

IAT359 Mobile Computing

Fall 2022

Instructor: Hanieh Shakeri

TA: Parnian Taghipour

lecture 4

- Intents
- Accessing and interacting with the hardware of the device

week 4 check-in

- Quiz 1 done (and grades posted)
- Quiz 2 next week
 - Will be released on Canvas after lecture (2:20PM) and will be open for 24 hours (due Wednesday October 12th at 2:20PM)
 - **Must be completed individually**
 - Will cover content from weeks 3 and 4
- Assignment 1 Part 2 due after this week's lab (submission will be open until Friday October 7th at midnight)
- Project teams should be formed by October 11, milestone 1 due on October 18

goals for today

- Quiz 1 review
- RecyclerView example
- Intents
 - Explicit and implicit
- Interacting with device hardware
- Sensor framework on Android
- Some examples from MobileHCI!
- Quiz 2 prep

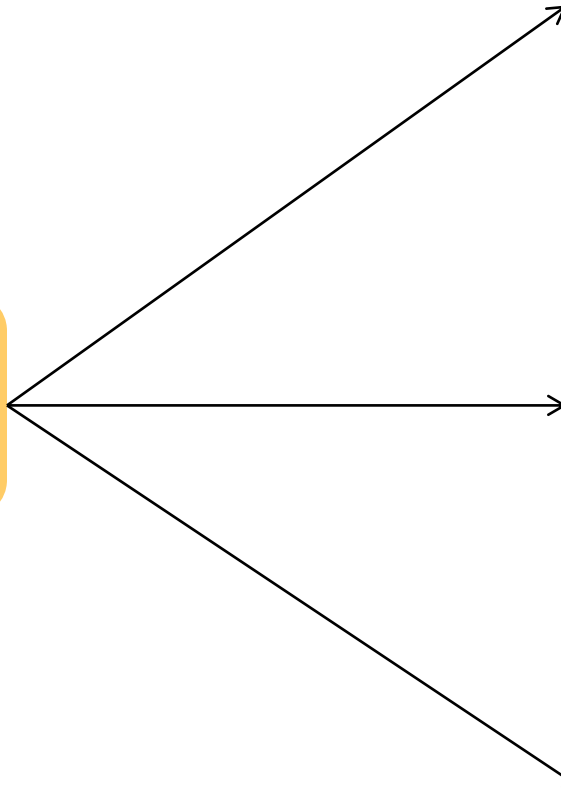
RecyclerView Example

what are intents?

- Asynchronous messages
 - Allow Android components to request functionality from other components of the Android system
 - Can also be used to signal the Android system that a certain event has occurred – other components can register to this event via an intent filter
- Intents are instances of the `android.content.Intent` class

