Lecture 14

New input devices: sensors on mobile computing devices

Sensors

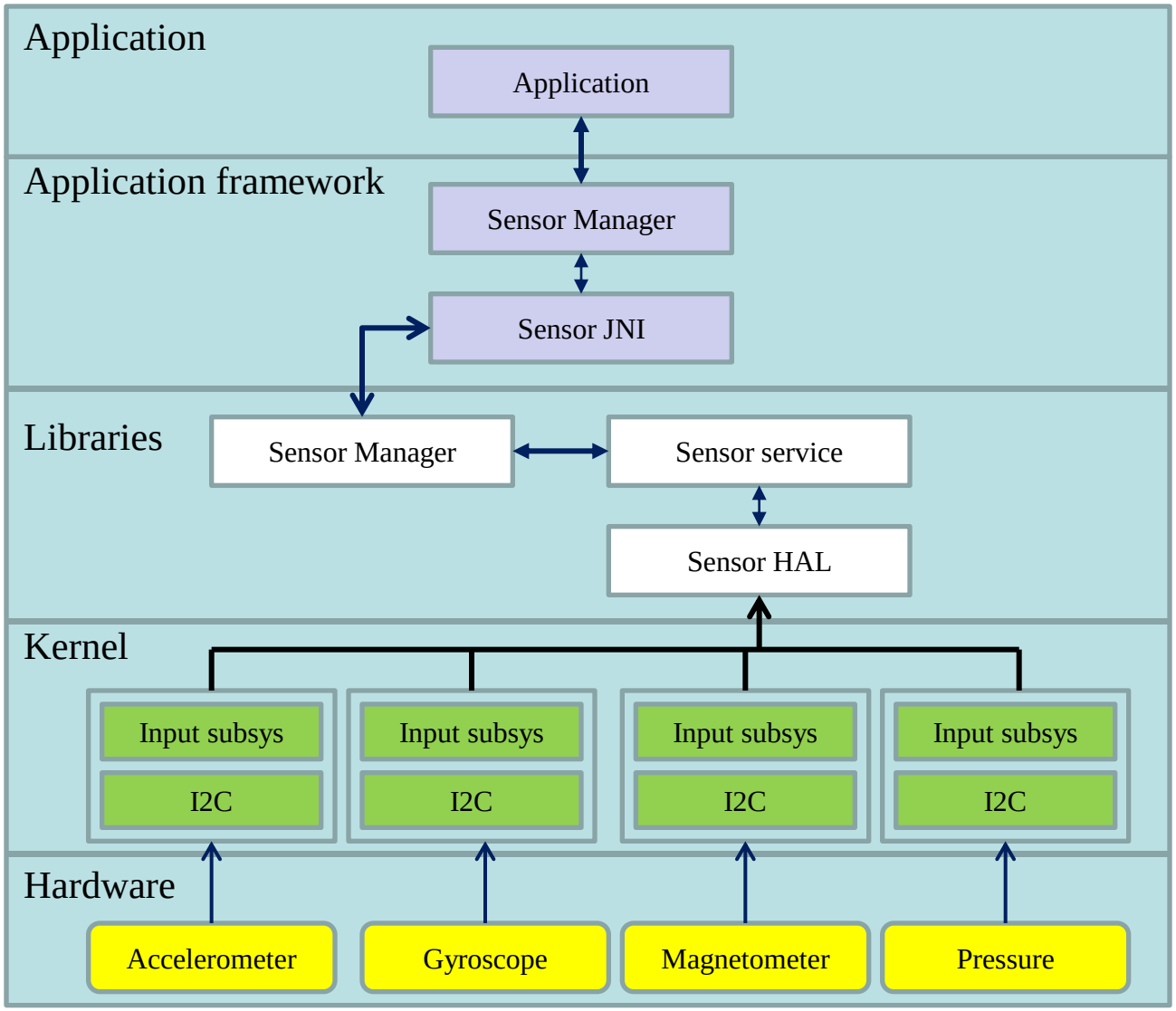
- Mobile devices support three broad categories of sensors:
  - motion sensors;
  - environmental sensors;
  - position sensors.
- The vast majority of sensors have an *official sensor type*. There can be several sensors of the same type. For example, two proximity sensors or two accelerometers.

- Some of the sensors are hardware based and some are software based sensors. Whatever the sensor is, the operating system allows users to get the raw data from these sensors and use it in applications.

