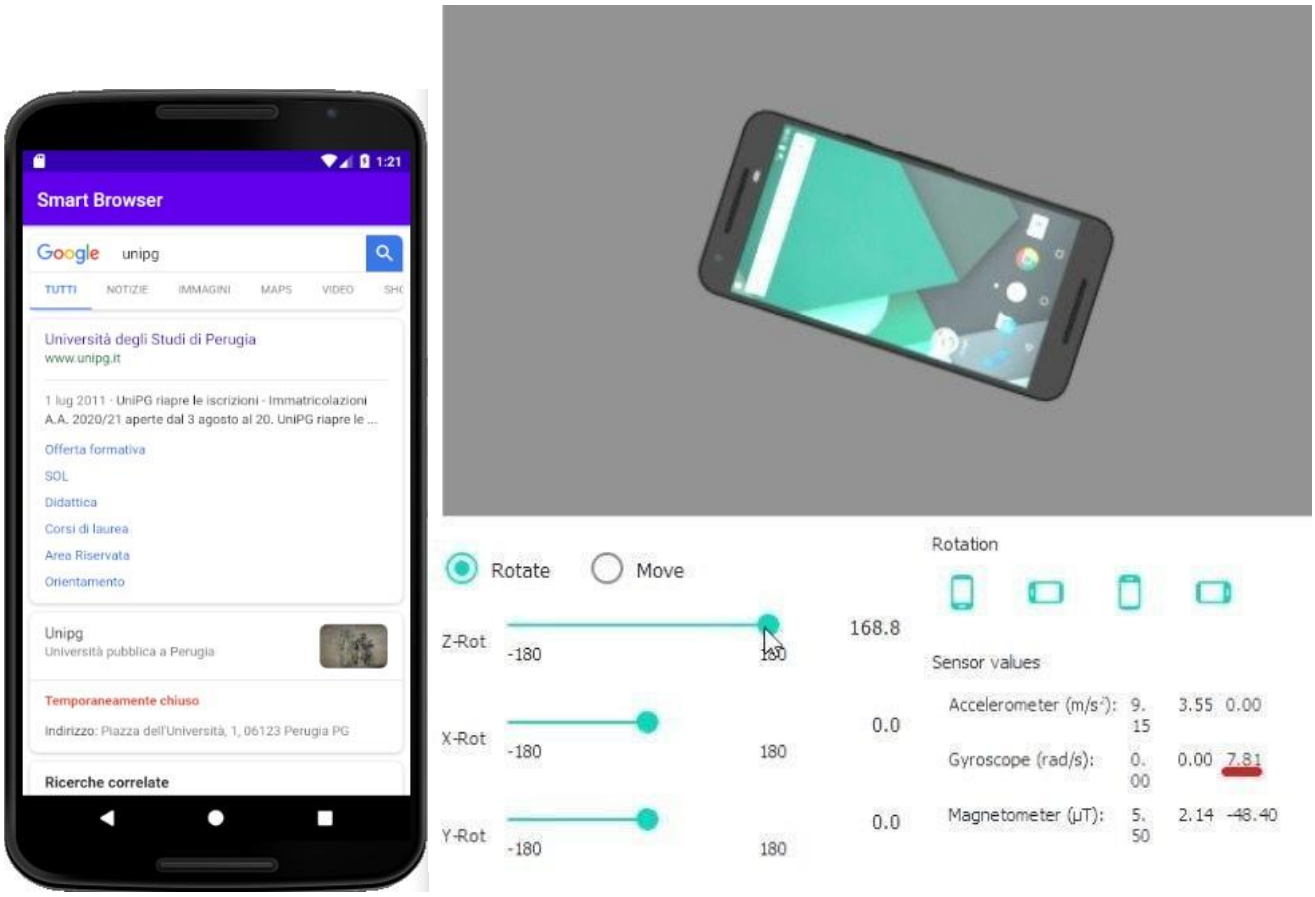

Rotate
Move
X
-7
7
0.0
Y
-4
4
0.0
Z
-4
4
4.0
Rotation
Sensor values
Accelerometer (m/s²): 0.00 8.46 16.63
Gyroscope (rad/s): -0.00 0.00 0.00
Magnetometer (µT): 0.00 9.89 -47.75
Rotation: ROTATION_0