Intent = asynchronous
messaging mechanism

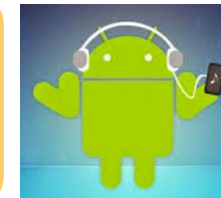
Intents



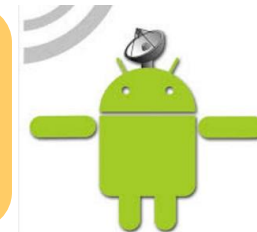
Activity



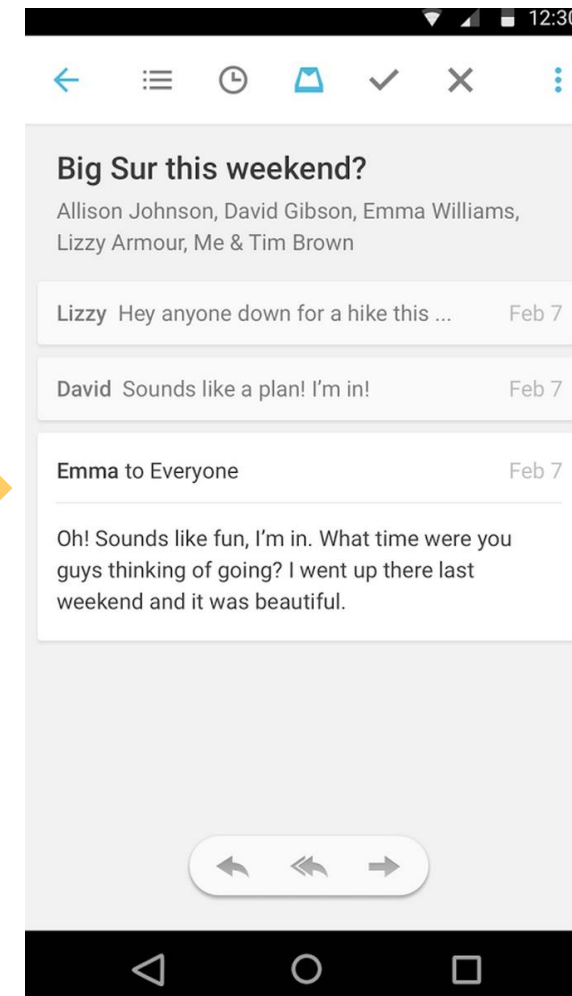
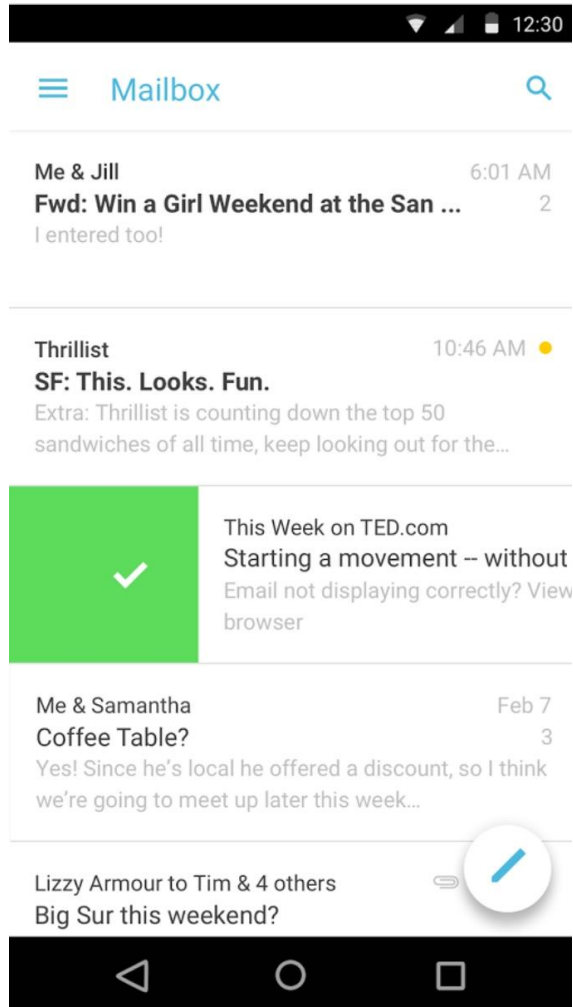
Service



Broadcast
Receiver



example: use intent to start a new activity



example: starting an activity

- Activity sends an intent to the Android system which starts another activity

//start the activity ActivityTwo

```
Intent i = new Intent(this, ActivityTwo.class);
```

```
startActivity(i);
```

context

Activity to be started

example: starting an activity

Explicit Intents

Explicitly define the **component** which should be called by the Android system

- Java class identifier

Implicit Intents

Specify the **action** which should be performed

- Can start any component that can handle the intended action

explicit intents

- Explicitly define the **component** which should be called by the Android system, by using the Java class as identifier



```
Intent i = new Intent (this, ActivityTwo.class);  
i.putExtra("value1", "This value 1 for Activity Two");  
i.putExtra("value2", "This value 2 for Activity Two");
```

Data: putExtra() – key/value pairs – key is always a string

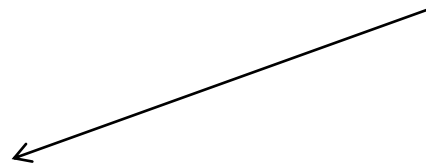
implicit intents

- Specify the **action** which should be performed
- Optionally – provides data for the action
- Can start any component that can handle the intended action

```
Intent i = new Intent(Intent.ACTION_VIEW);
```

```
i.setData(Uri.parse("http://www.sfu.ca"));
```

```
startActivity(i);
```



- typically, web browser is registered to this intent, but other components could register as well

implicit intents

- The Android system searches for all components which are registered for the specific action and the data type
 - If only one component is found, it is started directly
 - Several components: the user will get a selection dialog and can decide which component should be used for the intent

using intents to start other components

- If you send an Intent to the Android system, it needs to be specified which type of component the Intent should be sent – for example, activity or service
- Activities: `startActivity(Intent);`
- Services: `startService(Intent);`
 - We will explore services later in the course

sending the user to another app

- **Implicit intents**
- Based on the **action**
 - E.g. view, edit, send, get something ...
- Intents often include data – might be a Uri
 - E.g. address, email message, ...

```
Uri number = Uri.parse("tel: 5551234");
```

```
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```

more examples

View a map:

// Map point based on address

Uri location =

```
Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+California");
```

```
// Or map point based on latitude/longitude//
```

```
Uri location = Uri.parse("geo:37.422219,-122.08364?z=14"); // z param is zoom level
```

```
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```


more examples

- View a webpage:

```
Uri webpage = Uri.parse("http://www.android.com");
```

```
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```

extra data

- `putExtra()`
- Example: send an email with attachment:

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