Sensor Type	Description	Application
Accelerometer	Measures the acceleration force in m/s2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection
Temperature Sensor	Measures the ambient room temperature in degrees Celsius (°C).	Monitoring air temperatures
Gyroscope	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection
Magnetometer	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass, metal detection.
GPS	Gives the device's position longitude and latitude coordinate.	Location detection
Light Sensor	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
Heart rate sensor	Heart rate	Health

Camera, fingerprint sensor, microphone, and touch screen are not included in the table as they have their own reporting mechanism – more channels of higher data rate.

Android sensor subsystem



Framework's role

- The framework is responsible of linking several applications to the HAL (Hardware Abstraction Layer), otherwise only one application would connect to a sensor at a time. Its operation is the following:
  - When the first application registers to a sensor, the framework sends a request to the HAL to *activate the sensor*.
  - When additional applications register to the same sensor, the framework takes into account requirements from each application and sends the updated requested parameters to the HAL:
    - the *sampling frequency* will be the maximum of the requested sampling frequencies, meaning some applications will receive events at a frequency higher than the one they requested;
    - the *maximum reporting latency* will be the minimum of the requested ones.
  - When the last application un-registers from a sensor, the frameworks sends a request to the HAL to *deactivate the sensor*.

Use of sensors in applications

1. The first thing is to instantiate the object of *SensorManager* class. It can be achieved as follows:

```
SensorManager sMgr;
sMgr = (SensorManager)this.getSystemService(SENSOR_SERVICE);
```

   The SensorManager class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information.

2. Then, instantiate the object of Sensor class by calling the getDefaultSensor() method of the SensorManager class. The Sensor class provides various methods that let you determine a sensor's capabilities. Its syntax is given below:

```
Sensor light;
light = sMgr.getDefaultSensor(Sensor.TYPE_LIGHT);
```

The SensorEvent class is used to create a sensor event object, which provides information about a sensor event.

3. Register the sensor listener and override two methods which are *onAccuracyChanged* and *onSensorChanged*. Its syntax is as follows

```
sMgr.registerListener(this, light,SensorManager.SENSOR_DELAY_NORMAL);
public void onAccuracyChanged(Sensor sensor, int accuracy) {
 }
public void onSensorChanged(SensorEvent event) {
}
```

Sensor events

- The system uses the *SensorEvent* class to create a sensor event object, which provides information about a sensor event. A sensor event occurs every time a sensor detects a change in the parameters it is measuring.
- 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.
- The *SensorEventListener* interface is used to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

Event reporting modes

- Sensors can generate events in different ways called reporting modes; each sensor type has one and only one reporting mode associated with it.
  - *Continuous*: events are generated at a constant rate. Example: accelerometers, gyroscopes.
  - *On-change*: events are generated only if the measured values have changed. Activating the sensor at the HAL level also triggers an event, meaning the HAL must return an event immediately when an on-change sensor is activated. Example: the step counter, proximity, and heart rate sensor types.
  - *One-shot*: upon detection of an event, the sensor deactivates itself and then sends a single event through the HAL. No other event is sent until the sensor is reactivated. One-shot sensors are sometimes referred to as trigger sensors.

Examples

- To identify the sensors that are on a device you first need to get a reference to the sensor service, then list them:

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

- If you want to list all sensors of a certain type, replace TYPE_ALL by that type:

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

- In addition to listing the sensors that are on a device, you can use the public methods of the Sensor class to determine the capabilities and attributes of individual sensors. This is useful if you want your application to behave differently based on which sensors or sensor capabilities are available on a device. For example, you can use the getResolution() and getMaximumRange() methods to obtain a sensor's resolution and maximum range of measurement. getPower() method can be used to obtain a sensor's power requirements.

Further developments

- External sensors can be integrated by an *application-level driver framework* that enables reuse of sensor-specific code between applications.
- A single interface can control virtually any kind of sensor (both external and built-in) and reduces the amount of code needed to access a sensor.
- Additionally, the sensor framework automatically multiplexes the communication channels allowing different types of sensors to be used simultaneously by an application. For example, an application can easily use two USB and three Bluetooth sensors simultaneously to record several phenomena at once.

- Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. VanOrden and G. Borriello, Open Data Kit Sensors: A Sensor Integration Framework for Android at the Application-Level, MobiSys'12, June 25–29, 2012, Low Wood Bay, Lake District, UK
- https://source.android.com/devices/sensors/