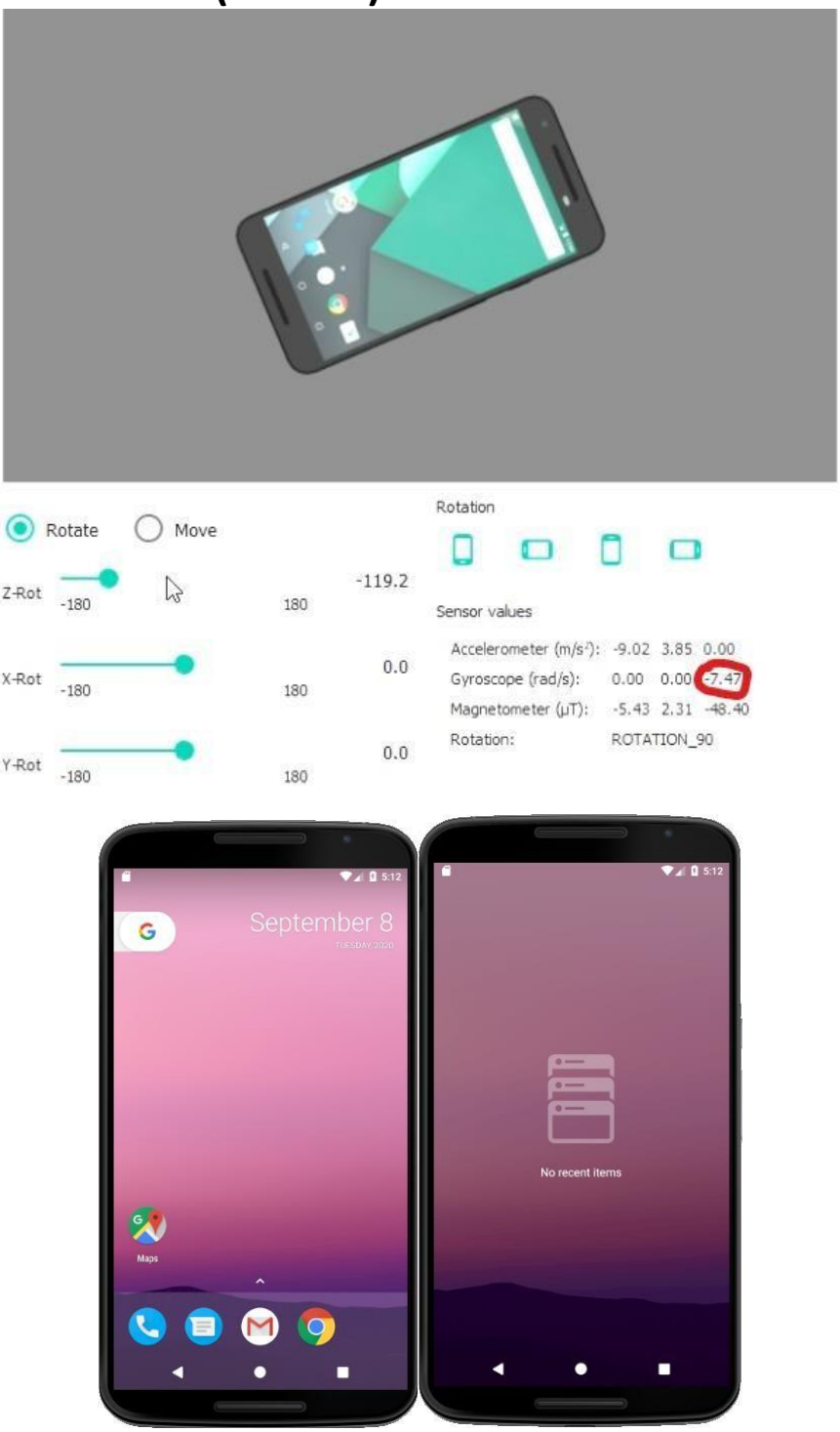

scuotendo il telecomando siamo stati riportati alla home come previsto.

## GESTURE GIRO-ANTIORARIO (AZIONE NON REGISTRATA)




Come previsto, possiamo osservare che la gesture giro anti-orario non causa nessuna azione, essendo una gesture non registrata nel telecomando.

# GESTURE GIRO ORARIO(CHIUDI)




Come previsto la gesture giro orario ha chiuso entrambe le app cancellandole anche dalla cache.