```
// The intent does not have a URI, so declare the "text/plain" type
```

```
emailIntent.setType(HTTP.PLAIN_TEXT_TYPE);
```

```
emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[] {"jon@example.com"}); // recipients
```

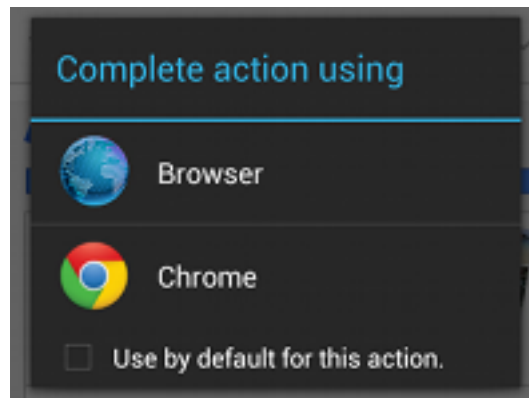
```
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email subject");
```

```
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message text");
```

```
emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("content://path/to/email/attachment")); // You can also  
attach multiple items by passing an ArrayList of Uris
```

verification step

- Verify that there is an app to receive your intent
- `queryIntentActivities()`
 - Returns a list of activities capable of handling the intent
 - If the list is not empty, intent can be safely used
- This check should be performed when the activity first starts



chooser dialog

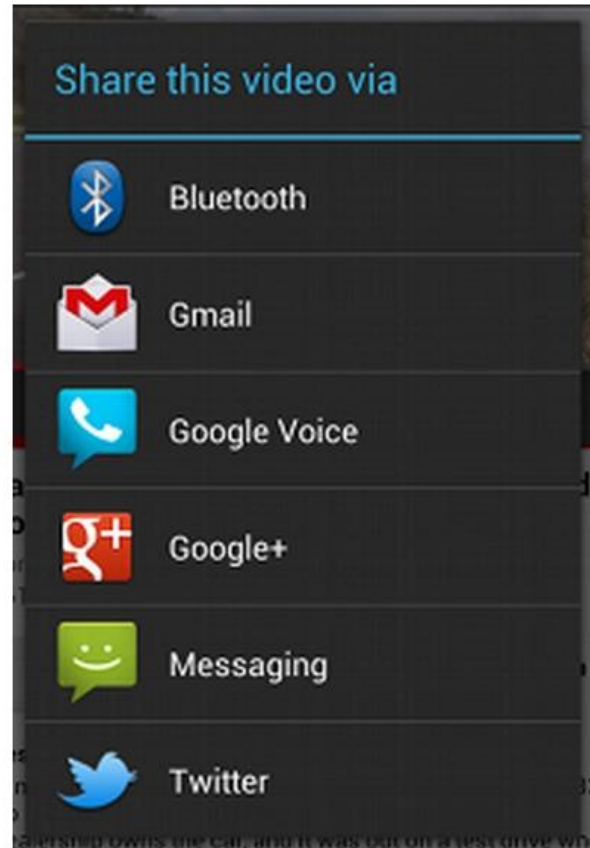


Figure 2. A chooser dialog.

example

```
// Build the intent
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+California");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);

// Verify it resolves
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(mapIntent, 0);
boolean isIntentSafe = activities.size() > 0;

// Start an activity if it's safe
if (isIntentSafe) {
    startActivity(mapIntent);
}
```

retrieving result data from the called activity

- Send the intent: `startActivityForResult()` method
- Once the called activity ends, the `onActivityResult()` method on the sending activity is called
 - The result from the triggered activity can be processed

result from started activity

- Start a camera app and receive the captured photo as result
- Start the Contacts app in order to select a contact; receive the contact details as result

trigger an activity for result

```
public void onClick (View View)
{
    Intent i = new Intent(this, ActivityTwo.class);
    i.putExtra("Value1", "This Value1 for ActivityTwo");
    i.putExtra("Value2", "This Value2 for ActivityTwo");
    //request code
    startActivityForResult(i, REQUEST_CODE);
}
```


request code

- This is the argument that identifies the request
- When the result is received by the initial app, the same request code is provided
 - Initial (sending) app identifies the result based on the request code and determines how to handle this result

send data (result) to the initial activity (on the second activity)

```
@Override  
public void finish()  
{  
    // Prepare data intent  
    Intent data = getIntent();  
    data.putExtra("returnKey1", "This is the result");  
    data.putExtra("returnKey2", "We are sending data back");  
    // Activity finished ok, return the data  
    setResult(RESULT_OK, data);  
    super.finish();  
}
```

receiving the result – back to the first activity

@Override

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {  
        if (data.hasExtra("returnKey1"))  
            Toast.makeText(this,  
data.getExtras().getString("returnKey1"),Toast.LENGTH_SHORT).show();  
    }  
}
```

onActivityResult() method

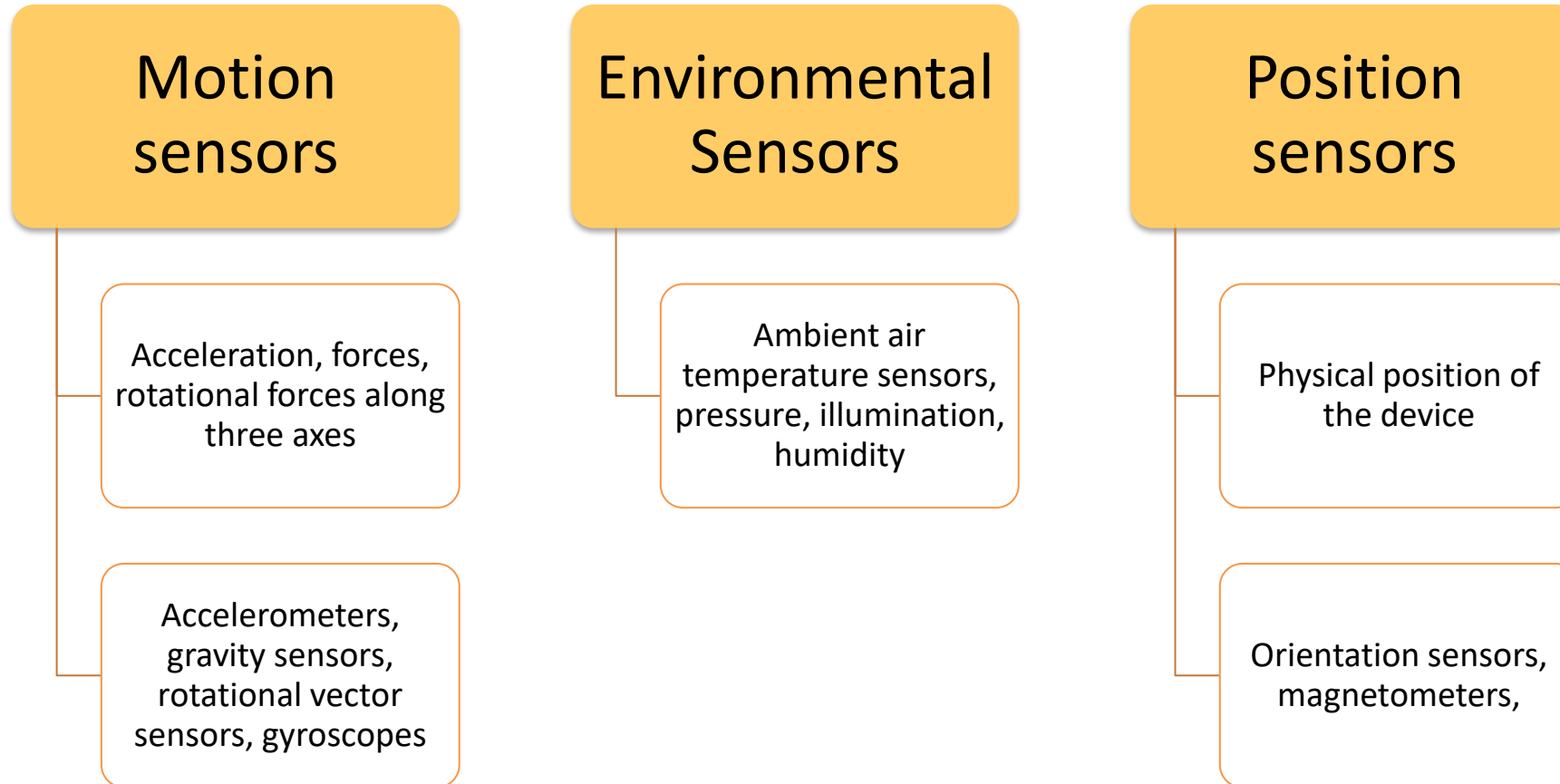
- This method has three arguments:
 - The **request code** that was passed to `startActivityForResult()`
 - A **result code** specified by the second activity
 - RESULT_OK: if operation was successful
 - RESULT_CANCELED: operation failed
 - The **result data**

device hardware

device hardware

- Built-in sensors
 - Motion, orientation, environment, conditions
- Data from sensors
 - Raw data, high precision and accuracy

types of sensors



guidelines when accessing hardware

- No assumptions
 - Hardware may or may not exist
- Always check and verify **optional** features
- Return value checking
- Hardware features = device resources
 - Acquire late
 - Release as soon as done with them
 - Do not drain the device battery by misusing hardware resources

android emulator

- Limited / no support for simulating hardware sensors and the device battery
 - Testing on real device
 - **Bring a physical device to the lab!!!**

android sensors framework

- Part of the android.hardware package

SensorManager

You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

Sensor

You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

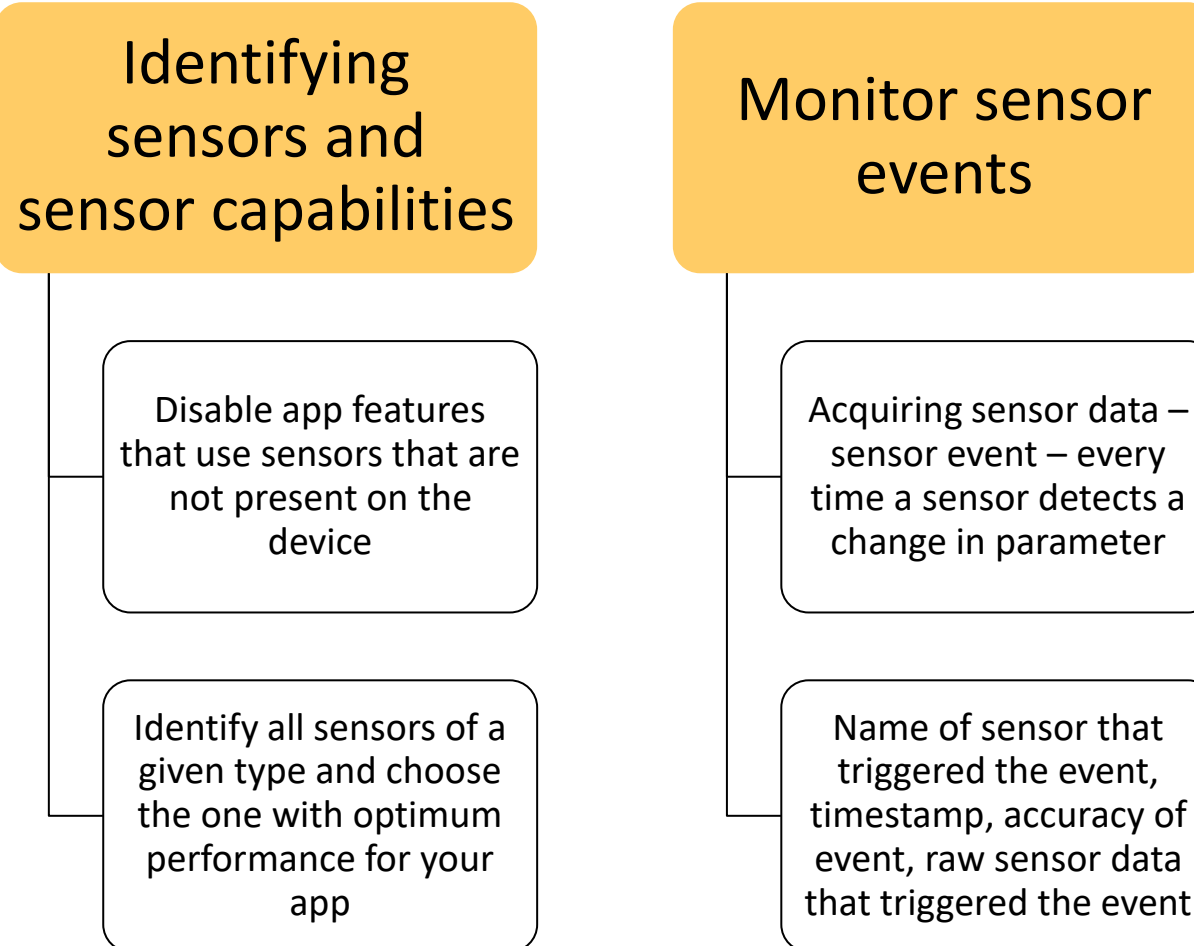
SensorEvent

The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

SensorEventListener

You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

sensor-related tasks



process in working with sensors

Determine which sensors are available on a device.



Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.



Acquire raw sensor data and define the minimum rate at which you acquire sensor data.



Register and unregister sensor event listeners that monitor sensor changes.

identifying sensors and capabilities

- First: get a reference to the sensor device
 - Get an instance of the SensorManager class:

```
private SensorManager mSensorManager;  
...  
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

- Get a listing of every sensor on the device

```
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

checking for a certain sensor

```
private SensorManager mSensorManager;  
...  
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){  
    // Success! There's a magnetometer.  
}  
else {  
    // Failure! No magnetometer.  
}
```

sensors list

- `public List<Sensor> getSensorList(int type);`
- type is one of:
 - TYPE_ACCELEROMETER
 - TYPE_GYROSCOPE
 - TYPE_LIGHT
 - TYPE_MAGNETIC_FIELD
 - TYPE_ORIENTATION
 - TYPE_PRESSURE
 - TYPE_PROXIMITY
 - TYPE_TEMPERATURE
 - TYPE_ALL

monitoring sensor events

- SensorEventListener interface: exposes two methods:
 - **onAccuracyChanged()**: accuracy changes, provides a reference to the Sensor object that changed, and the new accuracy value
 - **onSensorChanged()**: sensor reports a new value –new data, sensor that generated the data, timestamp, accuracy of data

sample code

```
public class SensorActivity extends Activity implements SensorEventListener {  
    private SensorManager mSensorManager;  
    private Sensor mLight;  
  
    @Override  
    public final void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
        mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);  
    }  
}
```

```

@Override
public final void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Do something here if sensor accuracy changes.
}

@Override
public final void onSensorChanged(SensorEvent event) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[0];
    // Do something with this sensor value.
}

@Override
protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}
}

```

AndroidManifest file - sensors

- <uses-feature> tag – indicates which sensors are required by the application

```
<uses-feature android:name="android.hardware.sensor.barometer" />  
<uses-feature  
    android:name="android.hardware.sensor.gyroscope"  
    android:required="false" />
```

acquiring a reference to a sensor

- getDefaultSensor() method of the SensorManager class
- This method takes a sensor type as parameter

```
Sensor accelSensor = sensors.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

reading sensor data

- After we have a valid sensor object, we can read the sensor data periodically
- Sensor values: `SensorEventListener` object – send back the sensor values to the app
 - The app must implement and register the `SensorEventListener` using the `registerListener()` method

methods to implement

- SensorEventListener interface
 - `onAccuracyChanged()` – called whenever the accuracy of a given sensor changes
 - `onSensorChanged()` – called whenever the values of the sensor change
 - This method is used to inspect sensor information

sample: onSensorChanged()

```
@Override
public void onSensorChanged(SensorEvent event) {
    StringBuilder sensorMessage =
        new StringBuilder(event.sensor.getName()).append(" new values: ");

    for (float value : event.values) {
        sensorMessage.append(" [").append(value).append("] ");
    }

    sensorMessage.append(" with accuracy ").append(event.accuracy);
    sensorMessage.append(" at timestamp ").append(event.timestamp);

    sensorMessage.append(".");

    Log.i(DEBUG_TAG, sensorMessage);
}
```

onSensorChanged() method

- One single parameter: SensorEvent object
- The SensorEvent class:
 - Contains all the data about the sensor
 - Which sensor caused the event
 - The accuracy of the sensor
 - The sensor's current readings
 - Timestamp
 - What data to expect from each sensor? -> SensorEvent class documentation in Android SDK

rate of sensor data

- Depending on the sensor in use
 - This rate might be high
- `onSensorChanged()` method – do as little as possible in this method
 - Processing data should not be done within this method

example: using sensors

```
private void createSensor() {  
    sensorManager =  
        (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
  
    showSensors();  
  
    sensorManager.registerListener(sensorEventListener,  
        sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION),  
        SensorManager.SENSOR_DELAY_UI);  
}
```

listing sensors on a device to log

```
private void showSensors() {  
  
    List<Sensor> sensors  
        = sensorManager.getSensorList(Sensor.TYPE_ALL);  
  
    Log.d(TAG, sensors.toString());  
  
    for(Sensor s : sensors) {  
        Log.d(TAG, s.getName() + " - minDelay: "  
            + s.getMinDelay() + ", power: " + s.getPower());  
        Log.d(TAG, "max range: " + s.getMaximumRange()  
            + ", resolution: " + s.getResolution());  
    }  
}
```

android system services

system services

- There is a wide list of services available
 - Power Service
 - KeyGuard Service
 - Vibration Service
 - Alarm Service
 - Sensor Service
 - Audio Service
 - Telephony Service
 - Connectivity Service
 - Wi-Fi Service

power service

- Android runs on limited capabilities devices
- It is crucial to use the battery wisely
- The power service gives us informations about the power of the system
- Get it with:

```
PowerManager pm = (PowerManager)  
getSystemService(Context.POWER_SERVICE);
```

vibration service

- Manages the vibration service
- Get it with:

```
Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
```

- Some methods:
 - `vibrate(long time);`
 - `cancel();`
 - `vibrate(long[] pattern, int repeat);`
- Needs `android.permission.VIBRATE`

alarm service

- Fires an Intent in the future
- Get it with

```
AlarmManager as = (AlarmManager) getSystemService(Context.ALARM_SERVICE);  
// set(int type, long triggerAtTime, PendingIntent operation);
```

- type is one of:
 - ELAPSED_REALTIME
 - ELAPSED_REALTIME_WAKEUP
 - RTC
 - RTC_WAKEUP

alarm service

- More methods
 - `setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation);`
 - Can use `INTERVAL_HOUR`, `INTERVAL_HALF_DAY`
 - `cancel(PendingIntent operation);`
 - Match with `filterEquals(Intent anotherIntent);`

telephony service

- Interacts with calls
- Get it with

```
TelephonyManager tm = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
```

- Ask the device about call information
 - `getCallState()`
 - `getDataState()`
 - `getDataActivity()`
 - `getNetworkType()`
 - `isNetworkRoaming()`

connectivity service

- Check device network state
- Get it with

```
String serId = Context.CONNECTIVITY_SERVICE;  
ConnectivityManager cm = (ConnectivityManager)Context.getSystemService(serviceId);
```

- Check WI-FI, GPRS
- Notify connection changes
- Needs
 - android.permission.ACCESS_NETWORK_STATE
 - android.permission.CHANGE_NETWORK_STATE

wi-fi service

- Manages the Wi-Fi connection
- Get it with

`WifiManager wfm = (WifiManager) getSystemService(Context.WIFI_SERVICE)`

- Check Wi-Fi
 - `getWifiState()`
 - Returns `WIFI_STATE_DISABLED`, `WIFI_STATE_DISABLING`, `WIFI_STATE_ENABLED`, `WIFI_STATE_ENABLING`, `WIFI_STATE_UNKNOWN`
 - `isWifiEnabled()` / `setWifiEnabled()`
- Lists all the configured wifi connections
 - `getConfiguredNetworks()`

wi-fi service

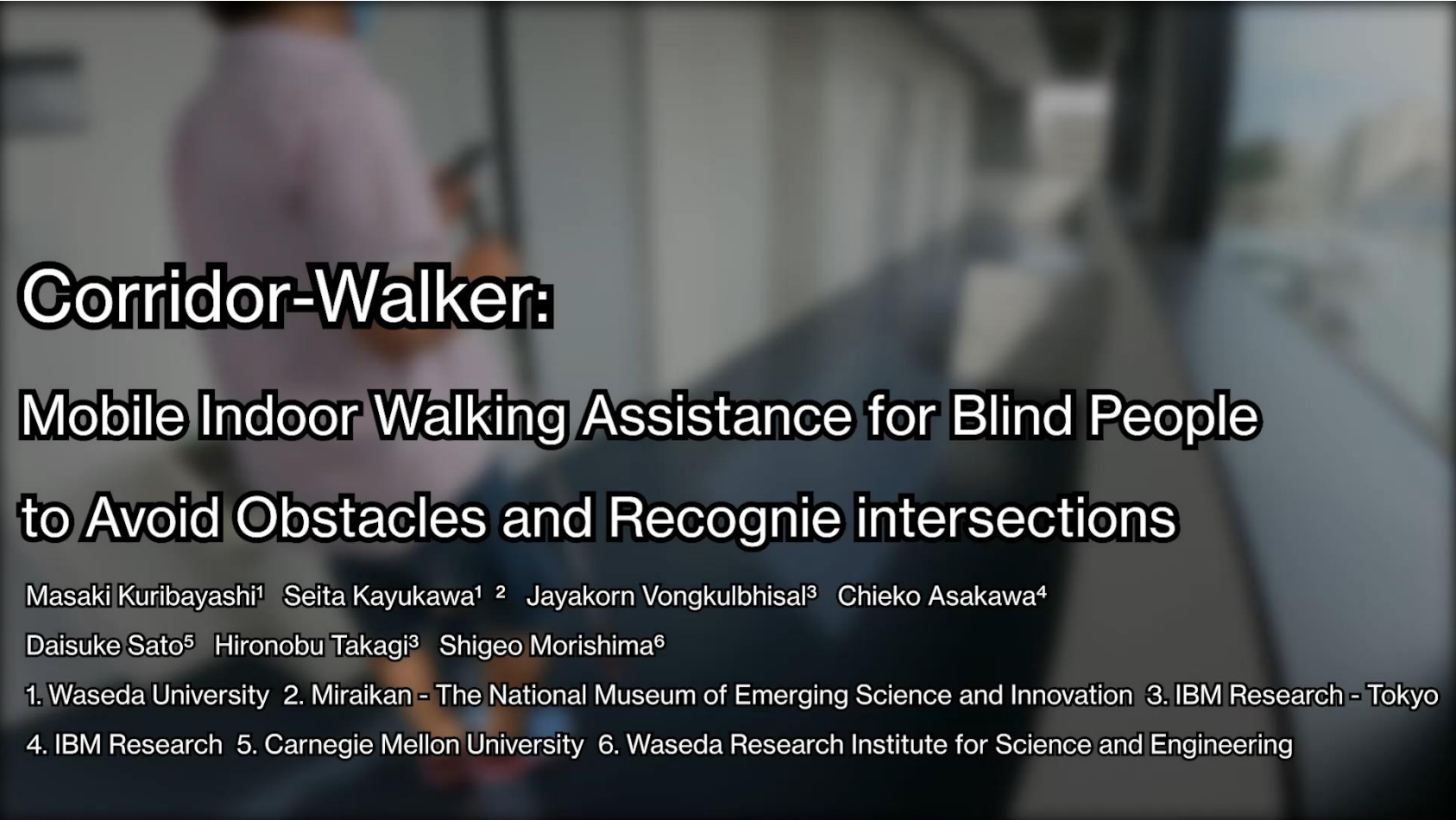
- Check/edit wi-fi connection
 - addNetwork(WifiConfiguration config)
 - updateNetwork(WifiConfiguration config)
 - removeNetwork(int netid)
- Scan for wi-fi networks
 - startScan()
- Be notified about wi-fi changes
 - Broadcast Intent: SCAN_RESULTS_AVAILABLE_ACTION
 - Call getScanResults()

MobileHCI

The ACM International Conference on Mobile Human-Computer Interaction

- International – took place in Vancouver this year!





Corridor-Walker:

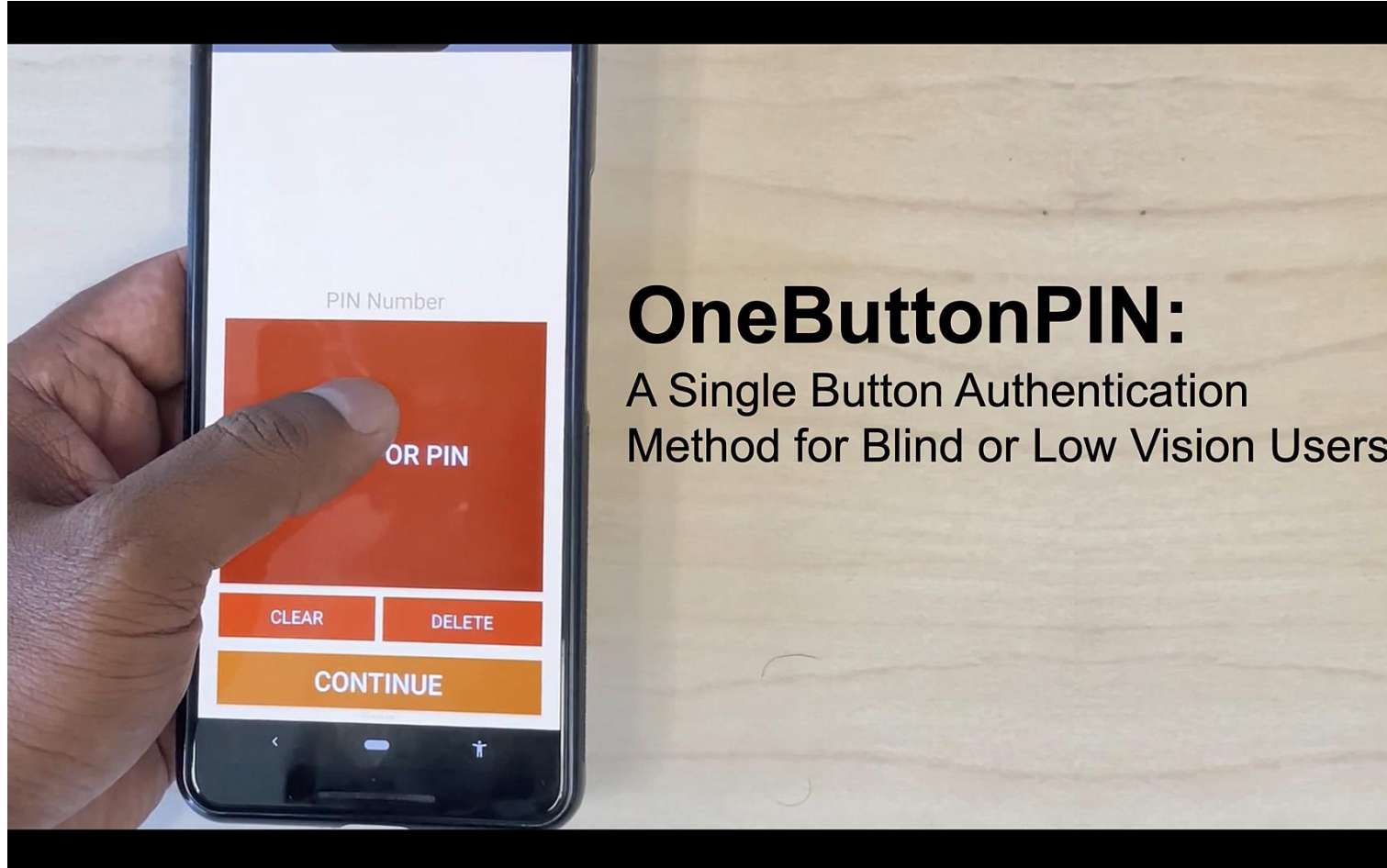
Mobile Indoor Walking Assistance for Blind People to Avoid Obstacles and Recognize intersections

Masaki Kuribayashi¹ Seita Kayukawa^{1 2} Jayakorn Vongkulbhisal³ Chieko Asakawa⁴

Daisuke Sato⁵ Hironobu Takagi³ Shigeo Morishima⁶

1. Waseda University 2. Miraikan - The National Museum of Emerging Science and Innovation 3. IBM Research - Tokyo

4. IBM Research 5. Carnegie Mellon University 6. Waseda Research Institute for Science and Engineering



OneButtonPIN:

A Single Button Authentication
Method for Blind or Low Vision Users

Quiz 2 prep

- Week 3:
 - Button click event in Android
 - View structure in Android
 - Linear Layout
 - RecyclerView
- Week 4:
 - Implicit and Explicit Intents
 - Accessing and using device hardware

resources

- Intents and intent filters:
<http://developer.android.com/guide/components/intents-filters.html>
- Android Sensors Overview:
http://developer.android.com/guide/topics/sensors/sensors_overview.html
- Android Sensors Framework:
http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-identify